

PYTHON

NOTES

CONTENTS

PageNo :

CHAPTER-1 : INTRODUCTION TO PYTHON

A. Real time applications/projects /product Developed by using python	13
B. History of python	13
C. Versions of python	14
D. Features of python	14
E. Data Representation in python	20

CHAPTER-2 : DATA TYPES IN PYTHON

A. Fundamental category Datatypes	24
i>Int	24
ii>Float	29
iii>Bool	30
iv>Complex	31
B. Sequence Category Datatypes	33
i>str (PART-1)	34
ii>Bytes	56
iii>Bytearray	58
iv>Range	58
c. List Category Datatypes	64
i>list	65
ii>Tuple	81
D. Set category Datatypes	87
i>Set	87
ii>Frozen Set	96
E. Dict category Datatypes	103
i>Dict	
F. None category Datatypes	118
i>None type	118

CHAPTER-3 : Number of Approaches to develop the Programs in python lang

	118
A. Definition of programs	119
B. Two types of approaches	119
i. Interactive approach	120
ii. Batch mode approach	126
C. Display the result of python program On the console	128
D. Reading the data or input Dynamically From keyboard	129

CHAPTER-4: Operators and Expressions in Python

A. Arithmetic Operators	
B. Assignment Operator	130
C. Relational Operators (Comparision Operators)	132
D. Logical Operators (Comparision Operators)	132
E. Bitwise Operators	137
F. MemberShip Operators	
i) in operator	
ii) not in operator	
G. Identity Operators	
i) is operator	
ii) is not operator	
H. Python ternary operators	144
I.String Handling (PART-2)	146

CHAPTER-5: FLOW CONTROL STATEMENTS IN PYTHON

	160
A) Purpose of Flow Control Statements in Python.	160
B) Types of Flow Control Statements in Python	160
I) Conditional (OR) Selection (OR) Branching Statements	160
i) Simple if statement	161
ii) if..else statement	
iii) if..elif..else statement	
iv) match case statement(Python Version3.10 Onwards)	167
=>Programming Examples	
II) Looping OR Iterative OR Repetative Statements	170
i)while Loop OR while ... else Loop	171
ii)for Loop OR for ...else loop	173
=>Programming Examples	
III) Transfer Flow Control Statements	179
i) break	180
ii) continue	182
iii) pass	
iv) return	
=>Programming Examples	
C) Combined Programming Examples on Conditional, Looping and Transfer Flow Control Statements.	

D) Inner OR Nested Loops	184
i) for loop in for loop	184
ii) while loop in while loop	185
iii) for loop in while loop	186
iv) while loop in for loop	185
=>Programming Examples	
<hr/>	
CHAPTER-6: FUNCTIONS IN PYTHON	193
A) FUNCTIONS	193
i) Purpose of Functions	193
ii) Types of Languages	193
iii) Definition of Function	
iv) Advantages of Functions	193
v) Parts of Functions	193
vi) Phases in Functions	193
vii) Syntax for Defining Functions	194
viii) Number of Approaches to Define Functions	194
ix) Programming Examples	
<hr/>	
B) Parameters and Arguments	198
Types of Arguments	200
i) Positional Arguments	200
ii) Default Arguments	200
iii) Keyword Arguments	202
iv) Variable Length Arguments	205
v) Keyword Variable Length Arguments	
vi) Programming Examples	
<hr/>	
C) Local and Global Variables	208
i) Programming Examples	
ii) Global Keyword	212
iii) Programming Examples	
iv) globals()	213
v) Programming Examples	
<hr/>	
D) Anonymous Function OR Lambda Functions	215
i) Implementation of Anonymous Function OR LambdaFunctions	215
ii) Programming Examples	
<hr/>	
E) Special Functions in Python	217
i) filter()	217
ii) map()	
iii) reduce()	219
iv) Programming Examples	

CHAPTER-7 : MODULES IN PYTHON	223
A) Modules	223
i) Purpose of Modules	224
ii) Types of Modules	224
iii) Steps for development of Modules	224
iv) Number of approaches to Re-Use Modules	
a) By Using import statement	
b) By using from...import statement	
v) Programming Examples	
B) Reloading the modules	231
i) implementation of imp modules OR importlib modules	
ii) Programming Examples	
C) Case Studies related to modules	
CHAPTER-8 : PACKAGES IN PYTHON	232
A) PACKAGES	232
i) purpose of packages	233
ii) create a package	233
iii) re-use package	234
B) differences between functions ,modules and packages	237
CHAPTER-9 : EXCEPTION HANDLING IN PYTHON	237
A) EXCEPTION HANDLING	237
i) Purpose of Exception Handling	237
ii) Types of Errors	237
a) Compile Time Errors	237
b) Logical Errors	238
c) Runtime Errors	238
iii) Definition of Exception	238
iv) Definition of Exception Handling	239
v) Building points in Exception handling	239
vi) Types of Exceptions	240
a) Pre-Defined Exceptions	240
b) Programmer OR User OR Custom Defined Exceptions	240
vii) Keywords Required for Handling Exceptions	
a) try b) except	
c) else d) finally e) raise	
viii) Syntax for Handling Exceptions	

ix) Programming Examples	
x) Various Forms of except block	241
xi) Programming Examples	
<hr/>	
B) Development of Programmer-Defined Exceptions	246
i) Programming Examples	
<hr/>	
C) Implementation of ATM Case Study with Pre-defined and Programmer Defined Exceptions	
<hr/>	
CHAPTER-10: FILES IN PYTHON	250
A) FILES	250
i) Purpose of Files	250
ii) Types of Applications in Files	250
a) Non-Persistent Applications	250
b) Persistent Applications	250
iii) Definition of Files	251
iv) Operations on Files	251
a) Write Operation with Steps	252
b) Read Operation with Steps	252
v) Types of Files	253
vi) File Opening Modes	254
a) r b) w c) a d) r+ e) w+ f) a+ g) x h) x+	254, 255
vii) Syntax for Opening the Files	255
a) By using Open()	256
b) By using " with open() as "	256
viii) Programming Examples.	
<hr/>	
B) Pickling (Object Serialization) and Un-Pickling (Object De-Serialization)	263
i) pickle module	263
ii) Implementation (steps) of Pickling and Un-Pickling Concept	264
iii) Programming Examples	
<hr/>	
C) Working with CSV Files	269
i) csv module	
ii) CSV File Operations	
a) read operation with csv.reader()	270
b) write operation with csv.writer()	269
c) Dict Read Operation with csv.DictReader()	270
d) Write Operation with csv.DictWriter()	269
iii) Programming Examples	
<hr/>	

D) Working with OS Based Operations	274
i) os module	274
ii) Programming Examples	
<hr/>	
CHAPTER-11: PYTHON DATABASE COMMUNICATION	278
A) PDDB	278
i) Limitations of Files	
ii) Advantages of Database Softwares	
iii) Purpose of Python Database Communication (PDDB)	
iv) Setup for Python Database Communication (PDDB)	
v) Steps for Developing Python Database Communication (PDDB) Applications/ Programs	279
a) Communication with Oracle Data Base	280
b) Communication with MySQL Data Base	292
<hr/>	
B) Queries and Types	
a) DDL Queries (create, alter, drop)	281
b) DML Queries (insert, delete, update)	283
c) DRL Queries (select)	288
i) Executing DDL Statements by Python Program	
a) Programming Examples	
ii) Executing DML Statements by Python Program	
a) Programming Examples	
iii) Executing DRL Statements by Python Program	
a) Programming Examples	
iv) Meta Data Programming by Python Programming	
a) Programming Examples	
<hr/>	
C) Case Studies	
<hr/>	
CHAPTER-12: Object Oriented Principles or Features or Concepts	304
A) OOPS	
i) What are the advantages of OOPs	
ii) List of Object Oriented Principles	305
<hr/>	
B) Classes	305

A) Importance and purpose of Classes concept	305
B) Syntax for Defining Class	305
c) Types of Data Members	
a) Instance Data Members	306
b) Class Level Data Members	306
D) Types of Methods	309
a) Instance Method	313
b) Class Level Method	314
c) Static Method	314
E) What is "self" and "cls"	
vi) Programming Examples	
C) Object	330
i) Importance and purpose of Object Concept	330
ii) Syntax for creating Object	330
iii) Programming Examples	
iv) PRAMMING Examples related to pickling and Data base communication with Classes and objects.	
<hr/>	
D) Constructors in OOPs	327
i) Importance and purpose of Constructors	327
ii) Types of Constructors	327
a) Default Constructors	327
b) Parameterized Constructors	328
iii) Rules for Constructors	327
iv) Programming Examples	
<hr/>	
E) Destructrors in OOPs with Garbage Collector	331
i) Importance and purpose of Detstructrors	331
ii) Syntax for defineng Detstructrors	332
iii) Internal flow of Detstructrors	332
iv) Relation between Detstructrors and Garbage Collector	332
v) gc module	
vi) Progammimg Examples	
<hr/>	
F) Data Encapsulation and Data Abstraction	337
i) Importance and purpose of Data Encapsulation	337
ii) Importaance and purpose of Data Abstraction	338
iii) Implementation of data encapsulation	337

and Data Abstraction
iv) Programming Examples

G) Inheritance	341
i) Importance and purpose of Inheritance	342
ii) Types of Inheritances	342
a) single	343
b) multi level	343
c) hierarchical	343
d) multiple	343
e) Hybrid	343
iii) Syntax for Inheritance	
v) Programming Examples	
H) Method Overriding in OOPs	347
i) Importance and purpose of Method Overriding	347
ii) Memory management in Method Overriding	347
iii) Programming Examples	
I) Polymorphism	347
i) Importance and purpose of Polymorphism	348
ii) Difference between Polymorphism and Inheritance	347
iii) Method Overriding with Polymorphism	347
iv) Constructor Overriding with Polymorphism	
v) super() and class name approaches in Polymorphism	
vi) Programming Examples	
CHAPTER-13:NUMPY MODULE IN PYTHON	353
A) NUMPY	353
i) Purpose of Numpy	
ii) Setup of numpy	354
iii) History of Numpy	354
iv) Advantages of Numpy	354
v) ndarray array in numpy	
vi) Differences between traditional list of Python and ndarray of numpy	355
vi) Various Programming Examples	

B) Basic Indexing and Slicing Operations on ndarray of Numpy	375
i) Advanced Indexing and Slicing Operations on ndarray of Numpy	376
ii) Filtering the Elements of ndarray of Numpy (OR) Random Selection pf Elements	
<hr/>	
C) Arithmetic Operations on Ndarray (OR) Matrix Operations on ndarray	381
i) Stastical Operations on ndarray	386
ii) Copy() and view() of ndarray	393
<hr/>	
D) Adding the Elements to ndarray of numpy	391
i) Deleting the Elements from ndarray of numpy	391
ii) Updating the the Elements of ndarray of numpy	391
<hr/>	
CHAPTER-14 : PANDAS IN PYTHON	398
A) PANDAS	398
i) Introduction to pandas	398
ii) Data structures in pandas	398
a) Series	401
b) Data Frame	
iii) Formula sheet of data frames	450
iv) DataFrame groupby()	456
<hr/>	
CHAPTER-15 : MULTI THREADING IN PYTHON	456
A) MULTI THREADING	456
i) Purpose of Multi Threading	456
ii) Types of applications	456
a) Process Based Applications	475
b) Therad Based Applications	475
iii) Definition of Thread	
iv) Purpose of Thread	
v) Programming Examples	
Non-Therading (Process Based) and Thread Based	
<hr/>	
B) Module Name for Developing Thread Based Applications	460

<p>---therading module</p> <ul style="list-style-type: none"> i) Detailed Discusstion about "therading" module ii) Programing Examples. 	
<hr/>	
<p>C) Synchroniation of threads</p> <p>(OR) Dead Lock Elimination</p> <ul style="list-style-type: none"> i) Implemenation of Lock class of thread module ii) Programming Examples 	473
<hr/>	
<p>CHAPTER-16:NETWORK PROGRAMMING IN PYTHON</p> <p>A) NETWORK PROGRAMMING</p> <ul style="list-style-type: none"> i) Purpose of Network Programming ii) Def of Network. iii) DNS, IP Address, Server, Client iv) Steps for developing Server Side Applications v) Steps for developing Client Side Applications vi) Module Name Required for Developing Server and Client Applications(socket) vii) Functions in socket module 	474
<hr/>	
<p>B) Programming Examples on NetWork Programming</p> <ul style="list-style-type: none"> i) 2-Tier Applications ii) 3-Tier Applications 	480
<hr/>	
<p>CHAPTER-17:RANDOM MODULE IN PYTHON</p> <p>A) Purpose of random module</p> <p>B) Pre-defined functions</p> <ul style="list-style-type: none"> i) randrange() ii) randint() iii) random() iv) uniform() v) choice() vi) shuffle() vii) sample() 	481
<hr/>	
<p>CHAPTER-18:Regular Expressions in Python</p> <p>A) Regular Expressions</p> <ul style="list-style-type: none"> i) Purpose of Regular Expressions ii) Deinition of Regular Expressions iii) Applications of Regular Expressions 	485

iv) Module Name of Regular Expressions (re)

B) Functions in re module

- i) search()
 - ii) findall()
 - iii) finditer()
 - iv) start()
 - v) end()
 - vi) group()
 - vii) sub()
 - viii) Programming Examples
-

C) Programmer-Defined Character Classes 485

- i) Programming Examples
 - ii) Pre-Defined Character Classes 495
 - iii) Programming Examples
 - iv) Quantifiers 495
 - v) Programming Examples
-

**D) Combined Programming Examples
on Programmer-Defined, Pre-Defined
and Quantifiers.**

=====

Real Time Applications / Projects / Products Developed by Using Python

=====

=>By Using PYTHON Programming Lang, In real time, we develop the following Applications / products.

1. Development of Web Applications
 - a) By using JAVA Lang---->Servlets, JSP, Hibernate, Spring.....etc
 - b) By Using C#.net Lang----->ASP.net, ASP.net Micro Service.....etc
 - c) By using PYTHON Lang--->Django, Bottle, Falsk, Pyramid....etc
 2. Gaming Applications
 3. Artificial Inteligence Application (Machine Learning, Deep Learning)
 4. Desktop Application (GUI Applications)
 5. Image Processing Applications.
 6. Text Processing Applications
 7. Business Applications (Amazon, Uber, umobile, flipcarrt)
 8. Audio and Video Based Applications
 9. Web Scrapping /Harvesting Applications
 10. Data Visualization
 11. Scientific Application
 12. Complex Math Calculations
 13. software development
 14. Operating System Installers
 15. CAD and CAM Based Applications
 16. Embedded System Applications
 17. Automation of Testing
 18. Language Development
 19. Animation Applications
 20. Data Analysis and Data Analytics
 21. Computer Vision....many more etc
- =====

History of Python Programming Lang

=====

=>Python Programming Lang Conceived in the Year 1980.

=>Python Programming Lang Implementation (Bring into Action) started in the year 1989.

=>Python Programming Lang Officially Released in to Industry in the year 1991 Feb 20.

=>Python Programming Lang Developed by "GUIDO VAN ROSSUM".

=>Python Programming Lang Developed at Centrum Wiskunde Informatica (CWI) Institute in Nether Land.

=>Python Programming Lang maintained and managed by a non-commercial Organization called "Python Software Foundation (PSF) and whose Official Website is "www.python.org".

=>"ABC Programming Lang" is the predecessor of Python Programming Lang.

Versions of Python Programming Lang

=>Python Programming Contains 3 Types of Versions. They are

1. Python 1.x ---->Here 1 is called Major Version and and x represents 0 1 2 3 4 5 6 7 8..etc and It is Minor Version
=>Python 2.x does not support Python1.x.Python Programming does not support Backward Compatability.

2. Python 2.x----->Here 2 is called Major Version and x represents 0 1 2 3 4 5 6 7 and it is Minor Versions
(Outdated--2020)

=>Python 3.x does not support Python2.x.Python Programming does not support Backward Compatability.

3. Python 3.x----->Here 3 is called Major Version and x represents 0 1 2 3 4 5 6 7, 8,9,10, 11 and it is Minor Versions
3.11.3--Latest Version
3.12---Future Release

Features of Python

=>Features of a Language are nothing but Services OR Facilities Provided by Language Developers in Languages and they are used by Real Time Programmers for developing Real Time applications.
=>Python Programming Language 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 Functional (Procedure Oriented) and Object Oriented Programming (OOPs)
9. Extensible
10. Embedded
11. Supports Third Party APIs such as Numpy, Pandas, Matplotlib, scipy, scikit, keras, NLP, seaborn..etc

1. Simple

=>Python Programming Language is one of The SIMPLE Programming Language bcoz of 3 Important Technical Factors. They are

Factor 1:

=>Python Programming Provides "Rich Set of Modules (Libraries)". So that Python Programmer can Re-Use the Pre-defined Code of MODULES in real Time application.

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

Factor 2:

=>Python Programming Provides In-Built Facility called "Garbage Collector". So that It Collects un-Used Memory space and Improves the Performance of Python Based Applications.

=>Definition of Garbage Collector:

=>A Garbage Collector is one of the Background Python Program which is Running along with regular Python Program whose Role is that To collect un-used memory space and improves performance of Python Based Applications.

=>Hence Garbage Collector Takes care automatic Memory Management.

Factor 3:

Python Programming Lang Provides User-Friendly Syntaxes . So that Python Programmer can develop Error-Free Programs in Limited Span of time.

2. Freeware and Open Source

Freeware:

If any Software downloaded Freely from Official Source Then that Software is called Freeware.

Example: PYTHON, Java...etc

Open Source:

=>The Standard Name of Python Software is "CPYTHON".

=>If any Software allowed for Customization then it is called Open Source.

=>In Later Days, Some Software Company Vendors Came forward and Customized "CPYTHON" and developed their own In-house Tools for using that Companies.

=>The Customized Version of Original Software (CPYTHON) are called Distributions.

=>The Various Python Distributions (Customized Versions of CPYTHON) are

- 1) Jpython OR Jython---->Used for Running Java Based Applications
 - 2) IronPython OR Ipython-->Used to run C#.net applications
 - 3) MicroPython---->Used To develop Micro Controllers
 - 4) Anaconda Python--->Used to Develop Big Data / Hadoop Application
 - 5) Stack Less Python--->Concurrency Based Applications...etc
-

3. Platform Independent Lang

=>A Platform is nothing but Type of OS being Used to run Application / Program.

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

1. Platform Dependent Lang
 2. Platform Independent Lang
-

1. Platform Dependent Lang

=>In Platform Dependent Lang, Data Types differ from One OS to Another OS.

Example: C,C++....etc

2. Platform Independent Lang

=>In Platform Independent Lang, Data Types memory remains Same on All Types OSes.

=>In Effective Platform Independent Lang, all types of Values will store in the form of OBJECTS and they can store Un-Limited amount of data.

=>Hence java Object contains Size Restricted where Python Objects contains Un-limited Size and unlimited values can store.

NOTE: IN PYTHON ALL VALUES ARE STORED IN THE OF OBJECTS.

Examples: Java, Python.

4. Dynamically Typed

=>In IT, we have Two Types of Programming languages. They are

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

1. Static Typed Programming Languages

=>In Static Typed Programming Languages, It is mandatory to Specify Variable Declaration for storing Inputs in main memory of Computer.

Examples Statements:

```
int x,y,z; // Variable Declaration-
            mandatory
x=10
y=30
z=x+y
```

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

2. Dynamically Typed Programming Languages

=>In Dynamically Typed Programming Languages, depends on type of value we are assiging, whose data type automatically / Implicitly assigned by Python Execution Environment

=> There is No Need to write Variable declaration in Dynamically Typed Programming Languages

Example Stmtns:

```
>>> a=100
>>> b=200
>>> c=a+b
>>> print(a,type(a))-----100
>>> print(b,type(b))-----200
                                <class'int'>
>>> print(c,type(c))-----300
                                <class 'int'>
```

Examples: Python

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 COMPILATION 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.

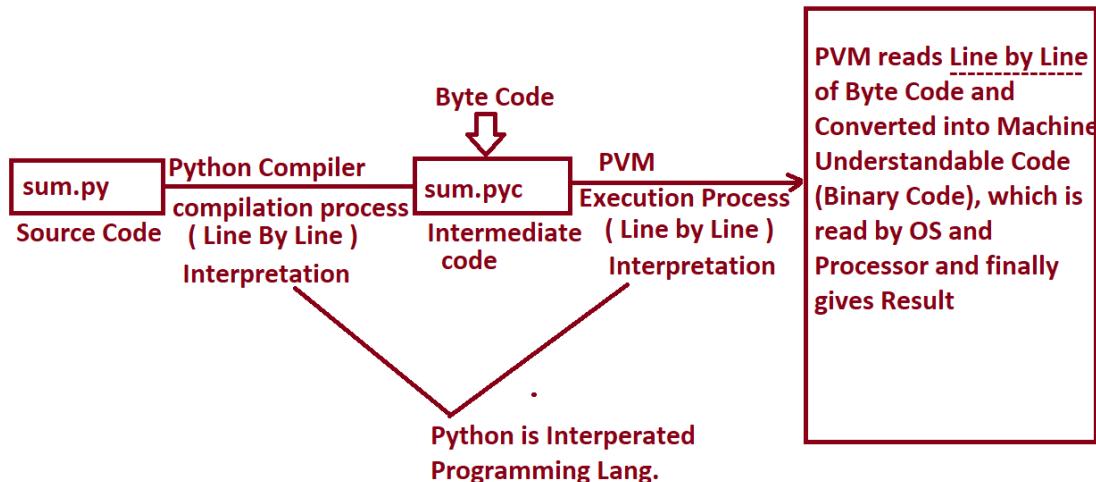
=>In EXECUTION PROCESS, The PVM reads the Python Intermediate Code(Byte Code) and Line by Line and Converted into Machine Understable Code (Executable or binary Code) and It is read by OS and Processor and finally Gives Result.

=>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)

Python Programming Execution Environment



6. High Level Programming

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

- 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=0b101010111110000
>>> b=0b10101111000
>>> print(a)-----22000
>>> print(b)-----1400
>>> a=0xBEE
>>> print(a)-----3054
>>> bin(22000)-----'0b101010111110000'
>>> hex(3054)-----'0xbbee'
```

8. Robust (Strong)

=>Python Programming is one of the Robust (Strong) Programming Language bcoz of "Exception Handling" Facility.

=>Definition of Exception: Runtime Errors of The program are called Exceptions

(Invalid Input ---->Runtime Errors----->Exceptions)

=>By default Exceptions generates Technical Error Messages. Which are understandable by Programmers but not by End-users. According Industry Standards, It recommended to generate User-Friendly Error Messages By using Exception Handling Concept

=>Definition of Exception Handling:

-->The Process of Converting Technical Error Messages into User-Friendly Error Message is Called Exception Handling.

9. Embedded

10. Extensible

Extensible:- Python Programming Provides Its Module Services to Other language

=>Programmers for easy to use by writing Less Code with More Meaning Since Python Programming Provides Its Services to Other languages and hence Python is One of The Extensible Programming Lang.

Embedded: Python Programming will use Other language Services / Libraries also. In Otherwords, we can call Other language code inside of Python Program and Hence Python Programming Lang is Embedded Programming Lang.

**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

=>Purpose of Data

=>Types of Literals / Values

=>Purpose of Identifiers OR Variables

=>Rules for Using Variables in Python Program

Data Representation in Python

Purpose of Data

=>The term "Data" is the most Important Factor in Organizations for Taking effective Decisions.
=>The Data which is used as of application / Program development can be stored either in Main Memory(RAM) OR Secondary Memory (HDD---Files , RDBMS Databases)

Types of Literals OR Values OR Data

=>In Programming Lang, we have 5 Types of Literals OR Values OR Data.They are

1. Integer Literals
2. Floating Point Literals
3. String Literals
4. Boolean Literals
5. Collection Literals (includes Dates....)

Important of Identifiers OR Variables

=>To Process values which are present in memory space , Programmers must give distinct names to cerated memory spaces. These distict names makes us to identify the values present in memory space and hence they are called IDENTIFIERS.

=>The IDENTIFIER Values are Changing/ Varying during Program Execution and hence IDENTIFIERS are also called VARIABLES.

=>hence All types of LITERALS are stored in the form VARIABLES and all Variables are called OBJECTS.

Rules for Using Identifiers OR Variables in Python

=>To use Variables in Python Program, we have 5 Types of Rules. They are

Rule1: The Variable Name is a Combination of Alphabets, Digits and a Special Symbol Under Score(_)

Examples:

```
- = 45---- Invalid  
$=56---- Invalid  
%=56---- Invalid
```

Rule-2: The First Letter of the Variable Name must starts either with an alphabet or Under Score (_).

Examples:

```
2=45---- Invalid  
2abc=45-- Invalid
```

```
    abc2=56 --valid
    _123=56--Valid
    _sal=56---Valid
    -sal=56---Invalid
    sal_=67---valid
```

Rule-3 : Within the Variable Name, No Special Symbols are allowed except under score (_)

Examples:

```
    sal   emp=56----Invalid
    sal-emp=56--invalid
    emp$sal=67--invalid
    emp_sal=56--valid
    emp_sal_tcs=67--valid
```

Rule-4: Keywords should not Taken as Variables Names (Keywords the Reserved words and they have some special meaning to the compilers)

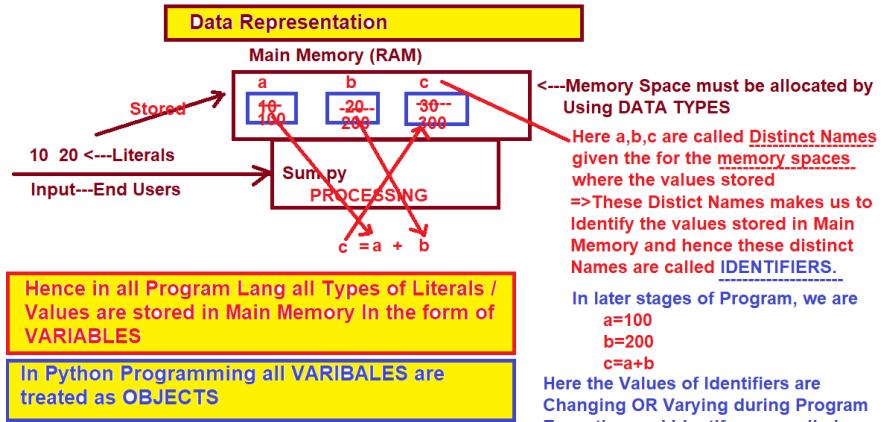
Examples:

```
if=23----Invalid
while=56---invalid
else=56--Invalid
True=56--invalid
false=78--Invalid
_while=67--valid
if1=56--Valid
_if=45---Valid
```

Rule-5: All the Variable Names in Python are Case
----- Sensitive

Examples:

```
>>> AGE=45
>>> age=44
>>> Age=43
>>> agE=42
>>> print(AGE,age,Age,agE) -----45 44 43 4
```



Data Types in Python (Most Imp---8 days)

=>The purpose of Data Types is that "To allocate sufficient Memory Space for Inputs / Data / Literals of the Program in the Main Memory of the Computer".

=>In Python Programming, we have 14 Data types and They are Classified into 6 Types.. 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 (Collections Data Type)

- 1) list

2) tuple

IV) Set Category Data Types (Collections Data Types)

1) set

2) frozenset

V) Dict Category Data Types (Collections Data Types)

1) dict

VI) NoneType Category Data Type

1) NoneType

=====

I) Fundamental Category Data Types

=====

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

=>In Fundamental Category , we have 4 Data Types. They are

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

=====

1. int

=====

Properties

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

=>The purpose of "int" data type is that "To Store Integer OR Integral Value OR Whole Numbers (Numbers without Decimal Values)". Ex: stno,empno,scno,htno,adcno....etc

Examples:

Python Instructions

Output

```
>>> a=100
>>> print(a)----- 100
>>> type(a)----- <class 'int'>
>>> print(a,type(a))-----100 <class 'int'>
>>> id(a)----- 140723540324232
>>> print(a,type(a),id(a))--- 100 <class 'int'> 140723540324232
```

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,type(a),id(a))----10 <class 'int'> 140723540321352
>>> print(b,type(b),id(b))----20 <class 'int'> 140723540321672
>>> print(c,type(c),id(c))----30 <class 'int'> 140723540321992
```

```
-----
>>> a=1000
>>> b=200
>>> c=a+b
>>> print(a,type(a),id(a))----1000 <class 'int'> 2188317454832
>>> print(b,type(b),id(b))----200 <class 'int'> 140723540327432
>>> print(c,type(c),id(c))----1200 <class 'int'> 2188314656176
```

=>By using "int" data type, we can also store Different Type of Number System Values.

=>In IT, 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.

=>It is one of the Default Number System followed by all Human Beings for Understanding their Day to data Transactions.

=>This Number System contains the following

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

=>The Base for Decimal Number System is 10

=>Hence Base 10 Literals are called Decimal Number System Values
Examples: 456, 345 , 789,367,34567812

2. Binary Number System.

=>This Number System data understandable by OS and Processor.

=>This Number System contains the following

Digits: 0 1 -----Total digits--(2)
Base : 2

=>The Base for Binary Number System is 2

=>Hence Base 2 Literals are called Binary Number System Values

=>In Python Programming, To store Binary Number System values , Binary Number System values Must Be Preceded by a letter 0b or 0B.

=>Syntax1: varname=0b Binary data
(OR)

=>Syntax2: varname=0B Binary data
=>Eventhough, we store the Binary Data, Internally Python execution environment Converts Binary Data into Decimal number System data.

Examples

```
>>> a=0b1100
>>> print(a,type(a))-----12 <class 'int'>
>>> bin(12)-----'0b1100'

>>> a=0b11011
>>> print(a,type(a))-----27 <class 'int'>
>>> bin(27)-----'0b11011'
>>> a=0B1111
>>> print(a,type(a))-----15 <class 'int'>
>>> bin(15)-----'0b1111'
NOTE: bin() is one of the pre-defined Function used for Converting any Data into Binary Number System Data.
```

3. Octal Number System

=>This Number System data understandable by Micro Processor Kits.

=>This Number System contains the following

Digits: 0 1 2 3 4 5 6 7 -----Total digits-- (8)
Base : 8

=>The Base for Octal Number System is 8

=>Hence Base 8 Literals are called Octal Number System Values

=>In Python Programming, To store Octal Number System values , Octal Number System values Must Be Preceded by a letter 0o or 0O.

=>Syntax1: varname=0o Octal data
(OR)

=>Syntax2: varname=0O Octal data

=>Eventhough, we store the Octal Number System Data, Internally Python execution environment Converts Octal Number System Data into Decimal number System data.

Examples

```
>>> a=0o22
>>> print(a,type(a))-----18 <class 'int'>
>>> oct(18)-----'0o22'
>>> a=0O123
>>> print(a,type(a))-----83 <class 'int'>
>>> oct(83)-----'0o123'
```

```

>>> a=0o128---SyntaxError: invalid digit '8' in octal literal
NOTE: oct() is one of the pre-defined Function used for
Converting any Data into Octal Number System Data.
-----
4. Hexa Decimal Number System.
-----
=>This Number System data Used in Development of OS.
=>This Number System contains the following
    Digits: 0 1 2 3 4 5 6 7 8 9
        A(10)  B(11)  C(12)  D(13)  E(14)  F(15)  -----
        Total digits--(16)
        Base : 16
=>The Base for Hexa Decimal Number System is 16
=>Hence Base 16 Literals are called Hexa Decimal Number System
data
=>In Python Programming, To store Hexa Decimal Number System
values , Hexa Decimal Number System values Must Be Preceded by
a letter 0x or 0X.
=>Syntax1: varname=0x Hexa Decimal Number System data
            (OR)
=>Syntax2: varname=0X Hexa Decimal Number System
=>Eventhough, we store the Hexa Decimal Number System Data,
Internally Python execution environment Converts Hexa Decimal
Number System Data into Decimal number System data.
-----
Examples
-----
>>> a=0xAC
>>> print(a,type(a))-----172 <class 'int'>
>>> a=0XAC
>>> print(a,type(a))-----172 <class 'int'>
>>> hex(172)-----'0xac'
-----
>>> a=0XBEE
>>> print(a,type(a))-----3054 <class 'int'>
>>> hex(3054)-----'0xbbee'
>>> a=0XFACE
>>> print(a,type(a))-----64206 <class 'int'>
>>> hex(64206)-----'0xface'
>>> hex(10)-----'0xa'
>>> hex(15)-----'0xf'
>>> a=0XBEER-----SyntaxError: invalid hexadecimal literal
>>> a=0XFACER----- SyntaxError: invalid hexadecimal literal
>>> x=0xACC
>>> print(x,type(x))-----2764 <class 'int'>
>>> hex(2764)-----'0xacc'
-----
```

NOTE: In Python Programming, we have 3 Types of Base Comverion Functions. They are

```
1. bin()
2. oct()
3. hex()

>>> #Conversion from Dec,Oct and Hexa into Binary Data
>>> a=15
>>> bin(a)-----'0b1111'
>>> a=0o22
>>> bin(a)-----'0b10010'
>>> a=0xAC
>>> bin(a)-----'0b10101100'
-----
>>> #Conversion from Dec,Oct and Binary into Hexa
>>> a=172
>>> hex(a)-----'0xac'
>>> a=0o15
>>> hex(a)-----'0xd'
>>> a=0b1111
>>> hex(a)-----'0xf'
-----
>>> #Conversion from Dec,Bin and Hex into oct
>>> a=13
>>> oct(a)-----'0o15'
>>> a=0b1010
>>> oct(a)-----'0o12'
>>> a=0xF
>>> oct(a)-----'0o17'
-----

NOTE:
-----
>>> a=0239-----SyntaxError: leading zeros in decimal
integer literals are not permitted; use an 0o prefix for octal
```

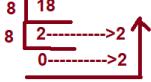
integers

Conversion from Decimal Data into Octal Data

Q1)

$$(18)_{10} \longrightarrow (x)_8 \text{ find } x=22$$

Sol:



$$\text{Hence } (18)_{10} \longrightarrow (22)_8$$

Conversion From Octal Data into Decimal

Q1)

$$(22)_8 \longrightarrow (x)_{10} \text{ find } x=18$$

Sol:

$$\begin{array}{r} 2 & 2 \\ 1 & 0 \\ \Rightarrow & 8 & 8 \\ \Rightarrow & 2 \times 8^1 + 2 \times 8^0 \\ \Rightarrow & 16 + 2 \\ \Rightarrow & 18 \end{array}$$

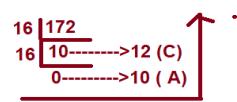
$$\text{Hence } (22)_8 \longrightarrow (18)_{10}$$

Conversion from Decimal to Hexa Decimal Values

Q1)

$$(172)_{10} \longrightarrow (x)_{16} \text{ find } x=AC$$

Sol:



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

Conversion from Hexa Decimal Values to Decimal

$$Q1) (AC)_{16} \longrightarrow (x)_{10} \text{ Find } x=172$$

Sol:

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

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

2. float

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

=>The purpose of 'float' data type is that "To store Real Constant Values OR Floating point Values (Numbers with Decimal values)". Examples: Percentange of marks, DA for employees...etc

=>This data type can also be used for storing Scientific Notation (It is one of the notation for representing floating point data).
=>The Advantage of Using Scientific Notation is that "To Minimize the memory space at time storing Normal floating Point Values".
=>this Data type does not support Binary, Octal and Hexa Decimal data in the form of floarting Pointvalues.

Examples

```
>>> a=2.3
>>> print(a,type(a))-----2.3 <class 'float'>
>>> a=10
>>> b=2.3
>>> c=a+b
>>> print(a,type(a))-----10 <class 'int'>
>>> print(b,type(b))-----2.3 <class 'float'>
>>> print(c,type(c))-----12.3 <class 'float'>
-----
>>> a=3e2
>>> print(a,type(a))-----300.0 <class 'float'>
>>> a=10e-2
>>> print(a,type(a))-----0.1 <class 'float'>
-----
>>> a=0.0000000000000000000000000000000000000000000000006
>>> print(a,type(a))-----6e-42 <class 'float'>
-----
>>> a=0b1111.0b1010-----SyntaxError: invalid decimal literal
>>> a=0xAC.0b1010-----SyntaxError: invalid decimal literal
>>> a=0b1111.0023-----SyntaxError: invalid syntax
```

3) bool

Properties

=>'bool' is one of the pre-defined class and treated as Fundamental data type.
=>The purpose of bool data type is that "To store True , False Value (Logical Values)".
=>In Python Programming, True is a Keyword and False is also a keyword and they are treated values for bool data type."
=>In Python Programming, Internally True is considered as 1 and False Considered as 0.
=>On Bool Values, we can perform Boolean Arithmetic Operation such as Addition, Substraction,Multiplication..etc

Examples

```
>>> True=45-----SyntaxError: cannot assign to True
>>> False=5.5-----SyntaxError: cannot assign to False
>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=False
>>> print(b,type(b))-----False <class 'bool'>
>>> a=true-----NameError: name 'true' is not defined.

>>> a=True
>>> b=False
>>> print(a+b)-----1
>>> print(a*b)-----0
>>> print(b*b)-----0
>>> print(a-b)-----1
>>> print(b-a)-----1
>>> print(a/b)-----ZeroDivisionError: division by zero
>>> print(2*True+3)-----5
>>> print(2+True*3)-----5
>>> print(2*True+4*False+3)----5

>>> print(0b1010*True)-----10
>>> print(0b1010*True+1.2)-----11.2
>>> print(0XF*False+1.2-True)-----0.19999999999999996
>>> print(round(0XF*False+1.2-True,2))---0.2
>>> print(0b1111*True-15)-----0
>>> print(0b1111*False-15)-----15
```

4. complex

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

=>The Purpose of complex data type of is that " To store Complex values".

=>The General format of complex data type is given bellow.

Format1:a+bj
Format2:a-bj

=>here 'a' and 'b' are called Real Part and Imaginary Parts.

=>here the letter 'j' Represents $\sqrt{-1}$ OR $\text{sqr}(j) = -1$

=>Internally, Real and Imaginary Parts are treated as Float Data Type.

=>To Extract Real Part and Imaginary Parts, we have Two Pre-Defined attributes in complex Object. They are

a) real
b) imag

Syntax1: complexobj.real====> Gives Real Part of Complex Object
 Syntax2: complexobj.imag====> Gives Imaginary Part of Complex Object.
=>On the Values of Complex ,we can Various Arithmetic Operations Like Addition,Substraction,Multiplication...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=-2-13j
>>> print(c,type(c))----- (-2-13j) <class 'complex'>

>>> a=2+3i-----SyntaxError: invalid decimal literal Invalid symbol 'i'

>>> a=2.3+4.5j
>>> print(a,type(a))----- (2.3+4.5j) <class 'complex'>
>>> b=-3.4-4.7j
>>> print(b,type(b))----- (-3.4-4.7j) <class 'complex'>

>>> a=3+4.6j
>>> print(a,type(a))----- (3+4.6j) <class 'complex'>
>>> b=-3-4.6j
>>> print(b,type(b))----- (-3-4.6j) <class 'complex'>

>>> a=2j
>>> print(a,type(a))----- 2j <class 'complex'>
>>> b=4.5j
>>> print(b,type(b))----- 4.5j <class 'complex'>

>>> a= -4j
>>> print(a,type(a))----- (-0-4j) <class 'complex'>
>>> b= -4.6j
>>> print(b,type(b))----- (-0-4.6j) <class 'complex'>
>>> a=4.5j
>>> print(a,type(a))----- 4.5j <class 'complex'>

>>> a=2+6j
>>> print(a,type(a))----- (2+6j) <class 'complex'>
>>> print(a.real)----- 2.0
>>> print(a.imag)----- 6.0

>>> a=-2.3-4.5

```

>>> print(a,type(a))----- -6.8 <class 'float'>
>>> print(a.real)----- -6.8
-----
>>> a=-2.3-4.5j
>>> print(a,type(a))----- (-2.3-4.5j) <class 'complex'>
>>> print(a.real)----- -2.3
>>> print(a.imag)----- -4.5
-----
>>> a=3.4-5.6j
>>> print(a,type(a))----- (3.4-5.6j) <class 'complex'>
>>> print(a.real)----- 3.4
>>> print(a.imag)----- 5.6
-----
>>> a=2j
>>> print(a,type(a))----- 2j <class 'complex'>
>>> print(a.real)----- 0.0
>>> print(a.imag)----- 2.0
-----
>>> a=-4j
>>> print(a,type(a))----- (-0-4j) <class 'complex'>
>>> print(a.real)----- -0.0
>>> print(a.imag)----- -4.0
-----
>>> a=0+0j
>>> print(a,type(a))----- 0j <class 'complex'>
>>> a=-0-0j
>>> print(a,type(a))----- 0j <class 'complex'>
>>> a=-0-3j
>>> print(a,type(a))----- -3j <class 'complex'>
>>> a=-3j
>>> print(a,type(a))----- (-0-3j) <class 'complex'>
-----
>>> a=2+3j
>>> b=3+4j
>>> c=a+b
>>> print(c,type(c))----- (5+7j) <class 'complex'>
>>> print(2+10j+11+9j)----- (13+19j)
-----
>>> a=2+3j
>>> b=3+4j
>>> c=a*b
>>> print(c,type(c))----- (-6+17j) <class 'complex'>
>>> a=2+3j
>>> print(a.imaginary)-----AttributeError: 'complex' object
has no attribute 'imaginary'
=====

```

II) Sequence Category Data Types

=====

=>The purpose of Sequence Category Data Types is that " To store 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(Part-1)

=====

Index

=>What is str

=>Purpose of str

=>Types of strs

=>Notations of str

=>Memory Management of str data

=>Syntaxes for Representing str data

=>Operations on str data

 a) Indexing

 b) slicing

=>Programming Examples

=====

Properties

=====

=>'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 tripple double quotes and tripple single quotes. "

=>Def. of str:

=>str is a collection of Characters or Alphanumeric data or numeric data or any type of data enclosed within double Quotes or single quotes or tripple double quotes and tripple 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:
-----
=>Syntax1:-      varname=" " " String Data1
                           String Data2
                           -----
                           String data-n " " "
                           (OR)
=>Syntax2:-      varname=' ' ' String Data1
                           String Data2
                           -----
                           String data-n ' ' '
=>With the help tripple double Quotes (" " " " ") and
Tripple single Quotes (' ' ' ' ') we can store single line
str data and multi line string data.
-----
```

Examples:

```

>>> s1="Python"
>>> print(s1,type(s1))-----Python <class 'str'>
>>> s2='Python'
>>> print(s2,type(s2))-----Python <class 'str'>
>>> s3="A"
>>> print(s3,type(s3))-----A <class 'str'>
>>> s4='A'
>>> print(s4,type(s4))-----A <class 'str'>
>>> s1="123456"
>>> print(s1,type(s1))-----123456 <class 'str'>
>>> s2="Python3.11"
>>> print(s2,type(s2))-----Python3.11 <class 'str'>
>>> s3="123$456_abc"
>>> print(s3,type(s3))-----123$456_abc <class 'str'>
>>> s4="@#$%^&8912"
>>> print(s4,type(s4))-----@#$%^&8912 <class 'str'>
>>> s1="Python Programming"
>>> print(s1,type(s1))-----Python Programming <class 'str'>
-----
```

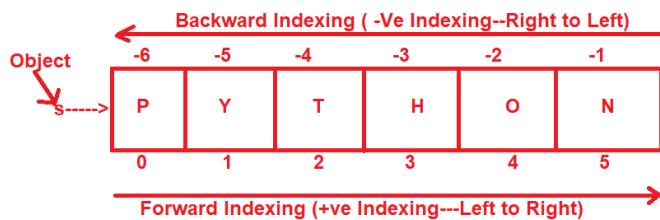
```
>>> addr1="Guido Van Rossum
SyntaxError: unterminated string literal (detected at line 1)
>>> addr1='Guido Van Rossum
SyntaxError: unterminated string literal (detected at line 1)
-----
>>> addr1="""Guido Van Rossum
... FNO:3-4, Hill Side
... Python Software Foundation
... Nether Lands-56"""
>>> print(addr1,type(addr1))
                    Guido Van Rossum
                    FNO:3-4, Hill Side
                    Python Software Foundation
                    Nether Lands-56 <class 'str'>
>>> addr2='''Travis Oliphant
... HNO:12-34, Sea Side
... Numpy Organization
... Nether lands-58'''
>>> print(addr2,type(addr2))
                    Travis Oliphant
                    HNO:12-34, Sea Side
                    Numpy Organization
                    Nether lands-58 <class 'str'>
-----
>>> s1="""Python Programming"""
>>> print(s1,type(s1))---- Python Programming <class 'str'>
>>> s1='''Python Programming'''
>>> print(s1,type(s1))-----Python Programming <class 'str'>
>>> s2="""A"""
>>> print(s2,type(s2))-----A <class 'str'>
>>> s2='''A'''
>>> print(s2,type(s2))-----A <class 'str'>
```

str memory management

Consider the following

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

When the above statement is executed then PVM stored the above data in Memory as follows



Operations on str data

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

1. Indexing
2. Slicing

1. Indexing

=>The Purpose of Indexing Concept in Python is that to get Single Value from given object.

=>Syntax: strobj [index]

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

=>If we enter Valid Index then PVM gives Corresponding Value of that Index

=>If we enter InValid Index then PVM gives IndexError

Examples

```
>>> s="PYTHON"
>>> print(s,type(s)) ----- PYTHON <class 'str'>
>>> print(s[0])-----P
>>> print(s[-1])-----N
>>> print(s[-6])-----P
>>> print(s[-5])-----Y
```

```

>>> print(s[1])-----Y
>>> print(s[-2])-----O
>>> print(s[4])-----O
>>> print(s[5])-----N
>>> print(s[15])-----IndexError: string index out of range
>>> print(s[-11])-----IndexError: string index out of range
=====
>>> s="Java Prog"
>>> print(s,type(s))-----Java Prog <class 'str'>
>>> print(s[4])-----Space
      OR
>>> s[4]----- ' '
>>> s[0]----- 'J'
>>> s[1]-----'a'
>>> len(s)-----9--->len(strobj) gives Number of
Characters in str obj
>>> s[len(s)-1]-----'g'
>>> s[len(s)]-----IndexError: string index out of
range
>>> s[-len(s)]-----'J'
-----
>>> s="PYTHON IS AN OOP LANG"
>>> print(s,type(s))----PYTHON IS AN OOP LANG <class 'str'>
>>> print(s[len(s)-1])-----G
>>> print(s[-len(s)])-----P
>>> print(s[0])-----P
=====
```

2. Slicing Operations

=>The process of obtaining Range of Characters or Sub String from given str object is called Slicing Operation.

Syntax1: strobj [BEGIN:END]

=>This Syntax gives Range of Characters from BEGIN Index Value END-1 Index Value Provided BEGIN<END otherwise we never get any result or we get Space as Result.

Examples

```

>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> print(s[0:4])-----PYTH
>>> print(s[2:5])-----THO
>>> print(s[4:6])-----ON
>>> print(s[1:4])-----YTH
>>> print(s[1:6])-----YTHON
```

```
>>> print(s[0:6])-----PYTHON
>>> print(s[4:2])-----Space
    OR
>>> s[4:2]-----' '
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> print(s[-6:-2])-----PYTH
>>> print(s[-5:-1])-----YTHO
>>> print(s[-4:-2])-----TH
>>> print(s[-6:-1])-----PYTHO
>>> print(s[-2:-4])-----space
    (OR)
>>> s[-2:-4]-----' '
=====
Sub Rule: strobj [ POSBEG:NEGEND ]
=====
```

=>This Syntax gives Range of chars from POSBEG Index to NEGEND-1 Index provided POSBEG>NEGEND (Intersection Area of Data) NEGEND>POSEND (Intersection Area of Data) otherwise we never get any result or we get Space as Result.

Examples

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[2:-1]-----'THO'
>>> print(s[2:-1])-----THO
>>> print(s[-1:2])-----space
    (OR)
>>> s[-1:2]-----' '
>>> s[1:-1]-----'YTHO'
>>> s[2:-2]-----'TH'
>>> s[-3:4]-----'H'
>>> s[-6:4]-----'PYTH'
>>> s[-2:0]-----''
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[2:-1]-----'THO'
>>> s[2:-2]-----'TH'
>>> s[2:-3]-----'T'
```

Syntax2: strobj [BEGIN :]

=>In this Syntax, we are Specifying BEGIN Index and we didn't specify END INDEX

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

OR

=>If we don't specify END Index then PVM Takes Chars from BEGIN INDEX Value to Last Character

Examples

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> len(s)-----6
>>> s[4:]-----'ON'
>>> s[2:]-----'THON'
>>> s[0:]-----'PYTHON'
>>> s[1:]-----'YTHON'
>>> s[3:]-----'HON'

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6:]-----'PYTHON'
>>> s[-4:]-----'THON'
>>> s[-5:]-----'YTHON'
>>> s[-3:]-----'HON'
>>> s[-2:]-----'ON'

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

Syntax3: strobj[:END]

=>In this Syntax, we are Specifying END Index and we didn't specify BEGIN INDEX

=>If we don't specify BEGIN Index then PVM Takes END Index as 0 OR -len(strobj)

(OR)

=>If we don't specify BEGIN Index then PVM Takes Chars from First Charcater OR 0th Character or -len(strobj)th Character Value to END-1 Index Value.

Examples:

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

```
>>> print(s)-----PYTHON
>>> print(s[:4])-----PYTH
>>> print(s[:3])-----PYT
>>> print(s[:6])-----PYTHON
>>> print(s[:-1])-----PYTHO
>>> print(s[:0])-----Space
>>> s[:0]-----'
>>> s[:-6]-----'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:3]-----'PYT'
>>> s[:-2]-----'PYTH'
>>> s[:-1]-----'PYTHO'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:-4]-----'PY'
>>> s[:-5]-----'P'
>>> s[:-1]-----'PYTHO'
>>> s[:0b0011]-----'PYT'
>>> s[:True+True]-----'PY'
```

Syntax4: strobj[:]

=>In this Syntax, we are not Specifying BEGIN and END Indices
=>This Syntax Give Characters from Fisr Character to Last Character (gives Complete data of str obj)

Examples

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:]-----'PYTHON'
>>> print(s[:])-----PYTHON
>>> print(s[0:6])-----PYTHON
>>> print(s[0:])-----PYTHON
>>> s="PYTHON PROG"
>>> print(s)-----PYTHON PROG
>>> s[:]-----'PYTHON PROG'
>>> print(s[:])-----PYTHON PROG
```

NOTE: In all the above Syntaxes, We are getting the characters from strobj in FORWARD DIRECTION with Default Step Value 1.

Syntax5: strobj[BEGIN:END:STEP]

RULE1: Here the Values of BEGIN, END and STEP can be either +VE or -VE

RULE2: If the Value of STEP is +VE then PVM Obtains the
 ---- Characters from BEGIN Index to END-1 Index
 In FORWARD DIRECTION provided BEGIN INDEX < END INDEX otherwise
 otherwise we never get any result or we get Space as Result.

RULE 3: If the Value of STEP is -VE then PVM Obtains the
 ----- Characters from BEGIN Index to END+1 Index
 in BACKWARD DIRECTION provided BEGIN INDEX > END INDEX otherwise
 otherwise we never get any result or we get Space as Result.

RULE4: In FORWARD DIRECTION, If we specify END INDEX as 0 Then
 we never get any result or we get Space as Result.

RULE5: In BACKWARD DIRECTION, If we specify END INDEX as -1
 Then we never get any result or we get Space as Result.

RULE-2

```

>>> s="PYTHON"
>>> print(s)
PYTHON
>>> print(s[0:4:1])
PYTH
>>> print(s[0:4])
PYTH
>>> print(s[:])
PYTHON
>>> print(s[:])
PYTHON
>>> print(s[::-1])
PTO
>>> print(s[0:4:-1])
PT
>>> print(s[0:6:-1])
PH
>>> print(s[0:6:-2])
PO
>>> print(s[1:5:-2])
YH
>>> print(s[5:1:-2])-----space
      (OR
>>> s[5:1:-2]
 ''
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> print(s[-6:-1:-1])
PYTHO
  
```

```
>>> print(s[-6:-1])
PYTHO
>>> print(s[-6:-1:2])
PTO
>>> print(s[-6:-1:3])
PH
>>> print(s[-6:-4:3])
P
```

RULE-3

```
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> s[::-1]
'NOHTYP'
>>> s[5:2:-1]
'NOH'
>>> s[4::-2]
'OTP'
>>> s[5:1:-2]
'NH'
>>> s[::-3]
'NT'
>>> s[::-4]
'NY'
>>> s[-1:-6:-1]
'NOHTY'
>>> s[-1:-7:-1]
'NOHTYP'
>>> s[-1:-7:-2]
'NHY'
>>> s[-6:5:-1]
''
```

```
>>> s[2:-1:-1]
''
```

RULE-4 and RULE-5

```
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> s[3:0:2]
''
>>> s[6:0:3]
''
>>> s[-3:-1:-1]
```

```

"""
>>> s[-6:-1:-2]
"""

-----
>>> s="MADAM"
>>> s[::-1]==s[::-1]
True
>>> s="LIRIL"
>>> s[::-1]==s[::-1]
True
>>> s="RACECAR"
>>> s[::-1]==s[::-1]
True
>>> s="DAD"
>>> s[::-1]==s[::-1]
True
>>> s="MOM"
>>> s[::-1]==s[::-1]
True
>>> s="MALAYALAM"
>>> s[::-1]==s[::-1]
True
>>> s="PYTHON"
>>> s[::-1]==s[::-1]
False
>>> s="MaDaM"
>>> s[::-1]==s[::-1]
True
>>> s="Madam"
>>> s[::-1]==s[::-1]
False
>>> s[::-1].lower()==s[::-1].lower()
True
>>> s[::-1].upper()==s[::-1].upper()
True
>>> s="PYTHON"
>>> s[::2]==s[::2][::-1]
False
>>> "PYTHON"[::2]==="PYTHON"[::-1][::-2]
True
>>> "PYTHON"[::-1][::-2]
'PTO'
>>> "PYTHON"[::2]
'PTO'
>>> "KVR"[::-1]!="KVR"[::2]
True

```

=====

Type Casting Techniques in Python

=====

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

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

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

=====

1.int()

=====

=>int() is used for Converting One Possible type of Value into int type Value.

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

Example1: float type ---->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'>
```

Example2: bool type ---->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'>
```

Example3: complex type ---->int type---->Not Possible

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

```

>>> b=int(a)----TypeError: int() argument must be a string, a
bytes-like object or a real number, not 'complex'
-----
Example4: str type ---->int type
=====
Case-1 : str int---->int--->Possible
-----
>>> a="12"
>>> print(a,type(a))-----12 <class 'str'>
>>> a-----'12'
>>> b=int(a)
>>> print(b,type(b))-----12 <class 'int'>
-----
Case-2: str float----->int--->Not Possible
-----
>>> a="12.45"
>>> print(a,type(a))-----12.45 <class 'str'>
>>> a-----'12.45'
>>> b=int(a)-----ValueError: invalid
literal for int() with base 10: '12.45'
-----
Case-3: str bool----->int--->Not Possible
-----
>>> a="True"
>>> print(a,type(a))-----True <class 'str'>
>>> a-----'True'
>>> b=int(a)-----ValueError: invalid literal for
int() with base 10: 'True'
-----
Case-4 : str complex---->int--->Not Possible
>>> a="2+3.4j"
>>> print(a,type(a))-----2+3.4j <class 'str'>
>>> a-----'2+3.4j'
>>> b=int(a)-----ValueError: invalid
literal for int() with base 10: '2+3.4j'
-----
Case-5 : pure str---->int--->Not Possible
-----
>>> a="PYTHON"
>>> print(a,type(a))-----PYTHON <class 'str'>
>>> a-----'PYTHON'
>>> b=int(a)-----ValueError: invalid
literal for int( ) with base 10: 'PYTHON'
=====
2.float( )
=====
```

```

=>float( ) is used for Converting One Possible type of Value
into float type Value.
=>Syntax: varname=float( int / bool / complex / str )
-----
Example1: int type ---->float type---->Possible
-----
>>> a=12
>>> print(a,type(a))-----12 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))-----12.0 <class 'float'>
>>> a=0
>>> print(a,type(a))-----0 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))-----0.0 <class 'float'>
-----
Example2: bool type ---->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'>
-----
Example3: complex type ---->float type---->Not Possible
-----
>>> a=2.3+4.5j
>>> print(a,type(a))----- (2.3+4.5j) <class
'complex'>
>>> b=float(a)-----TypeError: float() argument
must be a string or a real number, not 'complex'
>>> b=float(a.real)
>>> print(b,type(b))-----2.3 <class 'float'>
>>> b=float(a.imag)
>>> print(b,type(b))-----4.5 <class 'float'>
-----
Example4: str type ---->float type
=====
Case-1 : str int---->float--->Possible
-----
>>> a="34"
>>> print(a,type(a))-----34 <class 'str'>
>>> a-----'34'
>>> b=float(a)

```

```

>>> print(b,type(b))-----34.0 <class 'float'>
-----
Case-2: str float----float-->Possible
-----
>>> a="12.34"
>>> print(a,type(a))-----12.34 <class 'str'>
>>> a-----'12.34'
>>> b=float(a)
>>> print(b,type(b))-----12.34 <class 'float'>
>>> a="3e2"
>>> print(a,type(a))-----3e2 <class 'str'>
>>> a-----'3e2'
>>> b=float(a)
>>> print(b,type(b))-----300.0 <class 'float'>
-----
>>> a="12.34.56"
>>> print(a,type(a))-----12.34.56 <class 'str'>
>>> a-----'12.34.56'
>>> b=float(a)-----ValueError: could not
convert string to float: '12.34.56'
>>> a="3e2e4"
>>> print(a,type(a))-----3e2e4 <class 'str'>
>>> a-----'3e2e4'
>>> b=float(a)-----ValueError: could
not convert string to float: '3e2e4'
-----
Case-3: str bool----->float---->Not Possible
-----
>>> a="True"
>>> print(a,type(a))-----True <class 'str'>
>>> a-----'True'
>>> b=float(a)-----ValueError: could not
convert string to float: 'True'
-----
Case-4 : str complex---->float---->Not Possible
-----
>>> a="3+4.5j"
>>> print(a,type(a))-----3+4.5j <class 'str'>
>>> a-----'3+4.5j'
>>> b=float(a)-----ValueError: could
not convert string to float: '3+4.5j'
-----
Case-5 : pure str-----float---->Not Possible
-----
>>> a="PYTHON"
>>> print(a,type(a))-----PYTHON <class 'str'>
>>> a-----'PYTHON'

```

```

>>> b=float(a)-----ValueError: could not
convert string to float: 'PYTHON'
=====
3.bool( )
=====
=>bool() is used for Converting One Possible type of Value into
bool type Value.
=>Syntax: varname=bool( int / float / complex / str )
=>ALL NON-ZERO VALUES ARE TREATED AS TRUE
=>ALL ZERO VALUES ARE TREATED AS FALSE
-----
Example1: int type ---->bool type---->Possible
>>> a=100
>>> print(a,type(a))-----100 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=-123
>>> print(a,type(a))----- -123 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=0
>>> print(a,type(a))-----0 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
-----
Example2: float type ---->bool type---->Possible
-----
>>> a=12.34
>>> print(a,type(a))-----12.34 <class 'float'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=0.000000000000000000000000000000000000000000000000006
>>> print(a,type(a))-----6e-40 <class 'float'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=0.000000000000000000000000000000000000000000000000000000000
>>> print(a,type(a))-----0.0 <class 'float'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
-----
Example3: complex type ---->bool type---->Possible
-----
>>> a=2+3j
>>> print(a,type(a))-----(2+3j) <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=0.0+0j

```

```

>>> print(a,type(a))-----0j <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
-----
Example4: str type ---->bool type
Case-1 : str int---->bool--->Possible
-----
>>> a="0"
>>> print(a,type(a))-----0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----1
>>> a="00000"
>>> print(a,type(a))-----00000 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----5
>>> a="123"
>>> print(a,type(a))-----123 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
-----
Case-2: str float---->bool--->Possible
-----
>>> a="12.34"
>>> print(a,type(a))-----12.34 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a="0.0"
>>> print(a,type(a))-----0.0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a="2.3.4"
>>> print(a,type(a))-----2.3.4 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
-----
Case-3: str bool---->bool---->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'>
-----
```

```

Case-4 : str complex---->bool---->Possible
-----
>>> a="2+3j"
>>> print(a,type(a))-----2+3j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a="0+0j"
>>> print(a,type(a))-----0+0j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
-----
Case-5 : pure str-----bool---->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))----- space <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
>>> len(a)-----0
>>> a=" "
>>> print(a,type(a))----- spaces <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----3
=====

4.complex( )
=====
=>complex() is used for converting any possible type of value
into complex type value.
=>Syntax: varname=complex(int / float / bool / str)
-----
Example1: int value----->complex--->Possible
-----
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> b=complex(a)
>>> print(b,type(b))----- (10+0j) <class 'complex'>
>>> a=-10
>>> print(a,type(a))----- -10 <class 'int'>
>>> b=complex(a)
>>> print(b,type(b))----- (-10+0j) <class 'complex'>
-----
Example2: float value----->complex--->Possible
-----
```

```

>>> a=1.2
>>> print(a,type(a))-----1.2 <class 'float'>
>>> b=complex(a)
>>> print(b,type(b))----(1.2+0j) <class 'complex'>
-----
Example3:   bool value----->complex--->
-----
>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=complex(a)
>>> print(b,type(b))----(1+0j) <class 'complex'>
-----
Example4:   str type value----->complex
-----
Case-1: str int value----->complex--->Possible
-----
>>> a="12"
>>> print(a,type(a))-----12 <class 'str'>
>>> a-----'12'
>>> b=complex(a)
>>> print(b,type(b))----(12+0j) <class 'complex'>
-----
Case-2: str float value ----->complex--Possible
-----
>>> a="1.4"
>>> print(a,type(a))-----1.4 <class 'str'>
>>> a-----'1.4'
>>> b=complex(a)
>>> print(b,type(b))----(1.4+0j) <class 'complex'>
-----
Case-3: str bool value ----->complex---Not Possible
-----
>>> a="True"
>>> print(a,type(a))-----True <class 'str'>
>>> a-----'True'
>>> b=complex(a)-----ValueError: complex( ) arg is a
malformed string
-----
Case-4: str complex ----->complex---Possible
-----
>>> a="2+3j"
>>> print(a,type(a))-----2+3j <class 'str'>
>>> a-----'2+3j'
>>> b=complex(a)
>>> print(b,type(b))----(2+3j) <class
'complex'>

```

```

>>> a="2+3i"
>>> print(a,type(a))-----2+3i <class 'str'>
>>> a-----'2+3i'
>>> b=complex(a)-----ValueError: complex( ) arg is a
malformed string
-----
Case-5: pure str----->complex----Not Possible
-----
>>> a="kvr+mishraj"
>>> print(a,type(a))-----kvr+mishraj <class 'str'>
>>> a-----'kvr+mishraj'
>>> b=complex(a)-----ValueError: complex( )
arg is a malformed string
=====
5.Str( )
=====
=>str() is used for converting Any Type of Value into str type
value.
=>Syntax: varname=str(int / float / bool / complex)
-----
Examples
-----
>>> a=100
>>> print(a,type(a))-----100 <class 'int'>
>>> b=str(a)
>>> print(b,type(b))-----100 <class 'str'>
>>> b-----'100'
>>> a=12.34
>>> print(a,type(a))-----12.34 <class 'float'>
>>> b=str(a)
>>> print(b,type(b))-----12.34 <class 'str'>
>>> b-----'12.34'
>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=str(a)
>>> print(b,type(b))-----True <class 'str'>
>>> b-----'True'
>>> a=2+3.6j
>>> print(a,type(a))----- (2+3.6j) <class
'complex'>
>>> b=str(a)
>>> print(b,type(b))----- (2+3.6j) <class 'str'>
>>> b-----'(2+3.6j)'

```

S.no	int type	float type	bool type	Complex type	(≤str) Str int	Str float	Str bool	Str complex	Pure str
1> int()	✓	✓	✓	✗	✓	✗	✗	✗	✗
2> float()	✓	✗	✓	✗	✓	✓	✗	✗	✗
3> bool()	✓	✓	✗	✓	✓	✓	✓	✓	✓
Complex()	✓	✓	✓	✗	✓	✓	✗	✓	✗
str()	✓	✓	✓	✓	✓	✓	✓	✓	✓

2. bytes

Properties

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

=>The purpose of bytes data type is that "To store the range of values from 0 to 256(0 to 255 only) for the implementation End-to-End Encryption Conversion (Encrypted Format) "

=>bytes data type does not contain any symbolic notation for storing bytes data but we can convert any type data into bytes data by using bytes()

=>on the object of bytes , we can perform both Indexing and Slicing Operations.

=>An object of bytes data type maintains Insertion Order. which is nothing but whatever the order we organize,in the same order, data will be displayed.

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

Examples

```
>>> lst=[12,0,45,123,256,34]
>>> print(lst,type(lst))-----[12, 0, 45, 123, 256, 34]
<class 'list'>
>>> b=bytes(lst)----ValueError: bytes must be in range(0, 256)
```

```

>>> lst=[-12,0,45,123,255,34]
>>> print(lst,type(lst))-----[-12, 0, 45, 123, 255, 34]
<class 'list'>
>>> b=bytes(lst)-----ValueError: bytes must
be in range(0, 256)
-----
>>> lst=[12,0,45,123,255,34]
>>> print(lst,type(lst))-----[12, 0, 45, 123, 255,
34] <class 'list'>
>>> b=bytes(lst)
>>> print(b,type(b))-----b'\x0c\x00-\{\xff"'
<class 'bytes'>
>>> for kv in b:
...     print(kv)
...
12
0
45
123
255
34
>>> for kv in b[0:3]:
...     print(kv)
...
12
0
45
>>> for kv in b[::-2]:
...     print(kv)
...
12
45
255
>>> for kv in b[::-1]:
...     print(kv)
...
34
255
123
45
0
12
>>> print(b[0])-----12
>>> print(b[-1])-----34
-----
>>> lst=[10,20,45,67,89]
>>> print(lst,type(lst))-----[10, 20, 45,
67, 89] <class 'list'>
>>> b=bytes(lst)

```

```
>>> print(b,type(b),id(b))-----b'\n\x14-CY'  
<class 'bytes'> 2058373076704  
>>> b[0]-----10  
>>> b[-1]-----89  
>>> b[0]=123-----TypeError:  
'bytes' object does not support item assignment
```

Mutability and Immutability

Mutability

=>An object is said to Mutable if and only if It satisfies the following Property
"Mutable object allows us to perform Operation at Same Address"
Examples: bytearray, list, set, dict

Immutable

=>An object is said to Immutable if and only if It satisfies the following Properties.
a) Immutable Object NEVER allows us to perform Operations at Same Address (It Performs Operations / Modifies the values and Modified value placed at Different Address).
b) Immutable Objects never supports Item Assignment
Examples: int, float, bool, complex, str, ,range bytes,tuple,set,frozenset,NoneType

3. bytearray

Properties

=>'bytearray' is one of the pre-defined class and treated as sequence data type.
=>The purpose of bytearray data type is that "To store the range of values from 0 to 256(0 to 255 only) for the implementation of End-to-End Encryption Conversion (Encrypted Format) "
=>bytearray data type does not contain any symbolic notation for storing bytearray data but we can convert any type data into bytearray data by using bytearray()
=>on the object of bytearray , we can perform both Indexing and Slicing Operations.
=>An object of bytearray data type maintains Insertion Order. which is nothing but whatever the order we organize, in the same order, data will be displayed.
=>An object of bytearray belongs to mutable bcoz bytearray object supports item assignment.

Note: The Functionality of bytearray data type is exactly similar to bytes data type But an object of bytes belongs to immutable where as an object of bytearray belongs to Mutable.

Examples

```
>>> lst=[10,23,256,0,28,119]
>>> print(lst,type(lst))-----[10, 23, 256, 0, 28, 119]
<class 'list'>
>>> ba=bytearray(lst)-----ValueError: byte must be in range(0, 256)
>>> lst=[10,-23,255,0,28,119]
>>> print(lst,type(lst))-----[10, -23, 255, 0, 28, 119]
<class 'list'>
>>> ba=bytearray(lst)-----ValueError: byte must be in range(0, 256)
>>> lst=[10,23,255,0,28,119]
>>> print(lst,type(lst))-----[10, 23, 255, 0, 28, 119]
<class 'list'>
>>> ba=bytearray(lst)
>>> print(ba,type(ba))
bytearray(b'\n\x17\xff\x00\x1cw') <class 'bytearray'>
>>> print(ba,type(ba),id(ba))
bytearray(b'\n\x17\xff\x00\x1cw') <class 'bytearray'>
1619059523056
>>> print(ba[0])-----10
>>> print(ba[-1])-----119
>>> for val in ba:
...     print(val)
...     10
...     23
...     255
...     0
...     28
...     119
>>> ba[0]=200    # Updation
>>> for val in ba:
...     print(val)
...     200
...     23
...     255
...     0
...     28
...     119
```

```
>>> print(ba,type(ba),id(ba))-----
bytearray(b'\xc8\x17\xff\x00\x1cw') <class 'bytearray'>
1619059523056
>>> for val in ba[0:3]:
...     print(val)
...     200
...     23
...     255
```

range

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

=>The purpose of range data type is that "To store Sequnce of Numerical Integer values by maintaining equal Interval of Value i.e step".

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

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

=>To organize the range of values in an object of range class, we have Three Syntaxes. They are

Syntax-1: range(Value)

=>This Syntax generates range of values from 0 to Value-1

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 range(10):
...     print(v)
...     0
...     1
...     2
...     3
...     4
...     5
```

```

6
7
8
9
>>> r[2]-----2
>>> r[-1]-----5
>>>r=range(6)
>>> for v in r[::-1]:
...     print(v)

5
4
3
2
1
0
>>> r[1]=10-----TypeError: 'range' object does not
support item assignment
-----
Syntax-2: range(BEGIN,END)
-----
=>This Syntax Generates range of Values from BEGIN to END-1
Values
-----
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(15,21):
...     print(val)
        15
        16
        17
        18
        19
        20

>>> r=range(20,26)
>>> print(r,type(r))-----range(20, 26) <class 'range'>

```

```
>>> r[0]-----20
>>> r[-1]-----25
>>> for val in r[2:4]:
...     print(val)
        22
        23
```

=>In the above Two Syntaxes, we are getting range of Values in Forward Direction with default Step 1

Syntax-3: range(BEGIN,END,STEP)

=>This Syntax Generates range of Values from BEGIN to END-1 Values with Equal Interval of Value by maintaining Specified STEP Value either Forward OR BackWard Direction.

