

=>By using python Programming Lang and Some Combined technologies , we develop the following Real Time Applications.

1. Development of Web Applications (Websites)

a) Java Programming Lang<-----Tech: Servlets, JSP, Hibernate, Spring, Spring Boot, Spring with Micro Services...etc

(Sun Micro System INC USA--->Oracle Corp)

b) C#.net Programming Lang<-----Tech: ASP.net, ASP with Micro Services.

(Micro Soft)

c) PYTHON Programming Language<-----Tech: Django , Flask, Bottle, Pyramid

→Python---Says--"Less Lines of Code and Gives more Meaning"

Becoz of Rich set of MODULES are Present in PYTHON Learning

Python=Learning About Modules

=>By using python Programming Lang and Some Combined technolgies , we develop the following Real Time Applications.

1. Development of Web Applications (Web Sites)

a) Java Programming Lang<-----Tech: Servlets, JSP, Hibernate , Spring, Spring Boot, Spring with Micro Services...etc

(Sun Micro System INC USA--->Oracle Corp)

b) C#.net Programming Lang<-----Tech: ASP.net, ASP with Micro Services.

(Micro Soft)

c) PYTHON Programming Language<-----Tech: Django, Flask ,Bottle, Pyramid

Python---Says--"Less Lines of Code and Gives more Meaning"

Becoz of Rich set of MODULES are Present in PYTHON

Learning Python=Learning About Modules

Day(3) continued previous topic 15-03-2024.

2. Development of Gaming Applications.

3. Development of Artificial Intelligence(AI) Applications

a) Machine Learning (ML)

b) Deep Learning (DL)

4. Development of Image Processing Applications.

5. Development of Audio and Video Based Applications

6. Development of Web Scrapping / Web Harvesting

7. Development of Business Applications (Apps)

8. Development of Scientific Applications

9. Development of Software Development (Project Development)

10. Development of OS Installers

11. Development of Languages (Spark, Scala)

12. Development of Desktop GUI Applications

13. Development of Data Analysis and Data Analytics

14. Development of Automation of Testing

15. Development of Complex Math Operations

16. Development of Console Based Applications (Non-GUI)

17. Development of Animation Applications

18. Development of CAD and CAM Based Applications

19. Development of Computer Vision

20. Python Used in Education System

===== Day-4 18-03-2024 History of Python =====

=>The Python Programming Language Conceived (Foundation Stone Laid down) in the Year 1980

=>The Python Programming Language Implemented(Bring into an action) in the Year 1989.

=>The Python Programming Language Officially Released in the Year 1991 Feb.

=>The Python Programming Language Developed by "Guido Van Rossum" (Father OR Author of Python).

=>The Python Programming Language Developed at CWI (Centrum Wiskunde Informatica) Research Institute in the Country Nether Landas.

=>The Python Programming Language Maintained and Managed by a Non-Commercial Organization Called PSF (Python Software Foundation).

=>The official website of PSF is www.python.org.

Version of Python--Used in MNCs

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

1. Python 1.x --->Here 1 Is called Major Version and X Represents 0 1 2 3,4,...etc and these called Minor Version
Python 1.x was already Outdated
Python 2.x does not Support Backward Compatability with Python 1.x
 2. Python 2.x---->Here 2 Is called Major Version and X Represents 0 1 2 3,4,5,6,7,...etc and these called Minor Version
Python 2.x was already Outdated
Python 3.x does not Support Backward Compatability with Python 2.x.
 3. Python 3.x----Here 3 Is called Major Version and X Represents 0 1 2 3,4,5,6,7,8,9,10,11,12,13 and these called Minor Version
Here Python 3.x is called Current Version
Python 3.8,3.9,3.10---Industry Current Version Used for Developing Real Time Applications (Secured)
Python 3.11,3.12---Are the Latest Versions (BugFix Stage)
Python 3.13-----Pre-Release / Future Version
-