Examples

```
>>> r=range(10,17,2)
>>> print(r,type(r))
range(10, 17, 2) <class 'range'>
>>> for val in r:
...     print(val)
        10
        12
        14
        16
>>> for val in range(10,17,2):
...     print(val)
        10
        12
        14
        16
>>> for val in range(3,31,3):
...     print(val)
        3
        6
        9
        12
        15
        18
        21
        24
        27
        30
>>> for val in range(3,3*10+1,3):
...     print(val)
        3
```

```
6  
9  
12  
15  
18  
21  
24  
27  
30
```

Implementation Examples Sheet by using range data--
range(val), range(begin,end), range(beg,end,step)

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

OR range(0,10) OR range(0,10,1)

>>> for val in range(10):

```
...     print(val)  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Q2) 10 11 12 13 14 15 16 17 18 19 20---

range(10,21) OR range(10,21,1)

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

```
...     print(val)  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20
```

Q3) 50 52 54 56 58 60----range(50,61,2)

>>> for val in range(50,61,2):

```
...     print(val)
      50
      52
      54
      56
      58
      60
-----
Q4)  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
-----
Q5) -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
-----
Q6) -10   -9    -8    -7    -6    -5    -4    -3   --2   -1   --range (-
10,0, 1)
-----
>>> for val in range(-10,0,1):
...     print(val)
      -10
      -9
      -8
      -7
      -6
      -5
      -4
      -3
```

```

        -2
        -1
        (OR)
>>> for val in range(-10, 0):
...     print(val)
        -10
        -9
        -8
        -7
        -6
        -5
        -4
        -3
        -2
        -1
-----
Q7) -100  -90   -80   -70   -60   -50-----range(-100,-49,10)
>>> for val in range(-100, -49, 10):
...     print(val)
        -100
        -90
        -80
        -70
        -60
        -50
-----
Q1) -1000, -1020 , -1040 -1060-----range(-1000, -1061 ,-20)
>>> for val in range(-1000, -1061 ,-20):
...     print(val)
-1000
-1020
-1040
-1060
-----
Q8)  100   90    80    70    60    50-----range(100,49,-10)
-----
>>> for val in range(100,49,-10):
...     print(val)
        100
        90
        80
        70
        60
        50
-----
Q9)  -5    -4    -3    -2    -1     0     1     2     3     4     5    ----range(-
5,6,1) OR range (-5,6)

```

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

```
Q1) -1000 -2000 -3000 -4000 -5000----range(-1000,-5001, -1000)  
>>> for val in range(-1000,-5001, -1000):  
...     print(val)  
      -1000  
      -2000  
      -3000  
      -4000  
      -5000
```

List Category Data Types (Collections Data Types)

=>The purpose of List Category Data Types is that " To store Multiple Values either of Same Type or

Different type or Both Types in single object with Unique and duplicates ".

=>List Category contains 2 Data Types. They are

1. list (Mutable)

2. tuple (Immutable)

list

Index

=>Purpose of list

=>Types of lists

=>Operations on list

- a) Indexing
- b) slicing

=>Pre-defined Functions in list

=>Inner OR Nested 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 Multiple Values either of Same Type or

Different type or Both Types in single object with Unique and duplicates. "

=>The Data or Elements of list must written or Enclosed within Square Barckets [] and the elments must separated by comma.

=>An object of list maintains Insertion Order.

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

=>An object of list belongs to Mutable

=>To convert one type of object to list type object, we use list(object)

listobj=list(obj)

=>In Python Programming, we can cerate Two types of List objects. They are

- a) empty list
 - b) non-empty list
-

a) Empty List

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

=>Syntax1: listobj=[]

=>Syntax2: listobj=list()

b) Non-Empty List

```
=>A Non-Empty List is one, which contains Elements and whose  
length is >0  
=>Syntax1:      listobj=[Val1,Val2,...,Val-n]  
=>Syntax2:      listobj=list(object)
```

Examples

```
>>> lst1=[10,"Ashis",11.23,True,"OUCET","Python"]  
>>> print(lst1,type(lst1))-----[10, 'Ashis', 11.23, True,  
'OUCET', 'Python'] <class 'list'>  
>>> lst2=[10,20,30,40,10,10,10]  
>>> print(lst2,type(lst2))-----[10, 20, 30, 40, 10, 10,  
10] <class 'list'>  
>>> lst1=[10,"Ashis",11.23,True,"OUCET","Python"]  
  
>>> print(lst1,type(lst1),id(lst1))  
[10, 'Ashis', 11.23, True, 'OUCET', 'Python'] <class 'list'>  
2393969444096  
>>> lst1[0]-----10  
>>> lst1[-1]-----'Python'  
>>> lst1[1:3]-----['Ashis', 11.23]  
>>> lst1[::-2]-----[10, 11.23, 'OUCET']  
>>> lst1[::-1]-----['Python', 'OUCET', True, 11.23,  
'Ashis', 10]  
  
>>> print(lst1,type(lst1),id(lst1))  
[10, 'Ashis', 11.23, True, 'OUCET', 'Python'] <class 'list'>  
2393969444096  
>>> lst1[0]=100  
>>> print(lst1,type(lst1),id(lst1))---[100, 'Ashis', 11.23,  
True, 'OUCET', 'Python'] <class 'list'>  
>>> lst1[0:2]=[25,"Santosh"]  
>>> print(lst1,type(lst1),id(lst1))--[25, 'Santosh', 11.23,  
True, 'OUCET', 'Python'] <class 'list'> 2393969444096  
  
>>> lst1=[]  
>>> print(lst1,type(lst1), len(lst1))-----[] <class  
'list'> 0  
>>> lst2=list()  
>>> print(lst2,type(lst2), len(lst2))-----[] <class  
'list'> 0  
>>> lst3=[10,"Rossum",23.45,"Python"]  
>>> print(lst3,type(lst3), len(lst3))-----[10, 'Rossum',  
23.45, 'Python'] <class 'list'> 4  
  
>>> s1="PYTHON"
```

```
>>> lst1=list(s1)
>>> print(lst1,type(lst1))-----['P', 'Y', 'T', 'H', 'O',
'N'] <class 'list'>
>>> r1=range(10,21,2)
>>> lst2=list(r1)
>>> print(lst2,type(lst2))-----[10, 12, 14, 16, 18, 20]
<class 'list'>
=====
```

Pre-defined Functions in list

=>We know that on the object of list , we can both Indexing and slicing Operations.

=>Along with Indexing and slicing Operations, we can also Perform Various Other Operations by using Pre-defined Functions present in list object.

=>The list object contains the following Pre-defined Functions.

1) append()

Syntax: listobj.append(Value)

This Function is used for adding the value to list object at the end of existing content of list object (called Appending)

Examples

```
>>> lst1=[ ]
>>> print(lst1,type(lst1),id(lst1))-----[ ] <class
'list'> 2393966067968
>>> len(lst1)-----0
>>> lst1.append(100)
>>> lst1.append("Shahoo")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'Shahoo']
<class 'list'> 2393966067968
>>> len(lst1)-----2
>>> lst1.append(23.45)
>>> lst1.append("JNTU")
>>> print(lst1,type(lst1),id(lst1))----[100, 'Shahoo', 23.45,
'JNTU'] <class 'list'> 2393966067968
```

2. insert():

Syntax: listobj.insert(index,Value)

=>This Function is used for inserting the value at Specified Existing Index

=>If we specify Invalid Positive Index then the value inserted at Last

=>If we specify Invalid Negative Index then the value inserted at First

Examples

```
>>> lst=[10,"Rossum",12.34,"Python"]
>>> print(lst,id(lst))-----[10, 'Rossum', 12.34,
'Python'] 2393969457600
>>> lst.insert(2,"NL")
>>> print(lst,id(lst))-----[10, 'Rossum', 'NL',
12.34, 'Python'] 2393969457600
>>> lst.insert(90,"KVR")
>>> print(lst,id(lst))-----[10, 'Rossum', 'NL', 12.34,
'Python', 'KVR'] 2393969457600
>>> lst.insert(-90,"HYD")
>>> print(lst,id(lst))-----['HYD', 10, 'Rossum', 'NL',
12.34, 'Python', 'KVR'] 2393969457600
-----
```

3) clear()

Syntax: listobj.clear()
=>This function is used for removing all the elements of list object
=>If we call this Function upon empty list object then we get None OR Space as Result.

Examples

```
>>> lst=[10,"Rossum",12.34,"Python"]
>>> print(lst,id(lst),len(lst))-----[10, 'Rossum',
12.34, 'Python'] 2393969444096 4
>>> lst.clear()
>>> print(lst,id(lst),len(lst))-----[ ] 23939694440960
-----  

>>> lst=[10,"Rossum",12.34,"Python"]
>>> lst.clear( )
>>> print(lst,id(lst),len(lst))-----[ ] 23939694440960
-----  

>>> lst=[ ]
>>> print(lst,id(lst),len(lst))----- [ ]
23939694576000
>>> print(lst.clear( ))----- None
>>> print([ ].clear( ))-----None
OR
>>> [ ].clear( )-----Space  

>>> print(list( ).clear( ))-----None
(OR)
>>> list( ).clear( )-----Space
```

```
-----  
4) remove( )---Value Based Removal  
-----  
Syntax:    listobj.remove(Value)  
=>This Function is used for Removing The First Occurrence of  
Specified Element of list object.  
=>If the Specified Element is not existing in list object then  
we get ValueError.  
-----  
Examples  
-----  
>>> lst=[10,"Rossum",12.34,"Python"]  
>>> print(lst,id(lst),len(lst))-----[10, 'Rossum',  
12.34, 'Python'] 2393969458048 4  
>>> lst.remove("Rossum")  
>>> print(lst,id(lst),len(lst))-----[10, 12.34,  
'Python'] 2393969458048 3  
>>> lst.remove(10)  
>>> print(lst,id(lst),len(lst))-----[12.34, 'Python']  
2393969458048 2  
>>> lst.remove(12.34)  
>>> print(lst,id(lst),len(lst))-----['Python']  
2393969458048 1  
>>> lst.remove("Python")  
>>> print(lst,id(lst),len(lst))-----[ ] 2393969458048 0  
>>> lst.remove("python")-----ValueError:  
list.remove(x): x not in list  
-----  
>>> lst1=[10,20,30,10,40,20,60]  
>>> print(lst1,type(lst1))-----[10, 20, 30, 10, 40,  
20, 60] <class 'list'>  
>>> lst1.remove(10)  
>>> print(lst1,type(lst1))-----[20, 30, 10, 40, 20,  
60] <class 'list'>  
>>> lst1.remove(10)  
>>> print(lst1,type(lst1))-----[20, 30, 40, 20, 60]  
<class 'list'>  
>>> lst1.remove(20)  
>>> print(lst1,type(lst1))-----[30, 40, 20, 60]  
<class 'list'>  
>>> lst1.remove(200)-----ValueError:  
list.remove(x): x not in list  
-----  
5) pop(index)---Index Based Removal  
-----  
Syntax:    listobj.pop(index)
```

=>This Function is used for removing the value of list based on Index.

=>If the Value of Index is Invalid then we get IndexError

Examples

```
>>> lst1=[10,20,30,10,40,20,60]
>>> print(lst1,type(lst1))-----[10, 20, 30,
10, 40, 20, 60] <class 'list'>
>>> lst1.pop(3)-----10
>>> print(lst1,type(lst1))-----[10, 20, 30, 40,
20, 60] <class 'list'>
>>> lst1.pop(-2)-----20
>>> print(lst1,type(lst1))-----[10, 20, 30, 40,
60] <class 'list'>
>>> lst1.pop(0)-----10
>>> print(lst1,type(lst1))-----[20, 30, 40, 60]
<class 'list'>
>>> lst1.pop(10)-----IndexError: pop
index out of range
>>> lst1.pop(-10)-----IndexError: pop
index out of range
>>> list().pop(0)-----IndexError: pop
from empty list
```

5) pop()

Syntax: listobj.pop()

=>This Function Removes always Last Element from listobject

=>When we this function upon empty list object then we get
IndexError.

Examples

```
>>> lst=[10,"Rossum",34.56,True,2+3j]
>>> print(lst,type(lst),id(lst))----[10, 'Rossum', 34.56, True,
(2+3j)] <class 'list'> 2362516555008
>>> lst.pop( )-----(2+3j)
>>> print(lst,type(lst),id(lst))---[10, 'Rossum', 34.56, True]
<class 'list'> 2362516555008
>>> lst.pop( )-----True
>>> print(lst,type(lst),id(lst))---[10, 'Rossum', 34.56] <class
'list'> 2362516555008
>>> lst.pop( )-----34.56
>>> print(lst,type(lst),id(lst))----[10, 'Rossum'] <class
'list'> 2362516555008
>>> lst.pop( )----'Rossum'
```

```
>>> print(lst,type(lst),id(lst))---[10] <class 'list'>
2362516555008
>>> lst.pop( )-----10
>>> print(lst,type(lst),id(lst))---[] <class 'list'>
2362516555008
>>> lst.pop( )-----IndexError: pop from empty list
>>>>> [ ].pop( )-----IndexError: pop from empty list
>>> list( ).pop( )-----IndexError: pop from empty list
-----
```

7) copy()----- Shallow Copy

```
Syntax1: listobj2=listobj1.copy( )
=>This Function is used for Copying the content of one list
object into another list object ( Implements Shallow Copy )
```

Examples

```
>>> l1=[10,"Rossum"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class
'list'> 2362510195200
>>> l2=l1.copy( ) # Shallow Copy
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum'] <class
'list'> 2362516563904
>>> l1.append("Python")
>>> l2.append("NL")
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'Python']
<class 'list'> 2362510195200
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'NL']
<class 'list'> 2362516563904
>>> l2.insert(1,"Python")
>>> l1.pop( )-----'Python'
>>> print(l1,type(l1),id(l1))---[10, 'Rossum'] <class 'list'>
2362510195200
>>> print(l2,type(l2),id(l2))---[10, 'Python', 'Rossum', 'NL']
<class 'list'> 2362516563904
-----
```

Examples---Deep Copy

```
>>> l1=[10,"Rossum"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class
'list'> 2362516562688
>>> l2=l1 # Deep Copy
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum'] <class
'list'> 2362516562688
>>> l1.append("Python")
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'Python']
<class 'list'> 2362516562688
```

```
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'Python']
<class 'list'> 2362516562688
>>> l2.insert(2,"NL" )
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'NL',
'Python'] <class 'list'> 2362516562688
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'NL',
'Python'] <class 'list'> 2362516562688
-----
```

8) count()

```
=>Syntax: listobj.count(Value)
=>This Function is used for finding Number of Occurences of a
specified Element.
```

```
=>If the Value does not exist then we get its count as 0
```

Examples

```
>>> lst1=[10,20,30,10,10,30,40,10,20]
>>> print(lst1)-----[10, 20, 30, 10, 10, 30,
40, 10, 20]
>>> len(lst1)-----9
>>> lst1.count(10)-----4
>>> lst1.count(20)-----2
>>> lst1.count(30)-----2
>>> lst1.count(40)-----1
>>> lst1.count(50)-----0
>>> lst1.count("HYD")-----0
>>> [ ].count(100)-----0
>>> list( ).count(10)-----0
```

NOTE:

```
-----  
>>> s="MISSISSIPPI"
>>> lst=list(s)
>>> print(lst)-----['M', 'I', 'S', ' ', 'I', 'S',
'S', ' ', 'I', 'P', ' ', 'I']
>>> lst.count("M")-----1
>>> lst.count("I")-----4
>>> lst.count("S")-----4
>>> lst.count("P")-----2
>>> len1=lst.count("M")+lst.count("I")+lst.count("S")+
lst.count("P")
>>> print(len1)-----11
```

9) index()

```
Syntax: lstobj.index(Value)
```

=>This Function is used for Obtaining the Index of the First Occurrence of specified Element.

=>If the Specified Element does not exist then we get ValueError.

Examples

```
>>> lst=[10,"Rossum",34.56,True]
>>> print(lst)-----[10, 'Rossum', 34.56, True]
>>> lst.index("Rossum")----1
>>> lst.index(True)-----3
>>> lst.index(10)-----0
>>> lst.index(100)-----ValueError: 100 is not in list
>>> [ ].index(10)-----ValueError: 10 is not in list
>>> lst.index(100)-----ValueError: 100 is not in list
-----
```

```
>>> lst1=[10,20,30,10,20,30]
>>> print(lst1)-----[10, 20, 30, 10, 20, 30]
>>> lst1.index(10)-----0
>>> for i,v in enumerate(lst1):
...     print(i,"--->",v)
...         0 ---> 10
...         1 ---> 20
...         2 ---> 30
...         3 ---> 10
...         4 ---> 20
...         5 ---> 30
-----
```

10) reverse()

Syntax: lstobj.reverse()

=>This Function is used for reversing the Elements of List object in itself.

Examples

```
>>> lst=[10,"Rossum",34.56,True]
>>> print(lst,id(lst))-----[10, 'Rossum', 34.56, True]
2362513186624
>>> lst1=lst.reverse( ) # Important
>>> print(lst1)-----None
>>> print(lst,id(lst))-----[True, 34.56, 'Rossum', 10]
2362513186624
-----
```

```
>>> lst2=[10,20,-23,-56,12,0,34]
>>> print(lst2,id(lst2))-----[10, 20, -23, -56, 12, 0, 34]
2362513178688
>>> lst2.reverse( )
```

```

>>> print(lst2,id(lst2))-----[34, 0, 12, -56, -23, 20, 10]
2362513178688
-----
11) sort( )
=>Syntax1:      lstobj.sort( )-----Sorts the Data in
                           ASCENDING ORDER
=>Syntax2:      lstobj.sort(reverse=False)---Sorts the Data in
                           ASCENDING ORDER
=>Syntax3:      lstobj.sort(reverse=True)---Sorts the Data in
                           DECENDING ORDER
=>This Function is used for Sorting the Homogeneous (Similar
Type) of Data in Sorting Order.
-----
>>> lst2=[10,20,-23,-56,12,0,34]
>>> print(lst2,id(lst2))-----[10, 20, -23,56,
12, 0, 34] 2362516555008
>>> lst2.sort( )
>>> print(lst2,id(lst2))-----[-56,-23,0,10,12,20,34]
2362516555008
>>> lst2.reverse( )
>>> print(lst2,id(lst2))-----[34, 20, 12, 10, 0, -23, -56]
2362516555008
>>> #-----
>>> lst2=[10,20,-23,-56,12,0,34]
>>> print(lst2,id(lst2))-----[10, 20, -23, -56, 12, 0,
34] 2362513178688
>>> lst2.sort(reverse=True)
>>> print(lst2,id(lst2))-----[34, 20, 12, 10, 0, -23, -
56] 2362513178688
>>> #-----
>>> lst2=[10,20,-23,-56,12,0,34]
>>> print(lst2,id(lst2))-----[10, 20, -23, -56, 12, 0,
34] 2362516555008
>>> lst2.sort(reverse=False)
>>> print(lst2,id(lst2))-----[-56, -23, 0, 10, 12, 20,
34] 2362516555008
>>> #-----
>>> lst1=["trump","binden","modi","anil","travis"]
>>> print(lst1,id(lst1))-----['trump', 'binden', 'modi',
'anil', 'travis'] 2362513178688
>>> lst1.sort( )
>>> print(lst1,id(lst1))-----['anil', 'binden', 'modi',
'travis', 'trump'] 2362513178688
>>> #-----
>>> lst1=["trump","binden","modi","anil","travis"]
>>> print(lst1,id(lst1))-----['trump', 'binden', 'modi',
'anil', 'travis'] 2362510195200

```

```

>>> lst1.sort(reverse=True)
>>> print(lst1,id(lst1))-----['trump', 'travis', 'modi',
'binden', 'anil'] 2362510195200
-----
>>> lst=[10,"Rossum",23.45,True,2+3j]
>>> print(lst)-----[10, 'Rossum', 23.45, True, (2+3j)]
>>> lst.sort( )-----TypeError: '<' not supported
between instances of 'str' and 'int'
-----
12) extend( )

Syntax:      lstobj1.extend(listobj2)
              (OR)
                  lstobj1.extend(listobj2,listobj3.....listobj-n)#
TypeError To Avoid this TypeError, we use the following

Syntax:      lstobj1=lstobj1+lstobj2+.....+lstobj-n

=>This Function is used for Merging Listobj2 values with
Listobj1 Values.
=>This Function will merge one list object with another list
object But not with Multiple List Objects Elements.
=>To Merge Multiple List object Values in one single list
object, we use + Operator
-----

Examples:
-----
>>> l1=[10,20,30,40]
>>> l2=["RS","DR","TR"]
>>> print(l1,id(l1))-----[10, 20, 30, 40]
2362516555008
>>> print(l2,id(l2))-----['RS', 'DR', 'TR']
2362516561216
>>> l1.extend(l2)
>>> print(l1,id(l1))-----[10, 20, 30, 40, 'RS',
'DR', 'TR'] 2362516555008
>>> print(l2,id(l2))-----['RS', 'DR', 'TR']
2362516561216
-----
>>> l1=[10,20,30,40]
>>> l2=[10,20,100,200]
>>> print(l1,id(l1))-----[10, 20, 30, 40]
2362516562688
>>> print(l2,id(l2))-----[10, 20, 100, 200]
2362516555008
>>> l1.extend(l2)

```

```
>>> print(l1,id(l1))-----[10, 20, 30, 40, 10, 20, 100,  
200] 2362516562688  
-----  
>>> l1=[10,20,30,40]  
>>> l2=["RS","DR","TR"]  
>>> l3=[1,3,2,4]  
>>> l1.extend(l2,l3)-----TypeError: list.extend( )  
takes exactly one argument (2 given)  
        (OR)  
>>> l1=[10,20,30,40]  
>>> l2=["RS","DR","TR"]  
>>> l3=[1,3,2,4]  
>>> print(l1,id(l1))-----[10, 20, 30, 40]  
2362513013504  
>>> print(l2,id(l2))-----['RS', 'DR', 'TR']  
2362516561216  
>>> print(l3,id(l3))-----[1, 3, 2, 4] 2362516562688  
>>> l1=l1+l2+l3 # used + Operator  
>>> print(l1,id(l1))-----[10, 20, 30, 40, 'RS', 'DR',  
'TR', 1, 3, 2, 4] 2362516563904
```

Copy Techniques in Python--Most Imp

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

1. Shallow Copy
2. Deep Copy

1. Shallow Copy

=>The Properties of Shallow Copy are

1. The Initial Content of Both the Objects are Same
2. The Memory Address of Both the Objects are Different.
3. The Modifications / Updations are Independent (

Whatever the Changes we do on one object, whose modifications are not Reflecting to other Object)

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

=>Syntax: obj2=obj1.copy()

Examples

```
>>> l1=[10,"Rossum"]  
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class  
'list'> 2362510195200  
>>> l2=l1.copy() # Shallow Copy
```

```
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum'] <class
'list'> 2362516563904
>>> l1.append("Python")
>>> l2.append("NL")
>>> print(l1,type(l1),id(l1))----[10, 'Rossum', 'Python']
<class 'list'> 2362510195200
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'NL']
<class 'list'> 2362516563904
>>> l2.insert(1,"Python")
>>> l1.pop( )-----'Python'
>>> print(l1,type(l1),id(l1))----[10, 'Rossum'] <class 'list'>
2362510195200
>>> print(l2,type(l2),id(l2))----[10, 'Python', 'Rossum', 'NL']
<class 'list'> 2362516563904
```

2. Deep Copy

=>The Properties of Deep Copy are

1. The Initial Content of Both the Objects are Same
2. The Memory Address of Both the Objects are SAME.
3. The Modifications / Updations are Dependent
Whatever the Changes we do on one list object,
(whose modifications are Reflecting to other
Object)

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

=>Syntax: obj2=obj1

Examples

```
>>> l1=[10,"Rossum"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class
'list'> 2362516562688
>>> l2=l1 # Deep Copy
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum'] <class
'list'> 2362516562688
>>> l1.append("Python")
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'Python']
<class 'list'> 2362516562688
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'Python']
<class 'list'> 2362516562688
>>> l2.insert(2,"NL" )
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'NL',
'Python'] <class 'list'> 2362516562688
>>> print(l2,type(l2),id(l2))-----[10, 'Rossum', 'NL',
'Python'] <class 'list'> 2362516562688
```

===== Inner OR Nested List =====

=>The Process of Defining One List inside of another list is called Inner / Nested List.

=>Syntax:

```
listobj=[Val1,Val2,[Val11,Val12,...Val1n],[Val21,Val22....Val2n],....., Val-n]
```

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

=>Here [Val11,Val12..Val1n] is called One Inner / Nested List

=>Here [Val21,Val22..Val2n] is called Other Inner / Nested List

=>On inner List object, we can perform Various Operations By using Pre-defined Functions of list object.

Examples

```
>>> lst=[10,"Rossum",[18,17,19],[78,67,79],"OUCET"]
>>> print(lst,type(lst))-----[10, 'Rossum', [18, 17,
19], [78, 67, 79], 'OUCET'] <class 'list'>
>>> for val in lst:
...         print(val,type(val),type(lst))
...             10 <class 'int'> <class 'list'>
...             Rossum <class 'str'> <class 'list'>
...             [18, 17, 19] <class 'list'> <class 'list'>
...             [78, 67, 79] <class 'list'> <class 'list'>
...             OUCET <class 'str'> <class 'list'>
>>> print(lst[0])-----10
>>> print(lst[1])-----Rossum
>>> print(lst[2])-----[18, 17, 19]
>>> print(lst[-2])-----[78, 67, 79]
>>> print(lst[-1])-----OUCET
>>> print(lst[2])-----[18, 17, 19]
>>> print(lst[2][0])-----18
>>> print(lst[2][-3])-----18
>>> print(lst[2][-1])-----19
>>> print(lst[-2][-2])-----67
>>> lst[2].append(16)
>>> print(lst,type(lst))-----[10, 'Rossum', [18, 17, 19,
16], [78, 67, 79], 'OUCET'] <class 'list'>
>>> lst[-2].insert(1,72)
>>> print(lst,type(lst))-----[10, 'Rossum', [18, 17, 19, 16],
[78, 72, 67, 79], 'OUCET'] <class 'list'>
>>> lst[2].sort( )
>>> print(lst,type(lst))-----[10, 'Rossum', [16, 17, 18, 19],
[78, 72, 67, 79], 'OUCET'] <class 'list'>
>>> lst[3].sort(reverse=True)
```

```

>>> print(lst,type(lst))----[10, 'Rossum', [16, 17, 18, 19],
[79, 78, 72, 67], 'OUCET'] <class 'list'>
>>> lst[2].pop(2)----18
>>> print(lst,type(lst))---[10, 'Rossum', [16, 17, 19], [79, 78,
72, 67], 'OUCET'] <class 'list'>
>>> lst[3].remove(78)
>>> print(lst,type(lst))----[10, 'Rossum', [16, 17, 19], [79,
72, 67], 'OUCET'] <class 'list'>
>>> lst[2].clear()
>>> print(lst,type(lst))----[10, 'Rossum', [], [79, 72, 67],
'OUCET'] <class 'list'>
>>> del lst[3][1]
>>> print(lst,type(lst))---[10, 'Rossum', [], [79, 67], 'OUCET']
<class 'list'>
>>> del lst[3]
>>> print(lst,type(lst))---[10, 'Rossum', [], 'OUCET'] <class
'list'>
>>> del lst[2]
>>> print(lst,type(lst))---[10, 'Rossum', 'OUCET'] <class
'list'>
>>> lst.insert(2,[18,16,19])
>>> print(lst,type(lst))----[10, 'Rossum', [18, 16, 19],
'OUCET'] <class 'list'>
>>> lst.insert(3,[78,66,79])
>>> print(lst,type(lst))----[10, 'Rossum', [18, 16, 19], [78,
66, 79], 'OUCET'] <class 'list'>
>>> del lst[2][::2]
>>> print(lst,type(lst))----[10, 'Rossum', [16], [78, 66, 79],
'OUCET'] <class 'list'>
>>> del lst[-2][1:]
>>> print(lst,type(lst))----[10, 'Rossum', [16], [78], 'OUCET']
<class 'list'>
>>> lst[2].append(18)
>>> lst[2].append(19)
>>> print(lst,type(lst))----[10, 'Rossum', [16, 18, 19], [78],
'OUCET'] <class 'list'>
>>> lst[2].index(18)---1
>>> lst[2].index(29)----ValueError: 29 is not in list
=====
del operator
-----
=>'del' operator is used for Removing any object, Removing
Elements of Nay Object Either Based on Indexing OR Slicing
Operation of Mutable Objects only.

```

Syntax1: `del listobj[Index]`--Removes the Element Based Index of
Mutable Object

Syntax2: `del listobj[Beg:End]`---Removes the Element Based on Slicing of Mutable Object
Syntax3: `del listobj`----->Removes Complete Object of Mutable or Immutable

Examples

```
>>> lst=[10,"Sumnath",45.67,True,2+3j,"HYD"]  
>>> print(lst,id(lst))-----[10, 'Sumnath', 45.67, True, (2+3j),  
'HYD'] 1742369128960  
>>> del lst[2]  
>>> print(lst,id(lst))-----[10, 'Sumnath', True, (2+3j), 'HYD']  
1742369128960  
>>> del lst[::-2]  
>>> print(lst,id(lst))-----[ 'Sumnath', (2+3j)] 1742369128960  
>>> del lst  
>>> print(lst,id(lst))----NameError: name 'lst' is not defined.  
Did you mean: 'list'?  
-----  
>>> s="PYTHON"  
>>> print(s,type(s))-----PYTHON <class 'str'>  
>>> del s[4]-----TypeError: 'str' object  
doesn't support item deletion  
>>> del s[1:3]TypeError: 'str' object does not support item  
deletion  
>>> del s  
>>> print(s,type(s))----NameError: name 's' is not defined
```

Nested OR Inner List

Consider the following Statement
`lst=[10,"Rossum",[18,17,19],[78,67,79],"OUCET"]`

-5	-4	-3	-2	-1
10	Rossum	[18 17 19]	[78 67 79]	OUCET
0	1	2	3	4

=====

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 Multiple Values either of Same type or Different Types or Both Types with Unique and Duplicates in single object (Constant Values)".
=>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 contains any elements and whose length is 0
=>Syntax: tplobj=()
 OR
 tplobj=tuple()

2. Non-Empty tuple:

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

Examples:

```
-----  
tplobj=( Val1,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']

```

===== Set Category Data Types (Collections Data Types) =====

=>The purpose of Set Category Data Types is that " To store Multiple Values either of Same Type or Different type or Both Types in single object with Unique Values only (Duplicates are not allowed) ".

=>Set Category contains 2 Data Types. They are

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

===== set =====

Properties

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

=>The purpose of 'set' data type is that To store Multiple Values either of Same Type or Different type or Both Types in single object with Unique Values only (Duplicates are not allowed)".

=>The Elements of set Must be written / Organized / stored within Curly Braces{ } and Elements must be separated by comma.

=>The Elements of Set Never Maintains Insertion Order Bcoz PVM can display any order of Its Possibilities.

=>On the object of set , we can't perform Indexing and Slicing Operations bcoz set never maintains Insertion order.

=>An object of set belongs to Both Immutable (TypeError: 'set' object does not support item assignment) and Mutable (in the case Adding the elements to set object at same address)

=>In Python Programming, we have Types of sets. They are

- 1) Empty Set
- 2) Non-Empty Set

1.Empty Set

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

=>Syntax: setobj=set()

2.Non-Empty Set

=>A Non-Empty Set is one, which contains Elements and whose length is >0

=>Syntax: setobj={Val1,Val2....Val-n}

(OR)

setobj=set([Val1,Val2..Val-n])

OR

```
setobj=set( (Val1,Val2..Val-n) )
OR
setobj=set(object)
```

Examples

```
>>> s2={10,20,30,40,10,15,25,30}
>>> print(s2,type(s2))-----{20, 40, 25, 10, 30, 15}
<class 'set'>
>>> s1={10,"KVR",45.67,"PYTHON"}
>>> print(s1,type(s1))-----{'KVR', 10, 'PYTHON', 45.67}
<class 'set'>
>>> s1[0]-----TypeError: 'set' object is
not subscriptable
>>> s1[0:3]-----TypeError: 'set' object is not
subscriptable
-----
>>> s1={10,"Mahesh",45.67,True,2+3j}
>>> print(s1,type(s1))-----{True, 'Mahesh', 10, (2+3j),
45.67} <class 'set'>
>>> s1[0]=False-----TypeError: 'set' object does
not support item assignment
>>> s1={10,"Mahesh",45.67,True,2+3j}
>>> print(s1,type(s1),id(s1))-----{True, 'Mahesh', 10,
(2+3j), 45.67} <class 'set'> 2273844253120
>>> s1.add("PYTHON")
>>> print(s1,type(s1),id(s1))----{'PYTHON', True, 'Mahesh', 10,
(2+3j), 45.67} <class 'set'> 2273844253120
-----
>>> s1=set( )
>>> print(s1,type(s1))-----set( ) <class 'set'>
>>> len(s1)-----0
-----
>>> s2={10,"RS",23.45}
>>> print(s2,type(s2))-----{10, 'RS', 23.45} <class 'set'>
>>> len(s2)-----3
=====
```

Pre-Defined function set

```
=>On the object of set we can perform various Operation by using
the 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 object

=>When we call this function on empty set object then we get None OR Space as Result

Examples

```
>>> s1={10,"Siva",34.56,True}  
>>> print(s1,type(s1))-----{True, 10, 34.56, 'Siva'}  
<class 'set'>  
>>> len(s1)-----4  
>>> s1.clear()-----  
>>> print(s1,type(s1))-----set() <class 'set'>  
>>> len(s1)-----0  
>>> print(s1.clear())-----None  
>>> set().clear()-----Space  
OR  
>>> print(set().clear())-----None
```

2. add()

Syntax: setobj.add(Value)

=>This Function is used for adding the value(s) to set object.

Examples

```
>>> s1={10,"RS"}  
>>> print(s1,type(s1),id(s1))-----{10, 'RS'} <class  
'set'> 2273844252672  
>>> s1.append(100)-----AttributeError: 'set' object  
has no attribute 'append'  
>>> s1.add(100)  
>>> print(s1,type(s1),id(s1))-----{10, 100, 'RS'} <class 'set'>  
2273844252672  
>>> s1.add("HYD")  
>>> s1.add(2+3j)  
>>> print(s1,type(s1),id(s1))----{'HYD', 100, 10, (2+3j), 'RS'}  
<class 'set'> 2273844252672  
>>> s=set()  
>>> print(s,type(s),id(s))-----set() <class 'set'>  
2273844253120  
>>> s.add(10)  
>>> s.add(1.2)  
>>> s.add("PYthon")  
>>> print(s,type(s),id(s))----{'PYthon', 1.2, 10} <class 'set'>  
2273844253120
```

3. remove()

Syntax: setobj.remove(Value)
=>This Function is used for removing the Value from set object.
=>When we call remove() on empty set object then we get KeyError

Examples

```
>>> s1={10,"Rossum",34.56,True,2+3j}
>>> print(s1,type(s1))-----{True, 34.56, 10, (2+3j),
'Rossum'} <class 'set'>
>>> s1.remove(2+3j)
>>> print(s1,type(s1))-----{True, 34.56, 10, 'Rossum'}
<class 'set'>
>>> s1.remove(10)
>>> print(s1,type(s1))-----{True, 34.56, 'Rossum'} <class
'set'>
>>> s1.remove(True)
>>> print(s1,type(s1))-----{34.56, 'Rossum'} <class
'set'>
>>> s1.remove("Rossum")
>>> print(s1,type(s1))-----{34.56} <class 'set'>
>>> s1.remove(34.56)
>>> print(s1,type(s1))-----set() <class 'set'>
>>> s1.remove(100)-----KeyError: 100
>>> set().remove(10.25)-----KeyError: 10.25
-----
```

4) discard()

=>Syntax: setobj.discard(Value)
=>This Function is used for Removing the Value from set object
=>When we call discard() on empty set object then we never get
get KeyError

Examples

```
>>> s1={10,"Rossum",34.56,True,2+3j}
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 10, (2+3j),
'Rossum'} <class 'set'> 2354582181408
>>> s1.discard(10)
>>> print(s1,type(s1),id(s1))-----{True, 34.56, (2+3j),
'Rossum'} <class 'set'> 2354582181408
>>> s1.discard(True)
>>> print(s1,type(s1),id(s1))-----{34.56, (2+3j), 'Rossum'}
<class 'set'> 2354582181408
>>> s1.discard(34.56)
```

```

>>> print(s1,type(s1),id(s1))----{ (2+3j), 'Rossum' } <class
'set'> 2354582181408
>>> s1.discard("Rossum")
>>> print(s1,type(s1),id(s1))----{ (2+3j) } <class 'set'>
2354582181408
>>> s1.discard(2+3j)
>>> print(s1,type(s1),id(s1))-----set() <class 'set'>
2354582181408
>>> s1.discard("HYD") # Will not give KeyError
>>> print(s1,type(s1),id(s1))-----set() <class 'set'>
2354582181408
>>> s1.remove("HYD")-----KeyError: 'HYD'
-----
5) pop( )
-----
=>Syntax: setobj.pop( )
=>This Function is used for Removing any Arbitray Element from
set object (When Order of Display Not Given)
=>This Function is used for Removing First Element from set
object (When Order of Display Given)
=>When we call pop( ) on empty set object then we get KeyError.
-----
Example -1 --without Order of Display
-----
>>> s1={10,"Rossum",34.56,True,2+3j}
>>> s1.pop()-----True
>>> s1.pop()-----34.56
>>> s1.pop()-----10
>>> s1.pop()----- (2+3j)
>>> s1.pop()----- 'Rossum'
>>> print(s1,type(s1),id(s1))---set() <class 'set'>
2354582181632
>>> s1.pop()-----KeyError: 'pop from an empty set'
-----
Example-2--Order of Display
-----
>>> s1={"Sahoo","Nilesh","KVR",20,True,False,45.67,"Python"}
>>> print(s1,type(s1))----{False, True, 45.67, 'Nilesh',
'Sahoo', 'Python', 'KVR', 20} <class 'set'>
>>> s1.pop()-----False
>>> s1.pop()-----True
>>> s1.pop()-----45.67
>>> s1.pop()----- 'Nilesh'
>>> s1.pop()----- 'Sahoo'
>>> s1.pop()----- 'Python'
>>> s1.pop()----- 'KVR'
>>> s1.pop()-----20

```

```
>>> 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  
to another set object.( Implements Shallow Copy).  
-----  
Examples  
-----  
>>> s1={10,"Sanvi",34.56,True}  
>>> print(s1,type(s1),id(s1))-----{True, 10, 'Sanvi',  
34.56} <class 'set'> 2354582180288  
>>> s2=s1.copy() # Shallow Copy  
>>> print(s2,type(s2),id(s2))-----{True, 10, 'Sanvi', 34.56}  
<class 'set'> 2354582180960  
>>> s1.add("Python")  
>>> s2.add(1000)  
>>> print(s1,type(s1),id(s1))----{True, 34.56, 10, 'Python',  
'Sanvi'} <class 'set'> 2354582180288  
>>> print(s2,type(s2),id(s2))----{True, 34.56, 1000, 10,  
'Sanvi'} <class 'set'> 2354582180960  
-----  
7) isdisjoint( )  
-----  
Syntax: setobj1.isdisjoint(setobj2)  
=>This Function returns True provided setobj1 and setobj2 does  
not contain common elements.  
=>This Function returns False provided setobj1 and setobj2  
contains common elements.  
-----  
Examples  
-----  
>>> s1={10,20,30,40}  
>>> s2={15,25,35}  
>>> s3={10,45,55,65}  
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>  
>>> print(s2,type(s2))-----{25, 35, 15} <class 'set'>  
>>> print(s3,type(s3))-----{65, 10, 45, 55} <class 'set'>  
>>> s1.isdisjoint(s2)-----True  
>>> s1.isdisjoint(s3)-----False  
>>> s2.isdisjoint(s3)-----True  
-----  
8) 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.
=>This Function Returns False provided all the elements of setobj1 not present in setobj2 (OR) This Function Returns False provided setobj2 does not contain all the elements of setobj1.

Examples

```
>>> s1={10,20,30,40,50}
>>> s2={10,20,30}
>>> s3={10,20,30,45,55}
>>> s4={1,2,3,4}
>>> print(s1,type(s1))-----{50, 20, 40, 10, 30} <class
'>
'set'
>>> print(s2,type(s2))-----{10, 20, 30} <class 'set'>
>>> print(s3,type(s3))-----{20, 55, 10, 45, 30} <class
'>
'set'
>>> print(s4,type(s4))-----{1, 2, 3, 4} <class 'set'>
>>> s2.issubset(s1)-----True
>>> s3.issubset(s1)-----False
>>> s4.issubset(s1)-----False
>>> s1.issubset(s2)-----False
>>> s1.isdisjoint(s3)-----False
>>> s1.isdisjoint(s4)-----True
```

9) issuperset()

Syntax: setobj1.issuperset(setobj2)
=>This Function returns True provided setobj1 contains all the elements of setobj2 OR all the elements of setobj2 are present in setobj1.
=>This Function returns False provided setobj1 does not contain all the elements of setobj2 OR all the elements of setobj2 are not present in setobj1.

Examples

```
>>> s1={10,20,30,40,50}
>>> s2={10,20,30}
>>> s3={10,20,30,45,55}
>>> s4={1,2,3,4}
>>> print(s1,type(s1))-----{50, 20, 40, 10, 30} <class
'>
'set'
>>> print(s2,type(s2))-----{10, 20, 30} <class 'set'>
>>> print(s3,type(s3))-----{20, 55, 10, 45, 30} <class
'>
'set'
```

```
>>> print(s4,type(s4))-----{1, 2, 3, 4} <class 'set'>
>>> s1.issuperset(s2)-----True
>>> s1.issuperset(s3)-----False
>>> s1.issuperset(s4)-----False
```

```
-----  
10) union( )
```

```
Syntax: setobj3=setobj1.union(setobj2)
=>This Function is used for Taking all the Quinique Elements of
setobj1 and setobj2 and Place them in setobj3.
```

```
-----  
Examples
```

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

```
-----  
11) intersection( )
```

```
Syntax: setobj1.intersection(setobj2)
=>This Function is used for Obtaining Common Elements from both
setobj21 and setobj2
```

```
-----  
Examples
```

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



```
>>> s1={10,20,30,40}
>>> s2={100,200,150,250}
>>> s3=s1.intersection(s2)
>>> print(s3,type(s3))-----set() <class 'set'>
```

```
-----  
12) difference( )
```

```
Syntax: setobj3=setobj1.difference(s2)
```

=>This Function removes common Elements from setobj1 and setobj2 and Takes Remaining Elements from setobj1 and place them setobj3.

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,15,25}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))-----{25, 10, 20, 15} <class 'set'>
>>> s3=s1.difference(s2)
>>> print(s3,type(s3))-----{40, 30} <class 'set'>
>>> s3=s2.difference(s1)
>>> print(s3,type(s3))-----{25, 15} <class 'set'>
```

13) symmetric_difference()

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

=>This Function removes common Elements from setobj1 and setobj2 and Takes Remaining Elements from Both setobj1 and setobj2 and place them setobj3.

By Formula: setobj3=setobj1.symmetric_difference(setobj2)

```
setobj3=setobj1.union(setobj2).difference(setobj1.intersection(s
etobj2))
```

Examples:

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

OR

```
>>> s1={10,20,30,40}
>>> s2={10,20,15,25}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))-----{25, 10, 20, 15} <class 'set'>
>>> s3=s1.union(s2).difference(s1.intersection(s2)) # By Formula
>>> print(s3,type(s3))-----{40, 25, 30, 15} <class 'set'>
```

14) update()

```
-----  
=>Syntax:      setobj1.update(setobj2)  
=>This Function is used for adding all elements of setobj2 to  
setobj1  
-----  
Examples:  
-----  
>>> s1={10,20,30,40}  
>>> s2={10,20,15,25}  
>>> print(s1,type(s1))---->{40, 10, 20, 30} <class 'set'>  
>>> print(s2,type(s2))---->{25, 10, 20, 15} <class 'set'>  
>>> s1.update(s2)  
>>> print(s1,type(s1))---->{40, 10, 15, 20, 25, 30} <class 'set'>  
>>> s3=s1.update(s2)  
>>> print(s3)-----None  
=====
```

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

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 / add.

=>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 where as frozenset object belongs to immutable bcoz neither Item Assignment permitted nor additions is permitted.

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'
=====
```

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

=>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.

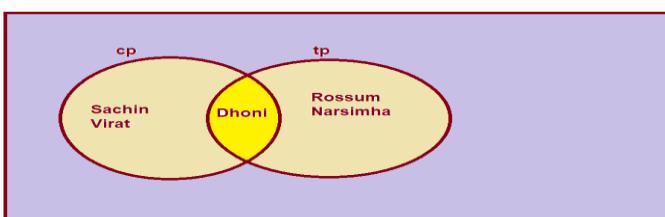
=>frozenset does not contain the following Functions

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

Examples:

```
>>> print(fs1,type(fs1),id(fs1))-----frozenset({50, 20, 70,
40, 10, 60, 30}) <class 'frozenset'> 1558323909504
>>> fs2=fs1.copy()
>>> print(fs2,type(fs2),id(fs2))-----frozenset({50, 20, 70, 40,
10, 60, 30}) <class 'frozenset'> 1558323909504
>>> print(fs1)-----frozenset({50, 20, 70, 40, 10, 60, 30})
-----
>>> fs1=frozenset({10,20,30,40,50,60,70})
>>> fs2=frozenset((10,20,30))
>>> fs1.issuperset(fs2)-----True
>>> fs2.issuperset(fs1)-----False
>>> fs2.issubset(fs1)-----True
>>> fs1.issubset(fs2)-----False
-----
>>> fs1=frozenset({10,20,30,40,50,60,70})
>>> fs2=frozenset((100,200,300))
>>> fs3=frozenset((10,2,3))
>>> fs1.isdisjoint(fs2)-----True
>>> fs1.isdisjoint(fs3)-----False
>>> print(fs1)-----frozenset({50, 20, 70, 40, 10, 60, 30})
>>> print(fs2)-----frozenset({200, 100, 300})
>>> fs1.union(fs2)-----frozenset({100, 70, 40, 200, 10,
300, 50, 20, 60, 30})
>>> fs1.intersection(fs2)-----frozenset()
>>> fs1.difference(fs2)--frozenset({70, 40, 10, 50, 20, 60, 30})
>>> fs2.difference(fs1)-----frozenset({200, 100, 300})
>>>
frozenset({10,20,30,40}).symmetric_difference(frozenset([10,20,5
0,60]) )
          frozenset({40, 50, 60, 30})
>>> fs1|fs2-----frozenset({100, 70, 40, 200, 10, 300,
50, 20, 60, 30})
```

Ven Diagram



===== Problem Statement by using Sets with Ven Diagrams =====

=>Let Us Consider the set of Cricket Players(cp) and Tennis Players(tp)

```
cp={"Sachin","Virat","Dhoni"}  
tp={"Dhoni","Rossum","Narshima"}
```

Solve the following Question

- Q1) Find All Player Names 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 playing EXCLUSIVELY Cricket and Tennis---symmetric_difference()

===== Solution---Approach1----Using Set Functions =====

```
=====Q1)  
Find All Player Names who are playing all the Games---union()  
ANS: >>> cp={"Sachin","Virat","Dhoni"}  
>>> tp={"Dhoni","Rossum","Narshima"}  
>>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}  
 <class 'set'>  
>>> print(tp,type(tp))----{'Rossum', 'Narshima',  
 'Dhoni'} <class 'set'>  
>>> allcptp=cp.union(tp)  
>>> print(allcptp,type(allcptp))----{'Rossum',  
 'Narshima', 'Sachin', 'Dhoni', 'Virat'} <class 'set'>
```

```
=====
Q2) Find All the Players who are playing Both Cricket and
Tennis---intersection()
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
                           <class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
                                    'Dhoni'} <class 'set'>
      >>> bothcptp=cp.intersection(tp)
      >>> print(bothcptp,type(bothcptp))----{'Dhoni'} <class
'set'>
=====
```

Q3) Find All the Players who are playing only Cricket but not tennis---difference()

```
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
                           <class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
                                    'Dhoni'} <class 'set'>
      >>> onlycp=cp.difference(tp)
      >>> print(onlycp,type(onlycp))----{'Sachin', 'Virat'}
                           <class 'set'>
=====
```

Q4) Find All the Players who are playing only Tennis but not Cricket--- difference()

```
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
                           <class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
                                    'Dhoni'} <class 'set'>
      >>> onlytp=tp.difference(cp)
      >>> print(onlytp,type(onlytp))----{'Rossum',
                                         'Narshima'} <class 'set'>
=====
```

Q5)Find All the Players who are playing EXCLUSIVELY Cricket and Tennis---symmetric_difference()

```
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
                           <class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
                                    'Dhoni'} <class 'set'>
```

```

>>> exclcptp=tp.symmetric_difference(cp)
>>> print(exclcptp)-----{'Rossum', 'Narshima',
                           'Sachin', 'Virat'}
=====
Solution---Approach2---(Don't Use Set Functions)---Now We can
Solve It By using BITWISE OPERATORS
=====
Q1) Find All Player Names who are playing all the Games---
Bitwise OR ( | )
=====
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
<class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
'Dhoni'} <class 'set'>
      >>> allcptp=cp|tp
      >>> print(allcptp)----{'Rossum', 'Narshima', 'Sachin',
'Dhoni', 'Virat'}
=====
Q2) Find All the Players who are playing Both Cricket and
Tennis---Bitwise AND ( & )
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
<class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
'Dhoni'} <class 'set'>
      >>> bothcptp=cp&tp
      >>> print(bothcptp)----{'Dhoni'}
=====
Q3) Find All the Players who are playing only Cricket but not
tennis--- Subtract Operator ( - )
=====
ANS: >>> cp={"Sachin","Virat","Dhoni"}
      >>> tp={"Dhoni","Rossum","Narshima"}
      >>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}
<class 'set'>
      >>> print(tp,type(tp))----{'Rossum', 'Narshima',
'Dhoni'} <class 'set'>
      >>> onlycp=cp-tp
      >>> print(onlycp,type(onlycp))---- {'Sachin',
'Virat'} <class 'set'>
=====
Q4) Find All the Players who are playing only Tennis but not
Cricket---Subtract Operator ( - )
=====
```

```
ANS: >>> cp={"Sachin","Virat","Dhoni"}  
>>> tp={"Dhoni","Rossum","Narshima"}  
>>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}  
                                <class 'set'>  
>>> print(tp,type(tp))----{'Rossum', 'Narshima',  
                                'Dhoni'} <class 'set'>  
>>> onlytp=tp-cp  
>>> print(onlytp,type(onlytp))----{'Rossum', 'Narshima'}  
                                <class 'set'>  
=====
```

Q6) Find All the Players who are playing EXCLUSIVELY Cricket and Tennis---Bitwise XOR (^)

```
ANS: >>> cp={"Sachin","Virat","Dhoni"}  
>>> tp={"Dhoni","Rossum","Narshima"}  
>>> print(cp,type(cp))----{'Dhoni', 'Sachin', 'Virat'}  
                                <class 'set'>  
>>> print(tp,type(tp))----{'Rossum', 'Narshima',  
                                'Dhoni'} <class 'set'>  
>>> exclcptp=cp^tp  
>>> print(exlcptp,type(exlcptp))----{'Rossum',  
                                'Narshima', 'Sachin', 'Virat'} <class 'set'>  
=====
```

**Dict Category Data Types (Collections Data Types)
(OR)
Dict data type**

=>'dict' ism one of the pre-defined class and treated dict data type
=>The purpose of dict data type is that "To store the data in the form of (Key,Value) ".
=>In (Key,value), the Values of Key are Unique and Values of Value may or may not be Unique.
=>To Store the data in the form of (Key,Value) in dict object, we use Curly Braces and (Key,value) Must separated comma.

=>Syntax: dictobj={Key1:Val1,Key2:Val2,.....,Key-n:Val-n}
----- here Key1,Key2....Key-n are called Values of Key
Here val1,val2....val-n are called Values of Value
=>An object of dict maintains Insertion Order.
=>In Python Programming, we can create Two Types of dict objects. They are

- 1) Empty Dict Object
- 2) Non-Empty Dict Object

1) Empty Dict Object

=>An Empty Dict Object is one, which does not contain any (Key,value) and whose length is 0

=>Syntax: dictobj={}
 (OR)
 dictobj=dict()

2) Non-Empty Dict Object

=>A Non-Empty Dict Object is one, which contains any (Key,value) and whose length is >0

=>Syntax: dictobj={Key1:Val1,Key2:Val2,.....,Key-n:Val-n}
 here Key1,Key2....Key-n are called Values of Key
 Here val1,val2....val-n are called Values of Value

=>On the object of Dict, we can't Perform Indexing and Slicing Operations bcoz Values of Key Itself acts as Indices.

=>An object of dict belongs to Mutable. In (Key,value), the Values of Key belongs to Immutable and Values of Value are belongs to Mutable.

=>We can the (Key,Value) to dict object by using the following Syntax

```
dictobj[Key1]=Val1
dictobj[Key2]=Val2
-----
-----
dictobj[Key-n]=Val-n
```

Examples:

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1))-----{10: 'RS', 20: 'MC', 30: 'DR',
40: 'RS'} <class 'dict'>
>>> d2={100:1.2,200:3.4,300:1.2,400:4.5}
>>> print(d2,type(d2))-----{100: 1.2, 200: 3.4, 300: 1.2,
400: 4.5} <class 'dict'>
-----
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1))-----{10: 'RS', 20: 'MC', 30:
'DR', 40: 'RS'} <class 'dict'>
>>> len(d1)-----4
>>> d2={}
>>> print(d2,type(d2))-----{} <class 'dict'>
>>> d3=dict()
>>> print(d3,type(d3))-----{} <class 'dict'>
```

```

>>> len(d2)-----0
>>> len(d3)-----0
-----
>>> d1={10:"RS",10:"VN",10:"TR"}
>>> print(d1,type(d1))-----{10: 'TR'} <class 'dict'>
-----
>>> d1={}
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'>
2383457965120
>>> len(d1)-----0
>>> d1["PYTHON"]="ROSSUM"
>>> print(d1,type(d1),id(d1))-----{'PYTHON': 'ROSSUM'} <class 'dict'> 2383457965120
>>> d1["JAVA"]="JGOSLING"
>>> d1["C++"]="STUP"
>>> d1["C"]="DR"
>>> print(d1,type(d1),id(d1))----{'PYTHON': 'ROSSUM', 'JAVA': 'JGOSLING', 'C++': 'STUP', 'C': 'DR'}
>>> d1["C"]="DENNIS"
>>> print(d1,type(d1),id(d1))----{'PYTHON': 'ROSSUM', 'JAVA': 'JGOSLING', 'C++': 'STUP', 'C': 'DENNIS'}

<class 'dict'> 2383457965120
-----
>>> d1=dict()
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'> 2383462276608
>>> len(d1)-----0
>>> d1[10]="Apple"
>>> d1[20]="Mango"
>>> d1[30]="Kiwi"
>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango', 30: 'Kiwi'} <class 'dict'> 2383462276608
>>> len(d1)-----3
=====
```

Pre-Defined Functions in dict object

=>To perform Various Operations on dict object, we use Pre-defined Functions present in dict object. They are

1) clear()

=>Syntax: dictobj.clear()

=>This Function is used for Removing all the (Key,value) from dict object

=>When we call this function on empty dict object then we get None as a Result

Examples

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}  
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30:  
'DR', 40: 'RS'} <class 'dict'> 2383457965120  
>>> len(d1)-----4  
>>> d1.clear()  
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'>  
2383457965120  
>>> len(d1)-----0  
>>> print(d1.clear( ))-----None  
>>> print({}.clear( ))-----None  
>>> print(dict( ).clear( ))-----None
```

2) copy()

Syntax: dictobj2=dictobj1.copy()
=>This Function is used for Copying the content of one Dict Object into another dict object(Shallow Copy)

Examples

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}  
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30: 'DR',  
40: 'RS'} <class 'dict'> 2383458322624  
>>> d2=d1.copy() # Shallow Copy  
>>> print(d2,type(d2),id(d2))-----{10: 'RS', 20: 'MC', 30:  
'DR', 40: 'RS'} <class 'dict'> 2383457965120  
>>> d1[50]="KV"  
>>> d1[15]="SS"  
>>> d2[15]="RS"  
>>> print(d1,type(d1))---{10: 'RS', 20: 'MC', 30: 'DR',  
40: 'RS', 50: 'KV', 15: 'SS'} <class 'dict'> 2383458322624  
>>> print(d2,type(d2))-----{10: 'RS', 20: 'MC', 30:  
'DR', 40: 'RS', 15: 'RS'} <class 'dict'> 2383457965120
```

3) pop()

=>Syntax: dictobj.pop(Key)
=>This function is used for Removing (Key,Value) from non-empty dict object.
=>If the Value of Key does not exist in non-empty dict object then we get KeyError
=>If we call this function on empty dict object then we get KeyError

Examples

```
-----
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30:
'DR', 40: 'RS'} <class 'dict'> 2383457964928
>>> d1.pop(20)-----'MC'
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 30: 'DR', 40:
'RS'} <class 'dict'> 2383457964928
>>> d1.pop(10)-----'RS'
>>> print(d1,type(d1),id(d1))-----{30: 'DR', 40: 'RS'}
<class 'dict'> 2383457964928
>>> d1.pop(40)-----'RS'
>>> print(d1,type(d1),id(d1))----{30: 'DR'} <class 'dict'>
2383457964928
>>> d1.pop(30)-----'DR'
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'>
2383457964928
>>> d1.pop(40)-----KeyError: 40
>>> dict().pop(10)-----KeyError: 10
>>> {}.pop(100)-----KeyError: 100
-----
```

4) popitem()

```
=>Syntax: dictobj.popitem( )
=>This Function is used for Removing Last (Key,Value) of non-
empty dict object.
=>If we call this function on empty dict object then we get
KeyError
-----
```

Examples:

```
-----
```

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1),id(d1))----{10: 'RS', 20: 'MC', 30: 'DR',
40: 'RS'} <class 'dict'> 2383462276096
>>> d1.popitem()-----{40, 'RS'}
>>> print(d1,type(d1),id(d1))----{10: 'RS', 20: 'MC', 30: 'DR'}
<class 'dict'> 2383462276096
>>> d1.popitem()-----{30, 'DR'}
>>> print(d1,type(d1),id(d1))----{10: 'RS', 20: 'MC'} <class
'dict'> 2383462276096
>>> d1.popitem()-----{20, 'MC'}
>>> print(d1,type(d1),id(d1))----{10: 'RS'} <class 'dict'>
2383462276096
>>> d1.popitem()-----{10, 'RS'}
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'>
2383462276096
>>> d1.popitem()-----KeyError: 'popitem(): dictionary is
empty'
```

```

>>> {}.popitem()-----KeyError: 'popitem(): dictionary
is empty'
>>> dict().popitem()-----KeyError: 'popitem(): dictionary
is empty'
-----
5) get( )
-----
=>Syntax: val=dictobj.get(Key)
=>This Function is used for obtaining Value of Value By passing
Value of Key.
=>If the Value of Key does not exist then we get None as a
Result
                (OR)
-----
=>Syntax: dictobj[Key]
This Syntax also gives Value of Value By passing Value of Key
If the Value of Key does not exist then we get KeyError
-----
Examples
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1),id(d1))----{10: 'RS', 20: 'MC', 30:
'DR', 40: 'RS'} <class 'dict'> 2383462276928
>>> val=d1.get(10)
>>> print(val)-----RS
>>> val=d1.get(20)
>>> print(val)-----MC
>>> val=d1.get(40)
>>> print(val)-----RS
>>> val=d1.get(120)
>>> print(val)-----None
>>> print({}.get(10))----None
>>> print(dict().get(200))----None
-----
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1),id(d1))----{10: 'RS', 20: 'MC',
30: 'DR', 40: 'RS'} <class 'dict'> 2383462276736
>>> print(d1[10])-----RS
>>> print(d1[20])-----MC
>>> print(d1.get(10))-----RS
>>> print(d1[120])-----KeyError: 120
>>> print(d1.get(120))-----None
-----
6) keys( )
-----
Syntax: Varname=dictobj.keys()
=>This Function is used for Obtaining Values of Key and Placed
in LHS Varname and whose type is <class, dict_keys>

```

Examples:

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}  
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC',  
30: 'DR', 40: 'RS'} <class 'dict'> 2383462276928  
>>> ks=d1.keys()  
>>> print(ks,type(ks))----dict_keys([10, 20, 30, 40]) <class  
'dict_keys'>  
>>> ks={}.keys()  
>>> print(ks,type(ks))-----dict_keys([]) <class 'dict_keys'>  
-----  
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}  
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30:  
'DR', 40: 'RS'} <class 'dict'> 2383462276096  
>>> ks=d1.keys()  
>>> for k in ks:  
...     print(k)  
...     10  
...     20  
...     30  
...     40  
>>> for k in d1.keys():  
...     print(k)  
...     10  
...     20  
...     30  
...     40
```

7) values()

Syntax: Varname=dictobj.values()
=>This Function is used for Obtaining Values of Value and
Placed in LHS Varname and whose type is <class, dict_values>

Examples

```
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}  
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30:  
'DR', 40: 'RS'} <class 'dict'> 2383462276736  
>>> vs=d1.values()  
>>> print(vs,type(vs))----dict_values(['RS', 'MC', 'DR',  
'RS']) <class 'dict_values'>  
>>> for v in vs:  
...     print(v)  
...     RS  
...     MC
```

```

        DR
        RS
>>> for v in d1.values():
...     print(v)
        RS
        MC
        DR
        RS
-----
8) items( )

Syntax:      Varname=dictobj.items( )
=>This Function is used for Obtaining (Key,Value) in the form
of tuples of list and Placed in LHS Varname and whose type is
<class, dict_items>.
-----
Examples
-----
>>> d1={10:"RS",20:"MC",30:"DR",40:"RS"}
>>> print(d1,type(d1),id(d1))-----{10: 'RS', 20: 'MC', 30:
'DR', 40: 'RS'} <class 'dict'> 2383462276928
>>> kvs=d1.items()
>>> print(kvs,type(kvs))----dict_items([(10, 'RS'), (20, 'MC'),
(30, 'DR'), (40, 'RS')]) <class 'dict_items'>
>>> for kv in kvs:
...     print(kv)
        (10, 'RS')
        (20, 'MC')
        (30, 'DR')
        (40, 'RS')
>>> for k,v in kvs:
...     print(k,"---->",v)
        10 ----> RS
        20 ----> MC
        30 ----> DR
        40 ----> RS
>>> for x,y in d1.items():
...     print(x,"--->",y)
        10 ---> RS
        20 ---> MC
        30 ---> DR
        40 ---> RS
-----
9) update( )

=>Syntax: dictobj1.update(dictobj2)
```

```

=>This Function is used for Updating / Adding One dict
(Key,Value) to Another dict object
-----
=>Examples
-----
>>> d1={10:"Apple",20:"Mango"}
>>> d2={100:"RS",200:"DR"}
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango'}
<class 'dict'>
>>> print(d2,type(d2))-----{100: 'RS', 200: 'DR'} <class
'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango',
100: 'RS', 200: 'DR'} <class 'dict'>
>>> print(d2,type(d2))-----{100: 'RS', 200: 'DR'} <class
'dict'>
>>> #-----
>>> d1={10:"Apple",20:"Mango"}
>>> d2={10:"KIWI",30:"SBerry"}
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango'}
<class 'dict'>
>>> print(d2,type(d2))-----{10: 'KIWI', 30: 'SBerry'}
<class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{10: 'KIWI', 20: 'Mango', 30:
'SBerry'} <class 'dict'>
>>> print(d2,type(d2))-----{10: 'KIWI', 30: 'SBerry'}
<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 of dict data type

```

======>The Process of Defining One Dict inside of another of another
dict Object is called Inner OR Nested dict.
=>Syntax:dictobj={"Key1":"Val1","Key2":"Val2", {"Key11":"Val11",
"Key12":"Val12"}..... "Keyn":"Valn"}
```

```

=>Here {"Key11":"Val11","Key12":"Val12"} Represents Inner Dict.
=>Here {"Key1":"Val1","Key2":"Val2", {"Key11":"Val11","Key12":"Val12"} ..... "Keyn":"Valn"} Represents Outer dict
-----
Examples
-----
>>>d1={"sno":10,"sname":"Rossum","IntMarks":{"cm":17,"cppm":16,"os":19}, "ExtMarks":{"cm":67,"cppm":77,"os":78}, "cname":"OUCET"}
>>> print(d1,type(d1))
{'sno': 10, 'sname': 'Rossum', 'IntMarks': {'cm': 17, 'cppm': 16, 'os': 19}, 'ExtMarks': {'cm': 67, 'cppm': 77, 'os': 78}, 'cname': 'OUCET'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"-->",v)
        sno --> 10
        sname --> Rossum
        IntMarks --> {'cm': 17, 'cppm': 16, 'os': 19}
        ExtMarks --> {'cm': 67, 'cppm': 77, 'os': 78}
        cname --> OUCET
-----
>>> for k,v in d1.items():
...     print(k,"-->",v,"-->",type(v),type(d1))
        sno --> 10 --> <class 'int'> <class 'dict'>
        sname --> Rossum --> <class 'str'> <class 'dict'>
        IntMarks --> {'cm': 17, 'cppm': 16, 'os': 19} -->
                    <class 'dict'> <class 'dict'>
        ExtMarks --> {'cm': 67, 'cppm': 77, 'os': 78} -->
                    <class 'dict'> <class 'dict'>
        cname --> OUCET --> <class 'str'> <class 'dict'>
-----
>>>d1={"sno":10,"sname":"Rossum","IntMarks":{"cm":17,"cppm":16,"os":19}, "ExtMarks":{"cm":67,"cppm":77,"os":78}, "cname":"OUCET"}
>>> for k in d1.keys():
...     print(k)
        sno
        sname
        IntMarks
        ExtMarks
        cname
>>> for k in d1.values():
...     print(k)
        10
        Rossum
        {'cm': 17, 'cppm': 16, 'os': 19}
        {'cm': 67, 'cppm': 77, 'os': 78}
        OUCET
>>> for k in d1.items():

```

```

...     print(k)
        ('sno', 10)
        ('sname', 'Rossum')
        ('IntMarks', {'cm': 17, 'cppm': 16, 'os': 19})
        ('ExtMarks', {'cm': 67, 'cppm': 77, 'os': 78})
        ('cname', 'OUCET')
>>>d1={"sno":10,"sname":"Rossum","IntMarks":{"cm":17,"cppm":16,"os":19},"ExtMarks":{"cm":67,"cppm":77,"os":78},"cname":"OUCET"}
>>> for x in d1:
...     print(x)
        sno
        sname
        IntMarks
        ExtMarks
        cname
-----
>>>d1={"sno":10,"sname":"Rossum","IntMarks":{"cm":17,"cppm":16,"os":19},"ExtMarks":{"cm":67,"cppm":77,"os":78},"cname":"OUCET"}
>>> for x in d1:
...     print(x,"--->",d1.get(x))
        sno ---> 10
        sname ---> Rossum
        IntMarks ---> {'cm': 17, 'cppm': 16, 'os': 19}
        ExtMarks ---> {'cm': 67, 'cppm': 77, 'os': 78}
        cname ---> OUCET
>>> #-----OR-----
>>> for x in d1:
...     print(x,"--->",d1[x])
        sno ---> 10
        sname ---> Rossum
        IntMarks ---> {'cm': 17, 'cppm': 16, 'os': 19}
        ExtMarks ---> {'cm': 67, 'cppm': 77, 'os': 78}
        cname ---> OUCET
-----
>>>d1={"sno":10,"sname":"Rossum","IntMarks":{"cm":17,"cppm":16,"os":19},"ExtMarks":{"cm":67,"cppm":77,"os":78},"cname":"OUCET"}
>>> for x in d1.keys():
...     print(x,"--->",d1[x])
        sno ---> 10
        sname ---> Rossum
        IntMarks ---> {'cm': 17, 'cppm': 16, 'os': 19}
        ExtMarks ---> {'cm': 67, 'cppm': 77, 'os': 78}
        cname ---> OUCET
>>> #-----OR-----
>>> for x in d1.keys():
...     print(x,"--->",d1.get(x))
        sno ---> 10

```

```

        sname ---> Rossum
        IntMarks ---> {'cm': 17, 'cppm': 16, 'os': 19}
        ExtMarks ---> {'cm': 67, 'cppm': 77, 'os': 78}
                           cname ---> OUCET
=====

list in dict--POSSIBLE
-----
>>>d1={"sno":10,"sname":"Rossum","Subs":["C","C++","DSA"],"cname":"OU"}
>>> print(d1,type(d1))
{'sno': 10, 'sname': 'Rossum', 'Subs': ['C', 'C++', 'DSA'],
 'cname': 'OU'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"-->",v,"--->",type(v),"-->",type(d1))
        sno --> 10 ---> <class 'int'> --> <class
                                         'dict'>
        sname --> Rossum ---> <class 'str'> -->
                                         <class 'dict'>
        Subs --> ['C', 'C++', 'DSA'] ---> <class
                                         'list'> --> <class 'dict'>
        cname --> OU ---> <class 'str'> --> <class
                                         'dict'>

>>> d1["Subs"]-----['C', 'C++', 'DSA']
>>> d1.get("Subs")-----['C', 'C++', 'DSA']
=====

tuple in dict--POSSIBLE
-----
>>>d1={"sno":10,"sname":"Rossum","Subs":("C","C++","DSA"),"cname":"OU"}
>>> print(d1,type(d1))
{'sno': 10, 'sname': 'Rossum', 'Subs': ('C', 'C++', 'DSA'),
 'cname': 'OU'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"-->",v,"--->",type(v),"-->",type(d1))
        sno --> 10 ---> <class 'int'> --> <class 'dict'>
        sname --> Rossum ---> <class 'str'> --> <class
                                         'dict'>
        Subs --> ('C', 'C++', 'DSA') ---> <class 'tuple'> --
                                         <class 'dict'>
        cname --> OU ---> <class 'str'> --> <class 'dict'>

-----

set in dict--POSSIBLE
-----
>>>d1={"sno":10,"sname":"Rossum","Subs":{"C","C++","DSA"},"cname":"OU"}
>>> print(d1,type(d1))

```

```

{'sno': 10, 'sname': 'Rossum', 'Subs': {'C++', 'C', 'DSA'},  

'cname': 'OU'} <class 'dict'>  

>>> for k,v in d1.items():  

...     print(k,"-->",v,"--->",type(v),"-->",type(d1))  

sno --> 10 ---> <class 'int'> --> <class 'dict'>  

sname --> Rossum ---> <class 'str'> --> <class  

                           'dict'>  

Subs --> {'C++', 'C', 'DSA'} ---> <class 'set'> -->  

                           <class 'dict'>  

cname --> OU ---> <class 'str'> --> <class 'dict'>  

>>> d1.get("Subs")-----{'C++', 'C', 'DSA'}  

>>> d1.get("Subs").add("KVR")  

>>> print(d1,type(d1))  

{'sno': 10, 'sname': 'Rossum', 'Subs': {'C++', 'C', 'KVR',  

'DSA'}, 'cname': 'OU'} <class 'dict'>  

>>> d1["Subs"].remove("KVR")  

>>> print(d1,type(d1))  

{'sno': 10, 'sname': 'Rossum', 'Subs': {'C++', 'C', 'DSA'},  

'cname': 'OU'} <class 'dict'>
=====
dict in list--POSSIBLE
-----
>>> l1=[10,"Rossum", {"DBMS":16,"OS":18,"CG":15}, "OUCET"]  

>>> print(l1,type(l1))----[10, 'Rossum', {'DBMS': 16, 'OS': 18,  

'CG': 15}, 'OUCET'] <class 'list'>  

>>> for val in l1:  

...     print(val,type(val))
    10 <class 'int'>
    Rossum <class 'str'>
    {'DBMS': 16, 'OS': 18, 'CG': 15} <class 'dict'>
    OUCET <class 'str'>
>>> for val in l1:  

...     print(val,type(val),type(l1))
    10 <class 'int'> <class 'list'>
    Rossum <class 'str'> <class 'list'>
    {'DBMS': 16, 'OS': 18, 'CG': 15} <class 'dict'>
                                         <class 'list'>
    OUCET <class 'str'> <class 'list'>  

>>> l1[2]-----{'DBMS': 16, 'OS': 18, 'CG': 15}  

>>> for k,v in l1[2].items():  

...     print(k,"-->",v)
    DBMS --> 16
    OS --> 18
    CG --> 15
>>> l1[2]["C++"]=17

```

```

>>> print(l1,type(l1))
[10, 'Rossum', {'DBMS': 16, 'OS': 18, 'CG': 15, 'C++': 17},
'OUCET'] <class 'list'>

=====
dict in tuple---Possible
-----
>>> t1=(10,"Rossum", {"DBMS":16,"OS":18,"CG":15}, "OUCET")
>>> print(t1,type(t1))---(10, 'Rossum', {'DBMS': 16, 'OS': 18,
'CG': 15}, 'OUCET') <class 'tuple'>
>>> for val in t1:
...     print(val,type(val),type(t1))
...         10 <class 'int'> <class 'tuple'>
...             Rossum <class 'str'> <class 'tuple'>
...                 {'DBMS': 16, 'OS': 18, 'CG': 15} <class 'dict'>
<class 'tuple'> OUCET <class 'str'> <class 'tuple'>

>>> t1[2]----{'DBMS': 16, 'OS': 18, 'CG': 15}
>>> t1[2] ["CM"]=12
>>> print(t1,type(t1))
(10, 'Rossum', {'DBMS': 16, 'OS': 18, 'CG': 15, 'CM': 12},
'OUCET') <class 'tuple'>
>>> t1[2] ["CG"]=20
>>> print(t1,type(t1))
(10, 'Rossum', {'DBMS': 16, 'OS': 18, 'CG': 20, 'CM': 12},
'OUCET') <class 'tuple'>
>>> t1[2].pop("OS")----18
>>> print(t1,type(t1))---(10, 'Rossum', {'DBMS': 16, 'CG': 20,
'CM': 12}, 'OUCET') <class 'tuple'>
=====
dict in set----Not Possible
=====
>>> s1={10,"Rossum", {"DBMS":16,"OS":18,"CG":15}, "OUCET"}---
TypeError: unhashable type: 'dict'
=====

Note:
-----
1) List in List-----Possible
2) List in tuple-----Possible
3) List in set-----Not Possible
4) List in dict-----Possible
-----
1) tuple in List-----Possible
2) tuple in tuple-----Possible
3) tuple in set-----Possible
4) tuple in dict-----Possible
-----

```

```

1) set in List-----Possible
2) set in tuple-----Possible
3) set in set-----Not Possible
4) set in dict-----Possible
-----
1) dict in List-----Possible
2) dict in tuple-----Possible
3) dict in set-----Not Possible
4) dict in dict-----Possible
=====Special Points for Converstion to dict type(Tuples of list into dict type)
-----
>>> l1=[(10,"Rossum"),(20,"Travis"),(30,"Kinney")]
>>> print(l1,type(l1))----[(10, 'Rossum'), (20, 'Travis'), (30, 'Kinney')] <class 'list'>
>>> for val in l1:
...     print(val,type(val),type(l1))
...         (10, 'Rossum') <class 'tuple'> <class 'list'>
...             (20, 'Travis') <class 'tuple'> <class 'list'>
...                 (30, 'Kinney') <class 'tuple'> <class 'list'>
>>> d1=dict(l1)
>>> print(d1,type(d1))----{10: 'Rossum', 20: 'Travis', 30: 'Kinney'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,v)
...
...         10 Rossum
...             20 Travis
...                 30 Kinney
-----
>>> t1=((10,"Rossum"),(20,"Travis"),(30,"Kinney"))
>>> print(t1,type(t1))
((10, 'Rossum'), (20, 'Travis'), (30, 'Kinney')) <class 'tuple'>
>>> d1=dict(t1)
>>> print(d1,type(d1))
{10: 'Rossum', 20: 'Travis', 30: 'Kinney'} <class 'dict'>
>>> l11=[[10,"Rossum"],[20,"Travis"],[30,"Kinney"]]
>>> print(l11,type(l11))
[[10, 'Rossum'], [20, 'Travis'], [30, 'Kinney']] <class 'list'>
>>> d1=dict(l11)
>>> print(d1,type(d1))
{10: 'Rossum', 20: 'Travis', 30: 'Kinney'} <class 'dict'>
=====
>>> l1=[10,20,30,40]
>>> l2=["RS","TR","MC","DR"]
>>> z=zip(l1,l2)

```

```

>>> d=dict(z)
>>> print(d,type(d))-----{10: 'RS', 20: 'TR', 30: 'MC',
40: 'DR'} <class 'dict'>
-----
>>> for k,v in d.items():
...     print(k,v)
        10 RS
        20 TR
        30 MC
        40 DR
=====
NoneType DataType
=====
=>'NoneType' is a pre-defined class and Treated as NoneType
DataType
=>None is a Keyword and It is the value of NoneType DataType.
=>None Value is not a space, False, zero
=>An object of NoneType can't be created bcoz It contains Only
Value not possible to convert other type of values into NoneType
=====
Examples
=====
>>> a=None
>>> print(a,type(a))-----None <class 'NoneType'>
>>> None=123-----SyntaxError: cannot assign
to None
>>> a=NoneType( )-----NameError: name
'NoneType' is not defined
>>> None==0-----False
>>> None==""-----False
>>> None==False-----False
>>> None==True-----False
>>> None==None-----True
-----
>>> print([ ].clear( ))-----None
>>> print({ }.clear( ))-----None
>>> print({ }.get(10))-----None
=====
```

Number of Approaches to develop the Programs in Python Lang

```

=====
-----
=>Definition of Program
-----
=>A Program is a collection Well-Defined Instructions to Perform
a Task.
```

=>The purpose of writing a Program is that "To develop OR Solve Real Time Problems".

=>In Python Language, Python Programmer can define Well-Defined Instructions in a single Unit and Save on Some File name with an extension .py(File Name.py)

=>In Python Programming Environment, we have TWO Approaches to Develop the Program. They are

- 1) By Using Interactive Approach.
- 2) By using Batch Mode Approach.

1) By Using Interactive Approach.

=>In this Mode of development, Python Programmer can Issue a single statement at a time and gets the Result of that statement Immediately.

=>The advantage of Interactive Approach is that "To test One Instruction at a time".

=>In Real Time, we don't use Interactive Approach for developing Program bcoz It never allows us to save the set of instructions on some file name and more over we can't re-use those un-named instructions in other part of the Project.

Examples

Q) Addition of TWO Values---Python Command Prompt

```
>>> a=100
>>> b=200
>>> c=a+b
>>> print(a) -----100
>>> print(b) -----200
>>> print(c) -----300
```

Examples Software:Python Command Prompt Python IDLE Shell

NOTE: The above Software will come on the Installation of Python Software.

2) By using Batch Mode Approach.

=>In This Mode of development, Python Programmer can develop Well-Defined Instructions (Program) for Solving a real Time Problem and giving a File Name with an Extension called .py (File Name.py) and whose advantage is that re-use those named instructions in other part of the Project.

Example IDEs for Developing Batch Mode Programming

- 1) PYTHON IDLE SHELL (On the installation of Python Software)
 - 2) PyChram IDE
 - 3) Anakonda Jupiter Note Book
 - 4) Spider
 - 5) atom
 - 6) VS Code
 - 7) Google Clab
 - 6) Sublime Text
 - 7) EDIT PLUS
 -etc
-

```
#program for adding Two Values
a=10
b=20
c=a+b
print(a)
print(b)
print(c)
```

```
#Program for adding of two values
a=12
b=23
c=a+b
print("Val of a=",a)
print("Val of b=",b)
print("Sum=",c)
```

```
#Program for adding two values
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
c=a+b
print("====")
print("First Value:",a)
print("Second Value:",b)
print("Sum=",c)
print("====")
```

Display the Result of Python Program on the Console

- =>To Display the Result of Python Program on the Console, we use a pre-defined function called `print()`.
- =>In otherwords, `print()` is one the pre-defined function used for displaying the result of Python program on the console.
- =>`print()` can be used in 6 Ways / Syntaxes
-

Syntax-1 : `print(Val1)`
 (OR)

```

        print(Val1,Val2,...,Val-n)
-----
=>This Syntax displays Values.
Examples:
-----
>>> a=10
>>> print(a)-----10
>>> b=12.34
>>> print(b)-----12.34
>>> print(a,b)-----10 12.34
-----
Syntax-2 :      print(Msg1)
                OR
                print(Msg1,Msg2,...Msg-n)
-----
=>This Syntax displays Message(s) which are in the form str type
-----
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
>>> s1="Python"
>>> s2="Prog"
>>> s3="Lang"
>>> print(s1,s2,s3)-----Python Prog Lang
>>> print(s1+s2+s3)-----PythonProgLang
>>> print(s1+" "+s2+" "+s3)-----Python Prog Lang
-----
>>> a=10
>>> b=20
>>> print(a+b)-----30 -- Here + Operator used for
addition
>>> a="Python"
>>> b="Lang"
>>> print(a+b)-----PythonLang
-----
>>> a="10"
>>> b="20"
>>> print(a+b)-----1020--- Here + Operator used for
concatination
>>> s=int(a)+int(b)
>>> print(s)-----30
-----
>>> a=10

```

```

>>> b="Python"
>>> c=a+b-----TypeError: unsupported
operand type(s) for +: 'int' and 'str'
>>> c=str(a)+b
>>> print(c)-----10Python
-----
MOST IMP: Here The operator * is originally used for Mul of
Numerical values Here The operator * can also be used for
Repetition Operator
-----
Examples
-----
>>> s="Python"
>>> s=s+s
>>> print(s)-----PythonPythonPython
-----
>>> s="Python"
>>> s=s*3
>>> print(s)-----PythonPythonPython
>>> s="Python"
>>> s=3*s
>>> print(s)-----PythonPythonPython
-----
>>> s="="
>>> print(s*20)----- =====
>>> print(40*s)-----=====
>>> print(40**"")-----
*****
>>> a=2
>>> b=3
>>> print(a*b)-----6
>>> print(9*"6")-----666666666
>>> print("9"*6)-----999999
>>> print("9""*6")-----TypeError: can't multiply sequence
by non-int of type 'str'
>>> print("9""*6")-----TypeError: can't multiply
sequence by non-int of type 'str'
-----
>>> print("9""*6""*5)-----TypeError: can't multiply
sequence by non-int of type 'str'
>>> print("9"+"6""*5)-----966666
>>> print(("9"+"6")*5)-----9696969696
-----
Syntax-3:      print(Values Cum Messages)
                           (OR)
                           print(Messages cum Values)

```

 =>This syntax displays Values cum Messages OR Messages Cum
 Values

 Examples:

```

>>>a=10
>>> print("Val of a=",a)-----Val of a= 10
>>> print("Val of a="+a)-----TypeError: can only
concatenate str (not "int") to str
>>> print("Val of a="+str(a))-----Val of a=10
>>> print(a," is the val of a")-----10 is the val of a
>>> print(str(a)+" is the val of a")-----10 is the val of a
    -----
```

```

>>> a=10
>>> b=20
>>> c=30
>>> 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(Values Cum Messages with format())
 (OR)
 print(Messages cum Values with format())

 =>This syntax displays Values cum Messages OR Messages Cum
 Values by using format()

 Examples

```

>>> a=10
>>> print("Val of a={}".format(a))-----Val of a=10
>>> print("{} is the val of a".format(a))-----10 is the val of
a
    -----
```

```

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

```
>>> print("sum of {} and {}={}".format(a,b,c))-----sum of  
10 and 20=30  
-----  
>>> sno=10  
>>> sname="Ram"  
>>> print("My Number is {} and Name is {}".format(sno,sname))---  
-My Number is 10 and Name is Ram  
-----  
>>> a=10  
>>> b=20  
>>> c=a+b  
>>> print("sum({},{})={}".format(a,b,c))-----sum(10,20)=30  
-----  
>>> a=4  
>>> b=5  
>>> c=a*b  
>>> print("{} x {}={}".format(a,b,c))-----4 x 5=20  
-----
```

Syntax-5 : print(Values Cum Messages with format specifiers)
(OR)
 print(Messages cum Values with format specifiers)

=>This syntax displays Values cum Messages OR Messages Cum
Values by using format Specifiers
=>In Python programming, %d is used for displaying Integer Data,
%f is for float data and %s is for str data. If any data does
not contain Format Specifier then we must convert the
corresponding value to str type by using str()

Examples

```
>>> a=10  
>>> b=20  
>>> c=a+b  
>>> print("val of a=%d" %a)-----val of a=10  
>>> print("Val of b=%d" %b)-----Val of b=20  
>>> print("sum=%d" %c)-----sum=30  
>>> print("Sum of %d and %d=%d" %(a,b,c))-----Sum of 10 and  
20=30  
>>> print("sum(%d,%d)=%d" %(a,b,c))-----sum(10,20)=30  
-----  
>>> a=1.2  
>>> b=2.3  
>>> c=a+b  
>>> print("sum(%f,%f)=%f" %(a,b,c))-----  
sum(1.200000,2.300000)=3.500000
```

```

>>> print("sum(%0.2f,%0.2f)=%0.2f" %(a,b,c))-----
sum(1.20,2.30)=3.50
-----
>>> sno=10
>>> sname="Ram"
>>> print("My Number is %d and Name is %s" %(sno,sname))-----
My Number is 10 and Name is Ram
>>> print("My Number is {} and Name '{}' ".format(sno,sname))---
My Number is 10 and Name 'Ram'
>>> print("My Number is %d and Name is '%s'" %(sno,sname))---My
Number is 10 and Name is 'Ram'
-----
>>> lst=[10,"Vinod",44.44,True,2+3j]
>>> print(lst)-----[10, 'Vinod', 44.44, True, (2+3j)]
>>> print("Content of lst=",lst)-----Content of lst= [10,
'Vinod', 44.44, True, (2+3j)]
>>> print("Content of lst={}".format(lst))-----Content of
lst=[10, 'Vinod', 44.44, True, (2+3j)]
>>> print("content of lst=%s" %str(lst))-----content of
lst=[10, 'Vinod', 44.44, True, (2+3j)]
-----
>>> a=10
>>> b=1.2
>>> c=a+b
>>> print("Sum of %d and %f=%f" %(a,b,c))----Sum of 10 and
1.200000=11.200000
>>> print("Sum of %f and %f=%f" %(a,b,c))----Sum of 10.000000
and 1.200000=11.200000
>>> print("Sum of %d and %d=%d" %(a,b,c))----Sum of 10 and 1=11
>>> print("Sum of %0.2f and %0.2f=%0.3f" %(a,b,c))----Sum of
10.00 and 1.20=11.200
-----
Syntax-6 : print(Value Cum Message, end=" ")
-----
=>This Display the result of python program in same line
-----
Examples
-----
>>> for val in range(6):
...         print(val,end=" ")-----0 1 2 3 4 5

>>> for val in range(100,120,2):
...     print("{}".format(val),end=" ")-----100 102 104 106 108 110
112 114 116 118

```

Reading the Data OR Input Dynamically From Key Board

=>To Read the Data OR Input Dynamically From Key Board, we have TWO Pre-defined Functions. They are

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

1) input()

=>Syntax: varname=input()

=>This Function is used for Reading the Input OR Data from Key Board and that data placed in LHS Variable always in the form of <class, str>

=>To convert the str data into our own format , we use Type Casting Functions

2) input(Message)

=>Here "Message" Represents User Prompting Message

=>This Function is also Used reading the data Dynamically from Key Board int the form of str and placed in LHS Variable and This Function additionally Prompts "User-Prompting Messages".

=>To convert the str data into our own format , we use Type Casting Functions

#program for accepting Two Values and find sum

```
#DataReadEx1.py
print("Enter First Value:")
s1=input()
print("Enter Second Value:")
s2=input()
#Convert s1 and s2 into float type
a=float(s1)
b=float(s2)
#add them
c=a+b
print("=*50)
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(c))
print("=*50)
```

#program for accepting Two Values and find sum

```
#DataReadEx2.py
print("Enter Two Values:")
s1=input()
s2=input()
```

```

a=float(s1)
b=float(s2)
c=a+b
print("=*50")
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(c))
print("=*50")


---


#program for accepting Two Values and find sum
#DataReadEx3.py
print("Enter Two Values:")
a=float(input())
b=float(input())
c=a+b
print("=*50")
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(c))
print("=*50")


---


#program for accepting Two Values and find sum
#DataReadEx4.py
print("Enter Two Values:")
c=float(input())+float(input())
print("=*50")
print("Sum={}".format(c))
print("=*50")


---


#program for accepting Two Values and find sum
#DataReadEx5.py
print("Enter Two Values:")
a=float(input())
b=float(input())
print("=*50")
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(a+b))
print("=*50")


---


#Program accepting Two Numerical values and multiply them
#DataRead6.py
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
c=a*b
print("Mul({},{})={}".format(a,b,c,))


---


#Program accepting Two Numerical values and multiply them
#DataRead7.py
a=float(input("Enter First value:"))
b=float(input("Enter Second value:"))

```

```

print("Mul({}, {})={}".format(a,b,a*b))
print("=====OR=====")
print("Mul({}, {})={}".format(a,b,round(a*b,2)))
print("=====OR=====")
print("Mul(%0.2f, %0.2f)=%0.2f" %(a,b,a*b))


---


#Program accepting Two Numerical values and multiply them
#DataRead7.py
print("Mul={}".format(float(input("Enter First value:"))*float(input("Enter Second value:"))))


---


#Write a python program which will calculate Area of clricle
#CircleArea.py
r=float(input("Enter radius:"))
ac=3.14*r*r
print("*"*50)
print("Radius:{}".format(r))
print("Area of Circle:{}".format(ac))
print("=====OR=====")
print("Area of Circle:{}".format(round(ac,2)))
print("*"*50)


---


#Write a python programe which will calculate area and perimeter
of rectangle?
#AreaPeriRect.py
l=float(input("\tEnter Length:"))
b=float(input("\tEnter Breadth:"))
#cal area
ar=l*b
#cal peri
pr=2*(l+b)
print("*"*50)
print("\t\tLength:{}".format(l))
print("\t\tBreadth:{}".format(b))
print("\t\tArea of Rect:{}".format(ar))
print("\t\tPerimeter of Rect:{}".format(pr))
print("*"*50)

```

Operators and Expressions in Python

- =>An Operator is a Symbol which is used for Performing an Operation on Objects Data / Values.
- =>If any Operator connects with Two Or More Objects then It is Called Expression.
- =>In Python Programming, we have 7 Types of Operators. They are

1. Arithmetic Operators
2. Assignment Operator
3. Relational Operators (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: Python Does not Support Unary Operators (`++` `--`) of C,C++, Java and C#.net

NOTE2: Python Does Not Support Ternary Operator (`? :`) of C, C++, Java and C#.net

NOTE3: Python Lang Contains its Own Ternary Operator
(`if...else` Operator) .

NOTE4: Python Lang Contains Short Hand Operators.

1. Arithmetic Operators

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

=>If Two or More Objects / Values Connected with Arithmetic Operators then It is Called Arithmetic Expression.

=>Arithmetic Operators are Classified into 7 Types. They are given in the following table.

SLNO	SYMBOL	MEANING	EXAMPLES	<code>a=10</code>	<code>b=3</code>
1	<code>+</code>	Addition	<code>print(a+b)</code>	-----	<code>>13</code>
2.	<code>-</code>	Substraction	<code>print(a-b)</code>	-----	<code>>7</code>
3.	<code>*</code>	Multiplication	<code>print(a*b)</code>	-----	<code>>30</code>
4.	<code>/</code>	Division (Float Quotient)	<code>print(a/b)</code>	<code>-- >3.33333333335</code>	
5.	<code>//</code>	Floor Division (Integer Quotient)	<code>print(a//b)</code>	-----	<code>>3</code>
6.	<code>%</code>	Modulo Division (Remainder)	<code>print(a%b)</code>	-----	<code>>1</code>
7.	<code>**</code>	Exponentiation (Power Operator)	<code>print(a**b)</code>	-----	<code>>1000</code>

NOTE:

```
>>> print(10/3)-----3.333333333333335
>>> print(10//3)-----3
>>> #-----
>>> print(10.0/3.0)-----3.333333333333335
>>> print(10.0//3.0)-----3.0
>>> print(10//3.0)-----3.0
>>> print(10.0//3)-----3.0
```

```
#Program for Demonstrating Arithmetic Operators
#ArithOpEx1.py
a=int(input("Enter First value:"))
b=int(input("Enter Second value:"))
print("*"*50)
print("\tResults of Arithmetic Operators")
print("*"*50)
print("\tSum({},{})={}".format(a,b,a+b))
print("\tSub({},{})={}".format(a,b,a-b))
print("\tMul({},{})={}".format(a,b,a*b))
print("\tNormalDiv({},{})={}".format(a,b,a/b))
print("\tFloor Div({},{})={}".format(a,b,a//b))
print("\tModuloDiv({},{})={}".format(a,b,a%b))
print("\tPower({},{})={}".format(a,b,a**b))
print("*"*50)
```

```
#Program for Demonstrating Arithmetic Operators
#ArithOpEx1.py
a=int(input("Enter First value:"))
b=int(input("Enter Second value:"))
print("*"*50)
print("\tResults of Arithmetic Operators")
print("*"*50)
print("\tSum({},{})={}".format(a,b,a+b))
print("\tSub({},{})={}".format(a,b,a-b))
print("\tMul({},{})={}".format(a,b,a*b))
print("\tNormalDiv({},{})={}".format(a,b,a/b))
print("\tFloor Div({},{})={}".format(a,b,a//b))
print("\tModuloDiv({},{})={}".format(a,b,a%b))
print("\tPower({},{})={}".format(a,b,a**b))
print("*"*50)
```

```
#Program for cal Square Root of a Given Number
#ArithOpEx3.py
n=float(input("Enter a Number for Cal Square Root:"))
res=n**0.5
print("SquareRoot({})={}".format(n,res))
```

=====

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

>>> a,b=10,20
>>> print(a,b)-----10 20
>>> a,b=b,a # Swapping Logic
>>> print(a,b)-----20 10

===== **3. Relational Operators (Comparision Operators)** =====

=>The Purpose of Relational Operators is that "To Compare Two Values".

=>If Two Objects / Values Connected with Relational Operators then It is called Relational Expression.

=>The Result of Relational Expression is Either True OR False (Bool Values)

=>The Relational Expression is also called Simple Test Condition.

=>In Python Programming, we have 6 Relational Operators. They are given in the following table

SLNO	SYMBOL	MEANING	EXAMPLE
1.	>	greater than	print(10>20) -----False print(100>20) ----True
2.	<	less than	print(10<20) -----True print(10<5) -----False
3.	==	Equality (Double Equal to)	print(10==20) ----False print(20==20) ----True
4.	!=	ot Equal to	print(10!=20) ----True print(10!=10) ----False
5.	>=	greater than	print(10>=2) -----True or equal to print(10>=20) ---False
6.	<=	less than or equal to	print(10<=20) ---True print(100<=20) ---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".

=>The Result of Logical Expression is Either True OR False (Bool Values)

=>The Logical Expression is also called Compound Test Condition.

=>In Python programming, we have 3 types of Logical Operators. They are given in the following table

=>General Syntax:

```

varname= RelExpr1 Logical Operator RelExpr2
-----
=====

SLNO           SYMBOL           MEANING
=====
1.             or               Physical ORing
2.             and              Physical ANDing
3.             not              -----
=====
1) or Operator (Physical ORing)
=====
Syntax:       varname= RelExpr1 or RelExpr2

```

=>The Functionality of "or" Operator Expressed in the following Truth Table

```

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

```

Truth Table

```

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

```

Examples

```

>>> (10>20) or (20>30)-----False
>>> (100>20) or (20>30)-----True---Short Circuit Evaluation
>>> (100>20) or (100>200) or (300>400)----True---Short Circuit
Evaluation
>>> (100>200) or (10>20) or (40>60) or (500>600)---False
>>> (100>200) or (30>20) or (40>60) or (500>600)---True--Short
Circuit Evaluation

```

Short Circuit Evaluation in the case of "or" Operator

=>If Multiple Relational Expressions are connected with "or" Operator and If Initial Relational Expression Result is True then PVM will not execute / Evaluate Rest of relational expressions and The total result of Logical Expression is Considered as True. This Process of Evaluation is called Short Circuit Evaluation

2) and Operator (Physical ANDing)

Syntax: varname= RelExpr1 and RelExpr2

=>The Functionality of "and" Operator Expressed in the following Truth Table

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

Truth Table

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

Examples

```
>>> (100>20) and (30>20)-----True
>>> (100>20) and (30<20)-----False---Short Circuit
Evaluation
>>> (100<20) and (30<20)-----False---Short Circuit
Evaluation
>>> (100<20) and (30<20) and (10>5)---False---Short Circuit
Evaluation
>>> (100>20) and (30<20) and (10>5)---False---Short Circuit
Evaluation
```

Short Circuit Evaluation in the case of "and" Operator

=>If Multiple Relational Expressions are connected with "and" Operator and If Initial Relational Expression Result is False then PVM will not execute / Evaluate Rest of relational expressions and The total result of Logical Expression is

Considered as False. This Process of Evaluation is called Short Circuit Evaluation

3 not Operator

Syntax: not RelExpr1

=>The Functionality of "not" Operator Expressed in the following Truth Table

RelExpr1	not RelExpr1
True	False
False	True

Examples

```
>>> not False-----True
>>> not True-----False
>>> not (10>20 or 20>30 )-----True
>>> 10<20 or 20>30-----True
>>> not 10<20 or 20>30-----False
>>> not (10>2 and 30>40)-----True
-----
>>> not "KVR"-----False
>>> not "0"-----False
>>> not 0-----True
>>> not -123-----False
>>> not (True=False)-----False
```

```
#Programming Swapping of any two values
a=input("Enter Val of a:")
b=input("Enter Val of b:")
print("*"*50)
print("Original value of a:{}".format(a))
print("Original value of b:{}".format(b))
#swapping Logic
t=a
a=b
b=t
print("*"*50)
print("Swapped value of a:{}".format(a))
print("Swapped value of b:{}".format(b))
print("*"*50)
```

```
#Programming Swapping of any two Integer values only But not
other type of values
a=int(input("Enter Val of a:"))
b=int(input("Enter Val of b:"))
print("*"*50)
print("Original value of a:{}".format(a))
print("Original value of b:{}".format(b))
#swapping Logic---works for Integer data But not For Other
Values
a=a+b
b=a-b
a=a-b
print("*"*50)
print("Swapped value of a:{}".format(a))
print("Swapped value of b:{}".format(b))
print("*"*50)
```

```
#Programming Swapping of any two values
a,b=input("Enter Val of a:"),input("Enter Val of b:")
print("*"*50)
print("Original value of a:{}".format(a))
print("Original value of b:{}".format(b))
#swapping Logic
a,b=b,a # Multi Line assignment
print("*"*50)
print("Swapped value of a:{}".format(a))
print("Swapped value of b:{}".format(b))
print("*"*50)
```

```
#Program for Demonstrating the Functionality of Relation
Operators
#RelationalOpEx1.py
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
print("====")
print("Results of Relational Operators")
print("====")
print("\t\t{} > {} = {}".format(a,b,a>b))
print("\t\t{} < {} = {}".format(a,b,a<b))
print("\t\t{}=={} = {}".format(a,b,a==b))
print("\t\t{} != {} = {}".format(a,b,a!=b))
print("\t\t{} >= {} = {}".format(a,b,a>=b))
print("\t\t{} <= {} = {}".format(a,b,a<=b))
print("====")
```

Special Points about Logical Operators (Comparision Operators)

```
>>> 10<20 and 20<30 and 40>20-----True
>>> 10>20 and 20<30 and 40>20-----False
```

```

-----
>>> False and True-----False
>>> True and False-----False
>>> False and False-----False
>>> True and True-----True
>>> 100 and 200-----200
>>> 100 and 0-----0
>>> -1000 and -2000----- -2000
>>> 0 and 200----- 0
>>> 100-100 and 200-198-----0
>>> 100 and 200 and 300-----300
>>> "" and "Java"----- ''
=====
>>> False or False-----False
>>> False or True-----True
>>> True or False-----True
>>> True or True-----True
>>> 100 or 200-----100
>>> 100 or 0-----100
>>> 0 or 200 or 0-----200
>>> "False" or "True"-----'False'
>>> False or 1-1-----0
>>> True or "False"-----True
>>> False or "False"-----'False'
>>> 100 or 200 and 300-----100
>>> 100 and 200 or 300 and -10 and 300---200
>>> "HYD" and "Bang" or "Eng" and "AUS" and False-----'Bang'
>>> "HYD" and 0 or "Eng" and "AUS" and False-----False
>>> "HYD" and 0 or "Eng" and "AUS" and True-----True
>>> bool(100) and bool(0)-----False
>>> bool(100) or bool(0)-----True
>>> "$" and "#"-----'#'
>>> "$" or "#"-----'$'
=====

```

5. Bitwise Operators (Most Imp)

=>Bitwise Operators are Applicable on Integer Data Only But Not on Floating Point Values bcoz Integer Data Provides Certainty where as Floating point Values does not provide Certainty.

=>The execution Process of Bitwise Operators is that "First Convert Integer Data into Binary Format, Secondly Perform the Operation on Binary Data in the form of Bit by Bit and Lastly the Result displayed in the format of Decimal Number System".

=>Since the Operations are performing on Binary Data in the form of Bit by Bit and Hence they 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. 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 Zeros 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
>>> print(12<<4)-----192
```

1. Bitwise Left Shift Operator (<<)

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

Examples:

16-Bit Register----a=10
>>>a=10-----> 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
>>b=3-----> 16-Bit Register----a=10
:-----> 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
>>c=a<<b-----> Fipped-off 16-Bit Register----a=10
c-----> 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
Result is 80

>>>print(c)-----80

By using Formula

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

No.of Bits

result= Given Data x 2

Examples: print(10<<3)----- 10×2^3
 $10 \times 8=80$

=====

2. Bitwise Right Shift Operator (>>)

=====

Syntax: varname=GivenNumber >> No. of Bits

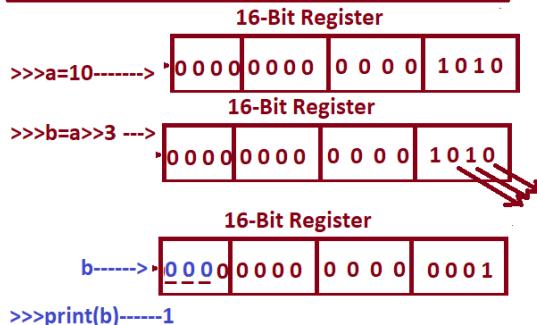
Concept: The Bitwise Right Shift Operator (>>) shifts No. of Bits Towards Right Side By Adding No. of Zeros at Left Side (depends on No. of Bits). So that result value will displayed in the form of decimal number system .

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
```

Bits Bitwise Right Shift Operator (>>):

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



Formula:

varname= Given Data>> No. of Bits

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

Ex: b=10>>3

$$\begin{aligned} b &= \frac{10}{2^3} \\ &= 10 // 8 \rightarrow 1 \end{aligned}$$

print(b)----1

=====

3. Bitwise OR Operator (|)

=====

=>Syntax: resultvar = Var1 | Var2

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

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

Examples

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

```
>>>c=a|b----- 1110
-----
```

```
>>>print(c)----14
>>print(10|15)----15
=====
```

4. Bitwise AND Operator (&)

=>Syntax: resultvar = Var1 & Var2
=>The Functionality of Bitwise AND Operator (&) is Shown in the following table

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

Examples

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

```
>>>c=a&----- 0000
-----
```

```
>>>print(c)----0
-----
```

```
>>print(10&15)----10
=====
```

5. Bitwise Complement Operator (~)

Syntax: varname = ~Value
=>The Complement Operator Internally will Invert the Bits of Given Value.

=>The Invert the Bits of Given Value is Nothing but Converting 0 to 1 and 1 to 0

Syntax: varname = ~Value

The Formula for Bitwise Complement Operator (\sim) = $-(\text{Value}+1)$

Examples

```
>>> a=10
>>> print(~a)----- -11
>>> a=-100
>>> print(~a)----- 99
>>> a=0
>>> print(~a)----- -1
>>> print(~4)----- -5
```

Proof 1:

```
>>>a=10-----Binary of a=10-----> 1010
>>>~a-----Complement of a-----> - (a+1)
                                         -(1010+0001)
                                         - ( 1010 0001)
-----  
-----  
----- (1011)---Result is -11
```

```
>>>print(a)----> -11
```

=====

Original Proof By Formula

```
=>Given a=10
=>Binary of a=1010
=> ~a-----> 0101 (Inverting the bits)---Negative Binary
Rep of Value 11
>>>print(a)---- -11
*****  
*****
```

Original Proof By Computing

```
=>Let us Consider a=11
=>Binary Representation of 11= 1011
=>1s Complement of 11 = 0100
=>2s Complement of 11=1s Complement of 11+1
                                         =0100+1
                                         =0100+0001
                                         OR
                                         =0100
                                         +0001
```

0101----It is the Result of 2s
Complement of 11(Nothing but -11)

Hence $\sim 10 = -11$ (See the proof above)

=====Original Proof By Formula

=====
=>Given a=4
=>Binary of a=0100
=> $\sim a \rightarrow 1011$ (Inverting the bits)---Negative Binary
Rep of Value 5
>>>print(a)---- -5

Original Proof By Computing

======
=>Let us Consider a=5
=>Binary Representation of 5= 0101
=>1s Complement of 5 = 1010
=>2s Complement of 5=1s Complement of 5+1
=1010+1
=1010+0001
OR
=1010
+0001

1011-----It is the Result of
2s Complement of 5 (Nothing but -5)

Hence $\sim 4 = -5$ (See the proof above)

6. Bitwise XOR Operator (^)

======
=>Syntax: resultvar = Var1 ^ Var2
=>The Functionality of Bitwise XOR Operator (^) is Shown in
the following table

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

Examples

```

>>>a=10----- 1010
>>>b=4----- 0100
-----
>>>c=a^b           1110
-----
>>>print(c)-----14
-----
>>print(10^15)----5
=====
```

Special Points about Bitwise Operators (Most Imp)

I) Set Operation--union() with Bitwise OR (|)

```

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

```

>>> s1={10,20,30}
>>> s2={30,40,50}
>>> s3=s1|s2 # Bitwise OR Operator ( | )
>>> print(s3,type(s3))-----{50, 20, 40, 10, 30} <class
'set'>
-----
```

II) Set Operation--intersection() with Bitwise AND (&)

```

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

```

>>> s1={10,20,30}
>>> s2={30,40,50}
>>> s3=s1&s2 # Bitwise AND Operator ( & )
>>> print(s3,type(s3))-----{30} <class 'set'>
-----
```

III) Set Operation--difference() with Arithmetic Subtraction Operator (-)

```

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

```

>>> print(s2)-----{ 40, 50, 30}
>>> s3=s1.difference(s2)
>>> print(s3,type(s3))-----{10,20} <class 'set'>
-----
>>> s1={10,20,30}
>>> s2={30,40,50}
>>> s3=s1-s2 # Substraction Operator ( - )
>>> print(s3,type(s3))-----{10, 20} <class 'set'>
>>> s3=s2-s1 # Substraction Operator ( - )
>>> print(s3,type(s3))-----{40, 50} <class 'set'>
-----
IV) Set Operation--symmetric_difference() with Bitwise XOR( ^ )
-----
>>> s1={10,20,30}
>>> s2={30,40,50}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{40, 50, 30}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3,type(s3))-----{40, 10, 50, 20} <class 'set'>
-----
>>> s1={10,20,30}
>>> s2={30,40,50}
>>> s3=s1^s2 # Bitwise XOR ( ^ )
>>> print(s3,type(s3))-----{40, 10, 50, 20} <class 'set'>
>>> s3=s2^s1 # Bitwise XOR ( ^ )
>>> print(s3,type(s3))-----{40, 10, 50, 20} <class 'set'>
-----
```

Examples:

```

>>> s1={"Apple","Mango","Kiwi"}
>>> s2={"Sberry","Kiwi","banana"}
>>> s3=s1|s2
>>> print(s3,type(s3))-----{'Mango', 'banana',
'Kiwi', 'Sberry', 'Apple'} <class 'set'>
-----
>>> s3=s1&s2
>>> print(s3,type(s3))-----{'Kiwi'} <class 'set'>
>>> s3=s1-s2
>>> print(s3,type(s3))-----{'Mango', 'Apple'} <class 'set'>
>>> s3=s1^s2
>>> print(s3,type(s3))-----{'Sberry', 'Mango', 'Apple',
'banana'} <class 'set'>
=====
```

Python Ternary Operator

=>In Python Programming, We have a Ternary Operator called "if..else " .

```
-----  
=>Syntax: varname = Expr1 if Test Cond else Expr2  
-----
```

Explanation

```
=>Here 'if' and 'else' are the Keywords.  
=>Here Test Condition is either Relational or Logical Expression  
and whose result can be either True OR False  
=>If the Result of Test Condition is True then PVM Evaluates  
Expr1 and whose Result Palced in LHS Varname.  
=>If the Result of Test Condition is False then PVM Evaluates  
Expr2 and whose Result Palced in LHS Varname.  
=>At any point of time PVM Evaluates Either Expr1 or Expr2 and  
whose Result placed in in LHS Varname.
```

```
#Program for finding Biggest and Smallest of Two Numebrs anc  
check for equality
```

```
#BigSmallEx.py
```

```
a=int(input("Enter First Value:")) # 10  
b=int(input("Enter Second Value:")) # 5  
bv=a if a>b else b if b>a else " Both Values are Eqaul"  
print("Big({},{})={}".format(a,b,bv))  
sv=a if a<b else b if b<a else "Both Values are Eqaul"  
print("Small({},{})={}".format(a,b,sv))
```

```
##Program for finding Biggest and Smallest of Two Numebrs anc  
check for equality
```

```
#BigSmallThreeEx.py
```

```
a=int(input("Enter First Value:"))  
b=int(input("Enter Second Value:"))  
c=int(input("Enter Third Value:"))  
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))
```

```
#Program for deciding Whether the given number is even or odd
```

```
#EvenOddEx1.py
```

```
n=int(input("Enter a Number:"))  
res= "Even" if n%2==0 else "Odd"  
print("{} is {}".format(n,res))
```

```
#Program for Deciding weather the given value is Palindrome or  
not
```

```
#PalindromeEx.py
```

```
n=input("Enter a Value:")  
res="PALINDROME" if n==n[::-1] else "NOT PALINDROME"  
print("{} is {}".format(n,res))
```

```
#Program for Deciding wether a given number is +Ve or -ve or  
zero
```

```
#PosNegZero.py
```

```
n=int(input("Enter any number:"))
res="POSSITIVE" if n>0 else "NEGATIVE" if n<0 else "ZERO"
print("{} is {}".format(n,res))
```

String Handling Part-2

=>On String Data, we can perform Indexing, Slicing Operations and with these operations, we can also perform different type of operations by using pre-defined functions present in str object.

Pre-defined Functions in str object

1) `capitalize()`

=>This Function is used for capitalizing the first letter First word of a given Sentence only.

=>Syntax: `strobj.capitalize()`
 (OR)
 `strobj=strobj.capitalize()`

Examples:

```
>>> s="python"
>>> print(s,type(s))-----python <class 'str'>
>>> s.capitalize()-----'Python'
>>> s="python is an oop lang"
>>> print(s,type(s))-----python is an oop lang <class 'str'>
>>> s.capitalize()-----'Python is an oop lang'
```

```
>>> s="python"
>>> print(s,type(s))-----python <class 'str'>
>>> s.capitalize()-----'Python'
>>> print(s,type(s))-----python <class 'str'>
>>> s=s.capitalize()
>>> print(s,type(s))-----Python <class 'str'>
```

2) `title():`

=>This is used for obtaining Title Case of a Given Sentence
(OR) Making all words First

Letters are capital.

Syntax: `s.title()`
 (OR)
 `s=s.title()`

Examples:

```
-----
>>> s="python"
>>> print(s,type(s))-----python <class 'str'>
>>> s.capitalize()-----'Python'
>>> s.title()-----'Python'
-----
>>> s="python is an oop lang"
>>> print(s,type(s))-----python is an oop lang
<class 'str'>
>>> s.capitalize()-----'Python is an oop lang'
>>> s.title()-----'Python Is An Oop Lang'
>>> print(s)-----python is an oop lang
>>> s=s.title()
>>> print(s)-----Python Is An Oop Lang
-----
3) index()
-----
=>This Function obtains Index of the specified Value
=>If the specified value does not exist then we get ValueError
=>Syntax:      strobj.index(Value)
=>Syntax:      indexvalue=strobj.index(value)
```

Examples:

```
-----
>>> s="python"
>>> s.index("p")-----0
>>> s.index("y")-----1
>>> s.index("o")-----4
>>> s.index("n")-----5
>>> s.index("K")-----ValueError: substring not found
```

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  
Alpabets OR Numerics or Alpha-Numerics only otherwise It returns  
False.  
=>Syntax: strobj. isalphanum()  
-----  
=>Examples:  
-----  
>>> s="python310"  
>>> s.isalnum()-----True  
>>> s="python"  
>>> s.isalnum()-----True  
>>> s="310"  
>>> s.isalnum()-----True  
>>> s="$python310"  
>>> s.isalnum()-----False  
>>> s="python 310"  
>>> s.isalnum()-----False  
>>> s="$python3.10"  
>>> s.isalnum()-----False  
>>> s="python3.10"  
>>> s.isalnum()-----False  
-----  
11) isspace()  
-----
```

=>This Function returns True provided str obj contains purely space otherwise it returns False.

=>Syntax: strobj.isspace()

Examples:

```
>>> s=" "
>>> s.isspace()-----True
>>> s=""
>>> s.isspace()-----False
>>> s="python Prog"
>>> s.isspace()-----False
>>> s="Prasana Laxmi"
>>> s.isspace()-----False
>>> s.isalpha()-----False
>>> s.isalpha() or s.isspace()-----False
```

12) split()

=>This Function is used for splitting the given str object data into different words base specified delimiter (- _ # % ^ ^ , ;etc)

=>The default 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'
-----
14) startswith():
-----
=>The startswith() Function returns True if the string starts
with the specified value, otherwise False.
Examples:
-----
>>>s="Python is an oop lang"
>>>s.startswith("Python")-----True
>>>s.startswith("python")-----False
-----
```

```
15) endswith():
-----
=>The endswith() Function returns True if the string ends with
the specified value, otherwise False.
Examples:
-----
>>>s="Python is an oop lang"
>>>s.endswith("Python")-----False
>>>s.endswith("lang")-----True
-----
16) swapcase()
-----
=>Make the lower case letters upper case and the upper case
letters lower case:
Examples:
>>>s="PyThOn"
>>>s.swapcase()-----pYtHoN
-----
MISc Examples
-----
>>> s="python"
>>> s.capitalize()
'Python'
>>> s="python is an oop lang"
>>> s.capitalize()
'Python is an oop lang'
>>> s="python is an oop lang.python is also fun lang"
>>> s.capitalize()
'Python is an oop lang.python is also fun lang'
>>>
>>> s="python"
>>> s.title()
'Python'
>>> s="python is an oop lang"
>>> s.title()
'Python Is An Oop Lang'
>>> s="python is an oop lang.python is also fun lang"
>>> s.title()
'Python Is An Oop Lang.Python Is Also Fun Lang'
>>> s="PYTHON"
>>> s.title()
'Python'
>>> s="PYTHON"
>>> s.capitalize()
'Python'
>>>
>>>
```

```
>>> s="PyThOn"
>>> s.swapcase()
'pYtHoN'
>>> s="12345"
>>> s.swapcase()
'12345'
>>> s="Python3.11"
>>> s.swapcase()
'pYTHON3.11'
>>> s="$%^&* () "
>>> s.swapcase()
'$%^&* () '
>>> s="PYTHON"
>>> s.lower()
'python'
>>> s="PYTHON"
>>> s.swapcase()
'python'
>>> s="PYThon"
>>> s.swapcase()
'pytHON'
>>> s.lower()
'python'
>>>
>>> s="python"
>>> s.lower()
'python'
>>> s.swapcase()
'PYTHON'
>>> s="python"
>>> s.upper()
'PYTHON'
>>> s="PYThon"
>>> s.upper()
'PYTHON'
>>> s="PYThon"
>>> s.lower()
'python'
```

```
>>>
>>> s="PYTHON"
>>> s.isupper()
True
>>> s="python"
>>> s.isupper()
False
>>> s="PYTHon"
>>> s.isupper()
False
>>> s="java"
>>> s.islower()
True
>>> s="JAvA"
>>> s.islower()
False
>>> s.isupper()
False
>>> s="1234"
>>> s.islower()
False
>>> s.isupper()
False
>>>
>>>
>>> s="python"
>>> s.index('p')
0
>>> s.index('o')
4
>>> s.index('k')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s.index('2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s="python"
>>> s.index('thon')
2
>>> s.index('khon')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s="python is an oop lang"
>>> s.index('is')
```

```
7
>>> s.index('o')
4
>>> s.index('an')
10
>>> s.index('10')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> s.index("an")
10
>>> s="Apple"
>>> s.isalpha()
True
>>> s="Apple123"
>>> s.isalpha()
False
>>> s="Ap ple"
>>> s.isalpha()
False
>>> s="123"
>>> s.isalpha()
False
>>> s="Pyth$on"
>>> s.isalpha()
False
>>> s="PYTHON311"
>>> s.isalnum()
True
>>> s="PYTHON"
>>> s.isalnum()
True
>>> s="311"
>>> s.isalnum()
True
>>> s="PYT HON"
>>> s.isalnum()
False
>>> s="PYTHON3.11"
>>> s.isalnum()
False
>>> s="PYTH$on"
>>> s.isalnum()
False
>>> s="123.56"
>>> s.isalnum()
False
```

```
>>> s="123"
>>> s.isnumeric()
True
>>> s="123.45"
>>> s.isnumeric()
False
>>> s="123$23"
>>> s.isnumeric()
False
>>> s="2"
>>> s.isnumeric()
True
>>> s="32"
>>> s.isdigit()
True
>>> s="PYTHON311"
>>> s.isdigit()
False
>>> s=" "
>>> s.isspace()
True
>>> s="123 456"
>>> s.isspace()
False
>>> s=""
>>> s.isspace()
False
>>> s="    "
>>> s.isspace()
True
>>>
>>> s="Apple is in red"
>>> s.split()
['Apple', 'is', 'in', 'red']
>>> x=s.split()
>>> print(x,type(x))
['Apple', 'is', 'in', 'red'] <class 'list'>
>>> len(x)
4
>>> s="08-07-2023"
>>> print(s)
08-07-2023
>>> x=s.split("-")
>>> print(x)
['08', '07', '2023']
>>> print("Day=",x[0])
Day= 08
```

```

>>> print("Month=",x[1])
Month= 07
>>> print("Year=",x[2])
Year= 2023
>>> s="Apple#Mango#kiwi-Banana"
>>> print(s)
Apple#Mango#kiwi-Banana
>>> x=s.split("#")
>>> print(x)
['Apple', 'Mango', 'kiwi-Banana']
>>> y=s.split("-")
>>> print(y)
['Apple#Mango#kiwi', 'Banana']
>>> y[0]
'Apple#Mango#kiwi'
>>> y[0].split("#")
File "<stdin>", line 1
    y[0].split("#")
^
SyntaxError: closing parenthesis ']' does not match opening
parenthesis '('
>>> y[0].split("#")
['Apple', 'Mango', 'kiwi']
>>> y[0:1]=y[0].split("#")
>>> print(y)
['Apple', 'Mango', 'kiwi', 'Banana']
>>>
>>> s="123$456$678$156$"
>>> print(s)
123$456$678$156$
>>> s.split("$")
['123', '456', '678', '156', '']
>>> s="123$456$678$156"
>>> s.split("$")
['123', '456', '678', '156']
>>>
>>>
>>>
>>> lst=["apple","mango","kiwi","guava"]
>>> print(lst,type(lst))
...
['apple', 'mango', 'kiwi', 'guava'] <class 'list'>
>>> k=""
>>> k.join(lst)
'apple mangokiwi guava'
>>> print(k)

```

```

>>> k
 ''
>>> lst=["apple","mango","kiwi","guava"]
>>> print(lst,type(lst))
['apple', 'mango', 'kiwi', 'guava'] <class 'list'>
>>> k=""
>>> k=k.join(lst)
>>> print(k)
apple mangokiwiguava
>>> k
'apple mangokiwiguava'
>>>
>>> lst=["apple","mango","kiwi","guava"]
>>> print(lst,type(lst))
['apple', 'mango', 'kiwi', 'guava'] <class 'list'>
>>> k=" "
>>> k=k.join(lst)
>>> print(k)
apple mango kiwi guava
>>> lst=["Python","is","an","oop","lang"]
>>> k=" "
>>> k=k.join(lst)
>>> print(k)
Python is an oop lang
>>> print(k,type(k))
Python is an oop lang <class 'str'>
>>> k.split()
['Python', 'is', 'an', 'oop', 'lang']
>>> s=" "
>>> s.isnull()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'isnull'
>>>
>>>
>>> s="Python is an oop lang"
>>> print(s)
Python is an oop lang
>>> s.startswith("Python")
True
>>> s.startswith("Pyt")
True
>>> s.startswith("p")
False
>>> s.startswith("p".upper())
True
>>> s.startswith("lang")

```

```
False
>>> s="Python is an oop lang"
>>> print(s)
Python is an oop lang
>>> s.endswith("Python")
False
>>> s.endswith("lang")
True
>>> s.endswith("la")
False
>>> s.endswith("ng")
True
>>> s.endswith("n")
False
>>> s.endswith("g")
True
>>> s.endswith("lang".upper())
False
```

**Flow Control Statements in Python
(OR)
Control Structures in Python---- 8 Days**

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

=>The purpose of Flow Control Statements in Python is that " To perform Certain Operation (X-Operation OR Y-Operation) Only Once OR Perform Certain Operation Repeatedly for finite number of times until Test Condition becomes False."

=>In Python Programming, we have 3 Types of Flow Control Statements in Python. They are

1. Conditional OR Selection OR Branching Statements
2. Looping OR Iterative OR Repeatative Statements
3. Transfer Flow Statements

1. Conditional OR Selection OR Branching Statements

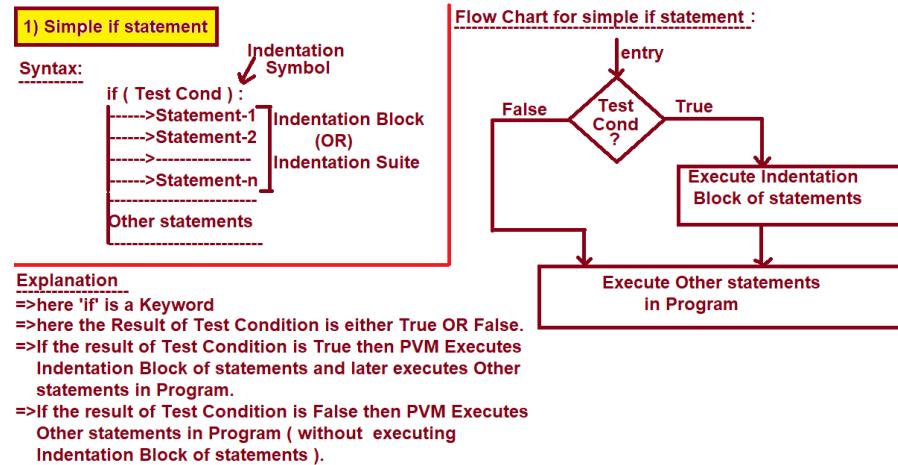
=>The purpose of Conditional OR Selection OR Branching Statements is that " To perform either

X-Operation in the case of True OR Y-Operation 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



```
#Moviee.py  
tkt=input("Do u have a Ticket(yes/no):")  
if(tkt.lower() == "yes"):  
    print("Enter into theater")  
    print("Watch the movie")  
    print("Enjoy")  
print("Goto Home")
```

```
#EvenOddEx1.py  
n=int(input("Enter a Number:")) # n=10  
if(n%2==0):  
    print("{} is EVEN".format(n))  
if(n%2!=0):  
    print("{} is ODD".format(n))  
print("Program execution Completed")
```

```
#PosNegZeroEx1.py  
n=int(input("Enter a Number:"))  
if(n>0):  
    print("{} is +VE".format(n))  
if(n<0):  
    print("{} is -VE".format(n))  
if(n==0):
```

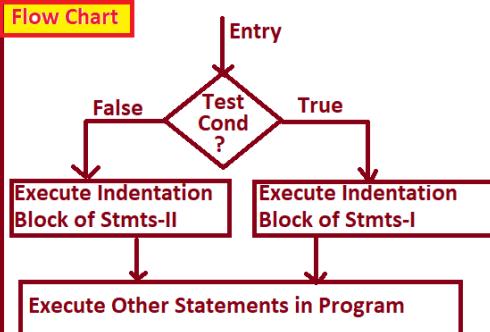
```
print("{} is ZERO".format(n))
```

2. if..else statement

Syntax:

```
if (Test Cond) :  
    Statement-1  
    Statement-2  
    Statement-n  
else:  
    Statement-11  
    Statement-12  
    Statement-n1  
Other Statements  
in Program
```

Flow Chart



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:-**

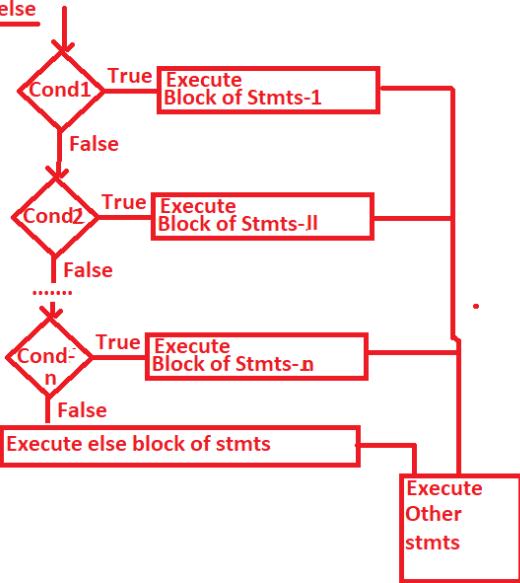
```

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.

flow chart for if..elif..else

```

=====
#DigitsEx1.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==8):
                                    print("{} is
EIGHT".format(d))
                                else:
                                    if(d==9):
                                        print("{} is
NINE".format(d))
                                    else:
                                        if d in [-1,-2,-3,-4,-
5,-6,-7,-8,-9]:
                                            print("{} is -Ve
Digit".format(d))
                                        else:
                                            if d<0:
                                                print("{} is -Ve
Number:".format(d))
                                            else:
                                                print("{} is
NUMBER".format(d))

```

```

#DigitsEx2.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))
    elif d in [-1,-2,-3,-4,-5,-6,-7,-8,-9]:
        print("{} is -Ve Digit".format(d))
    elif d<0:
        print("{} is -Ve Number:".format(d))
    else:
        print("{} is NUMBER".format(d))

```

```

#DigitsEx3.py
try:
    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))
    elif d in [-1,-2,-3,-4,-5,-6,-7,-8,-9]:

```

```

        print("{} is -Ve Digit".format(d))
    elif d<0:
        print("{} is -Ve Number:".format(d))
    else:
        print("{} is NUMBER".format(d))
except ValueError:
    print("Don't enter alnums, strs and symbols")

```

```

#DigitsEx4.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))
elif d in [-1,-2,-3,-4,-5,-6,-7,-8,-9]:
    print("{} is -Ve Digit".format(d))
elif d<0:
    print("{} is -Ve Number:".format(d))
elif (d>9) :
    print("{} is NUMBER".format(d))

```

```

#DigitsEx5.py
dictobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:
"SIX",7:"SEVEN",8:"EIGHT",9:"NINE",-1:"-ONE",-2:"-TWO",-3:"-THREE",
-4:"-FOUR",-5:"-FIVE",-6:"-SIX",-7:"-SEVEN",-8:"-EIGHT",
-9:"-NINE"}
d=int(input("Enter any digit:"))
res=dictobj.get(d) if dictobj.get(d) !=None else "Its a Number"
print("{} is {}".format(d,res))

```

```

#DigitsEx5.py
dictobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:
"SIX",7:"SEVEN",8:"EIGHT",9:"NINE",-1:"-ONE",-2:"-TWO",-3:"-"

```

```

THREE",-4:"-FOUR",-5:"-FIVE",-6:"-SIX",-7:"-SEVEN",-8:"-EIGHT",-  

9:"-NINE"}  

d=int(input("Enter any digit:"))  

print("{} is {}".format(d,dictobj.get(d) if dictobj.get(d) !=None  

else "Its a Number"))



---


#DigitsEx5.py  

dictobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:  

"SIX",7:"SEVEN",8:"EIGHT",9:"NINE",-1:"-ONE",-2:"-TWO",-3:"-  

THREE",-4:"-FOUR",-5:"-FIVE",-6:"-SIX",-7:"-SEVEN",-8:"-EIGHT",-  

9:"-NINE"}  

d=int(input("Enter any digit:"))  

print("{} is {}".format(d,dictobj.get(d) if dictobj.get(d) !=None  

else "Its a Number"))



---


#VowelConsonant.py  

word=input("Enter a Word:").lower() # APPle--->apple  

if 'a' in word or 'e' in word or 'i' in word or 'o' in word or  

'u' in word:  

    print("{} is Vowel Word".format(word))  

else:  

    print("{} is a Consonant Word".format(word))



---


#PosNegZeroEx2.py  

n=int(input("Enter a Number:")) # -10  

if(n>0):  

    print("{} is +VE".format(n))  

else:  

    if(n<0):  

        print("{} is -VE".format(n))  

    else:  

        print("{} Zero".format(n))
=====
```

d) match case statement.

=>Here "match case" is one the new feature in Python 3.10 Version onwards
=>The purpose of match case statement is that "To deal with Pre-defined Test Conditions"

=>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 Statements-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(|).

```

#MatchCaseEx1.py
import sys
print("-"*50)
print("\tArithmetic Operations")
print("-"*50)
print("\t1.Addition")
print("\t2.Substraction")
print("\t3.Multiplication")
print("\t4.Division")
print("\t5.Modulo Division")
print("\t6.Exponentiation")
print("\t7.Exit")
print("-"*50)
ch=int(input("Enter ur Choice:"))
match(ch):
    case 1:
        print("Enter Two values for addition:")

```

```

        a,b=float(input()),float(input())
        print("\tSum({},{})={}".format(a,b,a+b))
    case 2:
        print("Enter Two values for Substraction:")
        a, b = float(input()), float(input())
        print("\tSub({},{})={}".format(a, b, a - b))
    case 3:
        print("Enter Two values for Multiplication:")
        a, b = float(input()), float(input())
        print("\tMul({},{})={}".format(a, b, a * b))
    case 4:
        print("Enter Two values for Division:")
        a, b = float(input()), float(input())
        print("\tNormal Div({},{})={}".format(a, b, a / b))
        print("\tFloor Div({},{})={}".format(a, b, a // b))
    case 5:
        print("Enter Two values for Modulo Div:")
        a, b = float(input()), float(input())
        print("\tMod({},{})={}".format(a, b, a % b))
    case 6:
        a, b = float(input("Enter Base:")), float(input("Enter
                                                Power:"))
        print("\tpow({},{})={}".format(a, b, a ** b))
    case 7:
        print("Thx for using this Program")
        sys.exit()
    case _:
        print("Ur Selection of Operation is wrong-try again")
print("Program execution completed")

```

```

#MatchCaseEx2.py
wkd=input("Enter Week Name:")
match(wkd):
    case "MONDAY":
        print("{} is Working Day".format(wkd))
    case "TUESDAY":
        print("{} is Working Day".format(wkd))
    case "WEDNESDAY":
        print("{} is Working Day".format(wkd))
    case "THURSDAY":
        print("{} is Working Day".format(wkd))
    case "FRIDAY":
        print("{} is Working Day".format(wkd))
    case "SATURDAY":
        print("{} is Week-End--Plans Under Ground
                plans".format(wkd))
    case "SUNDAY":
        print("{} is HOLI Day".format(wkd))

```

```

    case _:
        print("{} is not a week day".format(wkd))
=====
#MatchCaseEx3.py
wkd=input("Enter Week Name:")
match(wkd.upper()):
    case "MONDAY"|"TUESDAY"|"WEDNESDAY"|"THURSDAY"|"FRIDAY":
        print("{} is Working Day".format(wkd))
        case "SATURDAY":
            print("{} is Week-End--Plans Under Ground".format(wkd))
    case "SUNDAY":
        print("{} is HOLI Day".format(wkd))
    case _:
        print("{} is not a week day".format(wkd))
=====
#MatchCaseEx4.py
wkd=input("Enter Week Name:")
match(wkd.upper()):
    case "MONDAY"|"TUESDAY"|"WEDNESDAY"|"THURSDAY"|"FRIDAY"|"MON"|"TUE"|"WED"|"THU"|"FRI":
        print("{} is Working Day".format(wkd))
    case "SATURDAY"|"SAT":
        print("{} is Week-End--Plans Under Ground".format(wkd))
    case "SUNDAY"|"SUN":
        print("{} is HOLI Day".format(wkd))
    case _:
        print("{} is not a week day".format(wkd))
=====
```

Looping or Iterative Or Repetative Statements

=>The purpose of Looping or Iterative Or Repetative Statements is that "To Perform Certain Operation Repeatedly for finite number of times until Test Condition becomes False".

=>In Python Programming, we have 2 Types of Loops. They are

1. while Loop OR while..else Loop
2. for loop OR for...else Loop

=>At the time of Dealing with Looping Statements, we must Ensure that, there must exist 3 Ponits. They are

1. Initlization Part (Form where to start)
2. Conditional Part (Where to stop)
3. Updation Part (Incrementation / Decrementation) -- (How much to increment / Decrement)

Syntax:

```
while( Test Cond ) :
    --->Statement-1
    --->Statement-2
    --->Statement-n
    Other statements in Prog
```

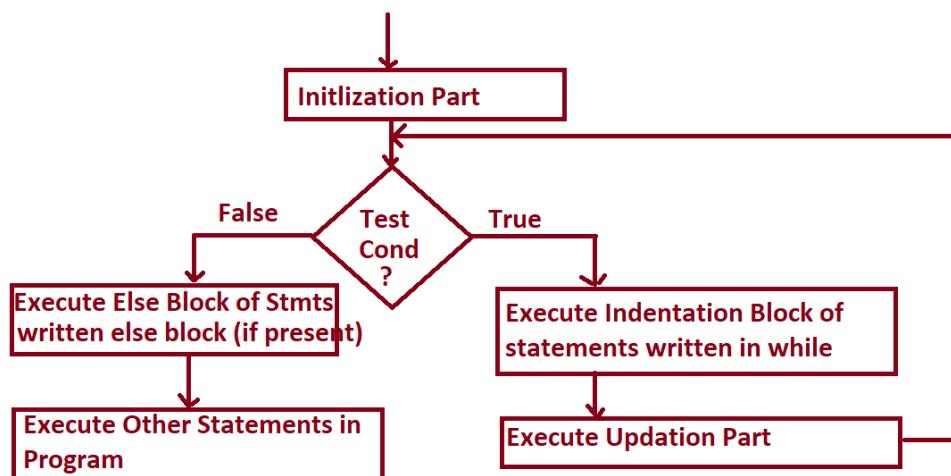
Syntax:

```
while( Test Cond ) :
    --->Statement-1
    --->Statement-2
    --->Statement-n
else:
    --->Else Block of Stmts
Other Stmtns 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.

Flow Chart for while loop OR while ..else Loop



#Program for generating 1 to n numbers where n is +ve
#WhileNumGenEx1.py

```
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
```

```

i=1 # Initialization part
while(i<=n): # Test Cond
    print("\t{}".format(i))
    i=i+1 # Incrementation
else:
    print("i am from while ..else statement")
    print("i am from while..else other statements")
print("Other statements in if..else")


---


#Program for generating n to 1 numbers where n is +ve
#WhileNumGenEx2.py
n=int(input("Enter How Many Numbers u want to generate:"))#n=4
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    print("-" * 40)
    print("Number of Values within:{}".format(n))
    print("-"*40)
    while(n>=1):
        print("\t{}".format(n))
        n=n-1
    else:
        print("-" * 40)


---


#Program for generating even numbers within n to 1
#WhileEvenNumberGenEx1.py
n=int(input("Enter How Even Numbers u want to generate with in
            given range:"))
if(n<=0): # if n is -ve or 0
    print("{} is Invalid Input".format(n))
else:
    print("=". * 50)
    print("Even Numbers within :{}".format(n))
    print("=". * 50)
    if(n%2!=0): # here n is +Ve Odd
        n=n-1
    while(n>=2):
        print("\t{}".format(n))
        n=n-2
    else:
        print("=". * 50)


---


#Program for generating even numbers within n to 1
#WhileEvenNumberGenEx2.py
n=int(input("Enter How Even Numbers u want to generate with in
given range:"))
if(n<=0): # if n is -ve or 0
    print("{} is Invalid Input".format(n))
else:

```

```

print("=" * 50)
print("Even Numbers within :{}".format(n))
print("=" * 50)
while(n>=2):
    if(n%2==0):
        print("\t{}".format(n))
    n=n-1
else:
    print("=" * 50)

```

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

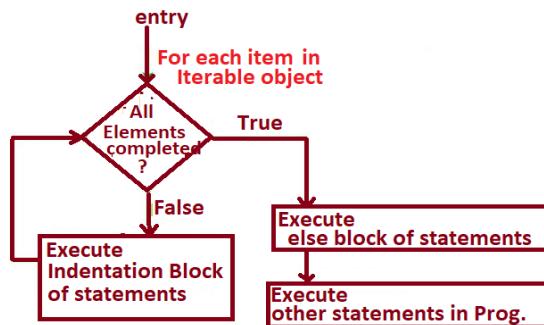
=>Here Iterable_object can be

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.

Flow Chart of for loop



```
#ForLoopEx1.py
lst=[10,"Rossum",45.67,True]
print("*"*50)
print("By using While Loop")
print("*"*50)
i=0
while(i<len(lst)):
    print("\t{}".format(lst[i]))
    i=i+1
else:
    print("*"*50)
print("By using for Loop")
print("*"*50)
for val in lst:
    print("\t{}".format(val))
else:
    print("*" * 50)
```

```
#ForLoopEx2.py
lst=[10,"Rossum",45.67,True]
print("*"*50)
print("By using While Looppin Backword Direction")
print("*"*50)
i=-1
```

```

while(i>=(-len(lst))):  

    print(lst[i])  

    i=i-1  

else:  

    print("=*50)  

print("By using for Loop Backword Direction--slicing")  

print("=*50)  

for val in lst[::-1]:  

    print(val)  

print("=*50)  

print("By using for Loop Backword Direction--logic")  

print("=*50)  

for i in range(len(lst)-1,-1,-1):  

    print("\t{}".format(lst[i]))  

print("=*50)  

print("=*50)  

print("By using for Loop Backword Direction--logic")  

print("=*50)  

for i in range(-len(lst),0,1):  

    print("\t{}".format(lst[i]))  

print("=*50)

```

```

#Program for Calculating factorial of a Number  

#FactorialEx1.py  

n=int(input("Enter a number for cal Factorial:"))  

if(n<0):  

    print("{} is Invalid Input".format(n))  

else:  

    fact=1  

    i=1  

    while(i<=n):  

        fact=fact*i  

        i=i+1  

    else:  

        print("Factorial({})={}".format(n,fact))

```

```

#Program for Calculating factorial of a Number  

#FactorialEx2.py  

n=int(input("Enter a number for cal Factorial:"))  

if(n<0):  

    print("{} is Invalid Input".format(n))  

else:  

    n1=n # We are preserving the value of n in n1  

    fact=1  

    while(n>=1):  

        fact=fact*n  

        n=n-1  

    else:

```

```

    print("Factorial({ })={ }".format(n1,fact))


---


#Program for generating Mul Tables for given +ve number
#MulTableEx1.py
n=int(input("Enter a Number for generating Mul Table:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("=*50)
    print("Mul Table for:{}".format(n))
    print("=*50)
    i=1
    while(i<11):
        print("\t{} x {}={}.".format(n,i,n*i))
        i=i+1
    else:
        print("=* * 50)


---


#Program for generating Mul Tables for given +ve number
#MulTableEx2.py
n=int(input("Enter a Number for generating Mul Table:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("=*50)
    print("Mul Table for:{}".format(n))
    print("=*50)
    for i in range(1,11):
        print("\t{} x {}={}.".format(n,i,n*i))
    else:
        print("=* * 50)


---


#Program for Multiplying first N-Natural Numbers
#NatNumsProductEx1.py
n=int(input("Enter How Many Nat Nums Product u want:"))
if(n<=0):
    print("{} is invalid Input: ".format(n))
else:
    print("-*50)
    print("Product of First {} Nat Nums ".format(n))
    print("- * 50)
    s=1
    i=1
    while(i<=n):
        print("\t{}".format(i))
        s=s*i # accumutaing the sum of n-nat nums
        i=i+1
    else:
        print("-*50)

```

```

        print("\tSum={ }".format(s))
        print("-" * 50)


---


#Program for adding first N-Natural Numbers
#NatNumsSumEx1.py
n=int(input("Enter How Many Nat Num sum u want:"))
if(n<=0):
    print("{} is invalid Input: ".format(n))
else:
    print("-" * 50)
    print("Sum of First {} Nat Num ".format(n))
    print("-" * 50)
    s=0
    i=1
    while(i<=n):
        print("\t{}".format(i))
        s=s+i # accumutaing the sum of n-nat num
        i=i+1
    else:
        print("-" * 50)
        print("\tSum={ }".format(s))
        print("-" * 50)


---


#DigitsSumEx1.py
n=int(input("Enter a Number:")) #n=3456
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    n1=n
    s=0
    while(n>0):
        d=n%10
        s=s+d
        n=n//10
    else:
        print("Sum of digits({})={}".format(n1,s))


---


#DigitsSumEx1.py
n=int(input("Enter a Number:")) #n=3456
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    n1=str(n) # n1="3456"
    s=0
    for v in n1:
        s=s+int(v)
    else:
        print("Sum of digits({})={}".format(n1, s))


---


#AmstrongNumEx1.py
```

```
n=int(input("Enter a Number:"))
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    n1=str(n)
    s=0
    for v in n1:
        s= s + int(v)**3
    else:
        if(s==int(n1)):
            print("{} is Amstrong Number:".format(n1))
        else:
            print("{} is not Amstrong Number:".format(n1))
#NOTE:0, 1, 153, 370,371,407 are called amstrong number
```

```
#SortNamesEx1.py
#SortNumbersEx1.py
n=int(input("Enter How Many Names u want Sort:"))
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    lst=list()
    for i in range(1,n+1):
        val=input("Enter {} Value:".format(i))
        lst.append(val)
    else:
        print("-" * 50)
        print("Given Names:{} ".format(lst))
        print("-" * 50)
        #Sort the given Data-Asc Order
        lst.sort()
        print("Asc Order of Names")
        print("\t{}".format(lst))
        # Sort the given Data-DESC Order
        lst.sort(reverse=True)
        print("DESC Order of Names")
        print("\t{}".format(lst))
```

```
#SortNumbersEx1.py
n=int(input("Enter How Many numbers u want Sort:"))
if(n<=0):
    print("{} is Invalid Input".format(n))
else:
    lst=list()
    for i in range(1,n+1):
        val=float(input("Enter {} Value:".format(i)))
        lst.append(val)
    else:
```

```

print("-" * 50)
print("Given Values:{}".format(lst))
print("-" * 50)
#Sort the given Data-Asc Order
lst.sort()
print("Asc Order of Element")
print("\t{}".format(lst))
# Sort the given Data-DESC Order
lst.sort(reverse=True)
print("DESC Order of Element")
print("\t{}".format(lst))


---


#VowelsConsDigsSS.py
text=input("Enter a Line of Text:")
print("-" * 50)
print("Given Text:{}".format(text)) # text=Apple7 is red$
print("-" * 50)
vlist=[]
conslist=list()
diglist=list()
syblist=[]
for ch in text:
    if(ch.isalpha() and ch.lower() in ['a','e','i','o','u']):
        vlist.append(ch)
    elif(ch.isalpha() and ch.lower() not in
['a','e','i','o','u']):
        conslist.append(ch)
    elif(ch.isdigit()):
        diglist.append(ch)
    else:
        syblist.append(ch)
else:
    print("=*50)
    print("Vowels List={}, Number of
Vowels={}".format(vlist, len(vlist)))
    print("Consonants List={}, Number of
Consonants={}".format(conslist, len(conslist)))
    print("Digits List={}, Number of Digits={}".format(diglist,
        len(diglist)))
    print("Special Symbols List={}, Number of Special
        Symbols={}".format(syblist, len(syblist)))
    l1=len(vlist)+len(conslist)+len(diglist)+len(syblist)
    print("Length Given String: {}={}{}".format(text, l1))
    print("=* 50)


---


=====
```

Transfer Flow Statements in Python

=>The purpose of Transfer Flow Statements in Python is that " To Transfer the flow of PVM one part of the program to another part of the program".

=>In Python, we have 3 Types of Transfer Flow Statements. They are

1. break
2. continue
3. pass
4. return

1. break statement

=>break is a key word

=>The purpose of break statement is that "To terminate the execution of loop logically when certain condition is satisfied and PVM control comes out of corresponding loop and executes other statements in the program".

=>when break statement takes place inside for loop or while loop then PVM will not execute corresponding else block(bcoz loop is not becoming False) but it executes other statements in the program

=>Syntax1:

```
-----  
for varname  in Iterable_object:  
-----  
    if (test cond):  
        break  
-----  
-----
```

=>Syntax2:

```
-----  
while(Test Cond-1):  
-----  
    if (test cond-2):  
        break  
-----  
-----
```

```
#BreakStmtEX1.py--By using for loop  
s="PYTHON"  
print("-"*50)  
for ch in s:  
    print("\t{}".format(ch))  
print("-"*50)  
#My Requirement today is : want to display only PYT without using  
Indexing and slicing  
for ch in s:  
    if(ch=="H"):
```

```

        break
    print("\t{}".format(ch))
else:
    print("i am from for loop--else statement")
print("Other statements in Program")


---


#BreakStmtEX1.py---while loop
s="PYTHON"
i=0
print("-"*50)
while(i<len(s)):
    print("\t{}".format(s[i]))
    i+=1 # here += is called Short hand + Operator
print("-"*50)
#My Requirement today is : want to display only PYT without using
Indexing and slicing
i=0
print("-"*50)
while(i<len(s)):
    if(s[i]=="H"):
        break
    print("\t{}".format(s[i]))
    i+=1
print("-"*50)


---


#program for Deciding whether the given number is prime or not
#BreakStmtEx3.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is Invalid Input".format(n))
elif(n==2):
    print("{} is Prime".format(n))
else:
    for i in range(2,n):
        if(n%i==0):
            break
    if(i+1==n):
        print("{} is Prime".format(n))
    else:
        print("{} is Not Prime".format(n))


---


#program for Deciding whether the given number is prime or not
#BreakStmtEx4.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is Invalid Input".format(n))
else:
    res="PRIME"
    for i in range(2,n):

```

```

        if(n%i==0):
            res="NOT PRIME"
            break
        if(res=="PRIME"):
            print("{} is {}".format(n,res))
        else:
            print("{} is {}".format(n, res))


---


#program for Deciding whether the given number is prime or not
#BreakStmtEx5.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is Invalid Input".format(n))
else:
    res=False
    for i in range(2,n):
        if(n%i==0):
            res=True
            break
    if(res):
        print("{} is NOT PRIME".format(n))
    else:
        print("{} is PRIME".format(n))


---


-----
```

continue statement

=>continue is a keyword
=>continue statement is used for making the PVM to go to the top of the loop without executing the following statements which are written after continue statement for that current Iteration only.
=>continue statement to be used always inside of loops.
=>when we use continue statement inside of loop then else part of corresponding loop also executes provided loop condition becomes false.

=>Syntax:-

```

                for varname    in Iterable-object:
-----  

                    if ( Test Cond):
                        continue
                        statement-1 # written after continue
statement
                        statement-2
                        statement-n
-----  

-----
```

=>Syntax:-

```
-----  
        while (Test Cond):  
-----  
            if ( Test Cond):  
                continue  
            statement-1 # written after continue  
            stateemnt  
            statement-2  
            statement-n  
-----  
-----
```

#WithoutContinueStmt.py

```
s="PYTHON"  
print("-"*50)  
for ch in s:  
    print("\t{}".format(ch))  
print("-"*50)  
#I want to display only PYHN  
for ch in s:  
    if(ch=="T") or (ch=="O"):pass  
    else:  
        print("\t{}".format(ch))  
else:  
    print("i am from for loop else part")  
print("Other statements in Program")
```

#ContinueStmtEx1.py

```
s="PYTHON"  
print("-"*50)  
for ch in s:  
    print("\t{}".format(ch))  
print("-"*50)  
#I want to display only PYHON  
for ch in s:  
    if(ch=="T"):  
        continue  
    print("\t{}".format(ch))  
else:  
    print("i am from for loop else part")  
print("Other statements in Program")
```

#ContinueStmtEx2.py

```
s="PYTHON"  
print("-"*50)  
for ch in s:
```

```

        print("\t{}".format(ch))
print("-"*50)
#I want to display only PYHN
for ch in s:
    if(ch=="T") or (ch=="O"):
        continue
    print("\t{}".format(ch))
else:
    print("i am from for loop else part")
print("Other statements in Program")
#ContinueStmtEx3.py
s="PYTHON"
print("-"*50)
i=0
while(i<len(s)):
    print("\t{}".format(s[i]))
    i+=1
print("-"*50)
#I want to display only PYHON
i=0
while(i<len(s)):
    if(s[i]=="T") or (s[i]=="O") :
        i = i + 1
        continue
    print("\t{}".format(s[i]))
    i=i+1
else:
    print("i am from for loop else part")
print("Other statements in Program")

```

===== Inner OR Nested Loops =====

=>The Process of Defining One Loop inside of another Loop is called Inner OR Nested Loop.

=>The Execution Process of inner OR Nested Loop is that " For Every Value of Outer Loop, Inner Loop executes Repeatedly for finite Number of times until Test Condition becomes False."

=>We can Inner OR Nested Loop in 4 Combinations. They are

Syntax1: For Loop in For Loop

```

        for Varname1  in Iterable-Object1:      # Outer Loop
        -----
        -----
        for Varname2 in Iterable-Object2:  #Inner Loop
        -----
```

```
-----  
    else:  
-----  
    else:  
-----  
-----  
Other Statements in Program  
-----  
-----  
Syntax2: While Loop in While Loop  
-----  
-----  
    while(Test Cond1): # Outer Loop  
-----  
-----  
        while(Test Cond2): # Inner Loop  
-----  
-----  
    else:  
-----  
-----  
    else:  
-----  
-----  
-----  
Other statements in Program  
-----  
-----  
-----  
Syntax3: while loop in for loop  
-----  
-----  
    for Varname1 in Iterable-Object1: # Outer Loop  
-----  
-----  
        while(Test Cond2): # Inner Loop  
-----  
-----  
    else:  
-----  
-----  
    else:  
-----  
-----  
-----  
Other Statements in Program  
-----
```

```
-----  
Syntax4:           for loop in while loop  
-----  
                  while(Test Cond1): # Outer Loop  
-----  
-----  
                  for Varname2 in Iterable-Object2: #  
Inner Loop  
-----
```

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

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

```
-----  
                  Other statements in Program  
-----
```

```
#InnerLoopEx1.py---for loop in for loop  
for i in range(1,6): # Outer Loop  
    print("Outer Loop i={}".format(i))  
    print("-----")  
    for j in range(1,4): # Inner Loop  
        print("\tInner Loop j={}".format(j))  
    else:  
        print("-----")  
        print("Out of Inner Loop")  
else:  
    print("Out of Outer Loop")
```

```
#InnerLoopEx2.py--while loop in while Loop  
i=1  
while(i<=5): # Outer Loop  
    print("Outer Loop i={}".format(i))  
    print("-----")  
    j=1  
    while(j<=3): # Inner Loop  
        print("\tInner Loop j={}".format(j))  
        j+=1  
    else:  
        i+=1  
    print("Out of Inner Loop")  
    print("-----")  
else:  
    print("Out of Outer Loop")
```

```
#InnerLoopEx3.py--while loop in for loop
```

```

for i in range(1, 6): # Outer Loop
    print("Outer Loop i={}".format(i))
    print("-----")
    j = 1
    while (j <= 3): # Inner Loop
        print("\tInner Loop j={}".format(j))
        j += 1
    else:
        print("Out of Inner Loop")
        print("-----")
else:
    print("Out of Outer Loop")


---


#InnerLoopEx4.py--for loop in while loop
i=5
while(i>=1): # Outer Loop
    print("Outer Loop i={}".format(i))
    print("-----")
    for j in range(3,0,-1): # Inner Loop
        print("\tInner Loop j={}".format(j))
    else:
        i-=1
        print("-----")
        print("Out of Inner Loop")
else:
    print("Out of Outer Loop")


---


#Program for Listing the Prime Numbers within n
#InnerLoopEx5.py
n=int(input("Enter the Prime Numbers Range:"))
if(n<=1):
    print("{} is Invalid:".format(n))
else:
    print("-----")
    print("List of Primes within:{}".format(n))
    print("-----")
    for num in range(2,n+1): # Outer Supply Number
        res="PRIME"
        for i in range(2,num):#inner Loop decides prime or not
            if(num%i==0):
                res="NOT PRIME"
                break
        if(res=="PRIME"):
            print("\t{}".format(num))
    print("-----")


---


#InnerLoopEx6.py
n=int(input("Enter the How Many Mul Tables u want:"))
if(n<=0):

```

```

        print("{} is Invalid:".format(n))
else:
    for num in range(1,n+1): # Outer For Loop-Supply Val
        print("-----")
        print("Mul Table for :{}".format(num))
        print("-----")
        for j in range(1,11):#Inner loop--generates mul table
            for outer loop supplied Value
                print("\t{} x {}={}".format(num,j,num*j))
        else:
            print("-----")


---


#Program generating Random Mul Tables
#InnerLoopEx7.py
n=int(input("Enter the How Many Mul Tables u want:"))
if(n<=0):
    print("{} is Invalid:".format(n))
else:
    lst=[]
    for i in range(1,n+1):
        val=int(input("Enter {} Value:".format(i)))
        lst.append(val)
    else:
        print("====")
        print("Given List:{} # lst=[16,19,0,-4,25]")
        print("====")
        for num in lst: # Outer for loop supply value from lst
            if(num<=0):
                continue
            print("-----")
            print("Mul table for {}".format(num))
            print("-----")
            for i in range(1,11): # Inner Loop will generate Mul
                table
                print("\t{} x {} = {}".format(num,i,num*i))
            else:
                print("-----")


---


#Program generating Random Mul Tables
#InnerLoopEx8.py
n=int(input("Enter the How Many Mul Tables u want:"))
if(n<=0):
    print("{} is Invalid:".format(n))
else:
    lst=[]
    for i in range(1,n+1):
        val=int(input("Enter {} Value:".format(i)))
        lst.append(val)

```

```

else:
    print("=====")
    print("Given List:{}".format(lst)) # lst=[16,19,0,-4,25]
    print("=====")
    for num in lst: # Outer for loop supply value from lst
        if(num<=0):pass
        else:
            print("-----")
            print("Mul table for {}".format(num))
            print("-----")
            for i in range(1,11): # Inner Loop will generate
                # Mul table
                print("\t{} x {} = {}".format(num,i,num*i))
            else:
                print("-----")


---


#InnerLoopEx9.py
n=int(input("Enter the How Many Mul Tables u want:"))
if(n<=0):
    print("{} is Invalid:".format(n))
else:
    lst=[]
    for i in range(1,n+1):
        val=input("Enter {} Value:".format(i))
        lst.append(val)
    else:
        print("=====")
        print("Given List:{}".format(lst)) # lst=[KVR,19,0,-4,25]
        print("=====")
        for val in lst: # Outer for loop supply value from lst
            try:
                num=int(val)
                if(num<=0):pass
                else:
                    print("-----")
                    print("Mul table for {}".format(num))
                    print("-----")
                    for i in range(1,11): # Inner Loop will
                        # generate Mul table
                    print("\t{} x {} ="
                          "{}".format(num,i,num*i))
                else:
                    print("-----")
            except ValueError:
                print("{} Is alnums,str, symbols".format(val))
                print("-----")


---


#PerfectNumberEx1.py

```

```

n=int(input("Enter a Number:"))
if(n<=0):
    print("{} is invalid Number".format(n))
else:
    s=0
    for i in range(1, (n//2)+1 ):
        if(n%i==0):
            print("\t{}".format(i))
            s+=i
    else:
        if(n==s):
            print("{} is Perfect Number:".format(n))
        else:
            print("{} is Not Perfect Number:".format(n))


---


#StudentMarksReport1.py
#Accept Student Number and Validate
while(True):
    sno=int(input("Enter Student Number(100-200):"))
    if(sno>=100) and (sno<=200):
        break
    print("\t{} is Invalid Student Number:".format(sno))
#Accept Student Name
sname=input("Enter Student Name:")
#Accept Marks in C Lang
while(True):
    cm=int(input("Enter Marks in C Lang:"))
    if(cm>=0) and (cm<=100):
        break
    print("\t{} is Invalid Marks in C -Lang:".format(cm))
#Accept Marks in CPP Lang
while(True):
    cppm=int(input("Enter Marks in C++ Lang:"))
    if(0<=cppm<=100):
        break
    print("\t{} is Invalid Marks in C---Lang:".format(cppm))
#Accept Marks in PYTHON Lang
while(True):
    pym=int(input("Enter Marks in PYTHON Lang:"))
    if(0<=pym<=100):
        break
    print("\t{} is Invalid Marks in PYTHON--Lang:".format(pym))
#Calculate totmarks and percentage
totmarks=cm+cppm+pym
percent=(totmarks/300)*100
#Decide Grade
if(cm<40) or (cppm<40) or (pym<40):
    grade="FAIL"

```

```

else:
    if(250<=totmarks<=300):
        grade="DISTINCTION"
    elif(200<=totmarks<=249):
        grade="FIRST"
    elif(150<=totmarks<=199):
        grade="SECOND"
    elif(120<=totmarks<=149):
        grade="THIRD"
#Display Marks Report
print("-"*50)
print("\tStudent Marks Report:")
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("-"*50)
print("\tSTUDENT TOTAL MARKS:{}".format(totmarks))
print("\tSTUDENT PERCENTAGE OF MARKS:{}".format(percent))
print("\tSTUDENT GRADE:{}".format(grade))
print("-"*50)


---


#StudentMarksReport2.py
#Accept Student Number and Validate
while(True):
    while(True):
        sno=int(input("Enter Student Number(100-200):"))
        if(sno>=100) and (sno<=200):
            break
        print("\t{} is Invalid Student Number:".format(sno))
    #Accept Student Name
    sname=input("Enter Student Name:")
    #Accept Marks in C Lang
    while(True):
        cm=int(input("Enter Marks in C Lang:"))
        if(cm>=0) and (cm<=100):
            break
        print("\t{} is Invalid Marks in C -Lang:".format(cm))
    #Accept Marks in CPP Lang
    while(True):
        ppm=int(input("Enter Marks in C++ Lang:"))
        if(0<=ppm<=100):
            break
        print("\t{} is Invalid Marks in C++-Lang:".format(ppm))
    #Accept Marks in PYTHON Lang
    while(True):

```

```

pym=int(input("Enter Marks in PYTHON Lang:"))
if(0<=pym<=100):
    break
print("\t{} is Invalid Marks in PYTHON-
Lang:".format(pym))
#Calculate totmarks and percentage
totmarks=cm+cppm+pym
percent=(totmarks/300)*100
#Decide Grade
if(cm<40) or (cppm<40) or (pym<40):
    grade="FAIL"
else:
    if(250<=totmarks<=300):
        grade="DISTINCTION"
    elif(200<=totmarks<=249):
        grade="FIRST"
    elif(150<=totmarks<=199):
        grade="SECOND"
    elif(120<=totmarks<=149):
        grade="THIRD"
#Display Marks Report
print("-"*50)
print("\tStudent Marks Report:")
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("-"*50)
print("\tSTUDENT TOTAL MARKS:{}".format(totmarks))
print("\tSTUDENT PERCENTAGE OF MARKS:{}".format(percent))
print("\tSTUDENT GRADE:{}".format(grade))
print("-"*50)
ch=input("\nDo u want to Generate another Student Marks
Report(yes/no):")
if(ch.lower() == "no"):
    print("Tq for this Program")
    break
elif (ch.lower() != "yes" ):
    print("Plz Learn Typing")
    break

```

```

#VoterEx1.py
age=int(input("Enter Age of Citizen:"))
if(age>=18):
    print("{} Years Citizen is Eligible to Vote:".format(age))
else:

```

```
    print("{} Years Citizen is not Eligible to  
Vote:".format(age))  
##VoterEx2.py  
while(True):  
    age=int(input("Enter the age of Citizen:"))  
    if(age>=18):  
        break  
    print("\t{} Years Citizen is Not Eligible to  
        Vote".format(age))  
print("{} Years Citizen is Eligible to Vote".format(age))
```

Functions in Python---8 Days
Functions (8) -----> Modules (2) -----> Packages (1)

Functions in Python

=>The purpose of Functions is that " To Perform Certain Operation /Task and Provides Code Re-Usability".

=>The Advantages of Functions in any languages 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 of the Code is Minimized
-

Definitions of Function

=>Sub Program of Main Program is Called Function.
(OR).

=>A Part of main program is Called Function.

Parts of Functions

=>At the time Developing functions in real time, we must ensure that, there must exist 2 Parts. they are

1. Function Definition
2. Function Calls

=>Every Function Definition Exists Only Once

=>Every Function call must contains a Function Defintion Otherwise we get NameError.

=>Function Definition will execute when we call by using function calls OR Without calling the Function by using Function Calls PVM will not execute Function Definition.

Phases in Functions

=>At the time Defining the functions, the Programmer must ensure that there must exist the following Phases.

1. Every Function Must take INPUT
2. Every Function Must PROCESS the Input
3. Every Function Must give OUTPUT or RESULT

Syntax for Defining a Function in Python

```
def functionname(list of formal Params if any): <---Function Heading  
    """doc string """  
    statement-1  
    statement-2  
    -----  
    statement-n <---Function Body  
Note: Function def= Function Heading +Function Body
```

Explanation:-

1. here 'def' is a keyword , which is used for defining Programmer-defined Functions.
2. "functionname" represents a valid variable name and treated as function name and every function name is an object of type `<class, 'function'>`
3. "list of formal params" represents list of variable names used in function heading and they are used for storing or holding the input(s) coming from function call(s).
4. "'''doc string''' " represents documentation string and it used for giving or writing the description about functionality of function.
5. The statement-1,statement-2....statement-n indentation block of statements and it is process the input or logic for problem solving and it is known Business Logic.
6. In the Function Body, we use some special variables and they are called Local Variables and whose purpose is to store the temporary results.
7. The values of Formal Params and Local Variables can be accessed only inside 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)

Number of Approaches to Define Functions

=>To Solve any problem / task / Operation by using Functions, We have 4 Approaches. They are

Approach-1

=>INPUT : From Function Call
=>PROCESSING : Inside of Function Body
=> RESULT / OUTPUT : to Function Call

Examples

```
def addop(a,b): # here a and b are called Formal Parameters  
    c=a+b # here c is called Local Variable  
    return c  
#main program  
x=float(input("Enter First Value:"))  
y=float(input("Enter Second Value:"))  
z=addop(x,y) # Function call  
print("sum({},{})={}".format(x,y,z))
```

Approach-2

```
=>INPUT: Inside of Function Body  
=>PROCESS: Inside of Function Body  
=>OUTPUT : Inside of Function Body  
  
Examples  
  
def addop():  
    #Taking INPUT  
    a=float(input("Enter First Value:"))  
    b=float(input("Enter Second Value:"))  
    #Do the PROCESS  
    c=a+b  
    #Display the RESULT / OUTPUT  
    print("sum({},{})={}".format(a,b,c))  
#main program  
addop() # Function call
```

Approach-3

```
=>INPUT: From Function Call  
=>PROCESS: Inside of Function Body  
=>OUTPUT : Inside of Function Body  
  
Examples
```

```
def addop(a,b):  
    #Doing the PROCESS  
    c=a+b  
    #Display the RESULT  
    print("sum({},{})={}".format(a,b,c))  
#main program  
a=float(input("Enter First Value:"))  
b=float(input("Enter Second Value:"))  
addop(a,b) # Function call
```

Approach-4

```
=>INPUT: Inside of Function Body  
=>PROCESS: Inside of Function Body  
=>OUTPUT : to Function Call  
  
Examples
```

```
def addop():  
    #Taking INPUT in Function Body
```

```

a = float(input("Enter First Value:"))
b = float(input("Enter Second Value:"))
#doing PROCESS
c=a+b
#give result back to Function Call
return a,b,c# a return kwd can return one or more values


---


#main program
x,y,z=addop() # Function call with Multi Line assigment
print("Sum({},{})={}".format(x,y,z))
print("-----OR-----")
res=addop() # Function call with Single Line assigment
#here res is an object of <class,tuple>
print("sum({},{})={}".format(res[0],res[1],res[2]))
print("-----OR-----")
print("sum({},{})={}".format(res[-3],res[-2],res[-1]))
print("*****")
print("sum({},{})={}".format(res[0:1],res[1:2],res[2:3]))
print("-----OR-----")
print("sum({},{})={}".format(res[-3:-2],res[-2:-1],res[-1:])))
print("*****")


---


#Define a function for cal addition of two numbers
#ApproachEx1.py
#INPUT : From Function Call
#PROCESSING : Inside of Function Body
#RESULT / OUTPUT : to Function Call
def addop(a,b): # here a and b are called Formal Parameters
    c=a+b # here c is called Local Variable
    return c


---


#main program
x=float(input("Enter First Value:"))
y=float(input("Enter Second Value:"))
z=addop(x,y) # Function call
print("sum({},{})={}".format(x,y,z))


---


#Define a function for cal addition of two numbers
#ApproachEx2.py
#INPUT: Inside of Function Body
#PROCESS: Inside of Function Body
#OUTPUT : Inside of Function Body
def addop():
    #Taking INPUT
    a=float(input("Enter First Value:"))
    b=float(input("Enter Second Value:"))
    #Do the PROCESS
    c=a+b
    #Display the RESULT / OUTPUT
    print("sum({},{})={}".format(a,b,c))

```

```

#main program
addop() # Function cal



---


#Define a function for cal addition of two numbers
#ApproachEx3.py
#INPUT: From Function Call
#PROCESS: Inside of Function Body
#OUTPUT : Inside of Function Body
def addop(a,b):
    #Doing the PROCESS
    c=a+b
    #Display the RESULT
    print("sum({},{})={}".format(a,b,c))
#main program
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
addop(a,b) # Function call


---


#Define a function for cal addition of two numbers
#ApproachEx4.py
#INPUT: Inside of Function Body
#PROCESS: Inside of Function Body
#OUTPUT : to Function Call
def addop():
    #Taking INPUT in Function Body
    a = float(input("Enter First Value:"))
    b = float(input("Enter Second Value:"))
    #doing PROCESS
    c=a+b
    #give result back to Function Call
    return a,b,c# a return kwd can return one or more values
#main program
x,y,z=addop() # Function call with Multi Line assigment
print("Sum({},{})={}".format(x,y,z))
print("-----OR-----")
res=addop() # Function call with Single Line assigment
#here res is an object of <class,tuple>
print("sum({},{})={}".format(res[0],res[1],res[2]))
print("-----OR-----")
print("sum({},{})={}".format(res[-3],res[-2],res[-1]))
print("*****")
print("sum({},{})={}".format(res[0:1],res[1:2],res[2:3]))
print("-----OR-----")
print("sum({},{})={}".format(res[-3:-2],res[-2:-1],res[-1:])))
print("*****")


---


#Function for cal area and perimiter of Circle

```

```

#AreaPeriCircle.py
def area():
    r=float(input("Enter Radius of Circle for Area:"))
    ac=3.14*r**2
    print("Area of Circle={}".format(ac))
def perimeter():
    r = float(input("Enter Radius of Circle for Perimeter:"))
    pc = 2*3.14 * r
    print("Perimeter of Circle={}".format(pc))
#main program
while(True):
    print("====")
    print("\tCircle Calculations")
    print("====")
    print("\tA:Area of Circle")
    print("\tP:Perimeter of Circle")
    print("\tE:Exit")
    print("====")
    ch=input("Enter Ur Choice:")
    if(ch.isalpha()):
        match(ch):
            case "A"|"a":
                area() # Function Call
            case "P"|"p":
                perimeter() # Function Call
            case "E"|"e":
                print("Tq for this program")
                break
            case _:
                print("Ur Selection of Operation is wrong-try again")
    else:
        print("Don't enter Digits and Symbols")
    -----
    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/Variable

Arguments:

=>Arguments are the variables / Values 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 Arguments (Actual variables OR Parameters)

=>The relationship between Arguments and Parameters is that all the Values of arguments are passing to Parameters. This Mechanism is called Arguments Passing.

#Program for cal Factorial of a Given Number
#ArgsParamsEx1.py
def factcal(n): # Here n is called Formal Parameter
if(n<0):
 print("{} Is Invalid Input".format(n))
else:
 fact=1
 for i in range(1,n+1):
 fact*=i
 else:
 print("Fact({})={}".format(n,fact))
#main program
n=int(input("Enter a Number for cal Factorial:"))
factcal(n) # Function call--here n is called argument

#Program for cal Factorial of a Given Number
#ArgsParamsEx2.py
def factcal(n): # Here n is called Formal Parameter
if(n<0):
 return ("{} Is Invalid Input".format(n))
else:
 fact=1 # Here Fact is called Local Variable
 for i in range(1,n+1):
 fact*=i
 else:
 return("Fact({})={}".format(n,fact))

```
#main program
n=int(input("Enter a Number for cal Factorial:"))
res=factcal(n) # Function call--here n is called argument
print(res)
=====
```

Types of Arguments

```
=>The Relation Between Arguments and Parameters is that Every
Value of Argument is passing to Formal Parameter.
=>Based on Passing Arguments to Formal Parameters, Arguments are
classified into 5 Types. They are
```

1. Possitional Arguments
2. Default Arguments
3. Keyword Arguments
4. Variable Length Arguments
5. Keyword Variable Length Arguments

1. Possitional Arguments

```
=>The Default Arguments mechanism in Function is Possitional
Arguments
```

```
=>Possitional Arguments concept says that Every Argument Passing
Every Formal Parameter Based on their Posstion by maintaining
Order and Meaning for Higher Accuracy. In Otherwords The number
of arguments must equal to Number of Formal Parameters.
```

```
=>Possitional Arguments concept always used for Passing Specific
Data from Function calls to Function Definitions.
```

```
=>PVM gives High Priority for Possitional Arguments
```

Syntax

```
def functionname(param1,parm2....., param-n):      # Function
Definition
-----
-----
-----
#main program
functionname(arg1,arg2,...arg-n) # Function Call
=>Here PVM Passes arg1, arg2,...arg-n to param-1,param-2...param-
n Respectively
```

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 Posstional Parameters)
```

Syntax for Function Definition with Default Parameters

```
def functionname(param1,param2,...param-n-1=Val1, Param-n=Val2):
```

Here param-n-1 and param-n are called "default Parameters".
and param1,param-2... are called "Positional parameters".

Rule:- When we use default parameters in the function definition, They must be used as last Parameter(s) otherwise we get Error(SyntaxError: non-default argument (Positional) follows default argument).

```
#Program for Demonstrating Possitonal Args  
#PossArgsEx1.py  
def studinfo(sno,sname,marks):  
    print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks))  
#main program  
print("-"*50)  
print("\tSno\tName\tMarks")  
print("-"*50)  
studinfo(10,"RS",34.56) # Function Call  
studinfo(20,"SB",24.56) # Function Call  
studinfo(30,"TR",44.56) # Function Call  
studinfo(40,"DR",74.56) # Function Call  
print("-"*50)
```

```
#Program for Demonstrating Possitonal Args  
#PossArgsEx2.py  
def studinfo(sno,sname,marks,crs):  
    print("\t{}\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))  
#main program  
print("-"*50)  
print("\tSno\tName\tMarks\tCourse")  
print("-"*50)  
studinfo(10,"RS",34.56,"PYTHON") # Function Call  
studinfo(20,"SB",24.56,"PYTHON") # Function Call  
studinfo(30,"TR",44.56,"PYTHON") # Function Call  
studinfo(40,"DR",74.56,"PYTHON") # Function Call  
print("-"*50)
```

```
#Program for Demonstrating Default Args  
#DefArgsEx1.py  
def studinfo(sno,sname,marks,crs="PYTHON"):  
    print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))  
#main program  
print("-"*50)
```



```

-----
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
#KwdArgsEx1.py
def disp(a,b,c):
    print("\t{}\t{}\t{}".format(a,b,c))
#main program
print("=".*50)
print("\tA\tB\tC")
print("=".*50)
disp(10,20,30) # Function call with Possitional args
disp(c=30,b=20,a=10) # Function call with Keyword args
disp(10,c=30,b=20) # Function call with Possitional and Keyword
                     args
#disp(b=20,a=10,30)---SyntaxError: positional argument follows
                     keyword argument
disp(b=20,a=10,c=30) # Function call with Keyword args
print("=".*50)
-----
#KeywordArgsEx2.py
#This Program will not execute as it is bcoz PVM remembers
latest Function Def only due to ints interpretation Process.
def dispvalues(sno,sname,marks):
    print(sno,sname,marks)
def dispvalues(eno,ename,sal,cname):
    print(eno,ename,sal,cname)
def dispvalues(sid,name,hb1,hb2,hb3):
    print(sid,name,hb1,hb2,hb3)
def dispvalues(a,b,c,d,e,f):
    print(a,b,c,d,e,f)
#main program
dispvalues(sno=10,sname="RS",marks=34.56) # Function call with 3
                                             Kwd args
dispvalues(eno=10,ename="RS",sal=4.56,cname="TCS") # Function
                                               call with 4 Kwd args

```

```

dispvalues(sid=10,name="Raj",hb1="Sleep",hb2="Eating",hb3="Chatting") # Function call with 5 Kwd args
dispvalues(a=10,b=20,c=30,d=40,e=50,f=60) #Function call with 6
                                            Kwd args


---


#Program for Demonstrating Keyword Args
#KwdArgsEx2.py
def dispstudinfo(sno,sname,marks,crs="PYTHON"):
    print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno, sname, marks,
                                             crs))

print("-"*50)
print("\tStno\tName\tMarks\tCourse")
print("-"*50)
dispstudinfo(10,"RS",34.56) # Function Call with positional
args
dispstudinfo(sname="TR",marks=45.67,sno=20)# Function Call with
keyword args
dispstudinfo(crs="Java",sname="TR",marks=45.67,sno=20)# Function
Call with keyword args
dispstudinfo(30,crs="DSA",sname="DR",marks=45.89)# Function Call
with Possitonal and keyword args
dispstudinfo(40,"MC",crs="HTML",marks=56.89) # Function Call
with Possitonal and keyword args
#dispstudinfo(50,"DR",crs="C++",56.89) SyntaxError: positional
argument follows keyword argument
dispstudinfo(50,"DR",58.78,crs="C++")# # Function Call with
Possitonal and keyword args
print("-"*50)


---


#KeywordArgsEx3.py
#This Program will not execute as it is bcoz PVM remembers
latest Function Def only due to ints interpretation Process.
def dispvalues(sno,sname,marks):
    print(sno,sname,marks)
dispvalues(sno=10,sname="RS",marks=34.56) # Function call with 3
Kwd args
#-----
def dispvalues(eno,ename,sal,cname):
    print(eno,ename,sal,cname)
dispvalues(eno=10,ename="RS",sal=4.56,cname="TCS") # Function
call with 4 Kwd args
#-----
def dispvalues(sid,name,hb1,hb2,hb3):
    print(sid,name,hb1,hb2,hb3)
dispvalues(sid=10,name="Raj",hb1="Sleep",hb2="Eating",hb3="Chatting") # Function call with 5 Kwd args
#-----
def dispvalues(a,b,c,d,e,f):

```

```
    print(a,b,c,d,e,f)
dispvalues(a=10,b=20,c=30,d=40,e=50,f=60) # # Function call with
6 Kwd args
#-----
```

4) Variables Length Parameters (or) arguments

=>When we have family of multiple function calls with Variable number of values / arguments then with normal python programming, we must define multiple function definitions. This process leads to more development time.

=>To overcome this process, we must use the concept of Variable length Parameters .

=>To Implement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called astrisk (* param) and the formal parameter with astrisk symbol is called Variable length Parameters and whose purpose is to hold / store any number of values coming from similar function calls and whose type is <class, 'tuple'>.

Syntax for function definition with Variables Length Parameters:

```
def functionname(list of Posstional 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).

```
#Non-VarArgsEx1.py
#Program for Demonstrating Variable length args
def disp1(a,b,c,d,e): # Function Def-1
    print(a,b,c,d,e)
def disp2(a,b,c,d): # Function Def-2
    print(a,b,c,d)
```

```

def disp3(a,b,c): # Function Def-3
    print(a,b,c)
def disp4(a,b): # Function Def-4
    print(a,b)
def disp5(a): # Function Def-5
    print(a)
#main program
print("-"*50)
disp1(10,20,30,40,50) # Function call-1 with 5 args
disp2(10,20,30,40) # Function call-2 with 4 args
disp3(10,20,30) # Function call-3 with 3 args
disp4(10,20) # Function call-4 with 2 args
disp5(10) # Function call-5 with 1 arg
print("-"*50)


---


#Program for Demonstrating Variable length args
#This Program will not execute as it is bcoz PVM remembers
Latest Function Definition Only due to its InterpretationProcess.
#VarArgsEx1.py
def disp(a,b,c,d,e): # Function Def-1
    print(a,b,c,d,e)
def disp(a,b,c,d): # Function Def-2
    print(a,b,c,d)
def disp(a,b,c): # Function Def-3
    print(a,b,c)
def disp(a,b): # Function Def-4
    print(a,b)
def disp(a): # Function Def-5
    print(a)
#main program
print("-"*50)
disp(10,20,30,40,50) # Function call-1 with 5 args
disp(10,20,30,40) # Function call-2 with 4 args
disp(10,20,30) # Function call-3 with 3 args
disp(10,20) # Function call-4 with 2 args
disp(10) # Function call-5 with 1 arg
print("-"*50)


---


#Program for Demonstrating Variable length args
#This Program will execute as it is
#VarArgsEx2.py
def disp(a,b,c,d,e): # Function Def-1
    print(a,b,c,d,e)
disp(10,20,30,40,50) # Function call-1 with 5 args
#-----
def disp(a,b,c,d): # Function Def-2
    print(a,b,c,d)
disp(10,20,30,40) # Function call-2 with 4 args
#-----
```

```

def disp(a,b,c): # Function Def-3
    print(a,b,c)
disp(10,20,30) # Function call-3 with 3 args
#-----
def disp(a,b): # Function Def-4
    print(a,b)
disp(10,20) # Function call-4 with 2 args
#-----
def disp(a): # Function Def-5
    print(a)



---


disp(10) # Function call-5 with 1 arg


---


#Program for Demonstrating Variable length args
#PureVarArgsEx1.py
def disp(*a): # Here *a is called Var Length arg and whose type
is tuple
    print(a,type(a))
#main program
print("-"*50)
disp(10,20,30,40,50) # Function call-1 with 5 args
disp(10,20,30,40) # Function call-2 with 4 args
disp(10,20,30) # Function call-3 with 3 args
disp(10,20) # Function call-4 with 2 args
disp(10) # Function call-5 with 1 arg
disp() #Function call-6 without args
print("-"*50)


---


#Program for Demonstrating Variable length args
#PureVarArgsEx2.py
def disp(*a): # Here *a is called Var Length arg and whose type
is tuple
    print("-----")
    for val in a:
        print(val,end=" ")
    print()
    print("-----")
#main program
print("-"*50)
disp(10,20,30,40,50) # Function call-1 with 5 args
disp(10,20,30,40) # Function call-2 with 4 args
disp(10,20,30) # Function call-3 with 3 args
disp(10,20) # Function call-4 with 2 args
disp(10) # Function call-5 with 1 arg
disp() #Function call-6 without args
print("-"*50)


---


#Program for Demonstrating Variable length args
#PureVarArgsEx3.py

```

```

def disp(sno,sname,*a): # Here *a is called Var Length arg and
whose type is tuple
    s=0
    print("-----")
    print("Student ID:{}".format(sno))
    print("Student Name:{}".format(sname))
    for val in a:
        print("\t{}".format(val))
        s+=val
    print("\tSum={}".format(s))
    print("-----")

#main program
disp(100,"Rossum",10,20,30,40,50) # Function call-1 with 5 args
disp(200,"KVR",10,20,30,40) # Function call-2 with 4 args
disp(300,"NRaju",10,20,30) # Function call-3 with 3 args
disp(400,"Shahoo",20) # Function call-4 with 2 args
disp(500,"Himansu",10) # Function call-5 with 1 arg
disp(600,"Avinash") #Function call-6 without args
disp(700,"Jaynanth",2.3,3.4,5.6) #Function call-7 with 3 args


---


#PureVarArgsEx4.py
#Program for Demonstrating Variable length args
def disp(sno,sname,*a,city="HYD"): # Here *a is called Var
Length arg and whose type is tuple
    s=0
    print("-----")
    print("Student ID:{}".format(sno))
    print("Student Name:{}".format(sname))
    print("Student City:{}".format(city))
    for val in a:
        print("\t{}".format(val))
        s+=val
    print("\tSum={}".format(s))
    print("-----")

#main program
disp(100,"Rossum",10,20,30,40,50) # Function call-1 with 5 args
disp(200,"KVR",10,20,30,40) # Function call-2 with 4 args
disp(300,"NRaju",10,20,30) # Function call-3 with 3 args
disp(400,"Shahoo",20) # Function call-4 with 2 args
disp(500,"Himansu",10) # Function call-5 with 1 arg
disp(600,"Avinash") #Function call-6 without args
disp(700,"Jaynanth",2.3,3.4,5.6,city="MUM") #Function call-7
with 3 args
#disp(800,"Srikanth",city="MP",3.4,5.6) ---Error
disp(800,"Srikanth",3.4,5.6,city="MP")
=====
```

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".

=>Local Variables Can be accessed inside of Corresponding Function Body But Not possible to access in other Part of the program.

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 fun2():
    -----
        def fun-n():
```

Here Var1, Var2..var-n are called Global variables and we can access those values inside of fun1(), fun2()....fun-n().

```
#LocalGlobalVarEx1.py
lang = "PYTHON" # here lang is called Global Variable
def learnML():
    crs1="ML" # Here crs1 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'"
          Lang".format(crs1,lang))
    #print(crs2,crs3) cant access bcozx they are local to other
    Functions
    print("-" * 50)
def learnDL():
    crs2="DL" # Here crs2 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'")
    Lang".format(crs2,lang)
    # print(crs1,crs3) cant access bcozx they are local to other
    Functions
    print("-" * 50)
def learnAI():
    crs3="AI" # Here crs3 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'"
          Lang".format(crs3,lang))
    # print(crs1,crs2) cant access bcozx they are local to other
    Functions
    print("-" * 50)
#main program
learnML()
learnDL()
learnAI()
```

```
#LocalGlobalVarEx2.py
def learnML():
    crs1="ML" # Here crs1 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'"
          Lang".format(crs1,lang))
    #print(crs2,crs3) cant access bcozx they are local to other
    Functions
    print("-" * 50)
def learnDL():
    crs2="DL" # Here crs2 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'"
          Lang".format(crs2,lang))
    #print(crs1,crs3) cant access bcozx they are local to other
    Functions
    print("-" * 50)
lang = "PYTHON" # here lang is called Global Variable
```

```

def learnAI():
    crs3="AI" # Here crs3 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'"
    # print(crs1,crs2) cant access bcozx they are local to other
        Functions
    print("-" * 50)
#main program
learnML()
learnDL()
learnAI()


---


#LocalGlobalVarEx3.py
def learnML():
    crs1="ML" # Here crs1 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'
          Lang".format(crs1,lang))
    #print(crs2,crs3) cant access bcozx they are local to other
        Functions
    print("-" * 50)
def learnDL():
    crs2="DL" # Here crs2 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'
          Lang".format(crs2,lang))
    # print(crs1,crs3) cant access bcozx they are local to other
        Functions
    print("-" * 50)
def learnAI():
    crs3="AI" # Here crs3 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'
          Lang".format(crs3,lang))
    # print(crs1,crs2) cant access bcozx they are local to other
        Functions
    print("-" * 50)
#main program
lang = "PYTHON" # here lang is called Global Variable
learnML()
learnDL()
learnAI()


---


#LocalGlobalVarEx4.py
def learnML():
    crs1="ML" # Here crs1 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'
          Lang".format(crs1,lang))
    #print(crs2,crs3) cant access bcozx they are local to other
        Functions
    print("-" * 50)
def learnDL():

```

```

crs2="DL" # Here crs2 is called Local Variable
print("'{ }' Applications Developed by Using '{ }'
      Lang".format(crs2,lang))
# print(crs1,crs3) cant access bcoz they are local to other
    Functions
print("-" * 50)
def learnAI():
    crs3="AI" # Here crs3 is called Local Variable
    print("'{ }' Applications Developed by Using '{ }'
Lang".format(crs3,lang))
    # print(crs1,crs2) cant access bcoz they are local to other
        Functions
    print("-" * 50)
#main program
#learnML()--cant access lang variable Value
lang = "PYTHON" # here lang is called Global Variable
learnDL()
learnAI()
=====

```

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
-----
```

```

#GlobalKwdEx1.py
a=10 # here a is called Global Variable
def increment():
    global a
```

```

        a=a+1
def multiply():
    global a
    a=a*2
def modifyval():
    c=a-2 # No Need to write global kwd bcoz we are not
           modifying the value of a, but we are just accessing
#main program
print("Val of a in main program before increment()=",a) # 10
increment()
print("Val of a in main program after increment()=",a) # 10
multiply()
print("Val of a in main program after multyiply()=",a) # 22
modifyval()
print("Val of a in main program after modifyval()=",a) # 22
=====
global and local variables and globals()
=====
```

=>When we come acrosss same global Variable names and Local Variable Names in same function definition then PVM gives preference for local variables but not for global variables.
=>In this context, to extract / retrieve the values of global variables names along with local variables, we must use `globals()` and it returns an object of `<class,'dict'>` and this dict object stores all global variable Names as Keys and global variable values as values of value.

=>Syntax:-

```

        var1=val1
        var2=val2
-----
        var-n=val-n # var1, var2...var-n are called global
                     Variables
        def      functionname():
-----
        var1=val11
        var2=val22
-----
        var-n=val-nn # var1, var2...var-n are
                     called local Variables
        # Extrarct the global variables values
        dictobj=globals()
-----
        globalval1=dictobj['var1'] # or
dictobj.get("var1") or globals()['var1'] or
globals().get('var1')
```

```

globalval2=dictobj['var2'] # or
dictobj.get("var2") or globals()['var2'] or
globals().get('var2')


---


#Program for demonstrating globals()
#In This Program local and global variable are different--no
need to use globals()
#GlobalsFunEx1.py
def operation():
    x=100
    y=200 # here x and y are called Local Variables
    #In This Program local and global variable are different
    res=a+b+x+y
    print("-" * 50)
    print("Val of a--Global Variable={}".format(a))
    print("Val of b--Global Variable={}".format(b))
    print("Val of x--Local Variable={}".format(x))
    print("Val of y--Local Variable={}".format(y))
    print("-" * 50)
    print("sum={}".format(res))
#main program
a=10
b=20 # Here a and b are called global variables
operation()


---


#Program for demonstrating globals()
#In This Program local and global variable are Same-- need to
use globals()
#GlobalsFunEx1.py
a=10
b=20 # Here a and b are called global variables
def operation():
    a=100
    b=200 # here a and b are called Local Variables
    #In This Program local and global variable are SAME
    print("-" * 50)
    print("Val of a--Global Variable={}".format(globals()['a']))
    print("Val of b--Global Variable={}".format(globals()['b']))
    print("Val of a--Local Variable={}".format(a))
    print("Val of b--Local Variable={}".format(b))
    print("-" * 50)
    res=a+b+globals()['a']+globals()['b']
    print("sum=",res)
#main program
operation()


---


#Program for demonstrating globals()
#In This Program local and global variable are Same-- need to
use globals()

```

```

#GlobalsFunEx1.py
a=10
b=20 # Here and b are called globals variables
def operation():
    a=100
    b=200
    #To get the global variables inside of function body
    gv=locals() # gv contains GVN:GVV--dict type
    #Here gv contains Present program global variables and
    Invisible Global variables
    print("-----")
    print("type of gv=",type(gv))
    print("-----")
    for gvn,gvv in gv.items():
        print("\t{}---->{}".format(gvn,gvv))
    print("-----")
    print("Total Globals=",len(gv))
    print("-----")
    print("Programmer-Defined Global Variables--Way-1")
    print("-----")
    print("Global Variable a=",gv['a'])
    print("Global Variable b=",gv['b'])
    print("-----")
    print("Programmer-Defined Global Variables--Way-2")
    print("-----")
    print("Global Variable a=", gv.get('a'))
    print("Global Variable b=", gv.get('b'))
    print("-----")
    print("Programmer-Defined Global Variables--Way-3")
    print("-----")
    print("Global Variable a=", globals()['a'])
    print("Global Variable b=", globals()['b'])
    print("-----")
    print("Programmer-Defined Global Variables--Way-4")
    print("-----")
    print("Global Variable a=", globals().get('a'))
    print("Global Variable b=", globals().get('b'))
    print("-----")
#Main program
operation()
=====

Anonymous Functions OR Lambda Functions
=====

=>Anonymous Functions are those which does not contain any
function name explicitly.
=>The purpose of Anonymous Functions is that "To Perform Instant
Operations".

```

=>Instant Operations are those which are performed at that point of time only and No Longer Interested to use in part of the project.

=>To develop Anonymous Functions, we use a keyword called "lambda" and Anonymous Functions are also called Lambda Functions.

=>Anonymous Functions contains Single Executable Statement but never contains Multiple statements.

=>Anonymous Functions returns the result automatically / Implicitly (No Need to use return statement).

Syntax: varname=lambda params-list : Statement

Explanation

=>varname represents an object of <class,'function'> and it can be used as function call

=>lambda is a keyword and It can be used for defining Anonymous Functions

=>params-list represents List of formal Parameters and they are used for storing values coming Function call

=>statement represents single executable statement and whose value returned automatically / Implicitly and there is no need to use return statement.

Question: Write a Function for cal Addition of Two Values

By using Normal Function

```
def sumop(a,b):
    c=a+b
    return c

#main program
res=sumop(10,20)
print("sum=",res) # 30
```

By using
Anonymous Function

```
sumop=lambda a,b : a+b

#main program
res=sumop(100,200)
print("sum=",res) #30 0
```

#AnonymousFunEx1.py

```
def sumop(a,b): # Normal Function Definition
    c=a+b
    return c
addop=lambda k,v: k+v # Anonymous Function Definition
#main program
print("-----")
print("Type of sumop=",type(sumop)) # <class 'function'>
res=sumop(10,20)
print("sum=",res)
```

```

print("-----")
print("-----")
print("Type of addop=", type(addop)) # <class 'function'>
res=addop(100,200)
print("sum=", res)
print("-----")


---


#AnonymousFunEx2.py
palindrome=lambda val: val.lower()==val[::-1].lower()
#main program
val=input("Enter a any value:")
if(palindrome(val)):
    print("{} is PALINDROME".format(val))
else:
    print("{} is NOT PALINDROME".format(val))


---


#AnonymousFunEx3.py
palindrome=lambda val: "PALINDROME" if val.lower()==val[::-1].lower() else "NOT PALINDROME"
#main program
val=input("Enter a any value:")
res= palindrome(val)
print("{} is {}".format(val,res))
=====
```

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 Filtering List of +Ve Elements from List of Values  
#FilterEx1.py
```

```
def pos(n):  
    if(n>0):  
        return True  
    else:  
        return False
```

```
#main program
```

```
lst=[10,-23,45,-46,-67,0,12,89]
```

```
filobj=filter(pos,lst)
```

```
print("type of filobj=",type(filobj)) # <class 'filter'>
```

```
print("content of filobj=",filobj) # <filter object at  
0x000002E341AB9D50>
```

```
#Convert filter object into list
```

```
pslist=list(filobj)
```

```
print("-----")
```

```
print("Given Elements={}".format(lst))
```

```
print("Positive Elements={}".format(pslist))
```

```
print("-----")
```

```
#Program for Filtering List of -Ve Elements from List of Values
```

```
#FilterEx2.py
```

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

```
#main program
```

```
lst=[10,-23,-46,45,-46,-67,0,12,89]
```

```
filobj=filter(negative,lst)
```

```
#Convert filter object into list
```

```
nglist=tuple(filobj)
```

```
print("-----")
```

```
print("Given Elements={}".format(lst))
```

```
print("Negative Elements={}".format(nglist))
```

```
print("-----")
```

```
#Program for Filtering List of +Ve and -Ve Elements from List  
of Values
```

```
#FilterEx3.py
```

```
pos=lambda n: n>0 # Anonymous Function
```

```
neg=lambda n: n<0 # Anonymous Function
```

```
#main program
```

```
lst=[10,-23,45,-46,-67,0,12,89]
```

```
pslist=list(filter(pos,lst))
```

```

nglist=tuple(filter(neg,lst))
print("-----")
print("Given Elements={}".format(lst))
print("Positive Elements={}".format(pslist))
print("Negative Elements={}".format(nglist))
print("-----")


---


#Program for Filtering List of +Ve and -Ve Elements from List
of Values
#FilterEx4.py
lst=[10,-23,45,-46,-67,0,12,89]
pslist=list(filter(lambda n:n>0,lst))
nglist=tuple(filter(lambda n:n<0,lst))
print("-----")
print("Given Elements={}".format(lst))
print("Positive Elements={}".format(pslist))
print("Negative Elements={}".format(nglist))
print("-----")


---


#FilterEx5.py
lst=[12,13,56,53,78,0,34,33,-34,-23]
evenlist=list(filter(lambda n:n%2==0 and n>=0,lst))
oddlist=list(filter(lambda n: n%2!=0 and n>0,lst))
print("-----")
print("Given Elements={}".format(lst))
print("Even Elements={}".format(evenlist))
print("Odd Elements={}".format(oddlist))
print("-----")


---


#Program for finding Length of words which are in list
#WordLengthCount.py
def findwordslength(lst):
    dictobj={}
    for word in lst:
        l1=len(word)
        dictobj[word]=l1
    return dictobj
#main program
lst=["Apple","Kiwi","Mango","Python","Java"]
obj=findwordslength(lst)
for word,length in obj.items():
    print("\t{}\t{}".format(word,length))
=====
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()

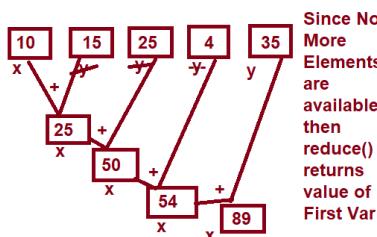
step-1:- Initially, reduce() selects First Two values of Iterable object and place them in First var and Second var .

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.

Consider the following list of Elements
lst=[10,15,25,4,35] -->Find Sum of List of Elements



def sumop(x, y):
 return (x+y)

res= functools.reduce(sumop,lst)

print(res) # 89

OR

res= functools.reduce(lambda x,y:x+y , lst)

print(res) # 89

We know that all functions belongs to Module
here reduce() belongs to a pre-defined Module called functools

varname=functools.reduce(functionname,Iterable-object)

```
#Program for adding list of values by using reduce()
#ReduceEx1.py
import functools
def sumop(x,y):
    return (x+y)
#main program
lst1=[10,34,23,56,78,12,47]
res=functools.reduce(sumop,lst1)
print("sum({})={}".format(lst1,res))
lst2=[1.2,3.4,4.5,6.7]
res=functools.reduce(sumop,lst2)
```

```

print("sum({ })={ }".format(lst2,res))


---


#Program for adding list of values by using reduce()
#ReduceEx2.py
import functools
lst1=[10,34,23,56,78,12,47]
res=functools.reduce(lambda a,b:a+b,lst1)
print("sum({ })={ }".format(lst1,res))
lst2=[1.2,3.4,4.5,6.7]
res=functools.reduce(lambda x,y:x+y,lst2)
print("sum({ })={ }".format(lst2,res))


---


#Program for finding max from list of values by using reduce()
#ReduceEx3.py
import functools
lst1=[10,34,23,56,78,12,47]
maxv=functools.reduce(lambda a,b:a if a>b else b,lst1)
print("Max({ })={ }".format(lst1,maxv))


---


#Program for concatenating list of words as line of text by
using reduce()
#ReduceEx5.py
import functools
lst=["Apple","is","red","in","color"]
line=functools.reduce(lambda x,y:x+" "+y,lst)
print("-----")
print("Given Words={ }".format(lst))
print("Line={ }".format(line))
print("-----")
lst=["Python","is","an","OOP","LANG"]
line=functools.reduce(lambda x,y:x+" "+y,lst)
print("-----")
print("Given Words={ }".format(lst))
print("Line={ }".format(line))
print("-----OR-----")
line=" ".join(lst)
print("Given Words={ }".format(lst))
print("Line={ }".format(line))
print("-----")


---


#DictComprehensionEx1.py
lst=["Python","DSA","HYDERABAD","Django"]
dobj=dict([(word,len(word)) for word in lst]) # List
Comprehension
print(dobj,type(dobj))


---


#DictComprehensionEx2.py
import math
lst=[2,3,1,0,-3,6,7,-8]
dobj=dict([(val,math.factorial(val)) for val in lst if val>=0])
print("-----")

```

```

print("Given List of Values={ }".format(lst))
print("Dict of Factorials={ }".format(dobj))
print("-----")


---


#DictComprehensionEx3.py
def kvrfact(n):
    if(n<0):
        return "No Factorial"
    else:
        f=1
        for val in range(1,n+1):
            f*=val
        return f
#main program
lst=[2,3,1,0,-3,6,7,-8]
dobj=dict([(val,kvrfact(val)) for val in lst if val>=0])
print("-----")
print("Given List of Values={ }".format(lst))
print("Dict of Factorials={ }".format(dobj))
print("-----")


---


#DictComprehensionEx3.py
def kvrfact(n):
    if(n<0):
        return "No Factorial"
    else:
        f=1
        for val in range(1,n+1):
            f*=val
        return f
#main program
lst=[2,3,1,0,-3,6,7,-8]
dobj=dict([(val,kvrfact(val)) for val in lst ])
print("-----")
print("Given List of Values={ }".format(lst))
print("Dict of Factorials={ }".format(dobj))
print("-----")


---


#DictComprehensionEx1.py
lst=["Python","DSA","HYDERABAD","Django","Apple"]
words=[word for word in lst if len(word)>=3 and len(word)<=5]
print(words)
print("-----OR-----")
words=list(filter(lambda word:len(word)>=3 and
len(word)<=5,lst))
print(words)
print("-----OR-----")
words=list(filter(lambda word:3<=len(word)<=5, lst))
print(words)


---



```

```

#ListComprehensionEx1.py
lst=[2,5,6,12,13,9]
lst1=[val**2 for val in lst] # List Comprehension
print("-----")
print("Given Elements={}".format(lst))
print("Squares={}".format(lst1))
print("-----")
lst2=[round(val**0.5,2) for val in lst] # List Comprehension
print("Given Elements={}".format(lst))
print("Square Roots={}".format(lst2))
print("-----")


---


#Find the squares of +ve numbers by using List Comprehension
#ListComprehensionEx2.py
lst=[2,5,-6,12,-13,9,0,-21]
lst1=[val**2 for val in lst if val>0 ] # List Comprehension
pselements=[val for val in lst if val>0] # List Comprehension
print("Given Elements={}".format(lst))
print("+Ve Elements={}".format(pselements))
print("Squares of +Ve Numbers={}".format(lst1))


---


#SetComprehensionEx1.py
lst=[10,-23,45,67,-45,-23,0]
pslist={val for val in lst if val>0} # Set Comprehension
print("Given Elements={}".format(lst))
print("Positive Elements={} and"
      "type={}".format(pslist,type(pslist)))


---


#TupleComprehensionEx1.py
# Tuple Comprehension is not possible directly but we can
convert generator object into tuple
lst=[10,-23,45,67,-45,-23,0]
pslist=(val for val in lst if val>0) # tuple Comprehension
print(type(pslist)) # <class 'generator'>
print("Content=",pslist) # <generator object <genexpr> at
0x00000167A18D9BD0>
#Convert Generator into tuple
tpl=tuple(pslist)
print("Given Elements={}".format(lst))
print("Positive Elements={} and type={}".format(tpl,type(tpl)))
=====
```

Modules in Python

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 Modules are already defined by Python Lang Developers and Available in Python Software and Used by all Python Lang Programmers and for dealing with Universal Requirements.

Examples:

functools,sys,math,calendar,re,pickle,threading,csv..etc

=>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 Modules 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...etc

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:

Name	<u><u>__pycache__</u></u>	<-----Folder
Aop.cpython-311.pyc	<-----Module Name	
mathsinfo.cpython-311.pyc	<-----Module Name	
icici.cpython-311.pyc	<-----Module Name	

```
#MathsInfo.py--File Name and acts as Module Name
PI=3.14
E=2.71 # Here PI and E are called Global variables
-----
import MathsInfo
print("Val of PI=",MathsInfo.PI)
print("Val of E=",MathsInfo.E)
-----
#Aop.py--File Name and acts as Module Name
def addop(a,b):
    print("sum({},{})={}".format(a,b,a+b))
def subop(a,b):
    print("sub({},{})={}".format(a, b, a - b))
def mulop(a,b):
    print("mul({},{})={}".format(a, b, a * b))
-----
#SE2.py
import Aop
Aop.addop(10,20)
Aop.subop(10,20)
Aop.mulop(10,20)
#Define a Function for cal simple Interest and total amt to pay
#icici.py--File Name and Module Name
bname="ICICI"
addr="HYDERABAD" # Global Variables
def simpleint(): # Function Definition
    p=float(input("Enter Principle Amount:"))
    t =float(input("Enter Time:"))
    r = float(input("Enter Rate of Interest:"))
    si=(p*t*r)/100
    totamt=p+si
    print("-"*50)
    print("Simple Interest Results")
```

```

print("-" * 50)
print("Principle Amount:{}".format(p))
print("Time:{}".format(t))
print("Rate of Interest:{}".format(r))
print("Simple Interest:{}".format(si))
print("Total Amount to Pay:{}".format(totamt))
print("-" * 50)
-----
import icici
print("Bank Name:{}".format(icici.bname))
print("Bank Address:{}".format(icici.addr))
icici.simpleint()
-----
#ImportStmtEx1.py
#Syntax1: import Module Name
import icici # Import single Module
print("Bank Name:{}".format(icici.bname))
print("Bank Address:{}".format(icici.addr))
icici.simpleint()
-----
#ImportStmtEx2.py
#Syntax2: import Module Name1,Module Name2....Module Name-n
import icici,MathsInfo,Aop # Import Multiple Modules
print("Bank Name:{}".format(icici.bname))
print("Bank Address:{}".format(icici.addr))
icici.simpleint()
print("-----")
print("Val of PI=",MathsInfo.PI)
print("Val of E=",MathsInfo.E)
print("-----")
Aop.addop(10,20)
Aop.subop(10,20)
Aop.mulop(10,20)
-----
#ImportStmtEx3.py
#Syntax2: import Module Name as alias name
import icici as i
import MathsInfo as M
import Aop as a
print("Bank Name:{}".format(i.bname))
print("Bank Address:{}".format(i.addr))
i.simpleint()
print("-----")
print("Val of PI=",M.PI)
print("Val of E=",M.E)
print("-----")
a.addop(10,20)
a.subop(10,20)

```

```

a.mulop(10,20)


---


#ImportStmtEx4.py
#Syntax4: import Module Name1 as alias name1,Module Name2 as
alias name2,..Module Name-n as alias name-n
import icici as i, MathsInfo as M, Aop as a
print("Bank Name:{}".format(i.bname))
print("Bank Address:{}".format(i.addr))
i.simpleint()
print("-----")
print("Val of PI=",M.PI)
print("Val of E=",M.E)
print("-----")
a.addop(10,20)
a.subop(10,20)
a.mulop(10,20)


---


#FromImportStmtEx1.py
#Syntax: from Module Name import
FuncName,GlobalVarname,ClassName
from icici import bname,addr,simpleint
print("Bank Name:{}".format(bname))
print("Bank Address:{}".format(addr))
simpleint()


---


#FromImportStmtEx2.py
#Syntax: from Module Name import FuncName as alias
name,GlobalVarname as alias name,ClassName as alias name
from icici import bname as bn,addr as a,simpleint as si
print("Bank Name:{}".format(bn))
print("Bank Address:{}".format(a))
si()


---


#FromImportStmtEx3.py
#Syntax: from Module Name import * --Not all recommended in
industry
from icici import *
print("Bank Name:{}".format(bname))
print("Bank Address:{}".format(addr))
simpleint()


---


#HighestLenWordEx1.py---File Name and acts as Module Name
def getwords():
    lst=[]
    nw=int(input("How Many words u have:"))
    if(nw<=0):
        return lst
    else:
        for i in range(1,nw+1):
            wrd=input("Enter {} Word:".format(i))
            lst.append(wrd)

```

```

        return lst
def findhighlenword():
    words=getwords()
    if(len(words)==0):
        print("No More Words--Can't Find Highest word length")
    else:
        d={} # Create empty dict
        for word in words:
            d[word]=len(word)
        else:
            ml=max(d.values())
            hwords=[word for word,length in d.items() if
                    length==ml]
            print("-----")
            print("Highest Length Word(s)")
            print("-----")
            for w in hwords:
                print("\t{}".format(w))
            print("-----")
-----
#HighestWordLenDemo.py
from HighestLenWordEx1 import findhighlenword as hw
hw()
=====
```

Arithmetic Operations

- 1. Addition
- 2. Substraction
- 3. Multiplication
- 4. Division
- 5. Modulo Division
- 6. Exponentiation
- 7. Exit

=====

Enter Ur Choice:

=====

File System

AopMenu.py<---File Name and Moudle Name
AopOperations.py<----File Name and Module Name
AopDemo.py<---Main program

Area of Different Figures

- 1. Circle

```

        2. Rectangle
        3. Square
        4. Triangle
        5. Exit
=====
Enter ur Choice:
=====

File System
-----
FigureMenu.py<---File Name and Module Name
    menu()
Circle.py<----File Name and Module Name
    area()
Rect.py<----File Name and Module Name
    area()
Square.py<----File Name and Module Name
    area()
Triangle.py<----File Name and Module Name
    area()
#FigureDemo.py-----Main program

=====
Temprature Converter
=====
1. C to F
2. C to K
3. F to C
4. F to K
5. K to C
6. K to F
7. Exit
=====

Enter Ur Choice:
=====

Celsius to Kelvin:  $K = C + 273.15$ 
Kelvin to Celcius:  $C = K - 273.15$ 
Fahrenheit to Celcius:  $C = (F-32) \cdot (5/9)$ 
Celsius to Fahrenheit:  $F = C(9/5) + 32$ 
Fahrenheit to Kelvin:  $K = (F-32) \cdot (5/9) + 273.15$ 
Kelvin to Fahrenheit:  $F = (K-273.15) \cdot (9/5) + 32$ 
=====

#AopMenu.py---File Name and Module Name
def menu():
    print("=*50)
    print("\tArithmetic Operations")
    print("=*50)
    print("\t1.Addition")
    print("\t2.Subtraction")

```

```

print("\t3.Multiplication")
print("\t4.Division")
print("\t5.Modulo Division")
print("\t6.Exponentiation")
print("\t7.Exit")
print("=*50)
-----
#AopOperations.py--File Name and Module Name
def readvalues(op):
    print("Enter Two Values for performing '{}'.format(op))"
    a,b=float(input()),float(input())
    return a,b
def addop():
    a,b=readvalues("Addition")
    print("\tSum({},{})={}".format(a,b,a+b))
def subop():
    a,b=readvalues("Substraction")
    print("\tSub({},{})={}".format(a, b, a - b))
def mulop():
    k,v=readvalues("Multiplication")
    print("\tMul({},{})={}".format(k, v, k * v))
def divop():
    a,b=readvalues("Division")
    print("\tNormal Div({},{})={}".format(a, b, a / b))
    print("\tFloor Div({},{})={}".format(a, b, a // b))
def modop():
    a,b=readvalues("Modulo Div")
    print("\tMod({},{})={}".format(a, b, a % b))
def expop():
    a, b = float(input("Enter Base:")), float(input("Enter Power:"))
    print("\tpow({},{})={}".format(a, b, a ** b))
-----
#AopRunner.py---File Name and Module Name
import sys
from AopMenu import menu
from AopOperations import addop,subop,mulop,divop,modop,expop
def aop():
    while(True):
        menu()
        ch=int(input("Enter Ur Choice:"))
        match(ch):
            case 1:
                addop()
            case 2:
                subop()
            case 3:

```

```

        mulop()
case 4:
    divop()
case 5:
    modop()
case 6:
    expop()
case 7:
    print("Thx for using this program")
    sys.exit()
case _:
    print("Ur Selection of Operation wrong-try
          again")
-----
#aop.py
From AOPRunner import Aop
K=Aop()
=====
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().
=====
=====X=====

#shares.py---file and treated as module name
def sharesinfo():
    d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}
    return d
#main program
#sharesdemo.py
import shares
import time
import importlib
def disp(d):
    print("-"*50)
    print("\tShare Name\tValue")

```

```

print("-"*50)
for sn,sv in d.items():
    print("\t{}\t\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)
-----
#SharesDemo.py
import Shares,time,importlib
def dispShares(d):
    print("=*50)
    print("Share name\tShare Value")
    print("=* 50)
    for sn,sv in d.items():
        print("\t{}\t\t:{}".format(sn,sv))
    print("=* 50)
#main program
d=Shares.sharesinfo()
dispShares(d)
print("I am going to sleep for 10 seconds")
time.sleep(20)
print("I am coming out-of sleep for 10 seconds")
importlib.reload(Shares) # Reloading Previous imported Module
d=Shares.sharesinfo()
dispShares(d)
print("I am going to sleep for 20 seconds")
time.sleep(20)
print("I am coming out-of sleep for 20 seconds")
importlib.reload(Shares) # Reloading Previous imported Module
d=Shares.sharesinfo()
dispShares(d)
=====
```

Package in Python

=>The Function concept is used for Performing some operation and provides code re-usability within the same program and unable to provide code re-usability across programs.

=>The Modules concept is a collection of Variables, Functions and classes and we can re-use the code across the Programs provided Module name and main program present in same folder but

unable to provide code re-usability across the folders / drives / environments.

=>The Package Concept is a collection of Modules.

=>The purpose of Packages is that to provide code re-usability across the folders / drives / environments.

=>To deal with the package, we need to learn the following.

- a) create a package
 - b) re-use the package
-

a) create a package:

=>To create a package, we use the following steps.

- i) create a Folder
 - ii) place / write an empty python file called `__init__.py` (Optional)
 - 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
    icicil.py   ---Module Name
    welcome.py  <--- Module Name
    greet.py    <---Module names
```

b) re-use the package

=>To re-use the modules of the packages across the folders / drives / environments, we have two approaches. They are

- i) By using sys module
 - ii) by using PYTHONPATH Environmental Variable Name
-

i) By using sys module:

Syntax:

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

=>sys is pre-defined module

=>path is a pre-defined object of list / 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("D:\\KVR-PYTHON-6pm\\PACKAGES\\BANK")  
                 (or)
```

```
sys.path.append("D:\\KVR-PYTHON-6PM\\ACKAGES\\BANK")  
                 (or)
```

```
sys.path.append("D:\\KVR-PYTHON-6PM/PACKAGES/BANK")  
-----
```

ii) by using PYTHONPATH Environmental Variables:

=>PYTHONPATH is one of the Environmental Variable

=>Search for Environmental Variable

Steps for setting :

```
-----  
     Var name : PYTHONPATH  
     Var Value : D:\\KVR-PYTHON-  
                  11am\\PACKAGES\\BANK
```

The overall path

```
PYTHONPATH= D:\\KVR-PYTHON-11am\\PACKAGES\\BANK  
=====
```

Differences Between Functions, Modules and Packages

```
=====
```

Functions:

```
-----  
=>Functions are used for Performing Operations and Provides Code  
Re-Usability within in the Same Program but not across the  
Programs
```

Modules

```
-----  
=>Module is a Collection of Global Variables, Function Names  
and Classes
```

```
=>Modules are used for Providing Code Re-Usability within in  
the Same Program and Across the Programs(Global Var, Functions  
and Classes) provided Module and main program present in Same  
Folder but not Possible to access across the Folders /  
Environments / Networks.
```

Packages

```
-----  
=>Packages is a Collection of Modules.
```

```
=>Package are used for Providing Code Re-Usability within in  
the Same Program , Across the Programs(Global Var, Functions and  
Classes) and across the Folders / Environments / Networks.
```

```
#AopMenu.py---File Name and Module Name
```

```

def menu():
    print("=*50)
    print("\tArithmetic Operations")
    print("=*50)
    print("\t1.Addition")
    print("\t2.Substraction")
    print("\t3.Multiplication")
    print("\t4.Division")
    print("\t5.Modulo Division")
    print("\t6.Exponentiation")
    print("\t7.Exit")
    print("=*50)
-----
#AopOperations.py--File Name and Module Name
def readvalues(op):
    print("Enter Two Values for performing '{}'.format(op))"
    a,b=float(input()),float(input())
    return a,b
def addop():
    a,b=readvalues("Addition")
    print("\tSum({},{})={}".format(a,b,a+b))
def subop():
    a,b=readvalues("Substraction")
    print("\tSub({},{})={}".format(a, b, a - b))
def mulop():
    k,v=readvalues("Multiplication")
    print("\tMul({},{})={}".format(k, v, k * v))
def divop():
    a,b=readvalues("Division")
    print("\tNormal Div({},{})={}".format(a, b, a / b))
    print("\tFloor Div({},{})={}".format(a, b, a // b))
def modop():
    a,b=readvalues("Modulo Div")
    print("\tMod({},{})={}".format(a, b, a % b))
def expop():
    a, b = float(input("Enter Base:")), float(input("Enter
                                                Power:"))
    print("\tpow({},{})={}".format(a, b, a ** b))
-----
#AopRunner.py---File Name and Module Name
import sys
from AopMenu import menu
from AopOperations import addop,subop,mulop,divop,modop,expop
def aop():
    while(True):
        menu()

```

```

ch=int(input("Enter Ur Choice:"))
match(ch):
    case 1:
        addop()
    case 2:
        subop()
    case 3:
        mulop()
    case 4:
        divop()
    case 5:
        modop()
    case 6:
        expop()
    case 7:
        print("Thx for using this program")
        sys.exit()
    case _:
        print("Ur Selection of Operation wrong-try
again")
-----
#Aop.py--File Name and acts as Module Name
def addop(a,b):
    print("sum({},{})={}".format(a,b,a+b))
def subop(a,b):
    print("sub({},{})={}".format(a, b, a - b))
def mulop(a,b):
    print("mul({},{})={}".format(a, b, a * b))
-----
#define a Function for cal simple Interest and total amt to pay
#icici.py--File Name and Module Name
bname="ICICI"
addr="HYDERABAD" # Global Variables
def simpleint(): # Function Definition
    p=float(input("Enter Principle Amount:"))
    t =float(input("Enter Time:"))
    r = float(input("Enter Rate of Interest:"))
    si=(p*t*r)/100
    totamt=p+si
    print("-"*50)
    print("Simple Interest Results")
    print("-" * 50)
    print("Principle Amount:{}".format(p))
    print("Time:{}".format(t))
    print("Rate of Interest:{}".format(r))
    print("Simple Interest:{}".format(si))
    print("Total Amount to Pay:{}".format(totamt))
    print("-" * 50)

```

```

#KvrMath.py---File Name and Module Name
def factorial(n):
    if(n<0):
        print("factorial can not define for negative values")
    else:
        fact=1
        for i in range(1,n+1):
            fact=fact*i
        else:
            print("Factotial({})={}".format(n,fact))

```

Exception Handling in Python-----4 Days

Types of Errors in Python

=>The purpose of Exception Handling in Python is that "To Build Robust(Strong) Applications ".
=>To develop any Real Time Project, we need to Choose a Programming language and By using that Programming Language we develop, compile and execute Various Programs. During this Process, we get 3 Types of Errors. They are

- 1. Compile Time Errors
 - 2. Logical Errors
 - 3. Runtime Errors
-

1. Compile Time Errors

=>Compile Time Errors are those which are occurring during Compilation Process(.py---->.pyc)

=>Compile Time Errors are occurring due to Syntaxes are not followed.

=>Compile Time Errors solved by Programmers at Development time.

2. Logical Errors

=>Logical Errors are those which are occurring during Execution / Run Time.

=>Logical Errors are occurring due to Wrong Representation of Logics.

=>Logical Errors always generates Wrong Results and Logical Errors solved by Programmers during Development Time.

3. Runtime Errors (Implementation Errors)

=>Runtime Errors are those which are occurring during Execution / Runtime .

=>Runtime Errors are occurring due to Invalid Inputs OR Wrong Inputs entered by Application Users OR End Users.

=>By Default Runtime Errors in all programming languages gives Technical Error Messages and they are understandable by Programmers But not applications or end users. So that Industry always Recommends Convert Technical Error Messages into User-Friendly Error Messages by using Exception Handling for making the application / project as Robust.

=====

Building points in Exception handling

=====

1) When the Application user enters Invalid Input then we get Runtime Error.

(Invalid Input----->Runtime Error)

2) By Default Every Run Time Error gives Technical Error Messages

(Invalid Input----->Runtime Error---->Technical Error Messages)

3) Definition of Exception: Every Runtime Error is Called Exception

(Invalid Input---->Runtime Error--->Exception---->Technical Error Message)

hence all Invalid Input gives Exceptions and generates Technical Error Message

4) Definition of Exception Handling:

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

5) When an exception occurs in Python Program, Internally 3 Steps Takes

- a) Program Execution Abnormally Terminated
- b) PVM Comes out of Program Flow
- c) PVM Generates Technical Error Messages

6) To do Step-(a), Step-(b) and Step-(c) , PVM Creates an object of appropriate exception class

7) When an exception occurs then PVM creates an object of appropriate exception class.

(ValueError, IndexError, KeyError, ZeroDivisionError...etc)

8) Hence Every Exception must considered as an object

- 9) a) To Project, If the End user enters Valid Input then end users Valid Result
b) To Project, If the End user enters InValid Input then end users Exception(It turns Converts Techincal Error Messages into User Friendly Error Messages)
-

Type of Exceptions in Python

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

1. Pre-Defined Exceptions.
 2. Programmer-Defined Exceptions.
-

1. Pre-Defined Exceptions

=>These exceptions are already defined by Python Lang Developers and avilable in Python Software and used by Python Lang Programmers for dealing with Universal Problems.

=>Some of the Universal Problems are

1. Invalid Number Conversions (ValueError)
 2. Wrong Operation of Data(TypeError)
 3. Division by zero (ZeroDivisionError)
 4. Wrong Index (IndexError)
 5. Invalid Key (KeyError)
 6. Invalid Module Name importing (ModuleNotFoundError) ...etc
-

2. Programmer-Defined Exceptions.

=>These exceptions are developed Python Programmers and they are available as a part of Python Project and they are used by Python Project Team Members and they deals with Common Problems.

=>Some of the Common Problems are.

- 1) Attempting to Enter Invalid PIN in ATM Applications.
 - 2) Attempting to Enter Wrong User Name and Password.
 - 3) Attempting withdraw more amount than existing bal in account.....etc
-

Syntax for Handling the Exceptions in python

=>Handling the Exceptions in python is nothing but converting Technical Error Messages into User-Friendly Error Messages.
=>For handling the exceptions in python, we have 5 Key words. They are

1. try
 2. except
 3. else
 4. finally
 5. raise
-

Syntax for Handling the exception

```
try:
    -----
        Block of statements generates exceptions
    -----
    except Exception-Class-Name1:
        -----
            User-Friendly Error Messages
        -----
    except Exception-Class-Name2:
        -----
            User-Friendly Error Messages
        -----
    -----
    -----
    except Exception-Class-Name-n:
        -----
            User-Friendly Error Messages
        -----
else:
    -----
        Block of statements dispalys Results
    -----
finally:
    -----
        Block of Statements executes Compulsorily
-----
```

```
#Program for Cal Division of Two Numbers
#Div1.py
print("Program execution started")
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1) # ValueError-----
b=int(s2) # ValueError-----
c=a/b # ZeroDivisionError--x
```

```

print("Div={}".format(c))
print("Program execution ended")


---


#Program for Cal Division of Two Numbers
#Div2.py
try:
    print("Program execution started")
    s1 = input("Enter First Value:")
    s2 = input("Enter Second Value:")
    a = int(s1) # ValueError-----
    b = int(s2) # ValueError-----
    c = a / b # ZeroDivisionError---
except ZeroDivisionError:
    print("DON'T ENTER ZERO FOR DEN....")
except ValueError:
    print("DON'T ENTER STRS, SYMBOLS AND ALPHA-NUMERIC")
else:
    print("-----else block-----")
    print("Div={}".format(c))
    print("-----")
finally:
    print("i am from finally block")


---


=====
```

Various Forms of except blocks

=>except block can be used in four ways. They are

Form1: An except block Can handle One Specific Exception at a Time

Syntax: try:

```

-----  

        Block of statements  

        generates exception  

-----  

        except exception-classname-1:  

-----  

        Block of statements  

        generates User-Freindly  

        Error Message  

-----
```

Form 2: A Single except block Can handle Multiple Specific Exceptions at a Time and such type except block is called Multi Exception handling Block

Syntax: try:

```
    Block of statements  
    generates exception  
-----  
except (exception-clsname-1,exception-clsname-  
2,...exception-clsname-n):  
-----
```

```
    Block of statements  
    generates User-Friendly  
    Error Message  
-----
```

Form 3: An except block with specific exception along alias can capture Third Party Softwares /Database / Could Related Exception Messages in Python Program

Syntax: try:

```
    -----  
    Block of statements  
    generates exception  
-----  
except exception-classname-1 as alias_name:  
-----  
    print(alias_name)  
-----  
except exception-classname-2 as alias_name:  
-----  
    print(alias_name)  
-----
```

Form 4: An except block without exception name is called default except block and It can handle all types of exception and It must be written always after specific exceptions But not recommended to write single default except block for handling all type of exception .

Syntax: try:

```
    -----  
    Block of statements generates exceptions  
-----  
except Exception-Class-Name1:  
-----  
    User-Friendly Error Messages  
-----  
except Exception-Class-Name2:  
-----  
    User-Friendly Error Messages  
-----
```

```
-----
-----  
except Exception-Class-Name-n:  
-----  
    User-Friendly Error Messages  
-----  
except : # Default Except Block  
-----  
    default Message  
-----  
else:  
-----  
    Block of statements dispalys Results  
-----  
finally:  
-----  
    Block of Statements executes Compulsorily  
-----  
(OR)
```

Syntax: try:

```
-----  
    Block of statements  
    generates exception  
-----  
except (exception-clsname-1,exception-clsname-  
2,...exception-clsname-n):  
-----  
    Block of statements  
    generates User-Friendly  
    Error Message  
-----  
except : # Default Except Block  
-----  
    default Message  
-----  
else:  
-----  
    Block of statements dispalys Results  
-----  
finally:  
-----  
    Block of Statements executes Compulsorily  
-----
```

#Program for Cal Division of Two Numbers

#Div3.py

try:

```
print("Program execution started")
s1 = input("Enter First Value:")
s2 = input("Enter Second Value:")
a = int(s1) # ValueError-----
b = int(s2) # ValueError-----
c = a / b # ZeroDivisionError--
except (ZeroDivisionError,ValueError):
    print("DON'T ENTER ZERO FOR DEN....")
    print("DON'T ENTER STRS, SYMBOLS AND ALPHA-NUMERIC")
else:
    print("-----else block-----")
    print("Div={}".format(c))
    print("-----")
finally:
    print("i am from finally block")
```

#Program for Cal Division of Two Numbers

#Div4.py

```
try:
    print("Program execution started")
    s1 = input("Enter First Value:")
    s2 = input("Enter Second Value:")
    a = int(s1) # ValueError-----
    b = int(s2) # ValueError-----
    c = a / b # ZeroDivisionError--
except ZeroDivisionError as z:
    print(z)
except ValueError as v:
    print(v)
else:
    print("-----else block-----")
    print("Div={}".format(c))
    print("-----")
finally:
    print("i am from finally block")
```

#Program for Cal Division of Two Numbers

#NotRecDiv.py

```
try:
    print("Program execution started")
    s1 = input("Enter First Value:")
    s2 = input("Enter Second Value:")
    a = int(s1) # ValueError-----
    b = int(s2) # ValueError-----
    c = a / b # ZeroDivisionError--
except :
    print("oooops Some thing went wrong!!!")
else:
```

```

        print("-----else block-----")
        print("Div={} ".format(c))
        print("-----")
finally:
    print("i am from finally block")

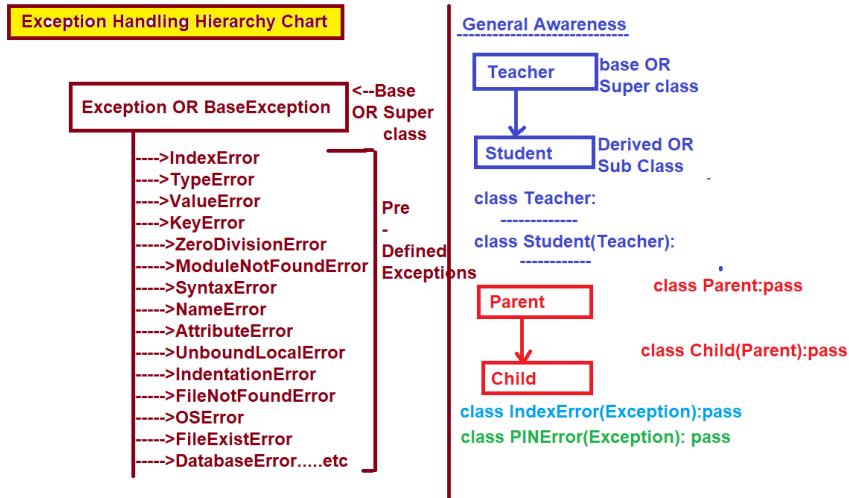

---


#Program for Cal Division of Two Numbers
#RecDiv.py----In 2023 Aug 5th KVR developed this code
#In 2025 Year Some Anil Kumar new Programmer who want add some
new statements
try:
    print("Program execution started")
    s1 = input("Enter First Value:")
    s2 = input("Enter Second Value:")
    a = int(s1)  # ValueError-----
    b = int(s2)  # ValueError-----
    c = a / b  # ZeroDivisionError---
    #2025--
    s="PYTHON"
    print(s[0])
except ZeroDivisionError:
    print("DON'T ENTER ZERO FOR DEN....")
except ValueError:
    print("DON'T ENTER STRS, SYMBOLS AND ALPHA-NUMERIC")
except IndexError:
    print("Plz Check the Index")
except: # Default except Block
    print("ooops some thing went wrong!!!!")
else:
    print("-----else block-----")
    print("Div={} ".format(c))
    print("-----")
finally:
    print("i am from finally block")


---



```



Development of Programmer-Defined Exceptions

=>These exceptions are developed Python Programmers and they are available as a part of Python Project and they are used by Python Project Team Members and they deals with Common Problems.
=>Some of the Common Problems are.

- 1) Attempting to Enter Invalid PIN in ATM Applications
 - 2) Attempting to Enter Wrong User Name and Password.
 - 3) Attempting withdraw more amount than existing bal in account
-etc

Steps for the Development of Programmer-Defined Exceptions

Step-1: Choose the Programmer-Defined Class Name

Step-2: The Programmer-Defined Class Name must Inherit From "Exception" OR "BaseException" for inheriting the features of Exception Handling and Programmer-Defined Class is called Programmer-Defined Exception Sub Class.

Step-3: Save the above Two Steps on some file name with an extension . py

Example: Develop a programmer-defined exception class for Invalid PIN

```
class PINError(Exception):pass
```

Example: Develop a programmer-defined exception class for Login issues.

```
class LoginError(BaseException):pass
```

Here PINError, LoginError are called Programmer-Defined Exception Classes.

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 whereas 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>

```
#ATNMenu.py
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
from ATMExcept import DepositError,WithDrawError,InSuffFundError
bal=500.00 # Global Variable
def deposit():
    damt=float(input("Enter the Deposit amount:"))#implcitly
    raises ValueError
```

```

if(damt<=0):
    raise DepositError
else:
    global bal
    bal=bal+damt
    print("Ur Account XXXXXX123 Credited with
INR:{}".format(damt))
    print("Now Ur Account XXXXXX123 after deposit
INR:{}".format(bal))
def withdraw():
    global bal
    wamt = float(input("Enter the withdraw amount:")) # 
    implicitly raises ValueError
    if(wamt<=0):
        raise WithDrawError
    elif((wamt+500)>bal)):
        raise InSuffFundError
    else:
        bal=bal-wamt
        print("Ur Account XXXXXX123 Debited by
INR:{}".format(wamt))
        print("Now Ur Account XXXXXX123 after withdraw
INR:{}".format(bal))
def balenq():
    print("Ur Account XXXXXX123 INR:{}".format(bal))
-----
#ATMMain.py
from ATMMenu import menu
from ATMExcept import DepositError,WithDrawError,InSuffFundError
from ATMOoperations import deposit,withdraw,balenq
while(True):
    try:
        menu()
        ch=int(input("Enter Ur Choice:"))
        match(ch):
            case 1:
                try:
                    deposit()
                except DepositError:
                    print("Don't try to deposit -Ve and Zero
Values")
                except ValueError:
                    print("Don't try to deposit alnums,space and
symbols")
            case 2:
                try:
                    withdraw()

```

```

        except WithDrawError:
            print("Don't try to withdraw -Ve and Zero
Notes")
        except InSuffFundError:
            print("U don't have suff funds--Read Python
Notes")
        except ValueError:
            print("Don't try to deposit alnums,space and
symbols")
    case 3:
        balenq()
    case 4:
        print("Thx for this Program")
        break
    case _:
        print("Ur Selection of Operation is wrong-try
again")
except ValueError:
    print("Don't enter alnums,strs and symbols for choice of
ATM Operation")
-----
```

```
#ATMExcept.py
class DepositError(Exception):pass
class WithDrawError(Exception):pass
class InSuffFundError(Exception):pass
-----
```

```
#kvr.py
#      (1)          (2)
class KvrDivisionError(Exception):pass
-----
```

```
#Division.py--File Name and Module Name
from kvr import KvrDivisionError
def divop(a,b): # a=10 b=0
    if(b==0):
        raise KvrDivisionError # hitting the exception
    else:
        return (a/b)
-----
```

```
#DivDemo.py---Main program
from Division import divop
from kvr import KvrDivisionError
try:
    a=int(input("Enter Value of a:"))# 100    100
    b=int(input("Enter Value of b:")) # 20      0
    res=divop(a,b) # Function call--Gives Result or Exception
except KvrDivisionError:
    print("DON'T ENTER ZERO FOR DEN...")
except ValueError:
```

```

        print("DON'T ENTER ALNUMS, STRS AND SYMBOLS")
else:
    print("Div({},{})={}".format(a,b,res))
=====

#MultabExcept.py--File Name and Module Name
class ZeroError(Exception):pass
class NegNumError(BaseException):pass
class HydValueError(Exception):pass
-----
=====

#MulTabDemo.py--main program
from MultabExcept import *
from MultTable import table
try:
    num=input("Enter a Number for generating Mul Table:")
    table(num) # Function Call--gives Result or exceptions
except ValueError:
    print("Don't Enter alnums,strs and symbols for numbers")
except ZeroError:
    print("Don't Enter Zero for Mul Table")
except NegNumError:
    print("Don't enter -Ve Number for Mul Table ")
except:
    print("OOOPS Some thing went wrong!!!")
finally:
    print("I am from finally Block")
=====

```

Files in Python-----4 days

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-Persistant Applications
- 2) Persistant Applications

=>In Non-Persistant Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form of objects, processed and whose results displayed on Moniter.

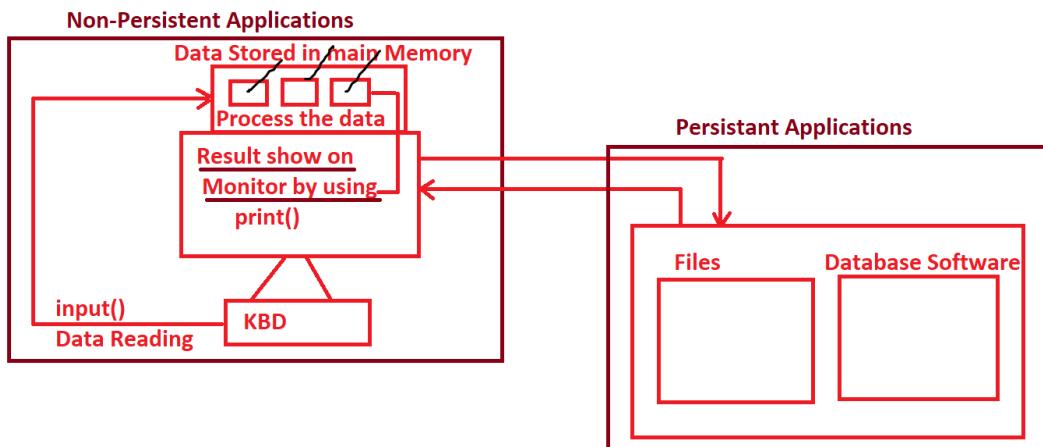
Examples: ALL our previous examples comes under Non-Persistant Applications.

=>We know that Data stored in Main Memory(RAM) is temporary.

=>In Persistant Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form of 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, MongoDB, DB2, PostgreSQL, SQL Server, SQLITE3..etc)



Data Persistency by Using 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:

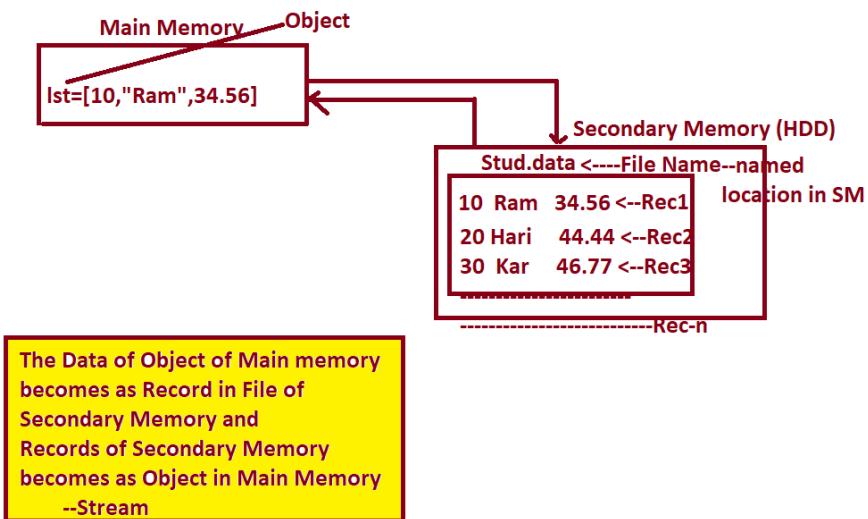
=>The purpose of read operation is that " To transfer or read the record from file of secondary memory into the object of main memory".

=>Steps

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

=>While we are performing read operations, we get the following exceptions.

- a) FileNotFoundException
- b) EOFError



===== Types of Files in Python =====

=>In any Programming lang, we have Two Types of Files. They are

1. Text Files
2. Binary Files

1. Text File

=>A Text File is One, which always contains alphabets, Digits and Special Symbols.

=>Text Files are denoted by a letter "t".

=>By deafult Python Programming Lang Treats every file as Text File.

Examples: .py .java .cpp .c
 .txt .xlsx, .doc...etc

2. Binary File

=>A Binary File is One, which contains the data in the form of Binary Format (Pixels).

=>Binary Files are denoted by a letter "b".

=>Examples

=> Images (.png, .jpeg,jpg, .gif....etc)
=> Audio and Video Files (.mvi, .avi....etc)
=> PDF File Images

File Opening Modes

=>The purpose of File Opening Modes is that "To Specify in which mode we are opening the File and Performing the File Operation".
=>In Python Programming, we have 8 types of 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 and we can Perform Read Operation only.

=>If open the File in "r" and if the File Name does not exist then we get FileNotFoundError

=>"r" Mode is one of the Default File Mode.

2. w

=>This Mode is used for Creating a File newly and On that File we can perform Write Operations only.

=>If we open New File in "w" mode then It will created newly and we can perform Write Operations.

=>If we open Existing File in "w" mode then It will Opened in write Mode and Existing data OVERLAPPED with new Data.

3. a

=>This Mode is used for Creating a File newly and On that File we can perform Write Operations.

=>If we open New File in "a" mode then It will created newly and we can perform Write Operations.

=>If we open Existing File in "a" mode then It will Opened in write Mode and Existing data

APPENDED with new Data.

4. r+

=>This Mode is used for Opening the File in Read Mode and First we can Perform Read Operation and additionally later we can perform Write Operation also.

=>If open the File in "r+" and if the File Name does not exist then we get FileNotFoundError

5. w+

=>This Mode is used for Creating a File newly and On that File we can perform Write Operations First and later we can also perform Read Operations also.

=>If we open New File in "w+" mode then It will created newly and we can perform Write Operations and later we can also perform Read Operations also.

=>If we open Existing File in "w+" mode then It will Opened in write Mode and Existing data OVERLAPPED with new Data.

6. a+

=>This Mode is used for Creating a File newly and On that File we can perform Write Operations first and later we can also perform Read Operation.

=>If we open New File in "a+" mode then It will created newly and we can perform Write Operations first and later we can also perform Read Operation..

=>If we open Existing File in "a+" mode then It will Opened in write Mode and Existing data APPENDED with new Data.

7. x

=>This Mode is used for Creating new file exclusively in write mode once and We can perform write operations only.

=>if we open Existing File in "x" mode then we get FileNotFoundError.

8. x+

=>This Mode is used for Creating new file exclusively in write mode once and We can perform write operations first and later we can perform read operations also.

=>if we open Existing File in "x+" mode then we get FileNotFoundError.

Syntax for Opening the Files

=>In Python Programming, we have Two approaches to Open the file. 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 a Valid Var name and It always points to the file and it is called File Pointer and whose type is <class, "io.TextIOWrapper">

=>open() is one of the pre-defined function belongs to "builtins" module and It used for opening the Specified File in Specified File Mode.

=>File Name Represents name of the File

=>File Mode Represents any one of the r,w,a,r+,w+,a+,x,x+

=>Once we open any file name with open() then we must close the File by using close() and It is mandatory for maintaining Consistency of Data(Manual Closing).

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

Explnation

=>"with" and "as" are the keywords

=>open() is a pre-defined function present in builtins module and It is used for Opening the Filename in Specified File Mode.

=>"FileName" represents Name of the file

=>" FileMode" represents r,w,a,r+,w+,a+,x,x+

=>Varname represents an object pointing the file and it is called File Pointer 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 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 close the file by using close() manually.

```
#FileOpenEx1.py
try:
    fp=open("kvr.data","r") # here fp is an object of
TextIOWrapper
except FileNotFoundError:
    print("File Does not exist")
else:
    print("-" * 50)
    print("Type of fp=",type(fp))
    print("File Opened in read mode sucessfully")
    print("-" * 50)
    print("Name of the File:",fp.name)
    print("File Mode:",fp.mode)
    print("Is File Readable:",fp.readable())
    print("Is File Writable:",fp.writable())
    print("Is File Closed:",fp.closed)
    print("-" * 50)
finally:
    print("\nI am from finally Block")
    fp.close()
    print("Is File Closed:", fp.closed)
```

```
#FileOpenEx2.py
fp=open("kvr.data","w")
print("-" * 50)
print("Type of fp=",type(fp))
print("File Created in write mode Sucessfully")
print("Type of fp=",type(fp))
print("-" * 50)
print("Name of the File:",fp.name)
print("File Mode:",fp.mode)
print("Is File Readable:",fp.readable())
print("Is File Writable:",fp.writable())
print("Is File Closed:",fp.closed)
print("-" * 50)
```

```
#FileOpenEx3.py
try:
    with open("kvr.data","r") as fp:
        print("-" * 50)
        print("Type of fp={}".format(type(fp)))
        print("File Opened in Read Mode Sucessfully")
```

```
    print("Name of the File:", fp.name)
    print("File Mode:", fp.mode)
    print("Is File Readable:", fp.readable())
    print("Is File Writable:", fp.writable())
    print("Is File Closed:", fp.closed)
    print("-" * 50)
    print("\nI am out off with-open() as Indentation")
    print("Is File Closed:", fp.closed)
except FileNotFoundError:
    print("File does not exist")
```

```
#FileOpenEx4.py
try:
    with open("kvr1.data","a+") as fp:
        print("-" * 50)
        print("Type of fp={}".format(type(fp)))
        print("File Opened in Write Sucessfully")
        print("Name of the File:", fp.name)
        print("File Mode:", fp.mode)
        print("Is File Readable:", fp.readable())
        print("Is File Writable:", fp.writable())
        print("Is File Closed:", fp.closed)
        print("-" * 50)
    print("\nI am out off with-open() as Indentation")
    print("Is File Closed:", fp.closed)
except FileNotFoundError:
    print("File does not exist")
```

```
#FileOpenEx5.py
try:
    with open("kvr2.data","x") as fp:
        print("-" * 50)
        print("Type of fp={}".format(type(fp)))
        print("File Opened in eXclusive created in Write mode
              sucessfully")
        print("Name of the File:", fp.name)
        print("File Mode:", fp.mode)
        print("Is File Readable:", fp.readable())
        print("Is File Writable:", fp.writable())
        print("Is File Closed:", fp.closed)
        print("-" * 50)
    print("\nI am out off with-open() as Indentation")
    print("Is File Closed:", fp.closed)
except FileExistsError:
    print("File already exist")
```

```
#FileOpenEx6.py
try:
    with open("kvr3.data","x+") as fp:
```

```

print("-" * 50)
print("Type of fp={}".format(type(fp)))
print("File Opened in eXclusive created in Write mode
      Sucessfully")
print("Name of the File:", fp.name)
print("File Mode:", fp.mode)
print("Is File Readable:", fp.readable())
print("Is File Writable:", fp.writable())
print("Is File Closed:", fp.closed)
print("-" * 50)
print("\nI am out off with-open() as Indentation")
print("Is File Closed:", fp.closed)
except FileExistsError:
    print("File already exist")
=====
Writing the Data to the File
=====
=>After Opening the File in any of the write modes, we must
write the data to the file.
=>To write the data to the file, we have 2 Pre-Defined Functions
present in TextIOWrapper. They are

```

- 1. write()
- 2. writelines()

1. write()

=>Syntax: varname=filepointerobj.write(strdata)

=>This Function is used for writing any type of data to the file
in the form of str.

2. writelines()

=>Syntax: varname=filepointerobj.writelines(Iterable-object)

=>This Function is used for writing any type of Iterable object
data to the file in the form of str.

NOTE: Here write() and writelines() will write any type of data
to the file in the form of Value by Value.

```
#FileWriteEx1.py
sno=300
sname="KINNEYMC"
subject="PANDAS"
marks=44.56
#perform write operations
with open("stud.data","a") as fp:
    fp.write(str(sno)+" ")
```

```

fp.write(sname+" ")
fp.write(subject+" ")
fp.write(str(marks)+" ")
fp.write("\n")
print("Data Written to the File")
=====
#FileWriteEx2.py
x={10:"Sujatha",20:"Swathi",30:"SriVidya"}
#write the above Iterable object to the file
with open("stud1.data","a") as fp:
    fp.writelines(str(x)+"\n")
    print("Data Written to the file")
=====
#DynamicDataWriteEx1.py
with open("C:\\NL\\PYTHON\\Hyd.data","a") as fp:
    print("Enter the Data and Press @ to stop:")
    while(True):
        kbdData=input()
        if(kbdData!="@"):
            fp.write(kbdData+"\n")
        else:
            print("Data Written to the File--verify:")
            break
=====

```

Reading the Data from the file

=>After Opening the File in Read Mode, we must read the data from File.

=>To Read the Data from the File, we have Two Pre-Defined Functions present in TextIOWrapper. They are

- 1. read()
- 2. readlines()

1. read()

=>Syntax: Varname=FilePointerObj.read()

=>This Function is used for Reading the Entire Data of the File and placed in LHS Variable and whose type is <class, 'str'>

2. readlines()

=>Syntax: Varname=FilePointerObj.readlines()

=>This Function is used for Reading the Entire Data of the File and placed in LHS Variable and whose type is <class, 'list'>

NOTE: Here read() and readlines() will read the data from the file in the form of Value by Value.

#FileReadEx1.py--read()

```

try:
    with open("stud.data","r") as fp:
        filedata=fp.read()
        print("-----")
        print("Content of File")
        print("-----")
        print(filedata)
        print("-----")
except FileNotFoundError:
    print("File Does not Exist")


---


#FileReadEx2.py--readlines()
try:
    with open("stud.data","r") as fp:
        filedata=fp.readlines()
        print("-----")
        print("Content of File")
        print("-----")
        for line in filedata:
            print(line,end="")
        print()
        print("-----")
except FileNotFoundError:
    print("File Does not Exist")


---


#This Program reads any file name and display its content
#FileContentDisp.py
filename=input("Enter File Name to display its content:")
try:
    with open(filename) as fp:
        filedata=fp.read()
        print("-----")
        print("content of File")
        print("-----")
        print(filedata)
        print("-----")
except FileNotFoundError:
    print("File does not exist")


---


#Copy the content of one into another file
#FileCopyEx1.py
srcfile=input("Enter Source File:")
try:
    with open(srcfile,"r") as rp:
        dstfile=input("Enter Destination File:")
        with open(dstfile,"a") as wp:
            srcdata=rp.read() #Reading the data from SCR FILE
            wp.write(srcdata) # writing to the dest file
            print("1 File Copied--Verify ")

```

```

except FileNotFoundError:
    print("Source File Does not Exist")


---


#Copy the Image of one into another file
#ImageFileCopyEx1.py
try:
    with open("D:\KVRAOP\hum.png", "rb") as rp:
        with open("pythonist.png", "wb") as wp:
            srcdata=rp.read()#Reading the data from SCR FILE
            wp.write(srcdata) # writing to the dest file
            print("Image Copied--Verify ")
except FileNotFoundError:
    print("Source File Does not Exist")


---


#Program for counting Number of Lines, Number of words and no.
of chars
#FileCountInformationEx.py
filename=input("Enter any File Name:")
try:
    nl,nw,nc=0,0,0
    with open(filename, "r") as fp:
        filedata=fp.readlines()
        for line in filedata:
            nl=nl+1
            nw=nw+len(line.split())
            nc=nc+len(line)
        else:
            if(nl==0):
                print("File is Empty")
            else:
                print("Number of Lines:{}".format(nl))
                print("Number of Words:{}".format(nw))
                print("Number of Chars:{}".format(nc))
except FileNotFoundError:
    print("File does not exist")


---


#RandomAccessFileEx1.py
#tell() give Index of File Pointer object
#seek() resets the file pointer object to the valid index in the
file data
with open("C:\\NL\\PYTHON\\hyd.data", "r") as fp:
    print("Initial Pos of FP={}".format(fp.tell()))
    filedata=fp.read(3)
    print("File Content:", filedata)
    print("Now Pos of FP={}".format(fp.tell()))
    print("-----")
    filedata = fp.read(11)
    print("File Content:", filedata)

```

```

print("Now Pos of FP={}".format(fp.tell()))
print("-----")
filedata = fp.read()
print("File Content:", filedata)
print("Now Pos of FP={}".format(fp.tell()))
print("-----")
#reset the file pointer to a particular Index
fp.seek(21)
print("-----")
filedata = fp.read(4)
print("File Content:", filedata)
print("Now Pos of FP={}".format(fp.tell()))
print("-----")
#reset the file pointer to a particular Index
fp.seek(0)
print("-----")
filedata = fp.read(14)
print("File Content:", filedata)
print("Now Pos of FP={}".format(fp.tell()))
print("-----")

```

**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 accepting student details save those student
details in file by using the concept of Pickling
#StudPickEx1.py
import pickle
def savestudercord():
    with open("stud.pick","ab") as fp:
        while(True):
            try:
                #accept the student details from KBD
                sno=int(input("Enter Student Number:"))
                sname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                #add student values to Iterable object
                lst=[]
                lst.append(sno)
                lst.append(sname)
                lst.append(marks)
                #Save the list obejct data into file
                pickle.dump(lst,fp)
                print("Student Data Saved in File Sucessfully")
                print("-"*50)
                ch=input("Do u want to enter another
                         record(yes/no):")
                if(ch.lower() == "no"):
                    print("Thx for this program")
                    break
            except ValueError:
                print("Don't Invalid Values for Student Number
                      and Marks")
#main program
savestudercord()

```

#Program for reading Student Records from File by using Un-Pickling concept

```

#StudUnPickEx1.py
import pickle
def readstudrecords():
    try:
        with open("stud.pick","rb") as fp:
            print("*"*50)
            while(True):
                try:
                    obj = pickle.load(fp)
                    for val in obj:
                        print("{}".format(val),end="\t")
                    print()
                except EOFError:
                    print("=* 50)

```

```

        break
    except FileNotFoundError:
        print("File Does not Exist:")
#main program
readstudrecords()


---


#MenuPickUnPick.py--File Name and Module Name
def menu():
    print("-"*50)
    print("Student Operations")
    print("-"*50)
    print("\t1. Pickling")
    print("\t2. Unpickling")
    print("\t3. Exit")
    print("-" * 50)
-----
#StudPickOperation.py---File and Module Name
import pickle
def savestudcord():
    with open("pythonstud.pick", "ab") as fp:
        while(True):
            try:
                #accept the student details from KBD
                sno=int(input("Enter Student Number:"))
                sname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                #add student values to Iterable object
                lst=[]
                lst.append(sno)
                lst.append(sname)
                lst.append(marks)
                #Save the list obejct data into file
                pickle.dump(lst,fp)
                print("Student Data Saved in File Sucessfully")
                print("-"*50)
                ch=input("Do u want to enter another
                         record(yes/no):")
                if(ch.lower() == "no"):
                    print("Thx for this program")
                    break
            except ValueError:
                print("Don't Invalid Values for Student Number
                      and Marks")
-----
#StudUnPickOperation.py--File Name and Module Name
import pickle
def readstudrecords():
    try:

```

```

        with open("pythononstud.pick", "rb") as fp:
            print("=" * 50)
            while (True):
                try:
                    obj = pickle.load(fp)
                    for val in obj:
                        print("{}".format(val), end="\t")
                    print()
                except EOFError:
                    print("=" * 50)
                    break
            except FileNotFoundError:
                print("File Does not Exist:")
-----
#PickUnPickDemo.py
from MenuPickUnPick import menu
from StudPickOperation import savestudercord
from StudUnPickOperation import readstudrecords
while(True):
    menu()
    try:
        ch=int(input("Enter Ur Choice:"))
        match(ch):
            case 1:
                savestudercord()
            case 2:
                readstudrecords()
            case 3:
                print('Thx for using this Program')
                break
            case _:
                print("Ur Selection of Operation is wrong--try again")
    except ValueError:
        print("Don't enter strs, alnums and symbols for choice")
-----
#Program for reading the data from CSV File
#CSVReadEx1.py--by using read()
try:
    with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\emp.csv") as fp:
        csvdata=fp.read()
        print(csvdata)
except FileNotFoundError:
    print("File does not exist")
-----
#Program for reading the data from CSV File
#CSVReadEx2.py--by using csv module reader()

```

```

import csv
try:
    with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\emp.csv") as
        fp:
            cr=csv.reader(fp)# Here cr is an object of
                            <class,'_csv.reader'>
            for record in cr:
                for val in record:
                    print("\t{}".format(val),end="")
            print()
except FileNotFoundError:
    print("File does not exist")


---


#SearchStudRecord.py--File Name and Module Name
import pickle
def searchstudrecords():
    try:
        sno=int(input("Enter Student Number for getting other
details student:"))
        with open("pythonstud.pick", "rb") as fp:
            print("=" * 50)
            lst=[] #
            while (True):
                try:
                    obj = pickle.load(fp)
                    if(obj[0]==sno):
                        lst.append(obj[0])
                        lst.append(obj[1])
                        lst.append(obj[2])
                        break
                except EOFError:
                    print("=" * 50)
                    break
            if(len(lst)>0):
                print("Student Number:{}".format(lst[0]))
                print("Student Name:{}".format(lst[1]))
                print("Student marks:{}".format(lst[2]))
            else:
                print("Student Record does not exist")

    except FileNotFoundError:
        print("File Does not Exist:")
    except ValueError:
        print("Dont enter strs,symbols and alnums for student
number")


---


#main program
searchstudrecords()

```

Working with CSV Files in Python

=>CSV stands for Comma Separated Values.
=>A CSV File is one of the simple file format used to store tabular data, such as a spreadsheet or database.
=>A CSV file stores tabular data (numbers and text) in plain text.
=>Each line of the CSV file is a data record. Each record consists of one or more fields, separated by commas.
=>Python provides an in-built module called csv to work with CSV files.
=>There are 2 classes provided by this module for writing the data to CSV File. They are

- 1) By using csv.writer class object
- 2) By Using csv.DictWriter class object

=>There are 2 classes provided by this module for Reading the data from CSV File. They are

- 1) By using csv.reader class object
 - 2) By Using csv.DictReader class object
-

1) By using csv.writer class object

=>The csv.writer class object is used to insert data to the CSV file.

=>To create an object of "csv.writer" class object, we use writer() and present in csv module.

=>"csv.writer" class object provides two Functions for writing to CSV file.

=>They are

- 1) writerow()
- 2) writerows()

1) writerow(): This 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.writerows(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)

#program for creating CSV File and writing the data Dynamically

#CSVWriteEx1.py

import csv

```

#step-1
hn=["eno","ename","sal","dsg"]
#step-2
records=[[100,"RS",3.4,"AUTHOR"],
          [200,"TR",4.5,"Scientist"],
          [300,"DR",1.5,"SE"],
          [400,"MC",2.5,"TL"]]
#step-3
with open("emp.csv","a") as fp:
    #Step-4
    csvwr=csv.writer(fp)
    #step-5--(a) and (b)
    csvwr.writerow(hn)
    csvwr.writerows(records)
    print("CSV File Created--Verify")


---


#program for adding record to existing CSV File
#CSVWriteEx2.py
import csv
#step-1
record=[500,"KV",0.0,"Triner"]
#step-2
with open("emp.csv","a") as fp:
    #Step-4
    csvwr=csv.writer(fp)
    #step-5--(b)
    csvwr.writerow(record)
    print("New Record added to CSV File --Verify")


---


#program for creating CSV File, Reading the data from KBD
# and writing the data Dynamically to CSV File
#CSVWriteEx3.py
import csv
print("Which Details u Want to Enter:")
details=input()
print("Enter How Many Header Names of '{}':".format(details))
noh=int(input())
print("Enter {} Col Names of '{}'".format(noh,details))
hn=[]
for i in range(1,noh+1):
    colname=input()
    hn.append(colname)
#get the records
records=[]
while(True):
    record = []
    for i in range(0,noh):
        vall=input("Enter Value of {}:".format(hn[i]))

```

```

        record.append(val1)
records.append(record)
ch=input("Do u want enter another record of
        '{}'.format(details))
if(ch.lower() == "no"):
    break
#Choose CSV File and Open in write Mode
csvfilename=input("Enter File Name with an extension .csv for
storing '{}' Details".format(details))
with open(csvfilename,"a") as fp:
    csvwr=csv.writer(fp)
    csvwr.writerow(hn)
    csvwr.writerows(records)
    print("CSV Created and Records written--verify")


---


#Program for reading the data from CSV File
#CSVReadEx1.py--by using read()
try:
    with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\emp.csv") as
fp:
    csvdata=fp.read()
    print(csvdata)
except FileNotFoundError:
    print("File does not exist")


---


#Program for reading the data from CSV File
#CSVReadEx2.py--by using csv module reader()
import csv
try:
    with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\emp.csv") as
fp:
        cr=csv.reader(fp)# Here cr is an object of
                           <class,'_csv.reader'>
        for record in cr:
            for val in record:
                print("\t{}".format(val),end="")
            print()
except FileNotFoundError:
    print("File does not exist")


---


#CSVReadEx3.py
import csv
try:
withopen("C:\\Users\\Kvr\\PycharmProjects\\9amfiles1\\book.csv",
"r") as fp:
    cr=csv.reader(fp)# Here cr is an object of
                      <class,'_csv.reader'>
    for record in cr:
        for val in record:

```

```

        print("\t{}".format(val),end="")
    print()
except FileNotFoundError:
    print("File does not exist")


---


#program for creating CSV File and writing the data Dynamically
#CSVDictWriteEx1.py
import csv
chn=["CID","NAME","COUNTRY"]
records=[{"CID":'1000',"NAME":"RS","COUNTRY":"NL"},  

        {"CID":'2000',"NAME":"KV","COUNTRY":"IND"},  

        {"CID":'3000',"NAME":"DR","COUNTRY":"USA"},  

        {"CID":'4000',"NAME":"MC","COUNTRY":"GERMANY"},  

        {"CID":'5000',"NAME":"TR","COUNTRY":"UK"}]
#Choose the CSV File and open in write mode
with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\citizen.csv","w")  

as fp:  

    csvdwr=csv.DictWriter(fp,fieldnames=chn)  

    csvdwr.writeheader()  

    csvdwr.writerows(records)  

    print("CSV File Created and Dict Data Written--verfiy")


---


#program for creating CSV File, Reading the data from KBD  

# and writing the data Dynamically to CSV File
#CSVDictWriteEx2.py
import csv
print("Which Details u Want to Enter:")
details=input()
print("Enter How Many Header Names of '{}':".format(details))
noh=int(input())
print("Enter {} Col Names of '{}'".format(noh,details))
hn=[]
for i in range(1,noh+1):
    colname=input()
    hn.append(colname)
#get the records
records=[]
while(True):
    record = dict() # Creating empty dict
    for i in range(0,noh):
        val1=input("Enter Value of {}:".format(hn[i]))
        record[hn[i]]=val1
    records.append(record)
    ch=input("Do u want enter another record of  

'{}'".format(details))
    if(ch.lower() == "no"):
        break
#Choose CSV File and Open in write Mode

```

```

csvfilename=input("Enter File Name with an extension .csv for
                  storing '{}' Details".format(details))
with open(csvfilename,"a") as fp:
    csvdwr=csv.DictWriter(fp,fieldnames=hn)
    csvdwr.writeheader()
    csvdwr.writerows(records)
    print("CSV Created and Records written--verify")

```

```

#CSVDictReadEx1.py
import csv
with
open("C:\\\\Users\\\\Kvr\\\\PycharmProjects\\\\9amfiles1\\\\book.csv") as
fp:
    csvdr=csv.DictReader(fp) # here csvdr is an object of
<class,csv.DictReader>
    for record in csvdr:
        print("-----")
        for k,v in record.items():
            print("\t{}\t{}".format(k,v))
        print("-----")

```

Working With OS Based Operations

=>In Python, "os" is one pre-defined module.
=>The purpose of os module is that "To perform some os related operations"
=>The Os based operations are

- 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 FileNotFoundError
=>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 \digits,\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 we specify any folder name with escape sequence ( \\n \\u
\\digits,\\t..etc) then we get OSError.
-----
```

```
#Program for Removing Folder / Directory
#rmdirsex.py
import os
```

```
try:  
    os.rmdir("D:\KVR")  
    print("Folder removed Successfully-verify")  
except FileNotFoundError:  
    print("folder name does not exist")  
except OSError:  
    print("rmdir() can remove those foilder which are empty--  
check ur path")
```

4) Removing Folders Hierarchy. (removedirs())

```
=>For Removing Removing Folders Hierarchy, we use removedirs()  
=>Syntax: os.removedirs("Folders Hierarchy")  
=>Here Folders Hierarchy represent Root Folder\sub folder\sub-  
sub folder so on...  
=>if the folder name not exist then we get FileNotFoundError  
=> if we specify any folder name with escape sequence ( \n \u  
\digits,\t..etc) then we get OSError.
```

Examples

```
#Program for Removing Folders Hierarchy  
#removedirsex.py  
import os  
try:  
    os.removedirs("D:\\India\\Hyd\\ampt\\python\\python")  
    print("Folders Hierarchy Removed Successfully-verify")  
except FileNotFoundError:  
    print("The specified folders hierachy does 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 Folder Name")
=>Syntax: os.rename("Old Folder Name", "New Folder Name")
=>If the Old Folder Name does not exist then we get
FileNotFoundException.

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
FileNotFoundException.

Examples:

```
#Program for Listing files in 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{}\n".format(filename))  
    print("-"*50)  
except FileNotFoundError:  
    print("Folder does not exist")
```

=====

Python Database Communication (PDBC) ---Most Imp---4 days

=====

=====

Python DataBase Communication (PDBC)

=====

=>Even we achieved the Data Persistency 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 depenedent)
4. Querying and Processing the data from Files is Very Complex bcoz file data is organized w.r.t Indices and idenfying 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 Persistency, we must use the concept of any RDBMS DataBase Softwares (Oracle, MYSQL, Mongo DB, DB2, SQL Server, Postgery 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 oranganized records in the form of Tables with Column Names.
5. The Data Present in any RDBMS DataBase Softwares oranganized 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.

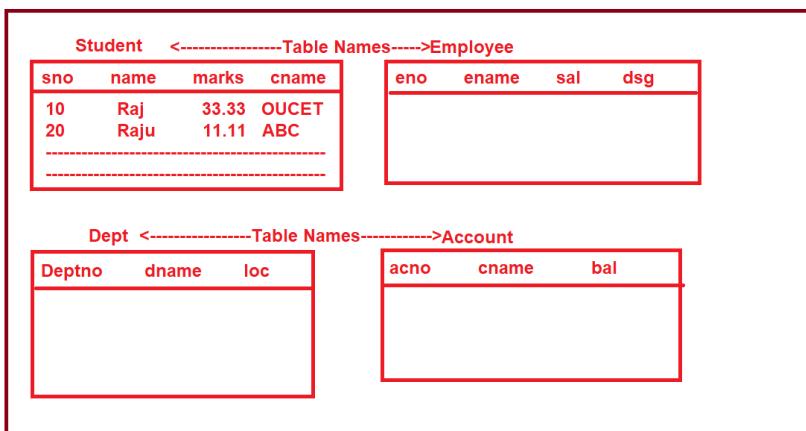
=>Examples : pip install cx_Oracle

=>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

Oracle Database <-----Collection of Tables <---Collection of Records



Steps for Developing Python Database Communication (PDBC) Applications / Programs

Steps

1. Import appropriate Database Module Name
2. Every Python Program must get the CONNECTION from Database Software.
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 Python Program must Must Process the Result, which is Received from Cursor object.

6. Every Python Program must Close the CONNECTION from Database Software.

=====

Communication between Python Program and Oracle Database

=====

Steps

1. Import appropriate Database Module Name
 2. Every Python Program must get the CONECTION from Database Software.
 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 Python Program must Must Process the Result, which is Received from Cursor object.
 6. Every Python Program must Close the CONECTION from Database Software.
- =====

Explanation

=====

Step-1: Import appropriate Database Module Name

=>If any Python Program want to communicate any database software, must import appropriate database module.

Examples: import cx_Oracle

=====

Step-2. Every Python Program must get the CONECTION from Database Software.

=>After Importing appropriate Database Module, Python Program must get the connection from database Software by using connect().

Syntax: varname=cx_Oracle.connect("Connection URL")

The Connection URL Represents :

UserName/Password@DNS/ServiceID

(OR)

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    <-----called Service ID
      -----
```

Examples:

```
conobj=cx_Oracle.connect("scott/tiger@localhost/xe")
(OR)
```

```
conobj=cx_Oracle.connect("scott/tiger@127.0.0.1/xe")
=>If we enter any Connection URL wrong then we get exception called "DatabaseError".
```

```
=====
Step-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 connection object.
```

```
=>Syntax: varname=conobj.cursor()
```

```
=>Here var name is an object of <class, cx_Oracle.Cursor>
```

```
=====
Step-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 Python Program.
```

```
=>To execute a Query, we have to use execute() and it is present in cursor object
```

```
=>Syntax: curobj.execute("Query")
```

```
=>In SQL, We have Different Types of Queries. They are DDL , DML and DRL
```

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 (sno number(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 droping the table from
Database Software:
=>Syntax: SQL> drop table tablename
=>Examples: SQL > drop table employee
=====
#Program for Demonstrating Connection from Oracle Database
#OracleConnTest1.py
import cx_Oracle
try:
    con=cx_Oracle.connect("system/manager@localhost/XE")
    print("Python Program got connection from Oracle database")
    print("Type of con=",type(con))
except cx_Oracle.DatabaseError as db:
    print("Problem in Data Base:",db)
=====
#Program for Demonstrating Connection from Oracle Database
#OracleConnTest2.py
import cx_Oracle
try:
    con=cx_Oracle.connect("system/manager@127.0.0.1/XE")
    print("Python Program got connection from Oracle database")
    print("Type of con=",type(con))
except cx_Oracle.DatabaseError as db:
    print("Problem in Data Base:",db)
=====
#Program for Demonstrating creating an object of Cursor
#OracleCursorObjEx1.py
import cx_Oracle # Step-2
try:
    con=cx_Oracle.connect("system/manager@localhost/XE") #Step-2
    print("Python Program got connection from Oracle database")
    cur=con.cursor() # Step-3
    print("Python Program created Cursor object:")
except cx_Oracle.DatabaseError as db:
    print("Problem in Oracle Database:",db)
=====
```

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 Perticular 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 creating a employee table

#OracleTableCreateEx1.py

import cx_Oracle

def createtable():

try:

con=cx_Oracle.connect("system/manager@localhost/xe")

cur=con.cursor()

#design the query, place it into cursor obj and execute

```

cq="create table employee(eno number(2) primary key,name
    varchar2(10) not null,sal number(5,2) not null)"
cur.execute(cq)
print("Table Created Sucessfully in Oracle Database-
    verify")
except cx_Oracle.DatabaseError as db:
    print("Problem in database:",db)
#main program
createtable()


---


#OracleTableCreateEx2.py
import cx_Oracle
def createtable():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        #design the query, place it into cursor obj and execute
        cq=input("Enter the Query for creating any table:\n")
        cur.execute(cq)
        print("Table Created Sucessfully in Oracle Database-
            verify")
    except cx_Oracle.DatabaseError as db:
        print("Problem in database:",db)
#main program
createtable()


---


#program for Inserting a record in employee table
#OracleRecordInsertEx1.py
import cx_Oracle
def emprecordinsert():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        # design the query, place it into cursor obj and execute
        iq="insert into employee values(30,'KV',0.0,'NIT')"
        cur.execute(iq)
        con.commit()
        print("Employee Record Inserted--Verify")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
emprecordinsert()


---


#program for Inserting a record in employee table
#OracleRecordInsertEx2.py
import cx_Oracle
def emprecordinsert():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")

```

```

        cur=con.cursor()
        #read the employee details from KBD
        print("-" * 50)
        empno=int(input("Enter employee Number:"))
        ename=input("Enter Employee Name:")
        sal=float(input("Enter Employee Salary:"))
        cname = input("Enter Employee Company Name:")
        print("-" * 50)
        # design the query, place it into cursor obj and execute
        iq="insert into employee values(%d,'%s',%f,'%s')"
        cur.execute(iq %(empno,ename,sal,cname))
        con.commit()
        print("{} Employee Record Inserted-
              Cerify".format(cur.rowcount))
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
emprecordinsert()


---


#program for Inserting a record in employee table
#OracleRecordInsertEx3.py
import cx_Oracle
def emprecordinsert():
    while(True):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            #read the employee details from KBD
            print("-" * 50)
            empno=int(input("Enter employee Number:"))
            ename=input("Enter Employee Name:")
            sal=float(input("Enter Employee Salary:"))
            cname = input("Enter Employee Company Name:")
            print("-" * 50)
            # design the query, place it into cursor obj and
            # execute
            iq="insert into employee values(%d,'%s',%f,'%s')"
            cur.execute(iq %(empno,ename,sal,cname))
            con.commit()
            print("{} Employee Record Inserted-
                  Cerify".format(cur.rowcount))
            print("-" * 50)
            ch=input("Do u want to another employee
                     record(yes/no):")
            if(ch.lower() == "no"):
                print("thx for using this program")
                break
        except cx_Oracle.DatabaseError as db:

```

```

        print("Problem in Database:",db)
    except ValueError:
        print("Don't enter alnums,symbols and strs for eno
              and sal")
    except:
        print("ooops some thing went wrong")

#main program
emprecordinsert()


---


#Program for adding new col name to the Employee Table
#OracleTableAlterAddEx1.py
import cx_Oracle
def altertableadd():
    try:
        con = cx_Oracle.connect("system/manager@127.0.0.1/xe")
        cur = con.cursor()
        # design the query, place it into cursor obj and execute
        aq="alter table employee add(cname varchar2(10) not
              null)"
        cur.execute(aq)
        print("Taqble altered-added Sucessfully in Oracle
              Database-verify")
    except cx_Oracle.DatabaseError as db:
        print("Problem in database:",db)
#main program
altertableadd()


---


#Program changing the col sizes of Employee Table
#OracleTableAlterModifyEx1.py
import cx_Oracle
def altertablemodify():
    try:
        con = cx_Oracle.connect("system/manager@127.0.0.1/xe")
        cur = con.cursor()
        # design the query, place it into cursor obj and execute
        aq="alter table employee modify(eno number(3),name
              varchar2(15),sal number(6,2))"
        cur.execute(aq)
        print("Taqble altered-modified Sucessfully in Oracle
              Database-verify")
    except cx_Oracle.DatabaseError as db:
        print("Problem in database:",db)
#main program
altertablemodify()


---


#Program for Removing the table
#OracleTableDropEx1.py
import cx_Oracle

```

```

def removetable():
    try:
        con = cx_Oracle.connect("system/manager@127.0.0.1/xe")
        cur = con.cursor()
        # design the query, place it into cursor obj and execute
        dq="drop table teacher"
        cur.execute(dq)
        print("Table dropped Sucessfully in Oracle Database-
              verify")
    except cx_Oracle.DatabaseError as db:
        print("Problem in database:",db)
#main program
removetable()
=====
```

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 for 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<= TotalNumber 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.

cur.execute("select * from employee")

Once the above query executed then cur object contains all the records of employee table.

ENO	NAME	SAL	CNAME
40	VK	1.2	TCS
50	NL	1.1	Wipro
70	MC	4.7	IBM
15	DR	5.6	Siemens
25	SK	4.5	TCS
55	SN	1.8	PayPal

To get the records from cur object, we have 3 pre-defined functions present in cursor object. They are

- 1) fetchone()
- 2) fetchmany(no.of recs)
- 3) fetchall()

```

#Program for Selecting Records from employee table
#OracleSelectRecordsEx1.py--fetchone()
import cx_Oracle
def readrecords():
    try:
        con = cx_Oracle.connect("system/manager@localhost/xe")
        cur = con.cursor()
        cur.execute("select * from employee")
        #get the records
        while(True):
            recs = cur.fetchone()
            if(recs!=None):
                for val in recs:
                    print("{}".format(val),end="\t")

```

```

        print()

    else:
        break

except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)
readrecords()


---


#Program for Selecting Records from employee table
#OracleSelectRecordsEx2.py--fetchmany()
import cx_Oracle
def readrecords():
    try:
        con = cx_Oracle.connect("system/manager@localhost/xe")
        cur = con.cursor()
        cur.execute("select * from employee")
        #get the records
        recs = cur.fetchmany(3)
        for rec in recs:
            for val in rec:
                print("{}".format(val),end="\t")
        print()

    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)

readrecords()


---


#Program for Selecting Records from employee table
#OracleSelectRecordsEx2.py--fetchall()
import cx_Oracle
def readrecords():
    try:
        con = cx_Oracle.connect("system/manager@localhost/xe")
        cur = con.cursor()
        cur.execute("select * from employee")
        #get the records
        recs = cur.fetchall()
        for rec in recs:
            for val in rec:
                print("{}".format(val),end="\t")
        print()

    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)

readrecords()

```

```

#Program for update a record based on employee number
#OracleUpdateRecordEx1.py
import cx_Oracle
def updaterecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("update employee set sal=sal+sal*20/100
                    where eno=15")
        con.commit()
        if(cur.rowcount>0):
            print("{} Employee Record
                  Updated:".format(cur.rowcount))
        else:
            print("Employee Records does not exist:")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
updaterecord()



---


#Program for update a record based on employee number
#OracleUpdateRecordEx2.py
import cx_Oracle
def updaterecord():
    while(True):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            #accept empno from KBD
            empno=int(input("Enter Employee Number for updating
                            other details:"))
            sal=float(input("Enter New Salary in New Company:"))
            cname=input("Enter Company Name:")
            cur.execute("update employee set sal=%f,cname='%s'
                        where eno=%d" %(sal,cname,empno))
            con.commit()
            if(cur.rowcount>0):
                print("{} Employee Record
                      Update:".format(cur.rowcount))
            else:
                print("Employee Records does not exist:")
            print("-"*50)
            ch=input("Do u want to Update another
                     record(yes/no):")
            if(ch.lower() == "no"):
                break
        except cx_Oracle.DatabaseError as db:
            print("Problem in Database:",db)

```

```

        except ValueError:
            print("Don't enter alnums,strs and symbols for
                  empno")
#main program
deleterecord()
=====
#Program for deleting a record based on employee number
#OracleDeleteRecordEx1.py
import cx_Oracle
def deleterecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        dq="delete from employee where eno=10"
        cur.execute(dq)
        con.commit()
        if(cur.rowcount>0):
            print("{} Employee Record
                  Deleted:".format(cur.rowcount))
        else:
            print("Employee Records does not exist:")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
deleterecord()
=====
#Program for deleting a record based on employee number
#OracleDeleteRecordEx1.py
import cx_Oracle
def deleterecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        dq="delete from employee where eno=10"
        cur.execute(dq)
        con.commit()
        if(cur.rowcount>0):
            print("{} Employee Record
                  Deleted:".format(cur.rowcount))
        else:
            print("Employee Records does not exist:")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
deleterecord()
=====
```

Communication between Python Program and MySQL Database

Steps

1. Import appropriate Database Module Name
 2. Every Python Program must get the CONECTION from Database Software.
 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 Python Program must Must Process the Result, which is Received from Cursor object.
 6. Every Python Program must Close the CONECTION from Database Software.
-

Explanation

Step-1: Import appropriate Database Module Name

=>If any Python Program want to communicate any database software, must import appropriate database module.

Examples: import mysql.connector

Step-2. Every Python Program must get the CONECTION from Database Software.

=>After Importing appropriate Database Module, Python Program must get the connection from database Software by using connect().

Syntax: varname=mysql.connector.connect(host="DNS/IP Address",
 user="User Name",
 passwd="password")

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 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)

Examples: conobj=mysql.connector.connect(host="localhost",
 user="root",
 passwd="root")

(OR)

```
conobj=mysql.connector.connect(host="127.0.0.1",
                                user="root",
                                passwd="root")
```

=>If we enter any Connection URL wrong then we get exception called "DatabaseError".

=====

Step-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 connection object.

=>Syntax: varname=conobj.cursor()

=>Here var name is an object of <class, mysql.connector.Cursor>

=====

Step-4: Every Python Program must Design the Query, place the query in Cursor object and execute.

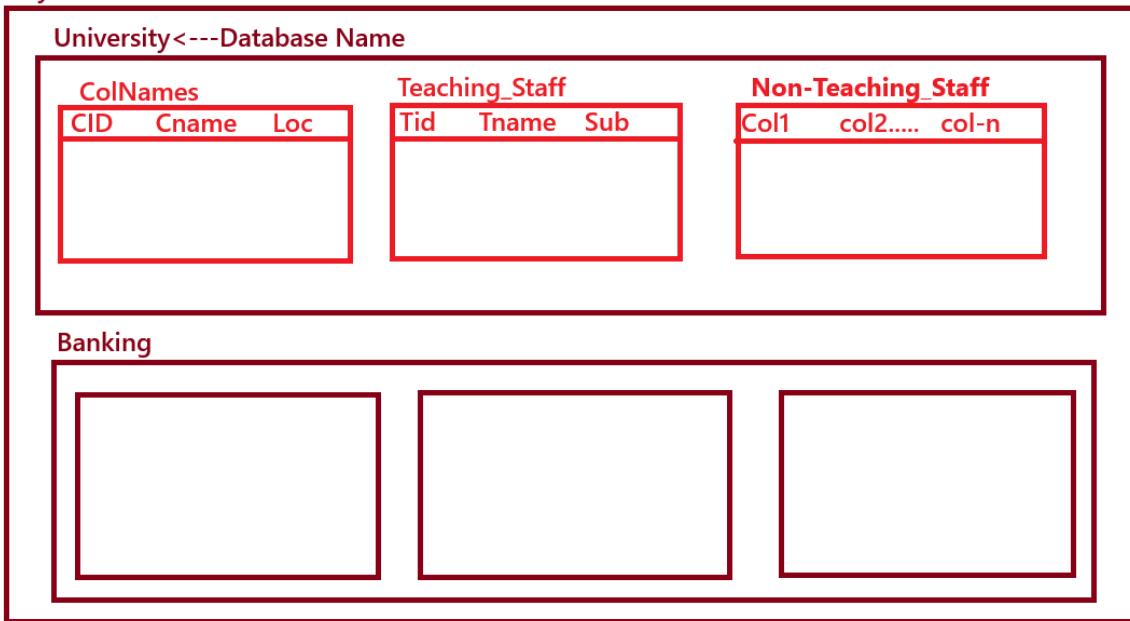
=>A Query a request / Question to the database from Python Program.

=>To execute a Query, we have to use execute() and it is present in cursor object

=>Syntax: curobj.execute("Query")

=>In SQL, We have Different Types of Queries. They are DDL , DML and DRL

MySQL Database<----Collection Databases<----Collection of Tables<----Collection of Records



```
#MySQLConnTestEx1.py
import mysql.connector
try:
    con=mysql.connector.connect(host="localhost",
                                user="root",
                                passwd="root1")
    print("Python Program got connection from MySQL DB")
except mysql.connector.DatabaseError as db:
    print("Problem in MySQL Database:",db)
```

```
#MySQLConnTestEx2.py
import mysql.connector
try:
    con=mysql.connector.connect(host="127.0.0.1",
                                user="root",
                                passwd="root")
    print("Python Program got connection from MySQL DB")
except mysql.connector.DatabaseError as db:
    print("Problem in MySQL Database:",db)
```

```
#MySQLCursorObjEx1.py
import mysql.connector
try:
    con=mysql.connector.connect(host="127.0.0.1",
                                user="root",
                                passwd="root")
    print("Python Program got connection from MySQL DB")
```

```

        cur=con.cursor()
        print("Python Program created Cursor object")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:",db)


---


#MySQLDatabaseCreateEx1.py
import mysql.connector
def databasecreate():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root")
        cur = con.cursor()
        dc="create database batch9am"
        cur.execute(dc)
        print("Database Created Sucessfully--verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:", db)


---


#main program
databasecreate()


---


#MySQLTableCreateEx1.py
import mysql.connector
def tablecrate():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur = con.cursor()
        tc="create table employee(eno int primary key, name
                                varchar(10) not null ,sal float not null)"
        cur.execute(tc)
        print("Table Created Sucessfully--verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:", db)


---


#main program
tablecrate()


---


#MySQLTableAlterEx1.py
import mysql.connector
def tablealter():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur = con.cursor()
        tc="alter table employee add(cname varchar(10)) "

```

```

        cur.execute(tc)
        print("Table altered Sucessfully--verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:", db)
#main program
tablealter()


---


#MySQLDatabaseDropEx1.py
import mysql.connector
def databasedrop():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                       user="root",
                                       passwd="root")
        cur = con.cursor()
        cur.execute("drop database batch4pm")
        print("Database droped Sucessfully--verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:", db)
#main program
databasedrop()


---


#program for Inserting a record in employee table
#MySQLRecordInsertEx.py
import mysql.connector
def emprecordinsert():
    while(True):
        try:
            con = mysql.connector.connect(host="127.0.0.1",
                                           user="root",
                                           passwd="root",
                                           database="batch9am")
            cur=con.cursor()
            #read the employee details from KBD
            print("-" * 50)
            empno=int(input("Enter employee Number:"))
            ename=input("Enter Employee Name:")
            sal=float(input("Enter Employee Salary:"))
            cname = input("Enter Employee Company Name:")
            print("-" * 50)
            # design the query, place it into cursor obj and
            # execute
            iq="insert into employee values(%d, '%s', %f, '%s')"
            cur.execute(iq %(empno,ename,sal,cname))
            con.commit()
            print("{} Employee Record Inserted-
                  Cerify".format(cur.rowcount))
            print("-" * 50)

```

```

        ch=input("Do u want to another employee
                  record(yes/no):")
        if(ch.lower() == "no"):
            print("thx for using this program")
            break
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL Database:",db)
    except ValueError:
        print("Don't enter alnums,symbols and strs for eno
              and sal")
    except:
        print("ooops some thing went wrong")
#main program
emprecordinsert()


---


#Program for update a record based on employee number
#MySQLRecordUpdateEx1.py
import mysql.connector
def updaterecord():
    while(True):
        try:
            con = mysql.connector.connect(host="127.0.0.1",
                                           user="root",
                                           passwd="root",
                                           database="batch9am")
            cur=con.cursor()
            #accept empno from KBD
            empno=int(input("Enter Employee Number for updating
                            other details:"))
            sal=float(input("Enter New Salary in New Company:"))
            cname=input("Enter Company Name:")
            cur.execute("update employee set sal=%f,cname='%s'
                        where eno=%d" %(sal,cname,empno))
            con.commit()
            if(cur.rowcount>0):
                print("{} Employee Record
                      Update:".format(cur.rowcount))
            else:
                print("Employee Records does not exist:")
            print("-"*50)
            ch=input("Do u want to Update another
                     record(yes/no):")
            if(ch.lower() == "no"):
                break
        except mysql.connector.DatabaseError as db:
            print("Problem in Database:",db)
        except ValueError:

```

```

        print("Don't enter alnums,strs and symbols for
              empno")

#main program
updaterecord()


---


#Program for Selecting Records from employee table
#MySQLSelectRecordsEx1.py
import mysql.connector
def readrecords():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur = con.cursor()
        cur.execute("select * from employee")
        #get the records
        recs = cur.fetchall()
        for rec in recs:
            for val in rec:
                print("{}".format(val),end="\t")
        print()

    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)
readrecords()


---


#MySQLSelectRecordsTableName.py
import mysql.connector
def readrecords():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur = con.cursor()
        cur.execute("select * from %s" %input("Enter Any Table
                                              Name:"))
        # Code for obtaining Col Names
        colnames = cur.description
        print("=" * 50)
        for colname in colnames:
            print(colname[0], end="\t")
        print()
        print("=" * 50)
        #get the records
        recs = cur.fetchall()
        for rec in recs:

```

```

        for val in rec:
            print("{}".format(val), end="\t")
        print()
        print("=" * 50)
    except mysql.connector.DatabaseError:
        print("In My DataBase Such Table does not exist" )
readrecords()

#Program for Selecting Records from employee table
#MySQLSelectRecordsWithColnamesEx1.py
import mysql.connector
def readrecords():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                       user="root",
                                       passwd="root",
                                       database="batch9am")
        cur = con.cursor()
        cur.execute("select * from student")
        # Code for obtaining Col Names
        colnames = cur.description
        print("=" * 50)
        for colname in colnames:
            print(colname[0], end="\t")
        print()
        print("=" * 50)
        #get the records
        recs = cur.fetchall()
        for rec in recs:
            for val in rec:
                print("{}".format(val), end="\t")
            print()
        print("=" * 50)
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)
readrecords()
=====
```

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.

```

```

#Non-DecEx1.py
def getval(): # This Function Defined by KVR
    return int(input("Enter Any Value:"))
def square(): # A student Mishra want to do square of the value
    given getval():
        n=getval()
        return (n**2)
def squareroot(): # A student Sumanth want to do square root of
    the value given getval()
    n=getval()
    return (n**0.5)
def cube(): # A student Naiyak want to do cube of the value
    given getval()
    n=getval()
    return (n**3)
#main program
print("Square =",square())
print("Square Root =",squareroot())
print("Cube Root =",cube())

```

```

#DecEx1.py
def getval(): # This Function Defined by KVR
    return int(input("Enter Any Number:"))
def square(gv): # here Square is called Decorator(Outer
    Function)
    def operation(): # Here operation is called Inner Function
        n=gv()
        res=n**2
        return res
    return operation
#main program
x=square(getval) # Here square() is one of the decorator .
result=x()
print('Square=',result)

```

```
#DecEx2.py
```

```

def square(gv): # here Square is called Decorator (Outer
Function)
    def operation(): # Here operation is called Inner Function
        n=gv()
        res=n**2
        return res
    return operation
@square
def getval(): # This Function Defined by KVR
    return int(input("Enter Any Number:"))
#main program
result=getval()
print("Square=",result)


---


#DecEx3.py
def getval():
    return float(input("Enter any number:"))
def cube(kvr): # Here cube is called Decorator (Outer Function )
    def cubeop(): # Here cubeop is called Inner Function
        v=kvr()
        res=v**3
        return res
    return cubeop
#main program
cop=cube(getval) # here cube is called Decorator
print("type of cop=",type(cop))
result=cop()
print("Cube=",result)


---


#DecEx4.py
def cube(kvr): # Here cube is called Decorator (Outer Function )
    def cubeop(): # Here cubeop is called Inner Function
        v=kvr()
        res=v**3
        return v,res
    return cubeop
@cube
def getval():
    return float(input("Enter any number:"))
#main program
v,cv=getval()
print("Cube({})={}".format(v,cv))


---


#DecEx5.py
def lenwords(gw):
    def calculate():
        d={}
        words=gw()
        for val in words:

```

```

        d[val]=len(val)
        return d
    return calculate

def getwords(gl):
    def operation():
        line=gl()
        words=line.split()
        return words
    return operation
@lenwords
@getwords
def getline():
    return (input("Enter a Line of Text:"))
#main program
wordslength=getline()
print(wordslength)
=====
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.
-----
def kvrrange(beg,end):
    while(beg<=end):
        yield beg

```

```

beg=beg+1
#main program
g=kvrrange(1,5) # Function Call
print("type of kvrrange={}, type of
g={}".format(type(kvrrange),type(g)))
#get the value from generator
print(g)
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))
#print(next(g)) # --StopIteration
=====
#GenEx2.py
def kvrrange(beg,end,step):
    while(beg<=end):
        yield beg
        beg=beg+step
#main program
g=kvrrange(10,20,2) # Function Call
for val in g:
    print(val)
print("====")
g=kvrrange(100,200,20) # Function Call
while(True):
    try:
        print(next(g))
    except StopIteration:
        break
=====
```

Object Oriented Principles or Features or Concepts (10 days)

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 get Effective Memory
 5. Provides Minimal Memory space for application development bcoz OOPs Provides Re-usable Mechanisms.
-

Object Oriented Programming Principles (OOPs)

=>In order to say a Programming lang is Object Oriented If and only if It can satisfy the following Object Oriented Principles.

1. Classes
 2. Objects
 3. Data Encapsulation
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing (already discussed)
-

1. Classes

=>The purpose of Classes Concept is that "To Develop Programmer-Defined Data Type + To develop any Real Time Application"

=>The Purpose of developing Programmer-Defined Data type is that "To Store Customized Data and to Perform Customized Operations"

=>In Python Programming, To develop Programmer-Defined Data type, we use Classes Concept by using 'class' keyword

=>Programmatically, Every Class Name is Treated as Programmer-Defined Data type.

=>All Types of OOPs Based Applications / Programs Must starts with Classes Concept.

Definition of Class

=>A Class is a Collection of Data Members (Instance Data Members and Class Level Data Members) and Methods (Instance Method, Class Level Method and Static Method)

=>When we define a Class , Memory Space is Not Created for Data Members and Methods But whose memory space is Created when we create an Object.

Syntax for Defining the Class in Python

```
class <ClsName>:  
    Class Level Data Members  
    def  instancemethodname(self,List of formal  
                           Parameters if any):  
        -----  
        Block of statements--Performs Specific  
        Operation  
        -----  
        @classmethod  
        def  classlevelmethodname(cls,List of formal  
                               Parameters if any):  
            -----  
            Block of statements--Performs Common  
            Operation  
            Specify--Class Level Data Members  
            -----  
            @staticmethod  
            def  staticmethodname(List of formal  
                               Parameters if any):  
                -----  
                Block of statements--Performs Utility  
                / Universal Operations  
                -----
```

Types of Data Members in a Class of Python

=>In a Class of Python, we can Define Two Types of Data Members. They are

1. Instance Data Members
2. Class Level Data Members

1. Instance Data Members

=>Instance Data Members are those , whose memory space created Each and Every Time when we create an object and hence Instance Data Members are called Object Level Data Members.

=>Instance Data Members are used to store Specific / Particular Data OR Objects contains Specific Data OR Instance Data.

=>Instance Data Members can be stored in Object in 3 ways

- Through Object name
- Through Instance Method Name
- Through Constructor

=>Instance Data Members must be accessed in 2 ways. They are

- By using object Name--->Syntax---
>ObjName.Instance Data Member name
- By using self----->Syntax---
>self.Instance Data Member name

```
#Program for storing sno,sname,marks by using OOPs
#InstanceDataMembersEx1.py
class Student:pass
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno,sname and marks are called Instance Data Members
s1.sno=100
s1.sname="RS"
s1.marks=11.11
#Place the Instance Data in s2
s2.sno=200
s2.sname="TR"
s2.marks=22.22
print("Now Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Now Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
```

```
#Program for storing sno,sname,marks by using OOPs
#InstanceDataMembersEx2.py
class Student:pass
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
```

```

print("-----")
#Place the Instance Data in s1
#Here sno, sname and marks are called Instance Data Members
s1.sno=100
s1.sname="RS"
s1.marks=11.11
#Place the Instance Data in s2
s2.sno=200
s2.sname="TR"
s2.marks=22.22
print("First Student Data")
print("-----")
print("Student Number:{}".format(s1.sno))
print("Student Name:{}".format(s1.sname))
print("Student Marks:{}".format(s1.marks))
print("-----")
print("Second Student Data")
print("-----")
print("Student Number:{}".format(s2.sno))
print("Student Name:{}".format(s2.sname))
print("Student Marks:{}".format(s2.marks))
print("-----")


---


#InstanceDataMembersEx3.py
#Program for storing sno, sname, marks by using OOPs
class Student:pass
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno, sname and marks are called Instance Data Members
s1.sno=100
s1.sname="RS"
s1.marks=11.11
#Place the Instance Data in s2
s2.sno=200
s2.sname="TR"
s2.marks=22.22
print("First Student Data")
print("-----")
for dmn,dmv in s1.__dict__.items():
    print("\t{}\t{}".format(dmn,dmv))
print("-----")

```

```

print("Second Student Data")
print("-----")
for dmn,dmv in s2.__dict__.items():
    print("\t{}\t{}".format(dmn,dmv))
print("-----")
#InstanceDataMembersEx4.py
class Student:pass
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno,sname and marks are called Instance Data Members
s1.sno=int(input("Enter First Student Number:"))
s1.sname=input("Enter First Student Name:")
s1.marks=float(input("Enter First Student Marks:"))
print("-----")
#Place the Instance Data in s2
s2.sno=int(input("Enter Second Student Number:"))
s2.sname=input("Enter Second Student Name:")
s2.marks=float(input("Enter Second Student Marks:"))
print("First Student Data")
print("-----")
print("Student Number:{}".format(s1.sno))
print("Student Name:{}".format(s1.sname))
print("Student Marks:{}".format(s1.marks))
print("-----")
print("Second Student Data")
print("-----")
print("Student Number:{}".format(s2.sno))
print("Student Name:{}".format(s2.sname))
print("Student Marks:{}".format(s2.marks))
print("-----")
=====
```

2. Class Level Data Members

- =>Class Level Data Members are those , whose memory space created Only Once for all the objects of Same Class.
- =>Class Level Data Members are used to store Common Data for all Objects of Same Class
- =>Class Level Data Members can Stored in Object in 2 ways
 - a) Inside of Class Name
 - b) Through Class Level Method Name

=>Class Level Data Members Must be accessed in 4 ways. They are

- a) By using Class Name--->Syntax--->ClsName.Class
Level Data Member name
- b) By using cls----->Syntax---->cls. Class
Level Data Member name
- c) By using object Name--->Syntax--->ObjName.Class
Level Data Member name
- d) By using self----->Syntax---
>self.Class Level Data Member name

```
#Program for storing sno,sname,marks and colname by using OOPs
#Here sno,sname and marks are called Instance Data Members
#Here colname is called Class Level Data Member bcoz ColName is
Common.
#ClassLevelDataMemberEx1.py
class Student:
    colname="OU" # Here colname is called Class Level Data
                  Member
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno,sname and marks are called Instance Data Members
s1.sno=100
s1.sname="RS"
s1.marks=11.11
#Place the Instance Data in s2
s2.sno=200
s2.sname="TR"
s2.marks=22.22
print("First Student Data")
print("-----")
print("Student Number:{}".format(s1.sno))
print("Student Name:{}".format(s1.sname))
print("Student Marks:{}".format(s1.marks))
print("Student Col Name:{}".format(Student.colname))#access the
class level Data member by using Class Name
print("-----")
print("Second Student Data")
print("-----")
print("Student Number:{}".format(s2.sno))
print("Student Name:{}".format(s2.sname))
print("Student Marks:{}".format(s2.marks))
```

```

print("Student Col Name:{}".format(Student.colname))#access the
          class level Data member by using Class Name
print("-----")


---


#Program for storing sno,sname,marks and colname by using OOPs
#Here sno,sname and marks are called Instance Data Memebers
#Here colname is called Class Level Data Member bcoz ColName is
Common.
#ClassLevelDataMemberEx2.py
class Student:
    colname="OU" # Here colname is called Class Level Data
                  Member
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno,sname and marks are called Instance Data Members
s1.sno=100
s1.sname="RS"
s1.marks=11.11
#Place the Instance Data in s2
s2.sno=200
s2.sname="TR"
s2.marks=22.22
print("First Student Data")
print("-----")
print("Student Number:{}".format(s1.sno))
print("Student Name:{}".format(s1.sname))
print("Student Marks:{}".format(s1.marks))
print("Student Col Name:{}".format(s1.colname))#access the class
          level Data member by using object name
print("-----")
print("Second Student Data")
print("-----")
print("Student Number:{}".format(s2.sno))
print("Student Name:{}".format(s2.sname))
print("Student Marks:{}".format(s2.marks))
print("Student Col Name:{}".format(s2.colname))#access the class
          level Data member by using object name
print("-----")


---


#Program for storing sno,sname,marks and colname by using OOPs
#Here sno,sname and marks are called Instance Data Memebers

```

```

#Here colname is called Class Level Data Member bcoz ColName is
Common.
#ClassLevelDataMemberEx3.py
class Student:
    colname="OU" # Here colname is called Class Level Data
                  Member
#Main Program
s1=Student() # Here s1 is called Object
s2=Student() # Here s2 is called Object
print("Initial Content of s1=",s1.__dict__)
print("Id Of s1=",id(s1))
print("Initial Content of s2=",s2.__dict__)
print("Id Of s2=",id(s2))
print("-----")
#Place the Instance Data in s1
#Here sno,sname and marks are called Instance Data Members
s1.sno=int(input("Enter First Student Number:"))
s1.sname=input("Enter First Student Name:")
s1.marks=float(input("Enter First Student Marks:"))
print("-----")
#Place the Instance Data in s2
s2.sno=int(input("Enter Second Student Number:"))
s2.sname=input("Enter Second Student Name:")
s2.marks=float(input("Enter Second Student Marks:"))
print("First Student Data")
print("-----")
print("Student Number:{}".format(s1.sno))
print("Student Name:{}".format(s1.sname))
print("Student Marks:{}".format(s1.marks))
print("Student Col Name:{}".format(s1.colname))#access the class
level Data member by using object name
print("-----")
print("Second Student Data")
print("-----")
print("Student Number:{}".format(s2.sno))
print("Student Name:{}".format(s2.sname))
print("Student Marks:{}".format(s2.marks))
print("Student Col Name:{}".format(s2.colname))#access the class
level Data member by using object name
print("-----")
=====
```

Types of Methods in a Class of Python

=>In a Class of Python, we can Define 3 Types of Methods. They are

1. Instance Method Name

2. Class Level Method Name
 3. Static Method Name
-

1. Instance Method Name

=>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 Name()
(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 not in Other Part of program bcoz It is a First Possitonal Formal Parameter .

2. Class Level Method Name

=>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 accessed 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 Method Name

=>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 storing Sno,sname and Marks by using Classes and
objects
#InstanceMethodEx1.py
class Student:
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("-----")

#main program
s1=Student()
s2=Student()
#Read the Data for s1 Object
print("Enter First Student Details")
s1.readstuddata()
print("Enter Second Student Details")
s2.readstuddata()
print("First Student Details")
s1.dispstuddata()
print("Second Student Details")
s2.dispstuddata()

```

```

#Program for storing Sno,sname and Marks by using Classes and
objects
#InstanceMethodEx2.py
class Student:
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("-----")

#main program

```

```

s1=Student()
s2=Student()
#Read the Data for s1 Object
print("Enter First Student Details")
s1.readstuddata()
print("First Student Details")
s1.dispstuddata()
print("Enter Second Student Details")
s2.readstuddata()
print("Second Student Details")
s2.dispstuddata()


---


#Program for storing Sno,sname and Marks by using Classes and
objects
#InstanceMethodEx3.py
class Student:
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")
        self.dispstuddata() # Calling the Instance Method by
                            using self
    def dispstuddata(self):
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("-----")
#main program
s1=Student()
s2=Student()
#Read the Data for s1 Object
print("Enter First Student Details")
s1.readstuddata()
print("Enter Second Student Details")
s2.readstuddata()
print("Second Student Details")
s2.dispstuddata()


---


#Program for Calculating Area of Circle by using Classes and
Obejcts
#ClassLevelMethodEx1.py
class Circle:
    @classmethod
    def getPI(cls): # Class Level Method
        cls.PI=3.14 # Class Level Data Member

```

```

def readrad(self):
    self.r=float(input("Enter Radious:"))
def area(self):
    self.readrad()
    self.ac=Circle.PI*self.r**2
    self.disparea()
def perimeter(self):
    self.readrad()
    self.pc=2*Circle.PI*self.r
    self.disppermeter()
def disparea(self):
    print("Radious={}".format(self.r))
    print("Area of Circle:{}".format(self.ac))
def disppermeter(self):
    print("Radious={}".format(self.r))
    print("Perimeter of Circle:{}".format(self.pc))
#main program
Circle.getPI() # Calling Class Level Method
c=Circle()
c.area()
print("-----")
c.perimeter()


---


#ClassLevelMethodEx2.py
class Student:
    @classmethod
    def getunivdet(cls):
        cls.uname="OU"
        Student.loc="HYD"
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("Student University Name:{}".format(self.uname))
        print("Student University Location:{}".format(self.loc))
        print("-----")
#main program
Student.getunivdet() # Calling Class Level Method w.r.t Class
                      Name
s1=Student()

```

```

s1.readstuddata()
s1.dispstuddata()


---


#ClassLevelMethodEx3.py
class Student:
    @classmethod
    def getunivdet(cls):
        cls.uname="OU"
        Student.loc="HYD"
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("Student University Name:{}".format(self.uname))
        print("Student University Location:{}".format(self.loc))
        print("-----")

#main program
s1=Student()
s1.getunivdet() # Calling Class Level Method w.r.t Object Name
s1.readstuddata()
s1.dispstuddata()


---


#ClassLevelMethodEx4.py
class Student:
    @classmethod
    def getunivdet(cls):
        cls.uname="OU"
        Student.loc="HYD"
    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        self.getunivdet() # Calling Class Level Method w.r.t
                           self
        print("-----")
        print("Student Number:{}".format(self.sno))

```

```

        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("Student University Name:{}".format(self.uname))
        print("Student University Location:{}".format(self.loc))
        print("-----")
#main program
s1=Student()
s1.readstuddata()
s1.dispstuddata()


---


#ClassLevelMethodEx5.py
class Student:
    @classmethod
    def getunivdet(cls):
        cls.uname="OU"
        cls.getunivloc() # Calling Class Level Method w.r.t cls
    @classmethod
    def getunivloc(cls):
        cls.loc="HYD"

    def readstuddata(self):
        print("-----")
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-----")

    def dispstuddata(self):
        self.getunivdet() # Calling Class Level Method w.r.t
                           self
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("Student University Name:{}".format(self.uname))
        print("Student University Location:{}".format(self.loc))
        print("-----")
#main program
s1=Student()
s1.readstuddata()
s1.dispstuddata()


---


#Program for Demonstrating Static Method
#StaticMethodEx1.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")

```

```

        self.sal=float(input("Enter Employee Salary:"))
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")
class Teacher:
    def getteadet(self):
        self.tno=int(input("Enter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.sub=input("Enter Teacher Subject:")
        self.exp=float(input("Enter Teacher Experience:"))
class Hyd:
    @staticmethod
    def dispobjdata(obj,objinfo):
        print("-" * 50)
        print("{} Deatils".format(objinfo))
        print("-" * 50)
        for dmn,dmv in obj.__dict__.items():
            print("\t{}\t{}".format(dmn,dmv))
        print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")
Hyd.dispobjdata(e,"Employee") # Calling Static Method w.r.t
                                Class Name
Hyd.dispobjdata(s,"Student") # Calling Static Method w.r.t Class
                                Name
Hyd.dispobjdata(t,"Teacher") # Calling Static Method w.r.t Class
                                Name


---


#Program for Demonstrating Static Method
#StaticMethodEx2.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")

```

```

class Teacher:
    def getteadet(self):
        self.tno=int(input("Enter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.sub=input("Enter Teacher Subject:")
        self.exp=float(input("Enter Teacher Experience:"))
class Hyd:
    @staticmethod
    def dispobjdata(obj,objinfo):
        print("-" * 50)
        print("{} Deatils".format(objinfo))
        print("-" * 50)
        for dmn,dmv in obj.__dict__.items():
            print("\t{}\t{}".format(dmn,dmv))
        print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")
h=Hyd()
h.dispobjdata(e,"Employee") # Calling Static Method w.r.t Object
                             Name
h.dispobjdata(s,"Student") # Calling Static Method w.r.t Object
                           Name
h.dispobjdata(t,"Teacher") # Calling Static Method w.r.t Object
                           Name

```

```

#Program for Demonstrating Static Method
#StaticMethodEx3.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")
class Teacher:
    def getteadet(self):
        self.tno=int(input("Enter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")

```

```

        self.sub=input("Enter Teacher Subject:")
        self.exp=float(input("Enter Teacher Experience:"))
class Hyd:
    @classmethod
    def getinfo(cls,obj,objinfo):
        Hyd.dispobjdata(obj,objinfo)# calling Static Method from
Class Level Method w.r.t class name

    @staticmethod
    def dispobjdata(obj,objinfo):
        print("-"*50)
        print("{} Deatils".format(objinfo))
        print("-" * 50)
        for dmn,dmv in obj.__dict__.items():
            print("\t{}\t{}".format(dmn,dmv))
        print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")
Hyd.dispobjdata(e,"Employee") # Calling Class Level Method w.r.t
                                Class Name
Hyd.dispobjdata(s,"Student") # Calling Class Level Method w.r.t
                                Class Name
Hyd.dispobjdata(t,"Teacher") # Calling Class Level Method w.r.t
                                Class Name

```

```

#Program for Demonstrating Static Method
#StaticMethodEx4.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")
class Teacher:
    def getteadet(self):
        self.tno=int(input("Enter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")

```

```

        self.sub=input("Enter Teacher Subject:")
        self.exp=float(input("Enter Teacher Experience:"))
class Hyd:
    @classmethod
    def getinfo(cls,obj,objinfo):
        cls.dispobjdata(obj,objinfo)# calling Static Method from
Class Level Method w.r.t cls

    @staticmethod
    def dispobjdata(obj,objinfo):
        print("-"*50)
        print("{} Deatils".format(objinfo))
        print("-" * 50)
        for dmn,dmv in obj.__dict__.items():
            print("\t{}\t{}".format(dmn,dmv))
        print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")
Hyd.dispobjdata(e,"Employee") #Calling Class Level Method w.r.t
                                Class Name
Hyd.dispobjdata(s,"Student") # Calling Class Level Method w.r.t
                                Class Name
Hyd.dispobjdata(t,"Teacher") # Calling Class Level Method w.r.t
                                Class Name

```

```

#Program for Demonstrating Static Method
#StaticMethodEx5.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")
class Teacher:
    def getteadet(self):
        self.tno=int(input("Enter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")

```

```

        self.sub=input("Enter Teacher Subject:")
        self.exp=float(input("Enter Teacher Experience:"))
class Hyd:
    def getinfo(self,obj,objinfo):
        self.dispobjdata(obj,objinfo)# calling Static Method
from Instance Method w.r.t self

    @staticmethod
    def dispobjdata(obj,objinfo):
        print("-"*50)
        print("{} Deatils".format(objinfo))
        print("-" * 50)
        for dmn,dmv in obj.__dict__.items():
            print("\t{}\t{}".format(dmn,dmv))
        print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")
h=Hyd()
h.dispobjdata(e,"Employee")#Calling Class Level Method w.r.t
                           Class Name
h.dispobjdata(s,"Student") #Calling Class Level Method w.r.t
                           Class Name
h.dispobjdata(t,"Teacher") #Calling Class Level Method w.r.t
                           Class Name

```

```

#Program for Demonstrating Static Method
#StaticMethodEx6.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
        Hyd.dispobjdata(self,"Employee")

class Student:
    def getstuddet(self):
        self.sno=int(input("Enter student Number:"))
        self.sname=input("Enter Student Name:")
        Hyd.dispobjdata(self, "Student")
class Teacher:

```

```

        def getteadet(self):
            self.tno=int(input("Enter Teacher Number:"))
            self.tname=input("Enter Teacher Name:")
            self.sub=input("Enter Teacher Subject:")
            self.exp=float(input("Enter Teacher Experience:"))
            Hyd.dispobjdata(self, "Teacher")
    class Hyd:
        @staticmethod
        def dispobjdata(obj,objinfo):
            print("-" * 50)
            print("{} Deatils".format(objinfo))
            print("-" * 50)
            for dmn,dmv in obj.__dict__.items():
                print("\t{}\t{}".format(dmn,dmv))
            print("-" * 50)
    #main program
    e=Employee()
    s=Student()
    t=Teacher()
    e.getempdet()
    print("-----")
    s.getstuddet()
    print("-----")
    t.getteadet()
    print("-----")


---


Program for Demonstrating Static Method
#StaticMethodEx7.py
class Employee:
    def getempdet(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
        Hyd().dispobjdata(self,"Employee")

    class Student:
        def getstuddet(self):
            self.sno=int(input("Enter student Number:"))
            self.sname=input("Enter Student Name:")
            Hyd().dispobjdata(self, "Student")
    class Teacher:
        def getteadet(self):
            self.tno=int(input("Enter Teacher Number:"))
            self.tname=input("Enter Teacher Name:")
            self.sub=input("Enter Teacher Subject:")
            self.exp=float(input("Enter Teacher Experience:"))
            Hyd().dispobjdata(self, "Teacher")
    class Hyd:

```

```

@staticmethod
def dispobjdata(obj,objinfo):
    print("-" * 50)
    print("{} Deatils".format(objinfo))
    print("-" * 50)
    for dmn,dmv in obj.__dict__.items():
        print("\t{}\t{}".format(dmn,dmv))
    print("-" * 50)
#main program
e=Employee()
s=Student()
t=Teacher()
e.getempdet()
print("-----")
s.getstuddet()
print("-----")
t.getteadet()
print("-----")


---


#Program for Cal sum of Two Numbers by using Classes and objects
#SumOperationInstanceMethodEx1.py
class Sum:
    def readvalues(self):
        self.a=float(input("Enter First Value:"))
        self.b=float(input("Enter Second Value:"))
    def addop(self):
        self.c=self.a+self.b
    def dispvalues(self):
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("Sum={}".format(self.c))
#main program
s=Sum()
s.readvalues()
s.addop()
s.dispvalues()


---


#Program for Cal sum of Two Numbers by using Classes and objects
#SumOperationInstanceMethodEx2.py
class Sum:
    def readvalues(self):
        self.a=float(input("Enter First Value:"))
        self.b=float(input("Enter Second Value:"))
        self.addop()
        self.dispvalues()
    def addop(self):
        self.c=self.a+self.b
    def dispvalues(self):

```

```
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("Sum={}".format(self.c))
#main program
s=Sum()
s.readvalues()
=====
```

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 initlize the object..

Syntax for defining Constructor:

```
def      __init__(self, list of formal params if any):
-----  
          Block of Statements  
          Performs Initlization  
-----
```

Rules or Properties of Constructors

1. The Name of the constructor is always def __init__(self,....)
2. Constructors will call automatically / implciitly 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, Conctructors can be Overridden

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 Initlize Multiple objects of same class with Same Values".
=>Syntax: def __init__(self):

Block of statements
Performs Initlization 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 Initlize Multiple objects of same class with Different Values".
=>Syntax: def __init__(self,list of formal params):

Block of statements
Performs Initlization Process

Examples:

```
#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
```

```
#EmployeeInfoWithDatabaseEx1.py
import cx_Oracle
class Employee:
    def __init__(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
    def saveempdata(self):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            iq="insert into employee1 values(%d, '%s', %f)"
            cur.execute(iq %(self.eno, self.ename, self.sal))
            con.commit()
            print("Employee Record Inserted Sucessfully")
        except cx_Oracle.DatabaseError as db:
            print("Problem in Data base:",db)
#main program
e=Employee() #Object Creation
```

```

e.saveempdata()
=====
#EmployeeInfoWithDatabaseEx1.py
import mysql.connector
class Employee:
    def __init__(self):
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
    def saveempdata(self):
        try:
            con=mysql.connector.connect(host="localhost",
                                         user="root",
                                         passwd="root",
                                         database="batch9am")
            cur=con.cursor()
            iq="insert into employeel values(%d, '%s', %f)"
            cur.execute(iq %(self.eno, self.ename, self.sal))
            con.commit()
            print("Employee Record Inserted Sucessfully")
        except mysql.connector.DatabaseError as db:
            print("Problem in Data base:",db)
#main program
e=Employee() #Object Creation
e.saveempdata()
=====

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

```
so=Student()
```

Example:- create an object Employee

```
eo=Employee(10,"Rossum")
```

Differences Between Classes and Objects

Class:

- 1) A class is a collection of Data Members and Methods
- 2) When we define a class, memory space is not created for Data Members and Methods and it can be treated as specification / model for real time application.
- 3) Definition of a particular class 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

=====

Constructors 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.

```
#Non-DestEx1.py  
class Employee:  
    def __init__(self,eno,ename):  
        print("-----")  
        print("I am from DC")  
        self.eno=eno  
        self.ename=ename  
        print("Employee Number:{}".format(self.eno))
```

```

        print("Employee Name:{}".format(self.ename))
        print("-----")
#main program
print("Program execution started")
e1=Employee(10,"Rossum")
e2=Employee(20,"Travis")
print("Program execution Ended")


---


#DestEx1.py
import sys
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
              Memory space")
        global totmemspace
        totmemspace=totmemspace-sys.getsizeof(self)
        print("Now Available Memory Space=",totmemspace)
#main program
print("Program execution started")
e1=Employee(10,"Rossum")
e2=Employee(20,"Travis")
e3=Employee(30,"Ritche")
totmemspace=sys.getsizeof(e1)+sys.getsizeof(e2)+sys.getsizeof(e3)
)
print("Total Memory space of Objects=",totmemspace) # 144
print("Program execution Ended")


---


#DestEx2.py
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
              Memory space")

```

```

#main program
print("Program execution started")
e1=Employee(10,"Rossum")
e2=Employee(20,"Travis")
e3=Employee(30,"Ritche")
print("Program execution Ended")


---


#DestEx3.py
import time
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{} ".format(self.eno))
        print("Employee Name:{} ".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
              Memory space")
#main program
print("Program execution started")
e1=Employee(10,"Rossum")
print("No Longer Interested to maintain e1 object memory")
time.sleep(5)
e1=None # GC Calls Destructor Forcefully
e2=Employee(20,"Travis")
print("No Longer Interested to maintain e2 object memory")
time.sleep(5)
e2=None #GC Calls Destructor Forcefully
e3=Employee(30,"Ritche")
print("No Longer Interested to maintain e3 object memory")
time.sleep(5)
e3=None # GC Calls Destructor Forcefully
print("Program execution Ended")


---


#DestEx1.py
import sys,time
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{} ".format(self.eno))
        print("Employee Name:{} ".format(self.ename))
        print("-----")

```

```

def __del__(self):
    print("GC Calls __del__(self) for Destroying Object
          Memory space")
#main program
print("Program execution started")
e1=Employee(10,"Rossum")
print("No Longer Interested to maintain e1 object memory")
time.sleep(5)
del e1 # GC Calls Destructor Forcefully
e2=Employee(20,"Travis")
print("No Longer Interested to maintain e2 object memory")
time.sleep(5)
del e2 # GC Calls Destructor Forcefully
e3=Employee(30,"Ritche")
print("No Longer Interested to maintain e3 object memory")
time.sleep(5)
del e3 # GC Calls Destructor Forcefully
print("Program execution Ended")

```

```

#DestEx5.py
import time
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{} ".format(self.eno))
        print("Employee Name:{} ".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
              Memory space")
#main program
print("Program execution started")
e1=Employee(10,"Rossum") # Object Creation
e2=e1 # Deep Copy
e3=e1 # Deep Copy
print("Program execution Ended")
time.sleep(10)

```

```

#DestEx5.py
import time
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno

```

```

        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
              Memory space")
#main program
print("Program execution started")
e1=Employee(10,"Rossum") # Object Creation
e2=e1 # Deep Copy
e3=e1 # Deep Copy
print("No Longer Interested to maintain e1 object memory")
time.sleep(5)
del e1#GC will not call Destructor Forcefully bcoz e2 and e3
points to object
print("No Longer Interested to maintain e2 object memory")
time.sleep(5)
del e2 #GC will not call Destructor Forcefully bcoz e3 points to
object
print("No Longer Interested to maintain e3 object memory")
time.sleep(5)
del e3 # #GC Calls Destructor Forcefully
print("Program execution Ended")
time.sleep(10)


---


#GCEX1.py
import gc
print("Initially, Is GC Running=",gc.isenabled())
a=10
b=20
print("a={} \tb={}".format(a,b))
gc.disable()
print("Now Is GC Running=",gc.isenabled())
c=a+b
gc.enable()
print("Now Is GC Running=",gc.isenabled())


---


#GCEX2.py
import gc
import sys
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        print("I am from DC")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))

```

```

        print("Employee Name:{} ".format(self.ename))
        print("-----")
    def __del__(self):
        print("GC Calls __del__(self) for Destroying Object
Memory space")
        global totmemspace
        totmemspace=totmemspace-sys.getsizeof(self)
        print("Now Available Memory Space=",totmemspace)
#main program
print("Program execution started")
print("Initially, IS GC Running:",gc.isenabled())
e1=Employee(10,"Rossum")
e2=Employee(20,"Travis")
e3=Employee(30,"Ritche")
gc.disable()
print("IS GC Running:",gc.isenabled())
totmemspace=sys.getsizeof(e1)+sys.getsizeof(e2)+sys.getsizeof(e3)
)
print("Total Memory space of Objects=",totmemspace) # 144
print("Program execution Ended")
=====
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 Level)

```

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 Initlized Data Memebrs can be encapsulated.

Note: We can also Encapsulate Class Name But In real Time Hiding the class name is of no use.

```
class __<clsname>:  
-----  
-----  
-----
```

#Account1.py--File Name and Module Name--Data Encapsulation

```
class Account:  
    def __init__(self):  
        self.__acno=100  
        self.cname="Rossum"  
        self.__bal=4.5  
        self.__pin=3456  
        self.bname="SBI"
```

#OtherProg1.py---Data Abstraction

```

from Account1 import Account
ac=Account()
print("-----")
#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))
print("-----")


---


#Account2.py--File Name and Module Name--Data Encapsulation
class Account:
    def getaccdetails(self):
        self.__acno=100
        self.cname="Rossum"
        self.__bal=4.5
        self.__pin=3456
        self.bname="SBI"
-----
#OtherProg2.py---Data Abstraction
from Account2 import Account
ac=Account()
ac.getaccdetails()
print("-----")
#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))
print("-----")


---


#Account3.py--File Name and Module Name--Data Encapsulation
class Account:
    def __getaccdetails(self):
        self.acno=100
        self.cname="Rossum"
        self.bal=4.5
        self.pin=3456
        self.bname="SBI"
-----
#OtherProg3.py---Data Abstraction
from Account3 import Account
ac=Account()
#ac.getaccdetails()
print("-----")
#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))
print("-----")


---


#Account4.py--File Name and Module Name--Data Encapsulation
#It is not Possible to Apply Data Encapsulation on Constructors
and Destructors
class Account:
    def _____init_____(self):
        self.acno=100
        self.cname="Rossum"
        self.bal=4.5
        self.pin=3456
        self.bname="SBI"
-----
#OtherProg4.py---Data Abstraction
from Account4 import Account
ac=Account()
ac._____init_____
print("-----")
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))
print("-----")


---


#Account5.py--File Name and Module Name--Data Encapsulation
class __Account: # Here we are Hiding Class Name
    def _____init_____(self):
        self.acno=100
        self.cname="Rossum"
        self.bal=4.5
        self.pin=3456
        self.bname="SBI"
-----
#OtherProg5.py
from Account5 import Account
ac=Account()


---


#PickOoopsEx1.py---Unpickling Process
import pickle
class ReadStudentData:
    def readrecords(self):
        try:
            with open("studpick.data","rb") as fp:
                print("*"*50)
                while(True):
                    try:
                        obj = pickle.load(fp)

```

```

        obj.dispstuddata()
    except EOFError:
        print("=" * 50)
        break

    except FileNotFoundError:
        print("File does not exist")
#main program
rs=ReadStudentData()
rs.readrecords()

```

```

#PickOopsEx1.py--Pickling Process
from PickStudent import Student
import pickle
class SaveStudentData:
    def readstuddata(self):
        with open("studpick.data", "ab") as fp:
            while(True):
                print("-" * 50)
                sno=int(input("Enter Student Number:"))
                sname = input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                #crate an object of student class
                s=Student(sno,sname,marks)
                #save object s data into file
                pickle.dump(s,fp)
                print("Student Object Data Saved into the File")
                print("-" * 50)
                ch=input("Do u want to Insert another
                         record(yes/no):")
                if(ch.lower() == "no"):
                    break

```

```

#main program
sp=SaveStudentData()
sp.readstuddata()

```

```

#PickStudent.py---File Name and Module Name
class Student:
    def __init__(self, sno, sname, marks):
        self.sno = sno
        self.sname = sname
        self.marks = marks
    def dispstuddata(self):
        print("\t{}\t{}\t{}".format(self.sno, self.sname,
                                    self.marks))
=====
```

Inheritance

=====

=>Inheritance is one of distinct features of OOPs

=>The purpose of Inheritance is that " To build Re-usable Applications in Python Object Oriented Programming".

=>Definition of Inheritance:

=>The Process obtaining Data members , Methods and Constructors (Features) of one class into another class is called Inheritance.

=>The class which is giving Data members , Methods and Constructors (Features) is called Super or Base or Parent Class.

=>The Class which is taking Data members , Methods and Constructors (Features) is called Sub or Derived or Child Class.

=>The Inheritance concept always follows Logical (Virtual) Memory Management. This Memory Management says that " Neither we write Source Code nor Takes Physical Memory Space ".

Advantages of Inheritance:

=>When we develop any inheritance based application, we get the following advantages.

1. Application Development Time is Less
2. Application Memory Space is Less
3. Application Execution time is Fast / Less
4. Application Performance is enhanced (Improved)
5. Redundancy (Duplication) of the code is minimized.

Types of Inheritances OR Re-Usable Techniques

=>Type of Inheritance is one of the Diagram OR Pattern Or Model, which makes us Understand How the features are Inherited from Base Class into Derived Class.

=>In Python Programming, we have 5 Types of Inheritances. They are

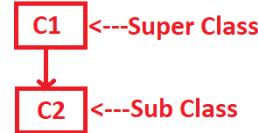
1. Single Inheritance
2. Multi Level Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

1) Single Inheritance

Definition:

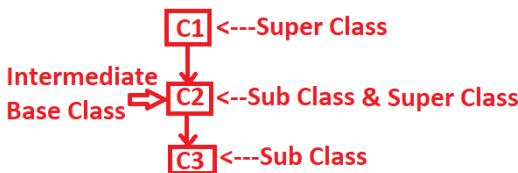
This Inheritance contains Single Base class and Single Derived Class

Diagram:



2. Multi Level Inheritance

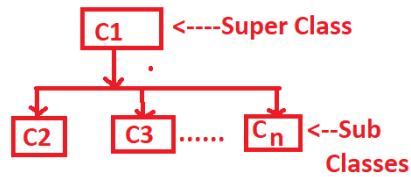
Definition: This Inheritance contains single base class , single derived class and Intermediate base class(es).



3. Hierarchical Inheritance

Definition: This Inheritance Contains Single Super Class and Multiple Sub Classes

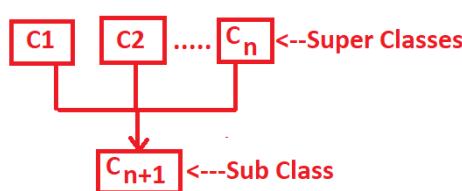
Diagram:



4. Multiple Inheritance

Definition: This Inheritance contains multiple Super Classes and single sub class.

Diagram:



5. Hybrid Inheritance:

Definition:
Hybrid Inheritance= Combination of any available Inheritance Types.

Diagram1:

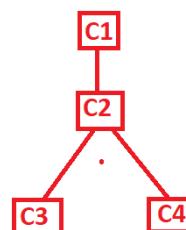
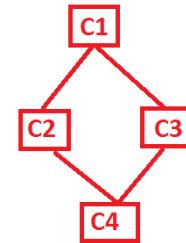


Diagram2:



Inheriting the Features of Base Class into Derived Class

=>To Inherit the Features of Base Class into Derived Class, we use the Following Syntax.

```
class <clsname-1>:
```

```

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

class <classname-2>:  

-----  

-----  

-----  

-----  

class <classname-n>:  

-----  

-----  

-----  

-----  

class <classname-n+1>(classname-1,classname-  

2,...,classname-n):  

-----  

-----  

-----  

-----  

-----
```

Explanation:

- =>Here <classname-1>,<classname-2>....<classname-n> are called Base OR Super Class.
 - =>Here <classname-n+1> is called Sub OR Derived Class Name.
 - =>Here All the Data Members, Methods and Constructors of <classname-1>,<classname-2>....<classname-n> are Inherited into <classname-n+1> and They are available Virtually OR Logically in <classname-n+1> .
 - =>When we Develop any Inheritance Application, It is always recommended to create an object of Bottom Most Derived Class bcoz we can access the features of Intermediate Base Classes and Top Most Base Class.
 - =>For Every Class in Python, there exist an implicit pre-defined class called "object" bcoz object class provides Garbage Collection Facility.
 - =>For all the data types, there exist a super type called "object".
-

#Program for Demonstrating Inheritance Process

#Non-InheritanceEx1.py

```

class C1:
    def disp1(self):
        print("C1-disp1()")
class C2:
    def disp2(self):
        print("C2-disp2()")
#main program
o1=C1()
o2=C2()
```

```

o1.disp1()
o2.disp2()


---


#Program for Demonstrating Inheritance Process
#InhProg1.py
class C1:
    def disp1(self):
        print("C1-disp1()")
class C2(C1):
    def disp2(self):
        print("C2-disp2()")
class C3(C2):
    def disp3(self):
        print("C3-disp3()")
#main program
o3=C3()
o3.disp1()
o3.disp2()
o3.disp3()


---


#Program for Demonstrating Inheritance Process
#InhProg2.py
class C1:
    def getA(self):
        self.a=10
class C2(C1):
    def getB(self):
        self.b=20
    def cal(self):
        self.c=self.a+self.b
        print("sum({},{})={}".format(self.a,self.b,self.c))
#main program
o2=C2()
o2.getA()
o2.getB()
o2.cal()


---


#Program for Demonstrating Inheritance Process
#InhProg3.py
class C1:
    def getA(self):
        self.a=10
class C2(C1):
    def getB(self):
        self.b=20
    def cal(self):
        self.getA()# calling Instance Method of Base Class from
                  Derived Class

```

```

        self.getB()# Calling Instance Method of Current Class
                  from Current Class Method
        self.c=self.a+self.b
        print("sum({},{})={}".format(self.a,self.b,self.c))
#main program
o2=C2()
o2.cal()


---


#Program for Demonstrating Inheritance Process
#InhProg4.py
class Father:
    def getFProp(self):
        self.fp=float(input("Enter Father Property:"))
class Mother:
    def getMProp(self):
        self.mp=float(input("Enter Mother Property:"))

class Child(Father,Mother):
    def getCProp(self):
        self.cp = float(input("Enter Child Property:"))
    def caltotprop(self):
        self.tp=self.fp+self.mp+self.cp
        print("====")
        print("Father Property:{} ".format(self.fp))
        print("Mother Property:{} ".format(self.mp))
        print("Child Property:{} ".format(self.cp))
        print("TOTAL PROPERTY:{} ".format(self.tp))
        print("====")
#main program
c=Child()
c.getFProp()
c.getMProp()
c.getCProp()
c.caltotprop()


---


#Program for Demonstrating Inheritance Process
#InhProg5.py
class Father:
    def getFProp(self):
        self.fp=float(input("Enter Father Property:"))
class Mother:
    def getMProp(self):
        self.mp=float(input("Enter Mother Property:"))

class Child(Father,Mother):
    def getCProp(self):
        self.cp = float(input("Enter Child Property:"))
    def caltotprop(self):

```

```

        self.getFProp()
        self.getMProp()
        self.getCPProp()
        self.tp=self.fp+self.mp+self.cp
        print("====")
        print("Father Property:{} ".format(self.fp))
        print("Mother Property:{} ".format(self.mp))
        print("Child Property:{} ".format(self.cp))
        print("TOTAL PROPERTY:{} ".format(self.tp))
        print("====")
#main program
c=Child()
c.caltotprop()
=====

```

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.

(POLYMORPHISM<----METHOD OVERRIDING<----INHERITANCE<----
CLASS AND OBJECTS)

**Number of approaches to call original methods / constructors
from Overridden methods / Constructors**

=>We have two approaches to call original method / constructors
of base class from overridden method / constructors of derived
class. They are

- 1) By using super()
- 2) By using Class Name

1) By using super():

=>super() is one of the pre-defined function, which is used for
calling super class original method / constructor from
overridden method / constructors of derived class.

Syntax1:- super().methodname(list of values if any)

Syntax2:- super().__init__(list of values if any)

=>with super() we are able to call only immediate base class
method but unable to call Specified method of base Class . To
do this we must use class name approach.

2) By using Class Name:

=>By using ClassName approach, we can call any base class method
/ constructor name from the context of derived class method /
constructor names.

Syntax1:- ClassName.methodname(self, list of values if
any)

Syntax2:- ClassName.__init__(self, list of values if any)

#Program for Demonstarting Polymorphism

#PolyEx1.py

class Circle:

```
    def draw(self): # Original Method(One Form)
        print("Circle--Draw")
        #super().draw()---AttributeError: 'super' object has no
                        attribute 'draw'
```

class Rect(Circle):

```
    def draw(self): # Overridden Method (Multiple Forms)
        print("Rect--Draw")
        super().draw() # calling Super Class draw() from Derived
```

Class draw()

class Square(Rect):

```

def draw(self): # Overridden Method (Multiple Forms)
    print("Square Draw")
    super().draw() # calling Super Class draw() from Derived
Class draw()
#Main program
print("w.r.t Square Class")
s=Square()
s.draw()


---


#Program for Demonstarting Polymorphism
#PolyEx2.py
class Circle:
    def draw(self): # Original Method(One Form)
        print("Circle--Draw")
class Rect(Circle):
    def draw(self): # Overridden Method (Multiple Forms)
        print("Rect--Draw")

class Square(Rect):
    def draw(self): # Overridden Method (Multiple Forms)
        print("Square Draw")
        Circle.draw(self) # calling Circle Class draw() from
Derived Class draw()
        Rect.draw(self)# calling Rect Class draw() from Derived
Class draw()
#Main program
print("w.r.t Square Class")
s=Square()
s.draw()


---


#Program for Demonstarting Polymorphism
#PolyEx2.py
class Circle:
    def draw(self): # Original Method(One Form)
        print("Circle--Draw")
class Rect:
    def draw(self): # Original Method(One Form)
        print("Rect--Draw")
class Triangle:
    def draw(self): # Original Method(One Form)
        print("Triangle--Draw")

class Square(Triangle,Circle,Rect):
    def draw(self): # Overridden Method
        print("Square--Draw")
        super().draw()
        Circle.draw(self)
        Rect.draw(self)

```

```

#Main program
print("w.r.t Square Class")
s=Square()
s.draw()


---


#Program for Demonstarting Polymorphism
#PolyEx2.py
class Circle:
    def draw(self): # Original Method(One Form)
        print("Circle--Draw")
class Rect:
    def draw(self): # Original Method(One Form)
        print("Rect--Draw")
class Triangle:
    def draw(self): # Original Method(One Form)
        print("Triangle--Draw")

class Square(Triangle,Circle,Rect):
    def draw(self): # Overridden Method
        print("Square--Draw")
        super().draw()
        Circle.draw(self)
        Rect.draw(self)


---


#Main program
print("w.r.t Square Class")
s=Square()
s.draw()


---


#Program for Demonstarting Polymorphism
#PolyEx4.py
class Circle:
    def __init__(self): # Original Constructor(One Form)
        print("Circle--Draw")
        super().__init__() # Valid
class Rect(Circle):
    def __init__(self): # Overridden Constructor
        print("Rect--Draw")
        super().__init__() # Calling Super Class Constructor
from Derived Class Constructor
class Square(Rect):
    def __init__(self): # Overridden Constructor
        print("Square--Draw")
        super().__init__() # Calling Super Class Constructor
from Derived Class Constructor
#main program
print("w.r.t Square Class")
s=Square() # Obejct Creation calls


---


#Program for Demonstarting Polymorphism

```

```

#PolyEx4.py
class Circle:
    def __init__(self): # Original Constructor(One Form)
        print("Circle--Draw")
        super().__init__() # Valid
class Rect:
    def __init__(self): # Overridden Constructor
        print("Rect--Draw")

class Square(Rect,Circle):
    def __init__(self): # Overridden Constructor
        print("Square--Draw")
        super().__init__()
        Circle.__init__(self)

#main program
print("w.r.t Square Class")
s=Square() # Object Creation calls --Constructor


---


#PolyEx6.py
class Circle:
    def area(self):
        self.r=float(input("Enter Radius:"))
        self.ac=3.14*self.r**2
        print("Area of Circle={}".format(self.ac))
class Square(Circle):
    def area(self):
        self.s=float(input("Enter Side:"))
        self.sa=self.s**2
        print("Area of Square={}".format(self.sa))
        print("-----")
        super().area()
class Rect(Square):
    def area(self):
        self.l=float(input("Enter Length:"))
        self.b = float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect={}".format(self.ar))
        print("-----")
        super().area()
#main program
r=Rect()
r.area()


---


#PolyEx7.py
class Circle:
    def area(self):
        self.r=float(input("Enter Radius:"))
        self.ac=3.14*self.r**2

```

```

        print("Area of Circle={}".format(self.ac))
class Square(Circle):
    def area(self):
        self.s=float(input("Enter Side:"))
        self.sa=self.s**2
        print("Area of Square={}".format(self.sa))
        print("-----")
        Circle.area(self)
class Rect(Square):
    def area(self):
        self.l=float(input("Enter Length:"))
        self.b = float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect={}".format(self.ar))
        print("-----")
        Square.area(self)
#main program
r=Rect()
r.area()



---


#PolyEx8.py
class Circle:
    def area(self,r):
        self.ac=3.14*r**2
        print("Area of Circle={}".format(self.ac))
class Square(Circle):
    def area(self,s):
        self.sa=s**2
        print("Area of Square={}".format(self.sa))
        print("-----")
        super().area(float(input("Enter Radius:")))
class Rect(Square):
    def area(self,l,b):
        self.ar=l*b
        print("Area of Rect={}".format(self.ar))
        print("-----")
        super().area(float(input("Enter Side:")))
#main program
r=Rect()
l = float(input("Enter Length:"))
b = float(input("Enter Breadth:"))
r.area(l,b)



---


#PolyEx9.py
class Circle:
    def area(self,r):
        self.ac=3.14*r**2
        print("Area of Circle={}".format(self.ac))

```

```

class Square(Circle):
    def area(self,s):
        self.sa=s**2
        print("Area of Square={}".format(self.sa))
        print("-----")
        Circle.area(self,float(input("Enter Radius:")))
    class Rect(Square):
        def area(self,l,b):
            self.ar=l*b
            print("Area of Rect={}".format(self.ar))
            print("-----")
            Square.area(self,float(input("Enter Side:")))
#main program
r=Rect()
l = float(input("Enter Length:"))
b = float(input("Enter Breadth:"))
r.area(l,b)


---


#PolyEx9.py
class Circle:
    def __init__(self,r):
        self.ac=3.14*r**2
        print("Area of Circle={}".format(self.ac))
        print("-----")
        Square.__init__(self,float(input("Enter Side:")))
class Square(Circle):
    def __init__(self,s):
        self.sa=s**2
        print("Area of Square={}".format(self.sa))
        print("-----")
class Rect(Square):
    def __init__(self,l,b):
        self.ar=l*b
        print("Area of Rect={}".format(self.ar))
        print("-----")
        Circle.__init__(self,float(input("Enter Radius:")))
#main program
l = float(input("Enter Length:"))
b = float(input("Enter Breadth:"))
r=Rect(l,b)
=====
numpy Module
=====
=====

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 efficient bcoz list object values takes more time to extract or retrieve(they are available at different Address) where as numpy based applications are efficient bcoz of ndarray object values takes less time to extract or retrieve(they are available at same Address / clustered) .

=>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.

```
In [1]: pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\kvr\anaconda3\lib\site-packages (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

```
In [2]: pip list
```

psutil	5.9.0
ptyprocess	0.7.0
py	1.11.0
pyasn1	0.4.8
pyasn1-modules	0.2.8
pycodestyle	2.8.0
pycosat	0.6.3
pycparser	2.21
pyct	0.4.8
pycurl	7.45.1
PyDispatcher	2.0.5
pydocstyle	6.1.1
pyerfa	2.0.0
pyflakes	2.4.0
Pygments	2.11.2
PyHamcrest	2.0.2
PyJWT	2.4.0
pylint	2.14.5
pyls-spyder	0.4.0
PyNaCl	1.5.0

```
In [3]: import numpy as np
```

```
In [4]: print(np.__version__)
```

```
1.21.5
```

```
In [5]: import sys
```

```
In [6]: lst=[10,20,30,40]
print(lst,type(lst))
totmem=sys.getsizeof(lst)
print("Total Memory Space=",totmem)
```

```
[10, 20, 30, 40] <class 'list'>
Total Memory Space= 120
```

```
In [8]: #convert Traditional Python List object into ndarray object
a=np.array(lst)
print(a,type(a))
totmem=sys.getsizeof(a)
print("Total Memory Space=",totmem)
```

```
[10 20 30 40] <class 'numpy.ndarray'>
Total Memory Space= 120
```

```
In [9]: lst=[10,20]
print(lst,type(lst))
totmem=sys.getsizeof(lst)
print("Total Memory Space=",totmem)
```

```
[10, 20] <class 'list'>
Total Memory Space= 72
```

```
In [10]: #convert Traditional Python List object into ndarray object
a=np.array(lst)
print(a,type(a))
totmem=sys.getsizeof(a)
print("Total Memory Space=",totmem)
```

```
[10 20] <class 'numpy.ndarray'>
Total Memory Space= 112
```

```
In [11]: lst=[10,20,30,40,50,60,70,80,90]
print(lst,type(lst))
totmem=sys.getsizeof(lst)
print("Total Memory Space=",totmem)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90] <class 'list'>
Total Memory Space= 152
```

```
In [12]: #convert Traditional Python List object into ndarray object
a=np.array(lst)
print(a,type(a))
totmem=sys.getsizeof(a)
print("Total Memory Space=",totmem)
```

```
[10 20 30 40 50 60 70 80 90] <class 'numpy.ndarray'>
Total Memory Space= 140
```

```
In [13]: lst=[10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90]
print(lst,type(lst))
totmem=sys.getsizeof(lst)
print("Total Memory Space=",totmem)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10, 20, 30, 40, 50, 60, 70, 80, 90, 10, 20, 30, 40, 50, 60, 70, 80, 90, 10, 20, 30, 40, 50, 60, 70, 80, 90] <class 'list'>
Total Memory Space= 280
```

```
In [14]: #convert Traditional Python List object into ndarray object
a=np.array(lst)
print(a,type(a))
totmem=sys.getsizeof(a)
print("Total Memory Space=",totmem)

[10 20 30 40 50 60 70 80 90 10 20 30 40 50 60 70 80 90 10 20 30 40 50 60
70 80 90] <class 'numpy.ndarray'>
Total Memory Space= 212
```



```
In [15]: lst=[10,20,30]
lst=lst+1 # TypeError: can only concatenate List (not "int") to list
```

```
-----
```

```
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8368\3497297203.py in <module>
      1 lst=[10,20,30]
----> 2 lst=lst+1

TypeError: can only concatenate list (not "int") to list
```



```
In [17]: a=np.array(lst)
print(a,type(a))
a=a+1 # Vector Based Operations
print(a,type(a))

[10 20 30] <class 'numpy.ndarray'>
[11 21 31] <class 'numpy.ndarray'>
```



```
In [18]: lst=[10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90,10,20,30,40,50,60,70,80,90]
print(lst,type(lst))
totmem=sys.getsizeof(lst)
print("Total Memory Space=",totmem)
```



```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10, 20, 30, 40, 50, 60, 70, 80, 90, 10,
20, 30, 40, 50, 60, 70, 80, 90] <class 'list'>
Total Memory Space= 280
```



```
In [20]: #convert Traditional Python List object into ndarray object
a=np.array(lst)
print(a,type(a))
totmem=sys.getsizeof(a)
print("Total Memory Space=",totmem)
```



```
[10 20 30 40 50 60 70 80 90 10 20 30 40 50 60 70 80 90 10 20 30 40 50 60
70 80 90] <class 'numpy.ndarray'>
Total Memory Space= 212
```

```
In [22]: lst=[10,20,30,40,50,60,70,80,90]
print(lst,type(lst),id(lst))
print("-----")
for val in lst:
    print(val,id(val))

[10, 20, 30, 40, 50, 60, 70, 80, 90] <class 'list'> 1574748768320
-----
10 1574664301136
20 1574664301456
30 1574664301776
40 1574664302096
50 1574664302416
60 1574664491216
70 1574664491536
80 1574664491856
90 1574664492176
```

```
In [23]: a=np.array(lst)
print(a,type(a),id(a))
print("-----")
for val in a:
    print(val,id(val))

[10 20 30 40 50 60 70 80 90] <class 'numpy.ndarray'> 1574770709872
-----
10 1574769757872
20 1574769757936
30 1574769757872
40 1574769757936
50 1574769757872
60 1574769757936
70 1574769757872
80 1574769757936
90 1574769757872
```

```
In [24]: lst=[10,20,30,40,50,60]
print(lst,type(lst))

[10, 20, 30, 40, 50, 60] <class 'list'>
```

```
In [30]: a=np.array(lst)
print(a,type(a))
print("Dimension=", a.ndim)
print("Shape=",a.shape)

[10 20 30 40 50 60] <class 'numpy.ndarray'>
Dimension= 1
Shape= (6,)
```

```
In [31]: lst.reshape(2,3) # List' object has no attribute 'reshape'

-----
AttributeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8368\3097928053.py in <module>
----> 1 lst.reshape(2,3)

AttributeError: 'list' object has no attribute 'reshape'

In [32]: a.reshape(2,3)

Out[32]: array([[10, 20, 30],
   [40, 50, 60]])

In [33]: a.reshape(3,2)

Out[33]: array([[10, 20],
   [30, 40],
   [50, 60]])

In [34]: print(a,type(a))

[10 20 30 40 50 60] <class 'numpy.ndarray'>

In [35]: lst=[10,20,30,40,50,60,70,80,90,15,25,35]
print(lst,type(lst))

[10, 20, 30, 40, 50, 60, 70, 80, 90, 15, 25, 35] <class 'list'>

In [36]: a=np.array(lst)
print(a,type(a))

[10 20 30 40 50 60 70 80 90 15 25 35] <class 'numpy.ndarray'>

In [38]: a.shape=(4,3)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)

[[10 20 30]
 [40 50 60]
 [70 80 90]
 [15 25 35]] <class 'numpy.ndarray'>
Dimension= 2
Shape= (4, 3)
```

```
In [39]: a.shape=(3,4)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)

[[10 20 30 40]
 [50 60 70 80]
 [90 15 25 35]] <class 'numpy.ndarray'>
Dimension= 2
Shape= (3, 4)
```

```
In [40]: a.shape=(6,2)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)

[[10 20]
 [30 40]
 [50 60]
 [70 80]
 [90 15]
 [25 35]] <class 'numpy.ndarray'>
Dimension= 2
Shape= (6, 2)
```

```
In [41]: a.shape=(2,6)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)

[[10 20 30 40 50 60]
 [70 80 90 15 25 35]] <class 'numpy.ndarray'>
Dimension= 2
Shape= (2, 6)
```

```
In [42]: a.shape=(3,6) # ValueError: cannot reshape array of size 12 into shape (3,6)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)
```

```
-----
ValueError                                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8368\4074935259.py in <module>
----> 1 a.shape=(3,6)
      2 print(a,type(a))
      3 print("Dimension=",a.ndim)
      4 print("Shape=",a.shape)
```

```
ValueError: cannot reshape array of size 12 into shape (3,6)
```

```
In [43]: a.shape=(2,2,3)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)
```

```
[[[10 20 30]
 [40 50 60]]

 [[70 80 90]
 [15 25 35]]] <class 'numpy.ndarray'>
Dimension= 3
Shape= (2, 2, 3)
```

```
In [44]: a.shape=(2,3,2)
print(a,type(a))
print("Dimension=",a.ndim)
print("Shape=",a.shape)
```

```
[[[10 20]
 [30 40]
 [50 60]]

 [[70 80]
 [90 15]
 [25 35]]] <class 'numpy.ndarray'>
Dimension= 3
Shape= (2, 3, 2)
```

```
In [46]: lst=[10,"RS",34.56,True]
print(lst,type(lst))
for val in lst:
    print(val,type(val))
```

```
[10, 'RS', 34.56, True] <class 'list'>
10 <class 'int'>
RS <class 'str'>
34.56 <class 'float'>
True <class 'bool'>
```

```
In [47]: a=np.array(lst)
print(a,type(a))
for val in a:
    print(val,type(val))
```

```
['10' 'RS' '34.56' 'True'] <class 'numpy.ndarray'>
10 <class 'numpy.str_'>
RS <class 'numpy.str_'>
34.56 <class 'numpy.str_'>
True <class 'numpy.str_'>
```

```
In [48]: a=np.array(lst,dtype="object")
print(a,type(a))
for val in a:
    print(val,type(val))

[10 'RS' 34.56 True] <class 'numpy.ndarray'>
10 <class 'int'>
RS <class 'str'>
34.56 <class 'float'>
True <class 'bool'>
```

```
In [49]: a.dtype
```

```
Out[49]: dtype('O')
```

```
In [50]: print(a.dtype)
```

```
object
```

```
In [51]: lst=[10,"RS",34.56,True]
print(lst,type(lst))
lst[0]=100
print(lst,type(lst))
```

```
[10, 'RS', 34.56, True] <class 'list'>
[100, 'RS', 34.56, True] <class 'list'>
```

```
In [56]: a=np.array(lst)
print(a,type(a),id(a))
a[0]=200
print(a,type(a),id(a))
```

```
['100' 'RS' '34.56' 'True'] <class 'numpy.ndarray'> 1574770934192
['200' 'RS' '34.56' 'True'] <class 'numpy.ndarray'> 1574770934192
```

```
In [ ]:
```

=====

Number of approaches to create an object of ndarray

=====

=>"ndarray" is one of the pre-defined class of numpy module and whose object is used for storing the data in numpy programming in the form of 1-D, 2-D and n-Dimensional Arrays.

=>In numpy programming, we have the following essential approaches to create an object of ndarray.

1. array()
2. arange()
3. zeros()
4. ones()
5. full()
6. identity()
7. hstack()
8. vstack()

1) array():

=>This Function is used for converting Traditional Python Objects into ndarray object.

=>Syntax:- varname=numpy.array(Object,dtype)
=>Here var name is an object of <class,ndarray>
=>here array() is pre-defined function of numpy module used for converting Traditional Python Objects into ndarray object.
=>object represents any Traditional Python Objects
=>dtype represents any numpy data type such as int8,int16,int32,float16, float 32, float64,....etc (Internal 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-----()
>>> 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-----()
>>> c.shape-----()
>>> 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-----()
-----
>>> 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.])
>>> 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]]])

```

```

[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]]])

>>> 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. 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 is function is used for building a matrix by specifying
fill value either 1-D or 2-D or n-D

```

=>Syntax:- varname=numpy.full(shape, fill_value, dtype)
=>varname is an obejct of <class, numpy.ndarray>
=>Here Shape can be 1-D(number of Fill_Value) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)
=>fill_value can be any number of programmer choice

Examples:

```
-----  
>>> a=np.full(3,1)  
>>> a-----array([1, 1, 1])  
>>>print(type(a))-----<class,numpy.ndarray>  
>>> a=np.full(3,9)  
>>> a-----array([9, 9, 9])  
>>> a=np.full(6,8)  
>>> a-----array([8, 8, 8, 8, 8, 8])  
>>> a.shape=(3,2)  
>>> a  
      array([[8, 8],  
              [8, 8],  
              [8, 8]])  
>>> a=np.full(6,9)  
>>> a-----array([9, 9, 9, 9, 9, 9])  
>>> a.reshape(2,3)  
      array([[9, 9, 9],  
              [9, 9, 9]])  
>>> a=np.full((3,3),9)  
>>> a  
      array([[9, 9, 9],  
              [9, 9, 9],  
              [9, 9, 9]])  
>>> a=np.full((2,3),6)  
>>> a  
      array([[6, 6, 6],  
              [6, 6, 6]])  
>>> a.reshape(3,2)  
      array([[6, 6],  
              [6, 6],  
              [6, 6]])  
>>> a=np.full((3,3,3),7)  
>>> a  
      array([[[7, 7, 7],  
              [7, 7, 7],  
              [7, 7, 7]],  
  
              [[[7, 7, 7],  
              [7, 7, 7],  
              [7, 7, 7]]],
```

```
[7, 7, 7]],  
[[7, 7, 7],  
[7, 7, 7],  
[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]]  
-----  
8.numpy.vstack()  
-----  
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.hstack(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:- 1dndrrayobj [begin:end:step]  

-----  

Examples:  

-----  

>>>a=np.array([10,20,30,40,50,60,70])

```

```
>>> a-----array([10, 20, 30, 40, 50, 60, 70])
>>> a[::-1]-----array([70, 60, 50, 40, 30, 20, 10])
>>> a[:, :]-----array([10, 20, 30, 40, 50, 60, 70])
```

2D Arrays Slicing:

Syntax:- ndrrayobj[i , j]
here 'i' represents Row Index
here 'j' represents Column Index
(OR)

Syntax:- 2dndrrayobj [Row Index, Column Index]

Syntax:- 2dndrrayobj [begin:end:step, begin:end:step]

Examples:

```
>>> a=np.array([[10,20,30],[40,50,60]])
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> a[0,0]
10
>>> a[0:,0:1]
array([[10],
       [40]])
>>> a[0:,1:2]
array([[20],
       [50]])
>>> a[1:,:]
array([[40, 50, 60]])
```

=====
3D Arrays Slicing

Syntax:- 3dndrrayobj[i,j,k]

here 'i' represents Which 2D matrix (Matrix Number-->0 1
2 3 4 5.....)
here 'j' represents which Rows in that 2D matrix
here 'k' represents which Columns in that 2D matrix
(OR)

Syntax:- 3dndrrayobj[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---Indexing and Slicing Operations of 1D,2D and 3D array

1D Arrays Slicing:

Syntax:- 1dndrrayobj [begin:end:step]

Examples:

```
>>> a=np.array([10,20,30,40,50,60,70])
>>> a-----array([10, 20, 30, 40, 50, 60, 70])
>>> a[::-1]-----array([70, 60, 50, 40, 30, 20, 10])
>>> a[:]-----array([10, 20, 30, 40, 50, 60, 70])
```

2D Arrays Slicing:

Syntax:- ndrrayobj[i , j]
here 'i' represents Row Index
here 'j' represents Column Index
(OR)

Syntax:- 2dndrrayobj [Row Index, Column Index]

Syntax:- 2dndrrayobj [begin:end:step, begin:end:step]

Examples:

```
>>> a=np.array([[10,20,30],[40,50,60]])
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> a[0,0]
10
>>> a[0:,0:1]
array([[10],
       [40]])
>>> a[0:,1:2]
array([[20],
       [50]])
>>> a[1:,:]
array([[40, 50, 60]])
```

3D Arrays Slicing

Syntax:- 3dndrrayobj [i,j,k]

here 'i' represents Which 2D matrix (Matrix Number--> 0 1
2 3 4 5.....)
here 'j' represents which Rows in that 2D matrix
here 'k' represents which Columns in that 2D matrix
(OR)

Syntax:- 3dndrrayobj[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--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 perfomed 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]])

```

```

[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,ndarray2)
=>This function is used for performing element-wise division of
ndarrayobj1, ndarrayobj2 and result can be displayed

>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
        array([[10, 20],
               [30, 40]])
>>> b
        array([[1, 2],
               [3, 4]])
>>> c=np.divide(a,b)
>>> c
        array([[10., 10.],
               [10., 10.]])
-----
>>> d=a/b      # we used operator / instead of divide()
>>> d
        array([[10., 10.],
               [10., 10.]])
-----
f) floor_divide()
-----
Syntax:- varname=numpy.floor_divide(ndarray1,ndarray2)
=>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,ndarray2)
=>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,ndarray2)
=>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 eleemnt= 4  
>>> print("Column Max eleemnts=",cmax) ---Columns Max elements=  
[4 4 3]  
>>> print("Row Max eleemnts=",rmax) ---Row Max eleemnts= [3 4 4]
```

b) amin():

=>This functions obtains minmum element of the entire matrix.
=>Syntax1:- varname=numpy.amin(ndarrayobject)

=>Syntax2:- varname=numpy.amin(ndarrayobject, axis=0) --->obtains 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 elements=  
[1 2 1]  
>>> print("Row Min eleemnts=",rmin) ---Row Min eleemnts= [1 1 2]
```

c) mean():

=>This is used for cal mean of the total matrix elements.
=>The formula for mean=(sum of all elements of matrix) / total
number of elements.

Syntax1:- varname=numpy.mean(ndarrayobject)
Syntax2:- varname=numpy.mean(ndarrayobject, axis=0) ---
 >Columnwise Mean
Syntax3:- varname=numpy.mean(ndarrayobject, axis=1) ---
 >Rowwise Mean

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
      [[1 2 3]  
      [4 2 1]  
      [3 4 2]]  
>>> m=np.mean(A)  
>>> cm=np.mean(A, axis=0)  
>>> rm=np.mean(A, axis=1)  
>>> print("Mean=",m) -----Mean= 2.4444444444444446  
>>> print("Column Mean=", cm) --Column Mean= [2.66666667  
2.66666667      2. ]  
>>> print("Row Mean=", rm) ---Row Mean= [ 2.        2.33333333  
3. ]
```

d) median():

=>This is used for calculating / obtaining median of entire
matrix elements.

=>Median is nothing but sorting the given data in ascending
order and select middle element.

=>If the number of sorted elements are odd then center or
middle element becomes median.

=>If the number sorted elements are even then select center or
middle of two elements, add them and divided by 2 and that
result becomes median.

Syntax1:- varname=numpy.median(ndarrayobject)
Syntax2:- varname=numpy.median(ndarrayobject, axis=0)
Syntax3:- varname=numpy.median(ndarrayobject, axis=1)

Examples:

```

>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> md=np.median(A)
>>> cmd=np.median(A, axis=0)
>>> rmd=np.median(A, axis=1)
>>> print("Median=",md) ---Median= 2.0
>>> print("Column Median=",cmd) ---Column Median= [3. 2. 2.]
>>> print("Row Median=",rmd) -----Row Median= [2. 2. 3.]
>>> l1=[[2,3],[4,1]]
>>> A=np.array(l1)
>>> print(A)
[[2 3]
 [4 1]]
>>> md=np.median(A)
>>> cmd=np.median(A, axis=0)
>>> rmd=np.median(A, axis=1)
>>> print("Median=",md) ---Median= 2.5
>>> print("Column Median=",cmd) ---Column Median= [3. 2.]
>>> print("Row Median=",rmd) ---Row Median= [2.5 2.5]
-----
e) var():
-----
Variance=  $\text{sqr}(\text{mean}-\text{xi}) / \text{total number of elements}$ 
here 'xi' represents each element of matrix.
-----
Syntax1:- varname=numpy.var(ndarrayobject)
Syntax2:- varname=numpy.var(ndarrayobject, axis=0)
Syntax3:- varname=numpy.var(ndarrayobject, axis=1)
-----
Examples:
-----
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> vr=np.var(A)
>>> cvr=np.var(A, axis=0)
>>> rvr=np.var(A, axis=1)
>>> print("Variance=",vr) ----Variance= 1.1358024691358024

```

```

>>> print("Column Variance=", cvr) ---Column Variance= [1.55555556
0.88888889
>>> 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
>>> print("Row Variance=", rvr) ---Row Variance= [0.66666667
1.55555556 0.66666667]
-----
>>> sd=np.std(A)
>>> csd=np.std(A, axis=0)
>>> rsd=np.std(A, axis=1)
>>> print("std=", sd) ---std= 1.0657403385139377
>>> print(" column std=", csd) --- column std= [1.24721913
0.94280904 0.81649658]
>>> print("Row std=", rsd) --Row std= [0.81649658 1.24721913
0.81649658]
=====X=====
Note: numpy module does not contain mode().
      mode() present in statistics module of Python

mode() gives Highest Frequent Element in given object
Examples:
-----

```

```

>>> import statistics as s
>>> l1=[10,20,30,10,20,40,10]
>>> s.mode(l1)-----10
>>> l1=[10,20,30,10,20,40,10,20]
>>> s.mode(l1)-----10
>>> l1=[20,10,30,10,20,40,10,20]
>>> s.mode(l1)-----20
>>> s.multimode(l1)-----[20, 10]
-----
>>> a=np.array(l1)
>>> s.mode(a)-----20
>>> s.multimode(a)-----[20, 10]
=====
numpy----append()
=====
Numpy module in python, provides a function to 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=numpy.insert( (ndarray , index, value)

Example.
import numpy as np
a=np.array([10,20,30,40])
b=np.insert(a,1,35)
print(b) # 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]
```

```
In [1]: #copy and view operations
```

```
In [2]: import numpy as np
```

```
In [4]: a=np.array([10,20,30,40,50,60])
print(a,type(a),id(a))
```

```
[10 20 30 40 50 60] <class 'numpy.ndarray'> 1980944308240
```

```
In [6]: b=a.copy() # Shallow Copy
print(b,type(b),id(b))
```

```
[10 20 30 40 50 60] <class 'numpy.ndarray'> 1980923335856
```

```
In [7]: a=np.append(a,100)
b=np.append(b,200)
print(a)
print(b)
```

```
[ 10  20  30  40  50  60 100]
[ 10  20  30  40  50  60 200]
```

```
In [8]: a=np.array([10,20,30,40,50,60])
print(a,type(a),id(a))
```

```
[10 20 30 40 50 60] <class 'numpy.ndarray'> 1980945211536
```

```
In [9]: b=a.view()
```

```
In [10]: print(b,type(b),id(b))
```

```
[10 20 30 40 50 60] <class 'numpy.ndarray'> 1980944790704
```

```
In [12]: #c=np.view(a) # AttributeError: module 'numpy' has no attribute 'view'
```

```
In [13]: a=np.append(a,100)
b=np.append(b,200)
print(a)
print(b)
```

```
[ 10  20  30  40  50  60 100]
[ 10  20  30  40  50  60 200]
```

```
In [ ]:
```

```
In [2]: #Sorting the data of ndarray----by using sort()
import numpy as np
a=np.array([12,5,6,45,12,78,15,22,19])
print(a,type(a))

[12  5   6 45 12 78 15 22 19] <class 'numpy.ndarray'>

In [3]: np.sort(a)

Out[3]: array([ 5,  6, 12, 12, 15, 19, 22, 45, 78])

In [4]: print(a)

[12  5   6 45 12 78 15 22 19]

In [5]: a=np.sort(a)
print(a)

[ 5  6 12 12 15 19 22 45 78]

In [6]: print(a[::-1])

[78 45 22 19 15 12 12  6  5]

In [7]: b=a[::-1]
print(b,type(b))

[78 45 22 19 15 12 12  6  5] <class 'numpy.ndarray'>

In [8]: a=np.array([12,5,6,45,12,78,15,22,19])
a.shape=(3,3)
print(a,type(a))

[[12  5   6]
 [45 12 78]
 [15 22 19]] <class 'numpy.ndarray'>

In [9]: np.sort(a) # # Row wise Sorting default axis=1

Out[9]: array([[ 5,  6, 12],
 [12, 45, 78],
 [15, 19, 22]])

In [10]: print(a)

[[12  5   6]
 [45 12 78]
 [15 22 19]]
```

```
In [11]: np.sort(a,axis=0) # Columnwise Sorting xis for column is 0
```

```
Out[11]: array([[12, 5, 6],  
                 [15, 12, 19],  
                 [45, 22, 78]])
```

```
In [12]: print(a)
```

```
[[12 5 6]  
 [45 12 78]  
 [15 22 19]]
```

```
In [13]: np.sort(a,axis=1) # Rowwise Sorting axis=1
```

```
Out[13]: array([[ 5,  6, 12],  
                 [12, 45, 78],  
                 [15, 19, 22]])
```

```
In [14]: a.shape=(9,  
           print(a)
```

```
[12 5 6 45 12 78 15 22 19]
```

```
In [19]: a=np.array([12,5,6,45,12,78,15,22,19])  
print(a,type(a))
```

```
[12 5 6 45 12 78 15 22 19] <class 'numpy.ndarray'>
```

```
In [20]: np.sort(a)
```

```
Out[20]: array([ 5,  6, 12, 12, 15, 19, 22, 45, 78])
```

```
In [21]: np.sort(a)[::-1]
```

```
Out[21]: array([78, 45, 22, 19, 15, 12, 12, 6, 5])
```

```
In [ ]:
```

```
In [1]: #Linear Algebra
# x+y=2
# 2x+3y=6
#Find x and y values
```

```
In [2]: import numpy as np
a=np.array([[1,1],[2,3]])
b=np.array([2,6])
print(a,type(a))
print(b,type(b))
```

```
[[1 1]
 [2 3]] <class 'numpy.ndarray'>
[2 6] <class 'numpy.ndarray'>
```

```
In [5]: #Solving Linear Algebra equations
np.linalg.solve(a,b)
```

```
Out[5]: array([0., 2.])
```

```
In [6]: #Solving Linear Algebra equations
sol=np.linalg.solve(a,b)
print(sol,type(sol))
```

```
[0. 2.] <class 'numpy.ndarray'>
```

```
In [7]: print("Val of x=",sol[0])
print("Val of y=",sol[1])
```

```
Val of x= 0.0
Val of y= 2.0
```

```
In [9]: print("Type of linalg object=",type(np.linalg))
```

```
Type of linalg object= <class 'module'>
```

```
In [10]: a=np.array([[10,20,30],[40,50,60]])
print(a)
```

```
[[10 20 30]
 [40 50 60]]
```

```
In [11]: a.T # Obtaining Transpose and here T is an attribute in ndarray used for obtain
```

```
Out[11]: array([[10, 40],
 [20, 50],
 [30, 60]])
```

9/12/23, 10:45 AM

Linear algebra and transpose operations on ndarray - Jupyter Notebook

In [12]: a

Out[12]: array([[10, 20, 30],
[40, 50, 60]])

In [13]: a=a.T
print(a)

[[10 40]
[20 50]
[30 60]]

In [14]: #Another way to get transpose
np.transpose(a)

Out[14]: array([[10, 20, 30],
[40, 50, 60]])

In [15]: print(a)

[[10 40]
[20 50]
[30 60]]

In [16]: a=np.transpose(a)
print(a)

[[10 20 30]
[40 50 60]]

In []:

=====

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 maketing 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 structires 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 following

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 Function
called DataFrame() which is present in pandas Module and returns
an object of DataFrame class.
=>We have 5 Ways to create an object of DataFrame. They are
    a) By using list / tuple
    b) By using dict
    c) By using set type
    d) By using Series
    e) By using ndarray of numpy
    f) By using CSV File (Comma Separated Values)
-----
=>Syntax for creating an object of DataFrame in pandas:
-----
    varname=pandas.DataFrame(object,index,columns,dtype)
-----
Explanation:
-----
=>'varname' is an object of
<class,'pandas.core.dataframe.DataFrame'>
=>'pandas.DataFrame()' is a pre-defined function present in
pandas module and it is used to create an object of DataFrame
for storing Data sets.
=>'object' represents list (or) tuple (or) dict (or) Series (or)
ndarray (or) CSV file
=>'index' represents Row index and whose default indexing starts
from 0,1,...n-1 where 'n' represents number of values in
DataFrame object.
=>'columns' represents Column index whose default indexing
starts from 0,1..n-1 where n number of columns.
=>'dtype' represents data type of values of Column Value.
=====
Creating an object DataFrame by Using list / tuple
-----
>>>import pandas as pd
>>>lst=[10,20,30,40]
>>>df=pd.DataFrame(lst)
>>>print(df)
      0
0  10
1  20
2  30
3  40
-----
```

```

lst=[[10,20,30,40], ["RS","JS","MCK","TRV"] ]
df=pd.DataFrame(lst)
print(df)
      0    1    2    3
0   10   20   30   40
1    RS    JS   MCK   TRV
-----
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst)
print(df)
      0    1
0   10   RS
1   20   JG
2   30   MCK
3   40   TRA
-----
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst, index=[1,2,3,4],columns=['Rno','Name'])
print(df)

      Rno Name
1    10   RS
2    20   JG
3    30   MCK
4    40   TRA
-----
tpl=(("Rossum",75),("Gosling",85),("Travis",65),
      ("Ritche",95),("MCKinney",60) )
df=pd.DataFrame(tpl, index=[1,2,3,4,5],columns=['Name','Age'])
print(df)

      Name    Age
1    Rossum    75
2    Gosling    85
3    Travis     65
4    Ritche     95
5  MCKinney    60
-----
Creating an object DataFrame by Using dict object
-----
=>When we create an object of DataFrame by using Dict , all the
keys are taken as Column Names and Values of Value are taken as
Data.
-----
Examples:
-----
>>> import pandas as pd

```

```

>>>
dictdata= { "Names": ["Rossum", "Gosling", "Ritche", "McKinney"], "Subjects": ["Python", "Java", "C", "Pandas"], "Ages": [65, 80, 85, 55] }
>>> df=pd.DataFrame(dictdata)
>>> print(df)
      Names    Subjects   Ages
0    Rossum     Python     65
1   Gosling       Java     80
2    Ritche        C     85
3  McKinney     Pandas     55
>>> df=pd.DataFrame(dictdata,index=[1,2,3,4])
>>> print(df)
      Names    Subjects   Ages
1    Rossum     Python     65
2   Gosling       Java     80
3    Ritche        C     85
4  McKinney     Pandas     55
-----
Creating an object DataFrame by Using Series object
-----
>>> import pandas as pd
>>> sdata=pd.Series([10,20,30,40])
>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
0  10
1  20
2  30
3  40
>>>
sdata=pd.Series({ "IntMarks": [10,20,30,40], "ExtMarks": [80,75,65,50] })
>>> print(sdata)
IntMarks    [10, 20, 30, 40]
ExtMarks    [80, 75, 65, 50]
dtype: object

>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
IntMarks  [10, 20, 30, 40]
ExtMarks  [80, 75, 65, 50]
>>> ddata={"IntMarks": [10,20,30,40], "ExtMarks": [80,75,65,50]}
>>> df=pd.DataFrame(ddata)
>>> print(df)
   IntMarks  ExtMarks
0         10        80

```

```

1          20        75
2          30        65
3          40        50
-----
Creating an object DataFrame by Using ndarray object
-----
>>> import numpy as np
>>> l1=[[10,60],[20,70],[40,50]]
>>> a=np.array(l1)
>>> df=pd.DataFrame(a)
>>> print(df)
      0    1
0   10   60
1   20   70
2   40   50
>>> df=pd.DataFrame(a,columns=["IntMarks","ExtMarks"])
>>> print(df)
      IntMarks  ExtMarks
0            10        60
1            20        70
2            40        50
-----
```

e) By using CSV File (Comma Separated Values)

```

-
import pandas as pd1
df=pd1.read_csv("D:\KVR-JAVA\stud.csv")
print("type of df=",type(df)) #type of df= <class
'pandas.core.frame.DataFrame'>
print(df)
----- OUTPUT -----
      stno     name      marks
0      10    Rossum    45.67
1      20    Gosling   55.55
2      30    Ritche    66.66
3      40    Travis    77.77
4      50      KVR    11.11
```

```
In [1]: import pandas as pd
```

```
In [2]: df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [4]: df.shape
```

```
Out[4]: (17, 8)
```

```
In [5]: df.ndim
```

```
Out[5]: 2
```

```
In [6]: df.describe()
```

```
Out[6]:
```

	htno	telugu	english	hindi	maths	science	social
count	17.000000	17.000000	17.000000	17.000000	17.000000	17.000000	17.000000
mean	107.294118	54.000000	70.705882	65.764706	82.235294	69.588235	59.882353
std	4.687279	7.141428	11.671358	20.504662	13.917298	14.339887	14.696438
min	100.000000	45.000000	48.000000	34.000000	55.000000	44.000000	35.000000
25%	104.000000	50.000000	63.000000	56.000000	76.000000	66.000000	49.000000
50%	107.000000	53.000000	67.000000	64.000000	87.000000	67.000000	55.000000
75%	111.000000	56.000000	78.000000	77.000000	93.000000	77.000000	73.000000
max	115.000000	67.000000	89.000000	99.000000	99.000000	98.000000	88.000000

In [7]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [8]: `df.head()`

Out[8]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51

In [9]: `df.head(3)`

Out[9]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77

In [10]: `df.head(6)`

Out[10]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88

In [11]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [12]: `df.tail()`

Out[12]:

	htno	name	telugu	english	hindi	maths	science	social
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [13]: df.tail(2)
```

Out[13]:

	htno	name	telugu	english	hindi	maths	science	social
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [14]: df.tail(3)
```

Out[14]:

	htno	name	telugu	english	hindi	maths	science	social
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [15]: for record in df.iterrows():
    print(record)
```

```
Name: 14, dtype: object
(15, htno      114
 name      Rajesh
 telugu     45
 english     67
 hindi      77
 maths      55
 science     66
 social      46
)
Name: 15, dtype: object
(16, htno      115
 name      Rakesh
 telugu     67
 english     78
 hindi      88
 maths      78
 science     67
 social      49
)
Name: 16, dtype: object)
```

9/14/23, 10:48 AM

Accessing the Data from DataFrame - Jupyter Notebook

In [16]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [17]: `df[6:10]`

Out[17]:

	htno	name	telugu	english	hindi	maths	science	social
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58

In [18]: `df[::2]`

Out[18]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
2	102	Rossum	56	88	56	99	44	77
4	104	Kalyan	51	63	62	93	67	51
6	106	Kambli	53	81	59	92	48	73
8	108	Ganesh	53	62	76	88	76	35
10	110	Biswa	66	48	86	95	48	47
12	104	Kalyan	51	63	62	93	67	51
14	113	sonu	60	89	98	87	77	68
16	115	Rakesh	67	78	88	78	67	49

9/14/23, 10:48 AM

Accessing the Data from DataFrame - Jupyter Notebook

In [19]: df[1::2]

Out[19]:

	htno	name	telugu	english	hindi	maths	science	social
1	101	Rajesh	45	67	34	67	66	78
3	103	Raji	56	78	34	56	88	55
5	105	Karthik	48	62	39	68	65	88
7	107	Praveen	46	88	74	86	78	45
9	109	Nags	55	77	44	77	86	58
11	111	Ritchi	66	68	64	76	98	75
13	112	shareef	50	63	99	90	76	67
15	114	Rajesh	45	67	77	55	66	46

In [20]: df[::-1]

Out[20]:

	htno	name	telugu	english	hindi	maths	science	social
16	115	Rakesh	67	78	88	78	67	49
15	114	Rajesh	45	67	77	55	66	46
14	113	sonu	60	89	98	87	77	68
13	112	shareef	50	63	99	90	76	67
12	104	Kalyan	51	63	62	93	67	51
11	111	Ritchi	66	68	64	76	98	75
10	110	Biswa	66	48	86	95	48	47
9	109	Nags	55	77	44	77	86	58
8	108	Ganesh	53	62	76	88	76	35
7	107	Praveen	46	88	74	86	78	45
6	106	Kambli	53	81	59	92	48	73
5	105	Karthik	48	62	39	68	65	88
4	104	Kalyan	51	63	62	93	67	51
3	103	Raji	56	78	34	56	88	55
2	102	Rossum	56	88	56	99	44	77
1	101	Rajesh	45	67	34	67	66	78
0	100	Ramesh	50	60	66	98	66	55

In [21]: df[6:7]

Out[21]:

	htno	name	telugu	english	hindi	maths	science	social
6	106	Kambli	53	81	59	92	48	73

```
In [22]: df["name"]
```

```
Out[22]: 0      Ramesh
          1      Rajesh
          2      Rossum
          3      Raji
          4      Kalyan
          5      Karthik
          6      Kamblis
          7      Praveen
          8      Ganesh
          9      Nags
          10     Biswa
          11     Ritchi
          12     Kalyan
          13     shareef
          14     sonu
          15     Rajesh
          16     Rakesh
Name: name, dtype: object
```

```
In [23]: df[["name", "maths"]]
```

```
Out[23]:
```

	name	maths
0	Ramesh	98
1	Rajesh	67
2	Rossum	99
3	Raji	56
4	Kalyan	93
5	Karthik	68
6	Kamblis	92
7	Praveen	86
8	Ganesh	88
9	Nags	77
10	Biswa	95
11	Ritchi	76
12	Kalyan	93
13	shareef	90
14	sonu	87
15	Rajesh	55
16	Rakesh	78

```
In [24]: df[["name","maths","science"]]
```

Out[24]:

	name	maths	science
0	Ramesh	98	66
1	Rajesh	67	66
2	Rossum	99	44
3	Raji	56	88
4	Kalyan	93	67
5	Karthik	68	65
6	Kamblji	92	48
7	Praveen	86	78
8	Ganesh	88	76
9	Nags	77	86
10	Biswa	95	48
11	Ritchi	76	98
12	Kalyan	93	67
13	shareef	90	76
14	sonu	87	77
15	Rajesh	55	66
16	Rakesh	78	67

```
In [25]: df[["name","maths","science"]][10:11]
```

Out[25]:

	name	maths	science
10	Biswa	95	48

```
In [26]: df[["name","maths","science"]][10:13]
```

Out[26]:

	name	maths	science
10	Biswa	95	48
11	Ritchi	76	98
12	Kalyan	93	67

```
In [27]: df[["name","maths","science"]][10:13:-1]
```

Out[27]:

	name	maths	science
--	------	-------	---------

9/14/23, 10:48 AM

Accessing the Data from DataFrame - Jupyter Notebook

In [28]: `df[["name", "maths", "science"]][13:10:-1]`

Out[28]:

	name	maths	science
13	shareef	90	76
12	Kalyan	93	67
11	Ritchi	76	98

In []:

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)
```

Data Filtering and Conditional Change

```
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"] ]
```

Data Filtering and updatations on DataFrame object

```
3) dataframeobj.loc[ (compund condition), ["Col Name"]  
                    ]=Expression
```

```
Ex: df.loc[ (df["percent"]>=60)&(df["percent"]<=80) , ["grade"]  
      ]="First" # cond updation.
```

Removing Column Name from Data Frame

```
1) dataframe.drop(columns="col name")  
2) dataframe.drop(columns=["col name1", "colname2"....])  
3) dataframe.drop(columns="col name", inplace=True)  
4) dataframe.drop(columns=DataFrameObj.columns[Col Number])  
Example: data = data.drop(columns=data.columns[3])
```

Removing Row from Data Frame

```
1) df.drop(labels=index Value, axis=0)  
2) df.drop(labels=[index Value1, Index Value2...], axis=0)  
Examples: df.drop(labels=0, axis=0) # Here axis=0 refers to Rows  
Examples: df.drop(labels=4, axis=0) # Here axis=0 refers to Rows  
# Delete some chosen rows by row numbers - 2nd, 10th, 30th:  
        df.drop(df.index[[1, 9, 29]])
```

sorting the data of dataframe

```
1) dataframeobj.sort_values(["colname"])
```

```

2) dataframeobj.sort_values(["colname"], ascending=False)
=====
knowing the duplicates data in dataframe
=====
1) dataframeobj.duplicated() -----gives boolean result
=====
Removing duplicated data from dataframe
=====
1) dataframeobj.drop_duplicates()
2) dataframeobj.drop_duplicates(inplace=True)
=====
Adding new Row to Data Frame
=====
1) dataframeobj.loc[len(DataFrame)] =[Val1, Val2...Val-n]
=====
Exporting Data Frame data to CSV File and Excel Files
=====
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 xlsx file
    df.to_excel("C:\\KVR\\finalresult.xlsx")
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)

```

```
In [1]: # Adding New Column Name and Data Filtering and Conditional Change
```

```
In [2]: import pandas as pd
df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [3]: df["total"]=0
```

```
In [4]: print(df)
```

	htno	name	telugu	english	hindi	maths	science	social	total
0	100	Ramesh	50	60	66	98	66	55	0
1	101	Rajesh	45	67	34	67	66	78	0
2	102	Rossum	56	88	56	99	44	77	0
3	103	Raji	56	78	34	56	88	55	0
4	104	Kalyan	51	63	62	93	67	51	0
5	105	Karthik	48	62	39	68	65	88	0
6	106	Kambli	53	81	59	92	48	73	0
7	107	Praveen	46	88	74	86	78	45	0
8	108	Ganesh	53	62	76	88	76	35	0
9	109	Nags	55	77	44	77	86	58	0
10	110	Biswa	66	48	86	95	48	47	0
11	111	Ritchi	66	68	64	76	98	75	0
12	104	Kalyan	51	63	62	93	67	51	0
13	112	shareef	50	63	99	90	76	67	0
14	113	sonu	60	89	98	87	77	68	0
15	114	Rajesh	45	67	77	55	66	46	0
16	115	Rakesh	67	78	88	78	67	49	0

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

```
In [5]: l""] = df[\"telugu\"]+df[\"hindi\"]+df[\"english\"]+df[\"maths\"]+df[\"science\"]+df[\"social\"]
```

```
In [6]: print(df)
```

	htno	name	telugu	english	hindi	maths	science	social	total
0	100	Ramesh	50	60	66	98	66	55	395
1	101	Rajesh	45	67	34	67	66	78	357
2	102	Rossum	56	88	56	99	44	77	420
3	103	Raji	56	78	34	56	88	55	367
4	104	Kalyan	51	63	62	93	67	51	387
5	105	Karthik	48	62	39	68	65	88	370
6	106	Kambli	53	81	59	92	48	73	406
7	107	Praveen	46	88	74	86	78	45	417
8	108	Ganesh	53	62	76	88	76	35	390
9	109	Nags	55	77	44	77	86	58	397
10	110	Biswa	66	48	86	95	48	47	390
11	111	Ritchi	66	68	64	76	98	75	447
12	104	Kalyan	51	63	62	93	67	51	387
13	112	shareef	50	63	99	90	76	67	445
14	113	sonu	60	89	98	87	77	68	479
15	114	Rajesh	45	67	77	55	66	46	356
16	115	Rakesh	67	78	88	78	67	49	427

```
In [9]: df[\"percent\"]=round((df[\"total\"])/600)*100,2)
```

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [10]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total  \
0     100   Ramesh     50       60     66     98      66      55    395
1     101  Rajesh      45       67     34     67      66      78    357
2     102  Rossum      56       88     56     99      44      77    420
3     103   Raji       56       78     34     56      88      55    367
4     104  Kalyan      51       63     62     93      67      51    387
5     105  Karthik      48       62     39     68      65      88    370
6     106  Kambli      53       81     59     92      48      73    406
7     107  Praveen      46       88     74     86      78      45    417
8     108  Ganesh      53       62     76     88      76      35    390
9     109   Nags      55       77     44     77      86      58    397
10    110   Biswa      66       48     86     95      48      47    390
11    111  Ritchi      66       68     64     76      98      75    447
12    104  Kalyan      51       63     62     93      67      51    387
13    112  shareef      50       63     99     90      76      67    445
14    113   sonu      60       89     98     87      77      68    479
15    114  Rajesh      45       67     77     55      66      46    356
16    115  Rakesh      67       78     88     78      67      49    427

      percent
0      65.83
1      59.50
2      70.00
3      61.17
4      64.50
5      61.67
6      67.67
7      69.50
8      65.00
9      66.17
10     65.00
11     74.50
12     64.50
13     74.17
14     79.83
15     59.33
16     71.17
```

In [11]: `df.loc[df["percent"]>75]`

Out[11]:

htno	name	telugu	english	hindi	maths	science	social	total	percent	
14	113	sonu	60	89	98	87	77	68	479	79.83

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [14]: df.loc[df["percent"]>68]

Out[14]:

	htno	name	telugu	english	hindi	maths	science	social	total	percent
2	102	Rossum	56	88	56	99	44	77	420	70.00
7	107	Praveen	46	88	74	86	78	45	417	69.50
11	111	Ritchi	66	68	64	76	98	75	447	74.50
13	112	shareef	50	63	99	90	76	67	445	74.17
14	113	sonu	60	89	98	87	77	68	479	79.83
16	115	Rakesh	67	78	88	78	67	49	427	71.17

In [15]: df.loc[df["maths"]>90]

Out[15]:

	htno	name	telugu	english	hindi	maths	science	social	total	percent
0	100	Ramesh	50	60	66	98	66	55	395	65.83
2	102	Rossum	56	88	56	99	44	77	420	70.00
4	104	Kalyan	51	63	62	93	67	51	387	64.50
6	106	Kambli	53	81	59	92	48	73	406	67.67
10	110	Biswa	66	48	86	95	48	47	390	65.00
12	104	Kalyan	51	63	62	93	67	51	387	64.50

In [16]: df.loc[(df["maths"]>60) & (df["maths"]<85)]

Out[16]:

	htno	name	telugu	english	hindi	maths	science	social	total	percent
1	101	Rajesh	45	67	34	67	66	78	357	59.50
5	105	Karthik	48	62	39	68	65	88	370	61.67
9	109	Nags	55	77	44	77	86	58	397	66.17
11	111	Ritchi	66	68	64	76	98	75	447	74.50
16	115	Rakesh	67	78	88	78	67	49	427	71.17

In [17]: df.loc[(df["maths"]>90) & (df["maths"]<98)]

Out[17]:

	htno	name	telugu	english	hindi	maths	science	social	total	percent
4	104	Kalyan	51	63	62	93	67	51	387	64.50
6	106	Kambli	53	81	59	92	48	73	406	67.67
10	110	Biswa	66	48	86	95	48	47	390	65.00
12	104	Kalyan	51	63	62	93	67	51	387	64.50

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [18]: `df.loc[(df["maths"]>95) & (df["maths"]<=99),["name","maths"]]`

Out[18]:

	name	maths
0	Ramesh	98
2	Rossum	99

In [19]: `df.loc[(df["maths"]>95) & (df["maths"]<=99),["name","maths","total","percent"]]`

Out[19]:

	name	maths	total	percent
0	Ramesh	98	395	65.83
2	Rossum	99	420	70.00

In [20]: `#Updating the DataFrame Data`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [21]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total  \
0     100   Ramesh     50       60      66     98      66      55    395
1     101  Rajesh      45       67      34     67      66      78    357
2     102  Rossum      56       88      56     99      44      77    420
3     103   Raji       56       78      34      56     88      55    367
4     104  Kalyan      51       63      62     93      67      51    387
5     105  Karthik      48       62      39     68      65     88    370
6     106  Kambli      53       81      59     92      48      73    406
7     107  Praveen      46       88      74     86      78      45    417
8     108  Ganesh      53       62      76     88      76      35    390
9     109   Nags      55       77      44     77      86      58    397
10    110   Biswa      66       48      86     95      48      47    390
11    111  Ritchi      66       68      64     76     98      75    447
12    104  Kalyan      51       63      62     93      67      51    387
13    112  shareef      50       63      99     90      76      67    445
14    113   sonu      60       89      98     87      77      68    479
15    114  Rajesh      45       67      77     55      66      46    356
16    115  Rakesh      67       78      88     78      67      49    427

      percent
0      65.83
1      59.50
2      70.00
3      61.17
4      64.50
5      61.67
6      67.67
7      69.50
8      65.00
9      66.17
10     65.00
11     74.50
12     64.50
13     74.17
14     79.83
15     59.33
16     71.17
```

In [22]: `df["grade"] = None`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [23]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total \
0     100   Ramesh     50       60     66     98      66      55    395
1     101  Rajesh      45       67     34     67      66      78    357
2     102  Rossum      56       88     56     99      44      77    420
3     103   Raji       56       78     34     56      88      55    367
4     104  Kalyan      51       63     62     93      67      51    387
5     105  Karthik      48       62     39     68      65      88    370
6     106  Kambli      53       81     59     92      48      73    406
7     107  Praveen      46       88     74     86      78      45    417
8     108  Ganesh      53       62     76     88      76      35    390
9     109   Nags       55       77     44     77      86      58    397
10    110   Biswa      66       48     86     95      48      47    390
11    111  Ritchi      66       68     64     76      98      75    447
12    104  Kalyan      51       63     62     93      67      51    387
13    112  shareef      50       63     99     90      76      67    445
14    113   sonu       60       89     98     87      77      68    479
15    114  Rajesh      45       67     77     55      66      46    356
16    115  Rakesh      67       78     88     78      67      49    427

      percent  grade
0      65.83  None
1      59.50  None
2      70.00  None
3      61.17  None
4      64.50  None
5      61.67  None
6      67.67  None
7      69.50  None
8      65.00  None
9      66.17  None
10     65.00  None
11     74.50  None
12     64.50  None
13     74.17  None
14     79.83  None
15     59.33  None
16     71.17  None
```

In [24]: `df.loc[(df["percent"]>=70), ["grade"]]="Distinction"`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [25]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total \
0     100   Ramesh     50       60      66      98      66      55    395
1     101   Rajesh     45       67      34      67      66      78    357
2     102  Rossum     56       88      56      99      44      77    420
3     103    Raji      56       78      34      56      88      55    367
4     104  Kalyan      51       63      62      93      67      51    387
5     105  Karthik     48       62      39      68      65      88    370
6     106  Kambli     53       81      59      92      48      73    406
7     107  Praveen     46       88      74      86      78      45    417
8     108   Ganesh     53       62      76      88      76      35    390
9     109    Nags      55       77      44      77      86      58    397
10    110   Biswa      66       48      86      95      48      47    390
11    111   Ritchi     66       68      64      76      98      75    447
12    104   Kalyan     51       63      62      93      67      51    387
13    112  shareef     50       63      99      90      76      67    445
14    113    sonu      60       89      98      87      77      68    479
15    114   Rajesh     45       67      77      55      66      46    356
16    115   Rakesh     67       78      88      78      67      49    427

      percent      grade
0      65.83      None
1      59.50      None
2      70.00  Distinction
3      61.17      None
4      64.50      None
5      61.67      None
6      67.67      None
7      69.50      None
8      65.00      None
9      66.17      None
10     65.00      None
11     74.50  Distinction
12     64.50      None
13     74.17  Distinction
14     79.83  Distinction
15     59.33      None
16     71.17  Distinction
```

In [26]: `df.loc[(df["percent"]>=65) & (df["percent"]<70), ["grade"]]="First"`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [27]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total \
0     100   Ramesh      50       60      66      98      66      55    395
1     101   Rajesh      45       67      34      67      66      78    357
2     102  Rossum      56       88      56      99      44      77    420
3     103   Raji       56       78      34      56      88      55    367
4     104  Kalyan      51       63      62      93      67      51    387
5     105  Karthik      48       62      39      68      65      88    370
6     106  Kambli      53       81      59      92      48      73    406
7     107  Praveen      46       88      74      86      78      45    417
8     108   Ganesh      53       62      76      88      76      35    390
9     109   Nags       55       77      44      77      86      58    397
10    110   Biswa       66       48      86      95      48      47    390
11    111   Ritchi      66       68      64      76      98      75    447
12    104   Kalyan      51       63      62      93      67      51    387
13    112  shareef      50       63      99      90      76      67    445
14    113   sonu       60       89      98      87      77      68    479
15    114   Rajesh      45       67      77      55      66      46    356
16    115   Rakesh      67       78      88      78      67      49    427

      percent      grade
0      65.83    First
1      59.50    None
2      70.00  Distinction
3      61.17    None
4      64.50    None
5      61.67    None
6      67.67    First
7      69.50    First
8      65.00    First
9      66.17    First
10     65.00    First
11     74.50  Distinction
12     64.50    None
13     74.17  Distinction
14     79.83  Distinction
15     59.33    None
16     71.17  Distinction
```

In [28]: `df.loc[(df["percent"]>=60) & (df["percent"]<65), ["grade"]]="Second"`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [29]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total \
0     100   Ramesh     50       60     66     98      66      55    395
1     101   Rajesh     45       67     34     67      66      78    357
2     102  Rossum     56       88     56     99      44      77    420
3     103   Raji      56       78     34     56      88      55    367
4     104  Kalyan     51       63     62     93      67      51    387
5     105  Karthik     48       62     39     68      65      88    370
6     106  Kambli     53       81     59     92      48      73    406
7     107  Praveen     46       88     74     86      78      45    417
8     108   Ganesh     53       62     76     88      76      35    390
9     109   Nags      55       77     44     77      86      58    397
10    110   Biswa      66       48     86     95      48      47    390
11    111   Ritchi     66       68     64     76      98      75    447
12    104   Kalyan     51       63     62     93      67      51    387
13    112  shareef     50       63     99     90      76      67    445
14    113   sonu      60       89     98     87      77      68    479
15    114   Rajesh     45       67     77     55      66      46    356
16    115   Rakesh     67       78     88     78      67      49    427

      percent      grade
0      65.83    First
1      59.50    None
2      70.00  Distinction
3      61.17    Second
4      64.50    Second
5      61.67    Second
6      67.67    First
7      69.50    First
8      65.00    First
9      66.17    First
10     65.00    First
11     74.50  Distinction
12     64.50    Second
13     74.17  Distinction
14     79.83  Distinction
15     59.33    None
16     71.17  Distinction
```

In [31]: `df.loc[(df["percent"]<60), ["grade"]]="Third"`

9/15/23, 10:36 AM

Adding new Column Name and Data Filtering and Conditional Change - Jupyter Notebook

In [32]: `print(df)`

```
      htno    name  telugu  english  hindi  maths  science  social  total \
0     100   Ramesh     50       60     66     98      66      55    395
1     101  Rajesh     45       67     34     67      66      78    357
2     102  Rossum     56       88     56     99      44      77    420
3     103   Raji      56       78     34     56      88      55    367
4     104  Kalyan     51       63     62     93      67      51    387
5     105  Karthik     48       62     39     68      65      88    370
6     106  Kambli     53       81     59     92      48      73    406
7     107  Praveen     46       88     74     86      78      45    417
8     108   Ganesh     53       62     76     88      76      35    390
9     109    Nags      55       77     44     77      86      58    397
10    110   Biswa      66       48     86     95      48      47    390
11    111  Ritchi      66       68     64     76      98      75    447
12    104  Kalyan     51       63     62     93      67      51    387
13    112  shareef     50       63     99     90      76      67    445
14    113    sonu      60       89     98     87      77      68    479
15    114  Rajesh     45       67     77     55      66      46    356
16    115  Rakesh     67       78     88     78      67      49    427

      percent      grade
0      65.83    First
1      59.50    Third
2      70.00  Distinction
3      61.17    Second
4      64.50    Second
5      61.67    Second
6      67.67    First
7      69.50    First
8      65.00    First
9      66.17    First
10     65.00    First
11     74.50  Distinction
12     64.50    Second
13     74.17  Distinction
14     79.83  Distinction
15     59.33    Third
16     71.17  Distinction
```

In [33]: `#Export the DataFrame Data to CSV File
df.to_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\\\FinalResult.csv")`

In []:

```
In [1]: #understanding about loc[]
```

```
In [2]: import pandas as pd
df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [3]: df.loc[6]
```

```
Out[3]: htno      106
         name     Kambli
         telugu    53
         english   81
         hindi     59
         maths     92
         science   48
         social    73
Name: 6, dtype: object
```

```
In [4]: df.loc[10]
```

```
Out[4]: htno      110
         name     Biswa
         telugu   66
         english  48
         hindi    86
         maths    95
         science  48
         social   47
Name: 10, dtype: object
```

9/15/23, 10:36 AM

Operations by using loc[] and iloc[] - Jupyter Notebook

In [5]: df.loc[6,['maths']]

Out[5]: maths 92
Name: 6, dtype: object

In [6]: df.loc[6,['name','maths']]

Out[6]: name Kambli
maths 92
Name: 6, dtype: object

In [7]: df.loc[6,['name','maths','science']]

Out[7]: name Kambli
maths 92
science 48
Name: 6, dtype: object

In [8]: df.loc[4:6]

Out[8]:

	htno	name	telugu	english	hindi	maths	science	social
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73

In [9]: df.loc[::-2]

Out[9]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
2	102	Rossum	56	88	56	99	44	77
4	104	Kalyan	51	63	62	93	67	51
6	106	Kambli	53	81	59	92	48	73
8	108	Ganesh	53	62	76	88	76	35
10	110	Biswa	66	48	86	95	48	47
12	104	Kalyan	51	63	62	93	67	51
14	113	sonu	60	89	98	87	77	68
16	115	Rakesh	67	78	88	78	67	49

9/15/23, 10:36 AM

Operations by using loc[] and iloc[] - Jupyter Notebook

In [10]: `df.loc[:,["name","maths","social"]]`

Out[10]:

	name	maths	social
0	Ramesh	98	55
1	Rajesh	67	78
2	Rossum	99	77
3	Raji	56	55
4	Kalyan	93	51
5	Karthik	68	88
6	Kambli	92	73
7	Praveen	86	45
8	Ganesh	88	35
9	Nags	77	58
10	Biswa	95	47
11	Ritchi	76	75
12	Kalyan	93	51
13	shareef	90	67
14	sonu	87	68
15	Rajesh	55	46
16	Rakesh	78	49

In [11]: `df.loc[4:8,['name','maths','social']]`

Out[11]:

	name	maths	social
4	Kalyan	93	51
5	Karthik	68	88
6	Kambli	92	73
7	Praveen	86	45
8	Ganesh	88	35

9/15/23, 10:36 AM

Operations by using loc[] and iloc[] - Jupyter Notebook

In [12]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [13]: `df.loc[0:11,"htno":"maths"]`

Out[13]:

	htno	name	telugu	english	hindi	maths
0	100	Ramesh	50	60	66	98
1	101	Rajesh	45	67	34	67
2	102	Rossum	56	88	56	99
3	103	Raji	56	78	34	56
4	104	Kalyan	51	63	62	93
5	105	Karthik	48	62	39	68
6	106	Kambli	53	81	59	92
7	107	Praveen	46	88	74	86
8	108	Ganesh	53	62	76	88
9	109	Nags	55	77	44	77
10	110	Biswa	66	48	86	95
11	111	Ritchi	66	68	64	76

9/15/23, 10:36 AM

Operations by using loc[] and iloc[] - Jupyter Notebook

In [14]: `df.loc[0:6,"htno":"maths":2]`

Out[14]:

	htno	telugu	hindi
0	100	50	66
1	101	45	34
2	102	56	56
3	103	56	34
4	104	51	62
5	105	48	39
6	106	53	59

In [15]: `df.loc[0:6,"name":"maths":2]`

Out[15]:

	name	english	maths
0	Ramesh	60	98
1	Rajesh	67	67
2	Rossum	88	99
3	Raji	78	56
4	Kalyan	63	93
5	Karthik	62	68
6	Kambli	81	92

In [16]: `#Understanding iloc[]`

In [17]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [18]: df.iloc[5]
```

```
Out[18]: htno      105
          name     Karthik
          telugu     48
          english    62
          hindi      39
          maths      68
          science    65
          social     88
          Name: 5, dtype: object
```

```
In [19]: df.iloc[10]
```

```
Out[19]: htno      110
          name     Biswa
          telugu     66
          english    48
          hindi      86
          maths      95
          science    48
          social     47
          Name: 10, dtype: object
```

```
In [22]: df.iloc[5:8]
```

```
Out[22]:   htno   name  telugu  english  hindi  maths  science  social
          5    105  Karthik     48       62     39      68      65      88
          6    106  Kamibli    53       81     59      92      48      73
          7    107  Praveen     46      88     74      86      78      45
```

```
In [25]: df.iloc[5,1]
```

```
Out[25]: 'Karthik'
```

```
In [26]: df.iloc[5,[1,3,5]]
```

```
Out[26]: name      Karthik
          english    62
          maths      68
          Name: 5, dtype: object
```

```
In [5]: import pandas as pd
df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [6]: df.iloc[3:8,1:6]
```

Out[6]:

	name	telugu	english	hindi	maths
3	Raji	56	78	34	56
4	Kalyan	51	63	62	93
5	Karthik	48	62	39	68
6	Kambli	53	81	59	92
7	Praveen	46	88	74	86

```
In [7]: df.iloc[10:15,1]
```

Out[7]: 10 Biswa
11 Ritchi
12 Kalyan
13 shareef
14 sonu
Name: name, dtype: object

9/15/23, 10:36 AM

Operations by using loc[] and iloc[] - Jupyter Notebook

In [8]: `df.iloc[[2,12,4,8]]`

Out[8]:

	htno	name	telugu	english	hindi	maths	science	social
2	102	Rossum	56	88	56	99	44	77
12	104	Kalyan	51	63	62	93	67	51
4	104	Kalyan	51	63	62	93	67	51
8	108	Ganesh	53	62	76	88	76	35

In [9]: `df.iloc[2:4,[1,3,5,7]]`

Out[9]:

	name	english	maths	social
2	Rossum	88	99	77
3	Raji	78	56	55

In [10]: `df.iloc[:,[1,6,7]]`

Out[10]:

	name	science	social
0	Ramesh	66	55
1	Rajesh	66	78
2	Rossum	44	77
3	Raji	88	55
4	Kalyan	67	51
5	Karthik	65	88
6	Kamblu	48	73
7	Praveen	78	45
8	Ganesh	76	35
9	Nags	86	58
10	Biswa	48	47
11	Ritchi	98	75
12	Kalyan	67	51
13	shareef	76	67
14	sonu	77	68
15	Rajesh	66	46
16	Rakesh	67	49

```
In [1]: #Removing Column Name from Data Frame
```

```
import pandas as pd
df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [2]: f["total"]=df["telugu"]+df["hindi"]+df["english"]+df["maths"]+df["science"]+df["social"]
```

```
In [3]: print(df)
```

	htno	name	telugu	english	hindi	maths	science	social	total
0	100	Ramesh	50	60	66	98	66	55	395
1	101	Rajesh	45	67	34	67	66	78	357
2	102	Rossum	56	88	56	99	44	77	420
3	103	Raji	56	78	34	56	88	55	367
4	104	Kalyan	51	63	62	93	67	51	387
5	105	Karthik	48	62	39	68	65	88	370
6	106	Kambli	53	81	59	92	48	73	406
7	107	Praveen	46	88	74	86	78	45	417
8	108	Ganesh	53	62	76	88	76	35	390
9	109	Nags	55	77	44	77	86	58	397
10	110	Biswa	66	48	86	95	48	47	390
11	111	Ritchi	66	68	64	76	98	75	447
12	104	Kalyan	51	63	62	93	67	51	387
13	112	shareef	50	63	99	90	76	67	445
14	113	sonu	60	89	98	87	77	68	479
15	114	Rajesh	45	67	77	55	66	46	356
16	115	Rakesh	67	78	88	78	67	49	427

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [4]: `df.drop(columns="total")`

out[4]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kamblji	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [5]: df

out[5]:

	htno	name	telugu	english	hindi	maths	science	social	total
0	100	Ramesh	50	60	66	98	66	55	395
1	101	Rajesh	45	67	34	67	66	78	357
2	102	Rossum	56	88	56	99	44	77	420
3	103	Raji	56	78	34	56	88	55	367
4	104	Kalyan	51	63	62	93	67	51	387
5	105	Karthik	48	62	39	68	65	88	370
6	106	Kambli	53	81	59	92	48	73	406
7	107	Praveen	46	88	74	86	78	45	417
8	108	Ganesh	53	62	76	88	76	35	390
9	109	Nags	55	77	44	77	86	58	397
10	110	Biswa	66	48	86	95	48	47	390
11	111	Ritchi	66	68	64	76	98	75	447
12	104	Kalyan	51	63	62	93	67	51	387
13	112	shareef	50	63	99	90	76	67	445
14	113	sonu	60	89	98	87	77	68	479
15	114	Rajesh	45	67	77	55	66	46	356
16	115	Rakesh	67	78	88	78	67	49	427

In [6]: df.drop(columns="total", inplace=True)

In [7]: print(df)

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [8]: df.drop(columns=["hindi", "science"], inplace=True)

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [9]: `print(df)`

	htno	name	telugu	english	maths	social
0	100	Ramesh	50	60	98	55
1	101	Rajesh	45	67	67	78
2	102	Rossum	56	88	99	77
3	103	Raji	56	78	56	55
4	104	Kalyan	51	63	93	51
5	105	Karthik	48	62	68	88
6	106	Kambli	53	81	92	73
7	107	Praveen	46	88	86	45
8	108	Ganesh	53	62	88	35
9	109	Nags	55	77	77	58
10	110	Biswa	66	48	95	47
11	111	Ritchi	66	68	76	75
12	104	Kalyan	51	63	93	51
13	112	shareef	50	63	90	67
14	113	sonu	60	89	87	68
15	114	Rajesh	45	67	55	46
16	115	Rakesh	67	78	78	49

In [10]: `df = df.drop(columns=df.columns[2])`

In [11]: `print(df)`

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [12]: `#Removing Row from Data Frame`

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [13]: `print(df)`

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [14]: `df.drop(labels=0, axis=0)`

Out[14]:

	htno	name	english	maths	social
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [15]: `df.drop(labels=0, axis=0)`

Out[15]:

	htno	name	english	maths	social
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [16]: `df.drop(labels=14, axis=0)`

Out[16]:

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [17]: `df.drop(df.index[[1,3,5]])`

Out[17]:

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
2	102	Rossum	88	99	77
4	104	Kalyan	63	93	51
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [18]: `print(df)`

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [19]: `#sorting the data of dataframe`

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [20]: `print(df)`

	htno	name	english	maths	social
0	100	Ramesh	60	98	55
1	101	Rajesh	67	67	78
2	102	Rossum	88	99	77
3	103	Raji	78	56	55
4	104	Kalyan	63	93	51
5	105	Karthik	62	68	88
6	106	Kambli	81	92	73
7	107	Praveen	88	86	45
8	108	Ganesh	62	88	35
9	109	Nags	77	77	58
10	110	Biswa	48	95	47
11	111	Ritchi	68	76	75
12	104	Kalyan	63	93	51
13	112	shareef	63	90	67
14	113	sonu	89	87	68
15	114	Rajesh	67	55	46
16	115	Rakesh	78	78	49

In [21]: `df.sort_values(["maths"])`

Out[21]:

	htno	name	english	maths	social
15	114	Rajesh	67	55	46
3	103	Raji	78	56	55
1	101	Rajesh	67	67	78
5	105	Karthik	62	68	88
11	111	Ritchi	68	76	75
9	109	Nags	77	77	58
16	115	Rakesh	78	78	49
7	107	Praveen	88	86	45
14	113	sonu	89	87	68
8	108	Ganesh	62	88	35
13	112	shareef	63	90	67
6	106	Kambli	81	92	73
4	104	Kalyan	63	93	51
12	104	Kalyan	63	93	51
10	110	Biswa	48	95	47
0	100	Ramesh	60	98	55
2	102	Rossum	88	99	77

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

```
In [22]: df.sort_values(["maths"], ascending=False)
```

Out[22]:

	htno	name	english	maths	social
2	102	Rossum	88	99	77
0	100	Ramesh	60	98	55
10	110	Biswa	48	95	47
4	104	Kalyan	63	93	51
12	104	Kalyan	63	93	51
6	106	Kambli	81	92	73
13	112	shareef	63	90	67
8	108	Ganesh	62	88	35
14	113	sonu	89	87	68
7	107	Praveen	88	86	45
16	115	Rakesh	78	78	49
9	109	Nags	77	77	58
11	111	Ritchi	68	76	75
5	105	Karthik	62	68	88
1	101	Rajesh	67	67	78
3	103	Raji	78	56	55
15	114	Rajesh	67	55	46

```
In [23]: df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\studentmarks1.csv")
print(df)
```

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

```
In [24]: #knowing the duplicates data in dataframe
#Removing duplicated data from dataframe
```

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [25]: `df.duplicated()`

Out[25]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
10   False
11   False
12   True
13   False
14   False
15   False
16   False
dtype: bool
```

In [26]: `df.drop_duplicates()`

Out[26]:

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kamblji	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [27]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	104	Kalyan	51	63	62	93	67	51
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	115	Rakesh	67	78	88	78	67	49

In [28]: `df.drop_duplicates(inplace=True)`

In [29]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
12	112	shareef	50	63	99	90	76	67
13	113	sonu	60	89	98	87	77	68
14	114	Rajesh	45	67	77	55	66	46
15	115	Rakesh	67	78	88	78	67	49

In [30]: `#Adding new Row to Data Frame`

In [31]: `len(df)`

Out[31]: 16

In [32]: `df.loc[len(df)]=[117, "KVR", 66, 77, 44, 66, 77, 78]`

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [33]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	117	KVR	66	77	44	66	77	78

In [34]: `df.loc[len(df)]=[118,"Nilesh",66,77,44,66,77,78]`

In [35]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	118	Nilesh	66	77	44	66	77	78

9/15/23, 10:35 AM

Removing Column Name from Data Frame - Jupyter Notebook

In [36]: `df.sort_values(["name"])`

Out[36]:

	htno	name	telugu	english	hindi	maths	science	social
10	110	Biswa	66	48	86	95	48	47
8	108	Ganesh	53	62	76	88	76	35
4	104	Kalyan	51	63	62	93	67	51
6	106	Kambli	53	81	59	92	48	73
5	105	Karthik	48	62	39	68	65	88
9	109	Nags	55	77	44	77	86	58
16	118	Nilesh	66	77	44	66	77	78
7	107	Praveen	46	88	74	86	78	45
1	101	Rajesh	45	67	34	67	66	78
15	114	Rajesh	45	67	77	55	66	46
3	103	Raji	56	78	34	56	88	55
0	100	Ramesh	50	60	66	98	66	55
11	111	Ritchi	66	68	64	76	98	75
2	102	Rossum	56	88	56	99	44	77
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68

In [37]: `print(df)`

	htno	name	telugu	english	hindi	maths	science	social
0	100	Ramesh	50	60	66	98	66	55
1	101	Rajesh	45	67	34	67	66	78
2	102	Rossum	56	88	56	99	44	77
3	103	Raji	56	78	34	56	88	55
4	104	Kalyan	51	63	62	93	67	51
5	105	Karthik	48	62	39	68	65	88
6	106	Kambli	53	81	59	92	48	73
7	107	Praveen	46	88	74	86	78	45
8	108	Ganesh	53	62	76	88	76	35
9	109	Nags	55	77	44	77	86	58
10	110	Biswa	66	48	86	95	48	47
11	111	Ritchi	66	68	64	76	98	75
13	112	shareef	50	63	99	90	76	67
14	113	sonu	60	89	98	87	77	68
15	114	Rajesh	45	67	77	55	66	46
16	118	Nilesh	66	77	44	66	77	78

In []:

=====

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:

Name	Age	Experience
------	-----	------------

```
City
Delhi    Riti    41      17
Mumbai   Aadi    46      11
Sydney   Jack    44      19
```

There were three unique values in the column "City", therefore 3 groups were created. The first() function fetched the first row of each of the Group and returned a DataFrame populated with that. The returned DataFrame has a row for each of the city and it is the first row from each of the city groups.

Get the count of number of DataFrame Groups

The DataFrameGroupBy object also provides a function size(), and it returns the count of rows in each of the groups created by the groupby() function. For example,

```
# Get the size of DataFrame groups
print(df.groupby('City').size())
```

Output:

```
Delhi      4
Mumbai    3
Sydney    3
dtype: int64
```

As there were three unique values in the column "City", therefore 3 groups were created by groupby() function. The size() function returned a Series containing the count of number of rows for each of the group.

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 .

```
#Non-IterEx1.py
lst=[10,"Ramesh",33.33,"HYD","Python",2+3j]# Iterable object
print(lst)# Here Total Object Data available at one Place and
leads to More Memory.
```

```
#IterEx1.py
lst=[10,"Ramesh",33.33,"HYD","Python",2+3j]# Iterable object
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
listiter=iter(lst)
print("Type of listiter=",type(listiter)) #<class,List_Iterator>
print(next(listiter))
print(next(listiter))
print(next(listiter))
while(True):
    try:
        print(next(listiter))
    except StopIteration:
        break
```

```
#IterEx2.py
tpl=(10,"Ramesh",33.33,"HYD","Python",2+3j) # Iterable object
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
tpliter=iter(tpl)
print("Type of tpliter=",type(tpliter)) #<class,tuple_Iterator>
while(True):
    try:
        print(next(tpliter))
    except StopIteration:
        break
```

```
#IterEx3.py
```

```

st={10,"Ramesh",33.33,"HYD","Python",2+3j} # Iterable object
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
stiter=iter(st)
print("Type of stiter=",type(stiter)) #<class,Set_Iterator>
while(True):
    try:
        print(next(stiter))
    except StopIteration:
        break


---


#IterEx4.py
d={10:"Apple",20:"Mango",30:"Kiwi",40:"Sberry"}
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
diter=iter(d)
print("Type of diter=",type(diter)) #<class,Dict_KeyIterator>
while(True):
    try:
        k=next(diter)
        print("{}\t{}".format(k,d[k]))
    except StopIteration:
        break


---


#IterEx5.py
s="PYTHON PROG"
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
siter=iter(s)
print("Type of siter=",type(siter)) #<class,Str_Iterator>
while(True):
    try:
        print("{}".format(next(siter)))
    except StopIteration:
        break


---


#IterEx6.py
r=range(10,51,10)
#Convert Iterable Object into Iterator Object by using iter()
#Syntax: iteratorobj=iter(Iterable object)
riter=iter(r)
print("Type of riter=",type(riter)) #<class,range_Iterator>
while(True):
    try:
        print("{}".format(next(riter)))
    except StopIteration:
        break

```

=====

Multi Threading in Python--3 Days--Most Imp

=====

=====

Types of Applications in 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

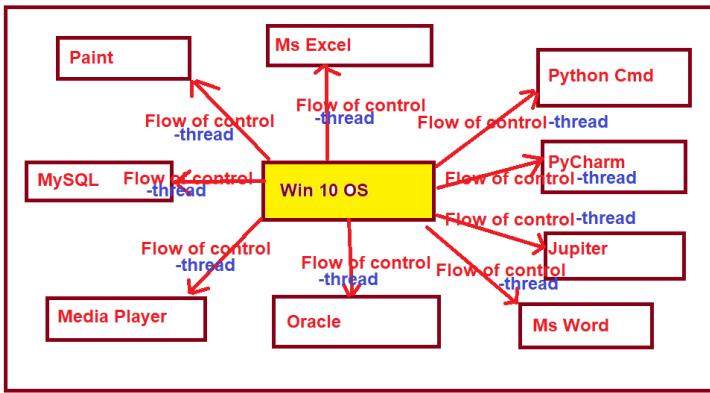
=>Thread Based Applications contains by default Single Thread and Programtically we can create Multiple Threads.

=>Therad Based Applications Provides Both Sequential Execution and Concurrent Execution.

=>Thread Based Applications Takes Less Execution Time

=>Thread Based Applications are treated as Light Weight Components.

=>Examples: Python, Java, C#.net...etc



Introduction to Thread Based Applications

=>The purpose of multi threading is that "To provide Concurrent / Simultaneous execution / Parallel Execution".

=>Concurrent Execution is nothing but executing the operations all at once.

=>The advantage of Concurrent execution is that to get less execution time.

=>If a Python Program contains multiple threads then it is called Multi Threading program.

=>Def. of thread:

=>A Flow of Control is called thread.

=>The purpose of thread is that "To Perform certain operation whose logic developed in Functions / Methods concurrently."

=>By default Every Python contains Single Thread and whose name is "MainThread" and It provides Sequential Execution.

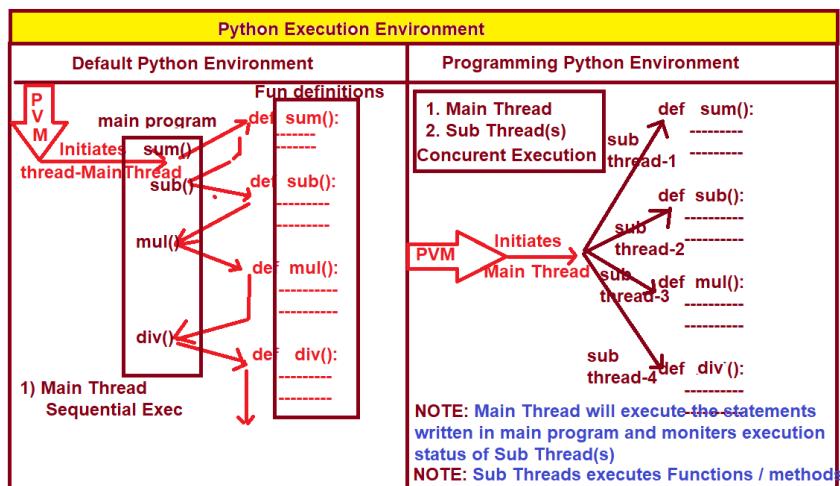
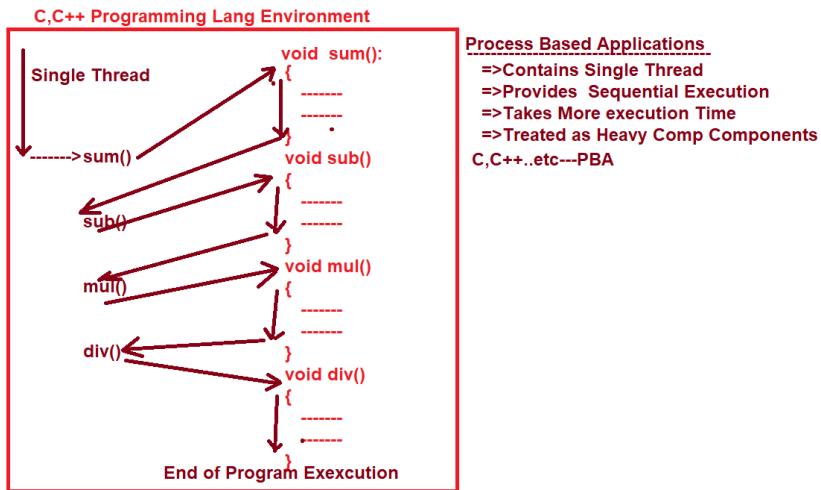
=>Programmatically, In a Python Program we can create multiple sub / Child threads and whose purpose is that "To execute operations whose logic is written in Functions / Methods Concurrently".

=>Hence Programmatically a Python Program contains two types of Threads. They are

- MainThread
- 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 ".



```
#WithoutSubThreadsEx1.py
import threading, time
def squares(lst):
```

```

        for val in lst:
            print("{}--"
>squares({})={}".format(threading.current_thread().name,val,val*
*2))
            time.sleep(1)

def cubes(lst):
    for val in lst:
        print("{}--"
>cubes({})={}".format(threading.current_thread().name,val,val**3
))
        time.sleep(1)
#main program
bt=time.time()
print("Program execution started---executed
      by:",threading.current_thread().name)
print("-----")
lst=[2,14,15,6,-8,12,19,22]
squares(lst)
cubes(lst)
print("-----")
print("Program execution ended---executed
      by:",threading.current_thread().name)
et=time.time()
print("Total Execution of non-threads Prog={}".format(et-bt))
#Total Execution of non-threads Prog=16.169081687927246


---


#WithSubThreadsEx1.py
import threading,time
def squares(lst):
    for val in lst:
        print("{}--"
>squares({})={}".format(threading.current_thread().name,val,val*
*2))
        time.sleep(1)

def cubes(lst):
    for val in lst:
        print("{}--"
>cubes({})={}".format(threading.current_thread().name,val,val**3
))
        time.sleep(1)
#main program
bt=time.time()
print("Program execution started---executed
      by:",threading.current_thread().name)
print("-----")
lst=[2,14,15,6,-8,12,19,22]

```

```

#create Two Sub Threads
t1=threading.Thread(target=squares,args=(lst,))# t1-->Thread-1
t2=threading.Thread(target=cubes,args=(lst,))# t2-->Thread-2
t1.start()
t2.start()
t1.join()
t2.join()
print("-----")
print("Program execution ended---executed
      by:",threading.current_thread().name)
et=time.time()
print("Total Execution of sub threads Prog={}".format(et-bt))
#Total Execution of non-threads Prog=16.169081687927246
=====
#Program for obtaininf default threads name
import threading
print("Default Name of the
therad=",threading.current_thread().name)
print("Number of therads in Python
Prog=",threading.active_count())
=====
#Program demonstrating default thread executes the functions one
by one
import threading
def hello():
    print("i am from hello()--executed
          by:",threading.current_thread().name)
def hi():
    print("i am from hi()--executed
          by:",threading.current_thread().name)
def welcome():
    print("i am from welcome()--executed
          by:",threading.current_thread().name)
#main program
print("Program execution started---executed
      by:",threading.current_thread().name)
print("-----")
hello()
hi()
welcome()
print("-----")
print("Program execution Ended--executed
      by:",threading.current_thread().name)
=====

Module Name Required for dealing with Thread based Applications
=====
=>The Module Name Required for dealing with Thread based
Applications is "threading".
=====
```

Module Name: threading

Functions in threading module

1) current_thread(): This Function is used for Obtaining Name of the thread, which is currently executing:

Syntax: `threadnameobj=threading.current_thread().name`

2) active_count() : This Function is used for Counting Number of active threads.

Syntax: `NoActivetherads=threading.active_count()`

Class Names in threading Module

- 1) Thread Class
- 2) Lock Class

1) Thread Class : The purpose of Thread is that to create sub threads for getting Concurrent Execution

a) Thread(target,args):

Syntax:

```
threadobj=threading.Thread(target=Function / Method name ,
                           args=(Val1,Val2,...,Val-n))
```

=>This Constructor is used for creating an object of Thread Class (called Sub/Child Thread) and by targetting the Function Name where it contains Logic of Python Executed by Sub Thread.

Examples: `t1=threading.Thread(target=squares,args=(obj,))`
`t2=threading.Thread(target=cubes,args=(obj,))`

b) start()

=>This Function is used for Dispatching the sub threads (Sending the target Function to execute)

=>Syntax: `threadobj.start()`

c) setName(str value)

=>This Function is used for Setting the Name of thread. This Function Deprecated on the name of an attribute "name".

Syntax: `threadobj.setName(Str Value) # Not Recommended`
(OR)
`threadobj.name=ThreadName in str format`
#Recommended

d) getName()

=>This Function is used for Obtaining the Name of thread. This Function Deprecated on the name of an attribute "name".

Syntax: `threadname=threadobj.getName() # Not Recommended
(OR)`
 `threadname=threadobj.name # Recommended`

e) `is_alive()`

=>This Function returns True Provided the Sub Thread is under execution.

=>This Function returns False Provided the Sub Thread is not under execution.

=>Syntax: `threadobj.is_alive()`

f) `join()`

=>This Function is used for Joining the Sub Thread(s) with MainThread after Complete Execution of Thread.

=>Syntax: `SubThreadobj1.join()
 SubThreadobj2.join()`

`SubThreadobj-n.join()`

Number of approaches to Therad Based Applications

=>In Python Programming, we have Two approaches to Develop Therad Based Applications. They are

1. By using Functional Programming
2. By Using Object Oriented Programming

1. By using Functional Programming

Steps

Step-1: import threading and Other Modules if Required.

Step-2: Define a Function, where It Contains Logic of Python Program, which is executed by Sub Thread.

Step-3: Create Sub Thread(s) for Executing the Target Function.

Step-4: Dispatch the Sub Thread(s) for Executing the Target Function.

2. By Using Object Oriented Programming

Steps

Step-1: import threading and Other Modules if Required.

Step-2: Choose the Programmer-Defined Class

Step-3: Define a Method in Programmer-Defined Class ,where It
----- Contains Logic of Python Program, which is executed by
SubThread.
Step-4: Create Sub Thread(s) for Executing the Target Function.

Step-5: Dispatch the Sub Thread(s) for Executing the Target
----- Function.

```
#program for demonstrating couting number of threads
#ActiveThreadCountEx.py
import threading
print("Number of Active Threads=",threading.active_count())
print("Name of Active Thread=",threading.current_thread().name)


---


#program for demonstrating current thread
#CurrentThreadEx1.py
import threading
th=threading.current_thread()
print(th) # <_MainThread(MainThread, started 9856)>
print("Default Thread Name:",th.name) # MainThread
print("=====OR=====")
print("Default Thread Name:",threading.current_thread().name)


---


#program for determining whether the sub thread is under
execution or not
#IsAliveThreadsEx1.py
import threading,time
def hello():
    print("hello()--executed by :",
          threading.current_thread().name)
    time.sleep(10)
    print("Sub Thread came out-of of sleep after 10 seconds")
#main program
print("Program execution
      started:",threading.current_thread().name)
#create sub thread
t1=threading.Thread(target=hello)
t1.name="KVR"
print("Active Number of Threads=",threading.active_count())#1
print("Execution status of Sub Thread before
      start=",t1.is_alive())#False
t1.start()
print("Execution status of Sub Thread after
      start=",t1.is_alive())#False
```

```

print("Active Number of Threads=",threading.active_count())#2
print("\nProgram execution
      Ended:",threading.current_thread().name)


---


#JoinSubThreadsEx1.py
import threading,time
def hello():
    print("hello()--executed by :",
threading.current_thread().name)
    time.sleep(10)
    print("Sub Thread came out-of sleep after 10 seconds")
#main program
print("Program execution
      started:",threading.current_thread().name)
#create sub thread
t1=threading.Thread(target=hello)
t1.name="KVR"
print("Active Number of Threads=",threading.active_count())#1
print("Execution status of Sub Thread before
start=",t1.is_alive())#False
t1.start()
print("Execution status of Sub Thread after
start=",t1.is_alive())#True
print("Active Number of Threads=",threading.active_count())#2
#Join Sub Threads with MainThread
t1.join()
print("Execution status of Sub Thread after
      join=",t1.is_alive())#False
print("Active Number of Threads=",threading.active_count())#1
print("\nProgram execution
      Ended:",threading.current_thread().name)


---


#Thread Based Application for generating 1 to n after each and
every second
#NumGenFunEx1.py
import threading,time#Step-1
def generate(n): # Step-2
    print("generate() executed
          by:{}".format(threading.current_thread().name))
    if(n<=0):
        print("{} Is Invalid Input".format(n))
    else:
        print("Numbers within:{}".format(n))
        for i in range(1,n+1):
            print("\tVal of i:{}".format(i))
            time.sleep(1)
#main program
#Create Sub Thread

```

```

t1=threading.Thread(target=generate,args=(int(input("Enter How
    Many Numbers u want to generate:"))),) # Step-3
t1.start() # Step-4


---


#Thread Based Application for generating 1 to n after each and
every second
#NumGenOOPEx1.py
import threading,time#Step-1
class Numbers: # Step-2
    def generate(self,n): # Step-3
        print("generate() executed
              by:{}".format(threading.current_thread().name))
        if(n<=0):
            print("{} Is Invalid Input".format(n))
        else:
            print("Numbers within:{}".format(n))
            for i in range(1,n+1):
                print("\tVal of i:{}".format(i))
                time.sleep(1)


---


#main program
#Create Sub Thread
n=Numbers()
t1=threading.Thread(target=n.generate,args=(int(input("Enter How
    Many Numbers u want to generate:"))),) # Step-3
t1.start() # Step-4


---


#Thread Based Application for generating 1 to n after each and
every second
#NumGenOOPEx2.py
import threading,time#Step-1
class Numbers: # Step-2
    def generate(self,n): # Step-3
        print("generate() executed
              by:{}".format(threading.current_thread().name))
        if(n<=0):
            print("{} Is Invalid Input".format(n))
        else:
            print("Numbers within:{}".format(n))
            for i in range(1,n+1):
                print("\tVal of i:{}".format(i))
                time.sleep(1)


---


#main program
#Create Sub Thread
t1=threading.Thread(target=Numbers().generate,args=(int(input("E
    nter How Many Numbers u want to generate:"))),) # Step-3
t1.start() # Step-4


---


#Program for Setting the thread name and getting thread name
#SetGetMethodsEx1.py

```

```

import threading
def hello():
    print("hello()--executed by :",
          threading.current_thread().name)
#main program
t1=threading.Thread(target=hello)
print("Default Name of Sub Thread before=",t1.name)
#getName() deprecated on the name of an attribute "name"
#t1.setName("HYD")--setName() deprecated on the name of an
                        attribute "name"
t1.name="HYD" # Setting the name of thread
print("Default Name of Sub Thread after setting=",t1.name)
#getName() deprecated on the name of an attribute "name"
#dispatch the sub therad
t1.start()
=====
#Program for creating sub thread
#SubThreadEx1.py
import threading
def hello():
    print("hello()--executed by
          :",threading.current_thread().name)
def hi(val):
    print("Hi,{ }--executed
          by:{}".format(val,threading.current_thread().name))
#main program
print("Program execution
started:",threading.current_thread().name)
#create sub threads
t1=threading.Thread(target=hello)#t1 is object--Thread-1
t2=threading.Thread(target=hi,args=("TR",))#t2 is object--
Thread-2
#Dispatch the sub threads
t1.start()
t2.start()
print("\nProgram execution
Ended:",threading.current_thread().name)
=====
```

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 acquiring and releasing the lock will be continued until all the thread objects completed their execution.

```
#Program for Demonstrating Dead Lock Result
#Non-SyncFunEx1.py
import threading,time
def multable(n):
    if(n<=0):
        print("{}--{} Invalid
Input".format(threading.current_thread().name,n))
    else:
        print("Mul Table for {}".format(n))
        for i in range(1,11):
            print("\t{}-->{} x {} =
{}".format(threading.current_thread().name,n,i,n*i))
            time.sleep(1)
#main program
t1=threading.Thread(target=multable,args=(6,))
t2=threading.Thread(target=multable,args=(16,))
t3=threading.Thread(target=multable,args=(19,))
#dispatch the sub threads
t1.start()
t2.start()
```

```

t3.start()


---


#Program for Demonstrating Dead Lock Result
#Non-SyncOOPEx1.py
import threading,time
class MulTable:
    def multable(self,n):
        if(n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,n))
        else:
            print("Mul Table for {}".format(n))
            for i in range(1,11):
                print("\t{}-->{} x {} =
{}".format(threading.current_thread().name,n,i,n*i))
                time.sleep(1)
#main program
t1=threading.Thread(target= MulTable().multable,args=(6,))
t2=threading.Thread(target= MulTable().multable,args=(16,))
t3=threading.Thread(target= MulTable().multable,args=(19,))
#dispatch the sub threads
t1.start()
t2.start()
t3.start()


---


#Program for Demonstrating Dead Lock Result
#SyncFunEx1.py
import threading,time
def multable(n):
    #Step-2---acquire the lock
    L.acquire()
    if(n<=0):
        print("{}--> {} Invalid
Input".format(threading.current_thread().name,n))
    else:
        print("Mul Table for {}".format(n))
        for i in range(1,11):
            print("\t{}-->{} x {} =
{}".format(threading.current_thread().name,n,i,n*i))
            time.sleep(1)
    #Step-3--Release the Lock
    L.release()
#main program
#Create an object of Lock of Class
L=threading.Lock() #Step-1
t1=threading.Thread(target=multable,args=(6,))
t1.name="Krishna"
t2=threading.Thread(target=multable,args=(16,))

```

```

t2.name="Raj"
t4=threading.Thread(target=multable,args=(-14,))
t4.name="KVR"
t3=threading.Thread(target=multable,args=(19,))
t3.name="Somnath"
#dispatch the sub threads
t1.start()
t2.start()
t4.start()
t3.start()


---


#Program for Demonstrating Dead Lock Result
#SyncOOPEx1.py
import threading,time
class MulTable:
    def multable(self,n):
        #Step-2
        L.acquire()
        if(n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,n))
        else:
            print("Mul Table for {}".format(n))
            for i in range(1,11):
                print("\t{}-->{} x {} =
{}".format(threading.current_thread().name,n,i,n*i))
                time.sleep(1)
        #Step-3
        L.release()
#main program
#Step-1
L=threading.Lock()
t1=threading.Thread(target= MulTable().multable,args=(6,))
t2=threading.Thread(target= MulTable().multable,args=(16,))
t3=threading.Thread(target= MulTable().multable,args=(19,))
#dispatch the sub threads
t1.start()
t2.start()
t3.start()


---


#TrainResFunEx1.py
import threading,time
def reservation(nos):
    L.acquire()
    global totnos
    if(nos>totnos):
        print("Hi:{}, {} Seats are Not Available--Try Next
Time".format(threading.current_thread().name,nos))

```

```

        time.sleep(2)
    else:
        totnos=totnos-nos
        print("Hi:{}, {} Seats are Reserved--Happy
Journey".format(threading.current_thread().name,nos))
        time.sleep(2)
        print("\tNow Available Number of
Seats:{}{}".format(totnos))
    L.release()
#main program
L=threading.Lock()
totnos=10
print("\tTOTAL NUMBER OF SEATS:{}{}".format(totnos))
p1=threading.Thread(target=reservation,args=(4,))
p1.name="Krishna"
p2=threading.Thread(target=reservation,args=(7,))
p2.name="Raj"
p3=threading.Thread(target=reservation,args=(4,))
p3.name="KVR"
p4=threading.Thread(target=reservation,args=(3,))
p4.name="Sumanth"
p5=threading.Thread(target=reservation,args=(2,))
p5.name="Vinod"
p6=threading.Thread(target=reservation,args=(2,))
p6.name="Shital"
#dispatch the threads
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()


---


#TrainResFunEx1.py
import threading,time
def reservation(nos):
    L.acquire()
    global totnos
    if(nos>totnos):
        print("Hi:{}, {} Seats are Not Available--Try Next
Time".format(threading.current_thread().name,nos))
        time.sleep(2)
    else:
        totnos=totnos-nos
        print("Hi:{}, {} Seats are Reserved--Happy
Journey".format(threading.current_thread().name,nos))
        time.sleep(2)

```

```

        print("\tNow Available Number of
Seats:{}".format(totnos))
        L.release()
#main program
L=threading.Lock()
totnos=10
print("\tTOTAL NUMBER OF SEATS:{}".format(totnos))
p1=threading.Thread(target=reservation,args=(4,))
p1.name="Krishna"
p2=threading.Thread(target=reservation,args=(7,))
p2.name="Raj"
p3=threading.Thread(target=reservation,args=(4,))
p3.name="KVR"
p4=threading.Thread(target=reservation,args=(3,))
p4.name="Sumanth"
p5=threading.Thread(target=reservation,args=(2,))
p5.name="Vinod"
p6=threading.Thread(target=reservation,args=(2,))
p6.name="Shital"
#dispatch the threads
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()


---


#Program for generating Odd Numbers by one thread and even
numbers by another thread
#OddEevnNumbersFunEx.py
import threading,time
def odd(n):
    if(n<=0):
        print("{}--{} Invalid
Input".format(threading.current_thread().name,n))
    else:
        for i in range(1,n+1,2):
            print("{}--Odd
Number:{}".format(threading.current_thread().name,i))
            time.sleep(1)

def even(n):
    if(n<=0):
        print("{}--{} Invalid
Input".format(threading.current_thread().name,n))
    else:
        for i in range(2,n+1,2):

```

```

        print("{}--Even
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)
#main program
t1=threading.Thread(target=odd,args=(10,))
t2=threading.Thread(target=even,args=(10,))
#dispatch the sub threads
t1.start()
t2.start()

```

```

#Program for generating Odd Numbers by one thread and even
numbers by another thread
#OddEevnNumbersOOPEx1.py
import threading,time
class OddNumbers:
    def __init__(self,n):
        self.n=n
    def odd(self):
        if(self.n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,self.n))
        else:
            for i in range(1,self.n+1,2):
                print("{}--Odd
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)

```

```

class EvenNumbers:
    def __init__(self,n):
        self.n=n

    def even(self):
        if(self.n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,self.n))
        else:
            for i in range(2,self.n+1,2):
                print("{}--Even
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)

```

```

#main program
on=OddNumbers(10)
en=EvenNumbers(10)
t1=threading.Thread(target=on.odd)
t2=threading.Thread(target=en.even)
#dispatch the sub threads
t1.start()
t2.start()

```

```

#Program for generating Odd Numbers by one thread and even
numbers by another thread
#OddEevnNumbersOOPEx2.py
import threading,time
class OddNumbers:
    def __init__(self,n):
        self.n=n
    def odd(self):
        if(self.n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,self.n))
        else:
            for i in range(1,self.n+1,2):
                print("{}--Odd
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)
class EvenNumbers:
    def __init__(self,n):
        self.n=n

    def even(self):
        if(self.n<=0):
            print("{}--{} Invalid
Input".format(threading.current_thread().name,self.n))
        else:
            for i in range(2,self.n+1,2):
                print("{}--Even
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)
#main program
t1=threading.Thread(target=OddNumbers(int(input("How Many Odd
    Numbers u want:"))).odd)
t2=threading.Thread(target=EvenNumbers(int(input("How Many Even
    Numbers u want:"))).even)
#dispatch the sub threads
t1.start()
t2.start()
=====

```

Network Programming in Python--2 Days--Outdated

Index

- =>Purpose of Network Programming
- =>Def of Network.
- =>DNS, IP Address, Server, Client
- =>Steps for developing Server Side Applications
- =>Steps for developing Client Side Applications

=>Module Name Required for Developing Server and Client Applications (socket)
=>Functions in socket module

=>Programming Examples on Network Programming
 2-Tier Applications
 3-Tier Applications

=====

Network Programming in Python

=====

=>The purpose of Network Programming is that "To share the data / information between Multiple Remote Machine / Devices across the Universe".

=>Definition of Network:

=>A Network is a Collection of Inter-connected Autonomous Computers Connected with Server.

Definition of Client Side Program

=>A Client Side Program is one, which always makes a Request to Server Side Program for obtaining the Result.

Definition of Server Side Program

=>A Server Side Program is one, which always Receives the Request from Client Side Program, Process the Request and Gives Response to Client Side Program.

Definition of DNS (Domain Naming Service)

=>DNS is nothing but name of the Physical Machine where Server Side Program resides. The Default DNS of Every Computer is "localhost".

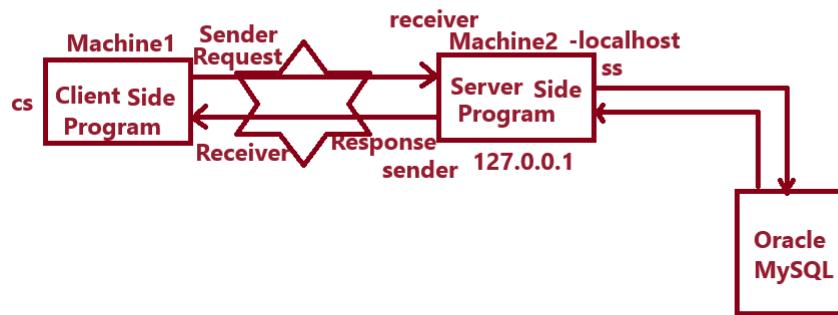
Definition of IP Address (Internet Protocol Address)

=>IP Address is nothing but Address of the Physical Machine where Server Side Program resides. The Default IP Address of Every Computer is "127.0.0.1" (loop back address).

Definition of Port Number

A Port Number is one of the Logical Numerical ID where Server Side Program is Running.

=>If Client Side Program want to Communicate with Server Side Program then Client Side Program Must get Connection from Server Side Program By Passing (DNS/IP Address, Portno).



Steps for Developing Networking Applications

=>To Develop Networking Applications, we develop Two Types of Programs. They are

1. Development Server Side Program
2. Development Client Side Program

1. Steps for Developing Server Side Program

Step-1: import socket module

Step-2: Every Server Side Must Resides / Binds with Certain -----
Machine along with Port Number.

Step-3: Every Server Side Program Must be configured in such way
that to how many Clients, A Server Side Program can
communicate.

Step-4: Every Server Side Program must ACCEPT Client Side
Program Request.

Step-5: Every Server Side Program must Read the Data of Client -
Side Program Request and Process.

Step-6: Every Server Side Must give RESPONSE / SEND the result -
to Client Side Program.

Step-7: Server Side Program Repeats Step-(4), Step-(5) and Step-
(6) until all Client Side Program Requests completes.

2. Steps for Developing Client Side Program

Step-1: import socket module

Step-2: Every Client Side Program Must get the CONECTION from -
----- Server Side Program by passing (DNS/IP Address, PortNo)

Step-3: Every Client Side Program Makes Request to Server Side
----- Program.

Step-4: Every Client Side Program Recevives the Response from
----- Server Side Program

Step-5: Every Client Side Program Repeats Step- (3) and Step- (4)
----- until Its Requests are Completed
=====

Module Name for Developing Networking Applications

=>The Module Name for Developing Networking Applications is
"socket".

Functions in socket module

1) socket()

Syntax: varname=socket.socket()
=>This Function is used for creating an object of socket.

=>If we create an object of socket at Client Side Program then
It is called Client Socket object.
=>If we create an object of socket at Server Side Program then
It is called Server Socket object.

Examples:s=socket.socket()
=>Hence An obejct of socket acts as Bi-Directional Communication

Object exist both at Client and Server Side Programs.

2) bind()

=>Syntax: serversocketobj.bind(("DNS/IPAddress",PortNo))
=>This Function is used for making Socket Object as Server

Socket and Makes the Program as Server Side Program.

=>Examples: s.bind(("localhost",4444)

(OR)

s.bind(("127.0.0.1",4444))

3) listen()

=>Syntax: serversocketobj.listen(Number of Clients)

=>This Function is used for making the server side program to be configured in such way that to how many Clients, A Server Side Program can communicate.

Examples: s.listen(2)

4) accept()

=>Syntax: varname1,varname2=serversockobj.accept()
=>This Function is used for accepting Client Program Request.
=>This Function returns Client Side Program socket Object (Client Socket) and Its address.

Examples

cs,ca=s.accept()

5) recv() with decode()

=>Syntax: strdata=socketobj.recv(1024 | 2048 | 4096).decode()
=>This function is used for Receiving the Data at Both Client and Server Programs

Examples: strdata=s.recv(1024).decode()

6) send() with encode()

Syntax: sockobj.send(strdata.encode())
=>This Function is used for Sending the Data at Both Client and Server Programs

Examples

s.send(str(any data).encode())

7) connect()

=>Syntax: clientsockobj.connect(("DNS/IP Address",PortNo))
=>This Function is used for Obtaining the Connection from Server Side Program by Client Side Program.

Examples:

cs=socket.socket()
cs.connect(("localhost",4444)
(OR)

```

    cs.connect(("127.0.0.1", 4444))


---


#ClientSquare.py
import socket
s=socket.socket()
s.connect(("localhost", 8558))
print("CSP Got Connection from SSP")
print("-----")
n=input("Enter Any Number for getting Square:")
s.send(n.encode())
res=s.recv(1024).decode()
print("Square({})={}".format(n, res))


---


#ServerSquare.py
import socket
s=socket.socket()
s.bind(("localhost", 8558))
s.listen(2)
print("SSP is Ready to accept any CSP Request")
while(True):
    try:
        cs, ca=s.accept()
        csdata=cs.recv(1024).decode()
        print("Client Side Value:{} ".format(csdata))
        n=int(csdata)
        res=n**2
        cs.send(str(res).encode())
    except ValueError:
        cs.send("Don't enter Alnums, strs and
                symbols".encode())


---


#ClientChat.py
import socket
while(True):
    s=socket.socket()
    s.connect(("127.0.0.1", 3600))
    csdata=input("Student-->")
    if(csdata.lower() == "bye"):
        s.send(csdata.encode())
        break
    else:
        s.send(csdata.encode())
        ssdata=s.recv(1024).decode()
        print("KVR-->{}".format(ssdata))


---


#ServerChat.py
import socket
s=socket.socket()
s.bind(("127.0.0.1", 3600))

```

```

s.listen(1)
print("SSP is Ready to Accept any CSP Request")
while(True):
    cs,ca=s.accept()
    csdata=cs.recv(1024).decode()
    print("Student-->{}".format(csdata))
    if(csdata.lower() == "bye"):
        cs.send("KVR-->bye-read well".encode())
    else:
        ssdata=input("KVR-->")
        cs.send(ssdata.encode())


---


#ClientDBEx1.py
import socket
s=socket.socket()
s.connect(("localhost",4444))
htno=input("Enter Ur Hall Ticket Number:")
s.send(htno.encode())
result=s.recv(1024).decode()
print("-----")
print(result)
print("-----")
-----
#ServerDBEx1.py
import cx_Oracle
import socket
s=socket.socket()
s.bind(("localhost",4444))
s.listen(2)
while(True):
    try:
        cs,ca=s.accept()
        htno=int(cs.recv(1024).decode())
        #PDBC Code
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from result where htno=%d"
                   "%htno")
        record=cur.fetchone()
        if(record!=None):
            s1="Hall Ticket Number:"+str(record[0])
            s2="Student Name:"+str(record[1])
            s3="Student Marks:"+str(record[2])
            res=s1+"\n"+s2+"\n"+s3
            cs.send(res.encode())
        else:
            cs.send(str("Record Does not
                        Exist").encode())
    
```

```
except ValueError:
    cs.send("Don't enter alnums,strs and symbols for
            htno".encode())
except cx_Oracle.DatabaseError as db:
    cs.send(("Problem in DB:"+db).encode())
=====
random module
=====

=>"random" is one of the pre-defined / in-built module in
python
=>The purpose of random module is that "To generate random
numbers".
=>The random module contains the following pre-defined
functions.
    a) randrange()
    b) randint()
    c) random()
    d) uniform()
    e) choice()
    f) shuffle()
    g) sample()
```

a) `randrange()`:

Syntax:- `random.randrange(start,stop)`
=>This function is used for generating random integer values
from specified start value to stop-1 value.(start value included
and stop value excluded)
=>Here start and stop values must be integer values only.

Examples:

```
>>> import random
>>> print(random.randrange(10,21))
11
>>> print(random.randrange(10,21))
10
>>> print(random.randrange(10,21))
16
>>> print(random.randrange(10,21))
15
>>> print(random.randrange(10,21))
19
>>> print(random.randrange(10,21))
13
>>> print(random.randrange(10,21))
20
>>> print(random.randrange(10,21))
```

13

```
-----  
#randrangeex1.py  
import random  
for i in range(1, 6):  
    print(random.randrange(1000, 9999))  
-----  
b) randint():  
-----  
Syntax:- random.randint(start, stop)  
=>This function is used for generating random integer values  
from specified start value to stop value.(start value and stop  
value included)  
=>Here start and stop values must be integer values only.
```

Examples:

```
-----  
#randintex1.py  
import random  
for i in range(1, 11):  
    print(random.randint(10000, 99999))  
-----  
>>> import random  
>>> print(random.randint(10, 15))  
15  
>>> print(random.randint(10, 15))  
12  
>>> print(random.randint(10, 15))  
10  
>>> print(random.randint(10, 15))  
12  
-----  
c) random():  
-----
```

```
=>Syntax:- random.random()  
=>This Function is used generating random floating point values  
between 0 and 1.(here 0 and 1 are excluded)
```

Examples:

```
-----  
>>> import random  
>>> print(random.random())  
0.18470704046738295  
>>> print(random.random())  
0.05181247786271237  
>>> print(random.random())  
0.7339066079590756
```

```

>>> print(random.random())
0.4754033595408611
>>> print(round(random.random(), 6))
0.717572
>>> print(round(random.random(), 2))
0.38
-----
#randomex1.py
import random
for i in range(1, 6):
    print(random.random())
-----
d) uniform():

=>Syntax:      random.uniform(start,stop)
=>This function is used for generating floating point random
values between start and stop (both of them are excluded).
=>Here start and stop values can be both integer and float
values .
Examples:
-----
>>> import random
>>> print(random.uniform(10,12))
10.709619666117199
>>> print(random.uniform(10,12))
10.476327243120252
>>> print(random.uniform(10,12))
11.872190617603886
>>> print(random.uniform(10,12))
10.642676007799324
>>> print(random.uniform(10,12))
10.138916048149817
>>> print(random.uniform(10,12))
11.445307273750647
-----
#uniformex1.py
import random
for i in range(1, 6):
    print(random.uniform(10.5,12))
-----
e) choice():

Examples:
-----
>>> s="PYTHON"
>>> import random
>>> s="PYTHON"

```

```

>>> print(random.choice(s))
Y
>>> print(random.choice(s))
T
>>> print(random.choice(s))
N
>>> print(random.choice(s))
>>> lst=[10,"KVR","HYD",23.45,True]
>>> print(random.choice(lst))
23.45
>>> print(random.choice(lst))
KVR
>>> print(random.choice(lst))
23.45
>>> print(random.choice(lst))
True
>>> print(random.choice(lst))
HYD
-----
#choiceex1.py
import random
tpl=(10,12.34,"Rossum",45.67,2+3j,True,False)
for i in range(1,6):
    print(random.choice(tpl))
-----
#choiceex2.py
import random
ccc="WeR3$tQ9#@ELaJhU8%^OcZ"
for i in range(1,6):
    print(random.choice(ccc),random.choice(ccc),random.choice(
        ccc),random.choice(ccc), random.choice(ccc))
-----
f) shuffle():
-----
=>Syntax:- random.shuffle(mutable_iterable_object)
=>This function is used for re-organizing the elements of any
mutable_Iterable_object.

```

Examples:

```

>>> import random
>>> lst=[10,"KVR","HYD",23.45,True]
>>> random.shuffle(lst)
>>> print(lst)
[True, 'KVR', 10, 23.45, 'HYD'] #re-organized elements
>>> random.shuffle(lst)
>>> print(lst)

```

```

['HYD', 23.45, 10, 'KVR', True] #re-organized elements
>>> random.shuffle(lst)
>>> print(lst)
[23.45, 10, 'KVR', 'HYD', True] #re-organized elements
-----
#shuffleex.py
import random
lst=[10,"KVR","HYD",23.45,True]
for i in range(1,6):
    random.shuffle(lst)
    print(lst)
-----
g) sample():

Syntax:-      random.sample(Iterable_object, k)
=>here 'k' represents "No. of Selections to be made randomly
from specified iterable object".
-----
Examples:
-----
>>> lst=[10,"KVR","HYD",23.45,True]
>>> random.sample(lst,2)
['HYD', 'KVR']
>>> random.sample(lst,2)
['KVR', 'HYD']
>>> random.sample(lst,2)
['KVR', 23.45]
>>> random.sample(lst,2)
['HYD', 23.45]
>>> random.sample(lst,2)
[10, 'HYD']
>>> random.sample(lst,3)
['HYD', True, 10]
>>> random.sample(lst,3)
[True, 10, 'HYD']
>>> random.sample(lst,3)
['HYD', 'KVR', True]
-----
#sampleex.py
import random
lst=[10,"KVR","HYD",23.45,True]
for i in range(1,6):
    print(random.sample(lst,3))

```

Regular Expressions in Python--2.5 Days

Index

=>Purpose of Regular Expressions
=>Definition of Regular Expressions
=>Applications of Regular Expressions
=>Module Name of Regular Expressions (re)
=>Functions in re module
 1) search()
 2).findall()
 3) finditer()
 4) start()
 5) end()
 6) group()
 7) sub()
=>Programming Examples

=>Programmer-Defined Character Classes
=>Programming Examples
=>Pre-Defined Character Classes
=>Programming Examples
=>Quantifiers
=>Programming Examples

=>Combined Programming Examples on Programmer-Defined, Pre-Defined and Quantifiers .

Regular Expressions in Python

=>The Regular Expressions in Python is that " To Develop Robust Applications By Validation of Data".
=>Regular Expressions concept is one of the Programming Language Independent Concept and It is Implemented in Various Programming Languages.
=>Regular Expressions Concept in Python Implemented by a Pre-Defined Module called "re".
=>To develop any Regular Expressions Programs / Applications, we use a pre-defined Module called "re".

Applications of Regular Expressions

=>Regular Expressions are used in Development of Language Compilers and Interpreters
=>Regular Expressions are used in Development of OSes
=>Regular Expressions are used in Universal Protocols
 Development (http, https, pop, nmp, smtp...etc)
=>Regular Expressions are used Electronics Circuits Designing.
=>Regular Expressions are used in Pattern Matching.

Definition of Regular Expressions

=>A Regular Expression is one of the Search Pattern which is a combination of Alphabets, Digits and Special Symbols which is used to Search OR Match OR Find in Given Data and Obtains Desired Results.

Programmer-Defined Character Classes

=>Programmer-Defined Character Classes developed by Python Programmers for Designing Search Patterns and It is used to Search OR Match OR Find in Given Data for Obtaining Desired Result.

=>Syntax: [Programmer-Defined Character Class]

=>The Programmer-Defined Character Classes are Classified as Follows

1. [abc]----->Searches for either 'a' or 'b' or 'c'
2. Only

2. [^abc]----->searches for all except 'a' or 'b' or 'c'

3. [A-Z]----->Searches for all Upper Case Alphabets
Only

4. [^A-Z]----->Searches for all Except Upper Case
Alphabets

5. [a-z]----->Searches for all Lower Case Alphabets
Only

6. [^a-z]----->Searches for all except Lower Case
Alphabets Only

7. [0-9]----->Searches for all digits Only

8. [^0-9]----->Searches for all except Digits

9. [A-Za-z]----->Searches all Alphabets (Lower +Upper)
Only.

10. [^A-Za-z]----->Searches all except Alphabets
(Lower +Upper) Only.

11. [A-Za-z0-9]----->Searches for all Alpha-Numeric Values
Only

12. [^A-Za-z0-9]----->Searches for Special Symbols (Except Alpha-Numeric Values)
 13. [A-Z0-9]----->Searches for all Upper Case Alphabets + Digits Only
 14. [^A-Z0-9]----->Searches for all except Upper Case Alphabets + Digits Only
 15. [a-z0-9]----->Searches for all Lower Case Alphabets + Digits Only
 16. [^a-z0-9]----->Searches for all except Lower Case Alphabets + Digits
-

```
#Search for a word "Python" in given Data.  
#RegExpr1.py  
import re  
gd="Python is an oop lang.Python is also Fun Prog Lang"  
sp="Python"  
md=re.search(sp, gd) # Here md is an object of <class 're.Match'>  
                      Or None  
if (md!=None):  
    print("Search is Sucessful")  
    print("Start Index:",md.start())  
    print("Start Index:",md.end())  
    print("Matched Value:",md.group())  
else:  
    print("Search is Un-Sucessful")  
    print("'{ }' is not Found".format(sp))  


---

  
#Search for a word "Python" in given Data.  
#RegExpr1.py  
import re  
gd="Python is an oop lang.Python is also Fun Prog Lang"  
sp="Python"  
md=re.search(sp, gd) # Here md is an object of <class 're.Match'>  
                      Or None  
if (md!=None):  
    print("Search is Sucessful")  
    print("Start Index:",md.start())  
    print("Start Index:",md.end())  
    print("Matched Value:",md.group())  
else:  
    print("Search is Un-Sucessful")  
    print("'{ }' is not Found".format(sp))  


---

  
#Search for a word "Python" in given Data.
```

```

#RegExpr2.py
import re
gd="Python is an oop lang.Python is also Fun Prog Lang"
sp="Python"
md=re.findall(sp,gd) # Here md is an object of <class 'list'>
print(md)
print("'{ }' Found {} Time(s)".format(sp,len(md)))
"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr2.py
['Python', 'Python']
'Python' Found 2 Time(s)
"""

#Search for a word "Python" in given Data along with Indices
#RegExpr3.py
import re
gd="Python is an oop lang.Python is also Fun Prog Lang"
sp="Python"
md=re.finditer(sp,gd) # Here md is an object of <class
'callable_iterator'
print("-----")
noc=0
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          Value:{}".format(mde.start(),mde.end(),mde.group()))
    noc=noc+1
print("-----")
print("'{ }' Found {} Time(s)".format(sp,noc))
print("-----")

#RegExpr4.py
import re
gd="aHfG4%KaQ5@6cDp%8b"
sp="[abc]"
md=re.finditer(sp,gd)
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          Value:{}".format(mde.start(),mde.end(),mde.group()))
print("-----")

"""

-----
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr4.py
-----
start Index:0 End Index:1 Value:a
start Index:7 End Index:8 Value:a

```

```

start Index:12 End Index:13 Value:c
start Index:17 End Index:18 Value:b
-----
"""
#RegExpr5.py
import re
md=re.finditer("[abc]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr5.py
-----
start Index:0 End Index:1 Value:a
start Index:7 End Index:8 Value:a
start Index:12 End Index:13 Value:c
start Index:17 End Index:18 Value:b
-----
"""

#RegExpr6.py
import re
md=re.finditer("[^abc]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr6.py
-----
start Index:1 End Index:2 Value:H
start Index:2 End Index:3 Value:f
start Index:3 End Index:4 Value:G
start Index:4 End Index:5 Value:4
start Index:5 End Index:6 Value:%
start Index:6 End Index:7 Value:K
start Index:8 End Index:9 Value:Q
start Index:9 End Index:10 Value:5
start Index:10 End Index:11 Value:@

```

```

start Index:11 End Index:12 Value:6
start Index:13 End Index:14 Value:D
start Index:14 End Index:15 Value:p
start Index:15 End Index:16 Value:%
start Index:16 End Index:17 Value:8
-----
"""
#searches for all Upper Case Alphabets
#RegExpr7.py
import re
md=re.finditer("[A-Z]","aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}\nValue:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr7.py
-----
start Index:1 End Index:2 Value:H
start Index:3 End Index:4 Value:G
start Index:6 End Index:7 Value:K
start Index:8 End Index:9 Value:Q
start Index:13 End Index:14 Value:D
-----
"""
#searches for all except Upper Case Alphabets
#RegExpr8.py
import re
md=re.finditer("[^A-Z]","aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}\nValue:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr8.py
-----
start Index:0 End Index:1 Value:a
start Index:2 End Index:3 Value:f
start Index:4 End Index:5 Value:4

```

```

start Index:5 End Index:6 Value:%
start Index:7 End Index:8 Value:a
start Index:9 End Index:10 Value:5
start Index:10 End Index:11 Value:@
start Index:11 End Index:12 Value:6
start Index:12 End Index:13 Value:c
start Index:14 End Index:15 Value:p
start Index:15 End Index:16 Value:%
start Index:16 End Index:17 Value:8
start Index:17 End Index:18 Value:b
-----"""
#searches for all Lower Case Alphabets
#RegExpr9.py
import re
md=re.finditer("[a-z]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr9.py
-----
start Index:0 End Index:1 Value:a
start Index:2 End Index:3 Value:f
start Index:7 End Index:8 Value:a
start Index:12 End Index:13 Value:c
start Index:14 End Index:15 Value:p
start Index:17 End Index:18 Value:b
-----
"""

#searches for all except Lower Case Alphabets
#RegExpr10.py
import re
md=re.finditer("[^a-z]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr10.py
-----
start Index:1 End Index:2 Value:H

```

```

start Index:3 End Index:4 Value:G
start Index:4 End Index:5 Value:4
start Index:5 End Index:6 Value:%
start Index:6 End Index:7 Value:K
start Index:8 End Index:9 Value:Q
start Index:9 End Index:10 Value:5
start Index:10 End Index:11 Value:@
start Index:11 End Index:12 Value:6
start Index:13 End Index:14 Value:D
start Index:15 End Index:16 Value:%
start Index:16 End Index:17 Value:8
-----
"""

#searches for all Digits Only
#RegExpr11.py
import re
md=re.finditer("[0-9]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}\nValue:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr11.py
-----
start Index:4 End Index:5 Value:4
start Index:9 End Index:10 Value:5
start Index:11 End Index:12 Value:6
start Index:16 End Index:17 Value:8
-----
"""

#searches for all except Digits
#RegExpr12.py
import re
md=re.finditer("[^0-9]", "aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}\nValue:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr12.py

```

```

-----
start Index:0  End Index:1 Value:a
start Index:1  End Index:2 Value:H
start Index:2  End Index:3 Value:f
start Index:3  End Index:4 Value:G
start Index:5  End Index:6 Value:%
start Index:6  End Index:7 Value:K
start Index:7  End Index:8 Value:a
start Index:8  End Index:9 Value:Q
start Index:10  End Index:11 Value:@
start Index:12  End Index:13 Value:c
start Index:13  End Index:14 Value:D
start Index:14  End Index:15 Value:p
start Index:15  End Index:16 Value:%
start Index:17  End Index:18 Value:b
-----
"""
#searches for all Alphabets
#RegExpr13.py
import re
md=re.finditer("[A-Za-z]","aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{}  End Index:{}\n        Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr13.py
-----
start Index:0  End Index:1 Value:a
start Index:1  End Index:2 Value:H
start Index:2  End Index:3 Value:f
start Index:3  End Index:4 Value:G
start Index:6  End Index:7 Value:K
start Index:7  End Index:8 Value:a
start Index:8  End Index:9 Value:Q
start Index:12  End Index:13 Value:c
start Index:13  End Index:14 Value:D
start Index:14  End Index:15 Value:p
start Index:17  End Index:18 Value:b
-----
"""
#searches for all except Alphabets
#RegExpr14.py
import re
md=re.finditer("[^A-Za-z]","aHfG4%KaQ5@6cDp%8b")

```

```

print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          "Value:{}".format(mde.start(),mde.end(),mde.group()))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr14.py
-----
start Index:4 End Index:5 Value:4
start Index:5 End Index:6 Value:%
start Index:9 End Index:10 Value:5
start Index:10 End Index:11 Value:@
start Index:11 End Index:12 Value:6
start Index:15 End Index:16 Value:%
start Index:16 End Index:17 Value:8
-----
"""

#searches for all Alpha-numeric Values
#RegExpr15.py
import re
md=re.finditer("[A-Za-z0-9]","aHfG4%KaQ5@6cDp%8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          "Value:{}".format(mde.start(),mde.end(),mde.group()))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr15.py
-----
start Index:0 End Index:1 Value:a
start Index:1 End Index:2 Value:H
start Index:2 End Index:3 Value:f
start Index:3 End Index:4 Value:G
start Index:4 End Index:5 Value:4
start Index:6 End Index:7 Value:K
start Index:7 End Index:8 Value:a
start Index:8 End Index:9 Value:Q
start Index:9 End Index:10 Value:5
start Index:11 End Index:12 Value:6
start Index:12 End Index:13 Value:c
start Index:13 End Index:14 Value:D
start Index:14 End Index:15 Value:p
start Index:16 End Index:17 Value:8
start Index:17 End Index:18 Value:b
-----

```

```
"""
=====
#searches for all Special Symbols
#RegExpr16.py
import re
md=re.finditer("[^A-Za-z0-9]","aH!fG4%KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
    print("start Index:{} End Index:{}"
          "Value:{}".format(mde.start(),mde.end(),mde.group()))
print("-----")
```

```
"""
-----
```

```
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr16.py
-----
start Index:2 End Index:3 Value:!
start Index:6 End Index:7 Value:%
start Index:11 End Index:12 Value:@
start Index:16 End Index:17 Value:*
```

```
"""
=====
```

Pre-Defined Character Classes

```
=>Pre-Defined Character Classes are already developed by Python
Lang Developers and available in Python software used for
Designing Search Patterns and It is used to Search OR Match OR
Find in Given Data for Obtaining Desired Result.
```

```
=>Syntax: "\Pre-Defined Character Class Name"
```

```
=>The Pre-Defined Character Classes are Classified as Follows
```

- ```
1) \s----->Searches for Space Character Only

2) \S----->Searches for all except Space Characters

3) \w----->Searches for Word Character Only]
 (alphabets+Numerics) OR [A-Za-z0-9]

4) \W----->Searches for all Special Symbols(except
 alphabets+Numerics) OR [^A-Za-z0-9]

5) \d----->Searches for Digits Only OR [0-9]

6) \D----->Searches for all except Digits OR [^0-9]
```

### **Quantifiers in Regular Expressions**

=====

=>Quantifiers in Regular Expressions are used for searching number of occurrences of the specified value (alphabets or digits or special symbols) used in search pattern to search in the given data and obtains desired result.

- 1) "k"----->It search for only one 'k' at a time
  - 2) "k+"----->It search for either one 'k' more 'k' s
  - 3) "k\*"----->It search for either zero 'k' or one 'k' and more 'k' s
  - 4) "k??"----->It search for either zero 'k' or one 'k'
  - 5) ". " ----->It searches for all
- 

Note:

-----

=>\ddd or \d{3}----->searches for 3 digits OR [0-9]{3}

=>\dd.\dd-----searches for 2 integer values and 2 decimal values

=>\d{2,4}----searches for min 2 digit number and max 4 digit number OR [0-9]{2,4}

=>[A-Za-z]+ ---searches one alphabet or More alphabets OR \w+

=>[0-9]+ or \d+-----searches one Digit or More Digits

-----

Special Points

- 
- 1) \d+---Searches for 1 or More Digits  
    \ddddddddd [0-9][0-9][0-9] [0-9][0-9][0-9] [0-9][0-9][0-9]  
        OR  
        \d{10} Or [0-9]{10}  
        \d{3,5} Or [0-9]{3,4}----\d{m,n} OR [0-9]{m,n}--  
Searches for Min m Digit Value and Max n-Digit Value
  - 2) [A-Za-z]{3,6}
  - 3) \w+
  - 4) \W
- 

```
#searches for Space Character
#RegExpr17.py
import re
md=re.finditer("\s","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}")

print("-----")
"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr17.py

```

```

start Index:1 End Index:2 Value:
start Index:8 End Index:9 Value:

"""
#searches for Space Character
#RegExpr17.py
import re
md=re.finditer("\s","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr17.py

start Index:1 End Index:2 Value:
start Index:8 End Index:9 Value:

"""

#searches for all except Space Character
#RegExpr18.py
import re
md=re.finditer("\S","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr18.py

start Index:0 End Index:1 Value:a
start Index:2 End Index:3 Value:H
start Index:3 End Index:4 Value:!
start Index:4 End Index:5 Value:f
start Index:5 End Index:6 Value:G
start Index:6 End Index:7 Value:4
start Index:7 End Index:8 Value:%
start Index:9 End Index:10 Value:K
start Index:10 End Index:11 Value:a
start Index:11 End Index:12 Value:Q
start Index:12 End Index:13 Value:5

```

```

start Index:13 End Index:14 Value:@
start Index:14 End Index:15 Value:6
start Index:15 End Index:16 Value:c
start Index:16 End Index:17 Value:D
start Index:17 End Index:18 Value:p
start Index:18 End Index:19 Value:*
start Index:19 End Index:20 Value:8
start Index:20 End Index:21 Value:b

"""
#searches for all word Characters Only
#RegExpr19.py
import re
md=re.finditer("\w","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}\n Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")
"""

```

```
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr19.py

```

```

start Index:0 End Index:1 Value:a
start Index:2 End Index:3 Value:H
start Index:4 End Index:5 Value:f
start Index:5 End Index:6 Value:G
start Index:6 End Index:7 Value:4
start Index:9 End Index:10 Value:K
start Index:10 End Index:11 Value:a
start Index:11 End Index:12 Value:Q
start Index:12 End Index:13 Value:5
start Index:14 End Index:15 Value:6
start Index:15 End Index:16 Value:c
start Index:16 End Index:17 Value:D
start Index:17 End Index:18 Value:p
start Index:19 End Index:20 Value:8
start Index:20 End Index:21 Value:b

```

```
"""

```

```
#searches for all Special Symbols Only
#RegExpr20.py
import re
md=re.finditer("\W","a H!fG4% KaQ5@6cDp*8b")
print("-----")
```

```

for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")
"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr20.py

start Index:1 End Index:2 Value:
start Index:3 End Index:4 Value:!
start Index:7 End Index:8 Value:%
start Index:8 End Index:9 Value:
start Index:13 End Index:14 Value:@
start Index:18 End Index:19 Value:*

"""

#searches for all Digits
#RegExpr21.py
import re
md=re.finditer("\d","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr21.py

start Index:6 End Index:7 Value:4
start Index:12 End Index:13 Value:5
start Index:14 End Index:15 Value:6
start Index:19 End Index:20 Value:8

"""

#searches for all except Digits
#RegExpr22.py
import re
md=re.finditer("\D","a H!fG4% KaQ5@6cDp*8b")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""

```

```
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr22.py
```

```

start Index:0 End Index:1 Value:a
start Index:1 End Index:2 Value:
start Index:2 End Index:3 Value:H
start Index:3 End Index:4 Value:!
start Index:4 End Index:5 Value:f
start Index:5 End Index:6 Value:G
start Index:7 End Index:8 Value:%
start Index:8 End Index:9 Value:
start Index:9 End Index:10 Value:K
start Index:10 End Index:11 Value:a
start Index:11 End Index:12 Value:Q
start Index:13 End Index:14 Value:@
start Index:15 End Index:16 Value:c
start Index:16 End Index:17 Value:D
start Index:17 End Index:18 Value:p
start Index:18 End Index:19 Value:
start Index:20 End Index:21 Value:b

```

```
"""
```

---

```
#searches for Exactly One K Only
#RegExpr23.py
import re
md=re.finditer("K","KVKKVKKVKV")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}
 Value:{}".format(mde.start(),mde.end(),mde.group()))
print("-----")
```

```
"""
```

```
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr23.py
```

```

start Index:0 End Index:1 Value:K
start Index:2 End Index:3 Value:K
start Index:3 End Index:4 Value:K
start Index:5 End Index:6 Value:K
start Index:6 End Index:7 Value:K
start Index:7 End Index:8 Value:K
start Index:9 End Index:10 Value:K

```

```
"""
```

---

```
#searches for One K OR More K's
#RegExpr24.py
import re
```

```

md=re.finditer("K+","KVKKVKKKVKV")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr24.py

start Index:0 End Index:1 Value:K
start Index:2 End Index:4 Value:KK
start Index:5 End Index:8 Value:KKK
start Index:9 End Index:10 Value:K

"""

#searches for Zero K or One K OR More K's
#RegExpr25.py
import re
md=re.finditer("K*","KVKKVKKKVKV")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""
D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr25.py

start Index:0 End Index:1 Value:K
start Index:1 End Index:1 Value:
start Index:2 End Index:4 Value:KK
start Index:4 End Index:4 Value:
start Index:5 End Index:8 Value:KKK
start Index:8 End Index:8 Value:
start Index:9 End Index:10 Value:K
start Index:10 End Index:10 Value:
start Index:11 End Index:11 Value:

"""

#searches for Zero K or One K
#RegExpr26.py
import re
md=re.finditer("K?", "KVKKVKKKVKV")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>

```

```

 print("start Index:{} End Index:{}\n"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")
"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr26.py

start Index:0 End Index:1 Value:K
start Index:1 End Index:1 Value:
start Index:2 End Index:3 Value:K
start Index:3 End Index:4 Value:K
start Index:4 End Index:4 Value:
start Index:5 End Index:6 Value:K
start Index:6 End Index:7 Value:K
start Index:7 End Index:8 Value:K
start Index:8 End Index:8 Value:
start Index:9 End Index:10 Value:K
start Index:10 End Index:10 Value:
start Index:11 End Index:11 Value:

"""

#searches for all
#RegExpr26.py
import re
md=re.finditer(".","KVK6KVKKKVKV")
print("-----")
for mde in md: # Here mde is an object of <class, re.match>
 print("start Index:{} End Index:{}\n"
 "Value:{}".format(mde.start(),mde.end(),mde.group())))
print("-----")

"""

D:\KVR-PYTHON-9AM\REGEXPRS>py RegExpr27.py

start Index:0 End Index:1 Value:K
start Index:1 End Index:2 Value:V
start Index:2 End Index:3 Value:K
start Index:3 End Index:4 Value:6
start Index:4 End Index:5 Value:K
start Index:5 End Index:6 Value:V
start Index:6 End Index:7 Value:K
start Index:7 End Index:8 Value:K
start Index:8 End Index:9 Value:K
start Index:9 End Index:10 Value:V
start Index:10 End Index:11 Value:K
start Index:11 End Index:12 Value:V

```

```

"""
#MobileNumberValidEx1.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("\nUr Mobile Number is Valid")
 break
 else:
 print("\tUr Mobile Number Should Not Non-
 Numerics-try again")
 else:
 print("\tInvalid Mobile Number Length--try again")
#MobileNumberValidEx2.py
import re
while(True):
 mno=input("Enter Ur Mobile Number:")
 if(len(mno)==10):
 res=re.search("[98][0-9]{8}",mno)
 if(res!=None):
 print("\nUr Mobile Number is Valid")
 break
 else:
 print("\tUr Mobile Number Should Not Non-
 Numerics-try again")
 else:
 print("\tInvalid Mobile Number Length--try again")
#Program for Extracting Marks of Students
#StudentMarksEx1.py
import re
gd="Rossum got 55 marks, Ritche got 66 marks, Travis got 44
 marks and Kinney got 33 marks"
markslist=re.findall("[0-9]{2}",gd)
print("-----")
print("Students Marks List")
print("-----")
for marks in markslist:
 print("\t{}".format(marks))
print("-----")
#Program for Extracting Names of Students
#StudentNamesEx1.py
import re
gd="Rossum got 55 marks, Ritche got 66 marks, Travis got 44
 marks and Kinney got 33 marks"

```

```

nameslist=re.findall("[A-Z][a-z]+",gd)
print("-----")
print("Students Names List")
print("-----")
for names in nameslist:
 print("\t{}".format(names))
print("-----")

#Program for Extracting Names and Marks of Students
#StudentNamesMarksEx1.py
import re
gd="Rossum got 55 marks, Ritche got 66 marks, Travis got 44
marks and Kinney got 33 marks"
nameslist=re.findall("[A-Z][a-z]+",gd)
markslist=re.findall("[0-9]{2}",gd)
print("-----")
print("Students Names \tMarks")
print("-----")
for names,marks in zip(nameslist,markslist):
 print("\t{}\t{}".format(names,marks))
print("-----")

#Program for obtaining names and mail ids from file
#StudentMailsNamesEx1.py
import re
try:
 with open("D:\\KVR-PYTHON-
 9AM\\REGEPRS\\INDEX\\mails.info","r") as fp:
 filedatal=fp.read()
 #get the nameslist
 nameslist=re.findall("[A-Z][a-z]+",filedata)
 #get the mail ids
 mailslist=re.findall("\S+@\S+",filedata)
 print("="*50)
 print("\tNames\tMails")
 print("="*50)
 for names,mails in zip(nameslist,mailslist):
 print("\t{}\t{}".format(names,mails))
 print("="*50)
except FileNotFoundError:
 print("File does not exist")

#Program for Extracting Names and Marks of Students
#StudentNamesMarksEx1.py
import re
try:
 with open("D:\\KVR-PYTHON-
 9AM\\REGEPRS\\INDEX\\students.data","r") as fp:
 filedatal=fp.read()

```

```

#get the nameslist
nameslist=re.findall("[A-Z] [a-z]+",filedata)
#get markslist
markslist=re.findall("\d{2}",filedata)
print("=*50)
print("\tNames\tMarks")
print("=*50)
for names,marks in zip(nameslist,markslist):
 print("\t{}\t{}".format(names,marks))
print("=*50)
except FileNotFoundError:
 print("File Does not Exist")
=====

#Program for Extracting Names and Marks of Students
#StudentNamesMarksEx1.py
import re
try:
 with open("D:\\KVR-PYTHON-
9AM\\REGEXPRS\\INDEX\\students.data","r") as fp:
 filedata=fp.read()
 #get the nameslist
 nameslist=re.findall("[A-Z] [a-z]+",filedata)
 #get markslist
 markslist=re.findall("\d{2}",filedata)
 print("=*50)
 print("\tNames\tMarks")
 print("=*50)
 for names,marks in zip(nameslist,markslist):
 print("\t{}\t{}".format(names,marks))
 print("=*50)
except FileNotFoundError:
 print("File Does not Exist")
=====

#SubFunEx.py
import re
txt = "The rain in Spain"
x = re.sub("\s", "->", txt)
print(x)
=====
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: `dictobj=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")
```

-----

```
#DictToJsonFile.py
import json
d={"sno":100,"name":"Rossum","marks":55.55,"cname":"PSF"}
with open("stud.json","w") as fp:
 json.dump(d,fp)
 print("Dict Object Data Saved in JSON File--verify:")
```

=====

```
#JsonFileToDict.py
import json
try:
 with open("stud.json","r") as fp:
 d=json.load(fp)
 print("-----")
 for k,v in d.items():
 print("\t{}\t{}".format(k,v))
 print("-----")
except FileNotFoundError:
 print("File does not exist")
```

=====

```
#JsonToDictEx.py
import json
emp='{"ID":"100", "NAME":"ROSSUM", "DSG":"AUTHOR"} '#JSON Data
 Format
#convert JSON Data into Dict Type
d=json.loads(emp) # Here loads() of json module converts json
data into dict object
#Here d is an object of dict
for k,v in d.items():
 print("\t{}\t{}".format(k,v))
```

=====

```
{"sno": 100, "name": "Rossum", "marks": 55.55, "cname": "PSF"}
```

=====

```
import qrcode
generating a QR code using the make() function
qr_img = qrcode.make("God Bless U, Course Completed!!!")
saving the image file
qr_img.save("kvr2.jpg")
```

=====