DownLoading and Installation Process of Python

=>Goto www.python.org

=>Choose DownLoads---> Python 3.12.2 (click)

=>Ensure that Python 3.12.2 downloaded

=>Double Click on Python 3.12.2---ensure u Must set the Path--->Choose Install Now

Python & Full Stack Python @ 4:00 PM (IST) By Mr. K.V.Rao

Day-1 <https://youtu.be/GQmtAVZ4DkM>

Day-2 <https://youtu.be/mj-CP4yZM74>

Day-3 <https://youtu.be/5uMVFLx50w0>

Day-4 <https://youtu.be/7J4UcwMnLec>

Day-5 <https://youtu.be/OBUeqhOo2t4>

Day-6 <https://youtu.be/DFTIZwpIUu0>Day-7 https://youtu.be/197qSFAsa_8

➲ Drive Link: https://drive.google.com/drive/folders/1oHOpc9RC0ll3g0EO_ZOLneaX_oeOfM_V

Day-5 19-03-2024

Features of Python

=>Features of a Language are Nothing but Services OR Facilities Provided by Language Developers in the Languages, which are used by Language Programmers in development of Real Time Applications.

=>In Python Programming, we have 11 Features. They are

1. Simple
2. Platform Independet Langauge
3. Dynamically Typed
4. Interpreted
5. High Level
6. Robust (Strong)
7. Freeware and Open Source
8. Extensible
9. Embedded
10. Both Functional and Object Oriented Programming Lang
11. Supports Third Party APIs Such as numpy, pandas, matplotlib, scipy, scikit,

nltk,keras,scilearn..etc

Day-6 20-03-2024

1. Simple

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

Factor-1: Python Programming Provides "Rich Set of MODULES"

----- So that Python Programmer can Re-Use the Pre-Defined Code which is Present in MODULES without writing Our Own Source Code

Examples: calendar , random, math, cmath, os, oracledb, mysql. connector...etc

Definition of Module

A Module is a Collection of Functions, Data Members and Class names

Factor-2: Python Programming Provides In-Built Facility Called Garbage Collector.
So Garbage Collector Collects Un-Used Memory Space and Improves the performance of Python Based Applications.

Definition of Garbage Collector

=>A Garbage Collector is one of Python Background Program running Behind of Regular Python Program and whose Role is that To Collect Un-used memory Space and Improves the Performance of Python Based Applications.
=>Hence Garbage Collector takes care about automatic Memory Management.

Factor-3:

=>The Python Programming Provides User-Friendly Syntaxes and Makes to Programmer to write Error-Free Programs in Limited span of Time.

DAY-7 21-03-2024

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

1. Platform Dependent Languages
2. Platform Independent Languages

1. Platform Dependent Lang

=>In Platform Dependent Lang, Data Types differes from One OS to Another OS and These Lang are PLATFORM DEPENDENT.

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

C, C++ , Programming Lang Environment.

Data Types: Used for allocating memory space and for storing inputs

Data type	DOS	UNIX
Int	2	4
Float	4	8
Double	8	16
Char	1	1

- Since c, c++ Lang Related data types occupies different space in different OSes so it is called as platform dependent.

2. Platform Independent Language--Most Imp

=====
=>A platform is Nothing but Type of OS Being Used to Run our Application / Project / Program.
Ex: Java, python.

Java Programming Lang Environment.

Data Types: Used for allocating memory space and for storing inputs

Data type	DOS	UNIX
Byte	1	1
Short	2	2
Int	4	4
Long	8	8
Float	4	4
Double	8	8
Boolean	0	0
Char	2	2

- ❖ These data types are called predefined data types (or) primitive (or) scalar data types
- ❖ Hence Java is a platform independent.
- ❖ Java pre-defined data types takes same memory space on all types of oses and java Lang is platform independent language

Python Programming Lang Environment.

Data Types: Used for allocating memory space and for storing inputs

Definition of OS(Operating System):

=====
=>OS is a Software and It acts as Resource Allocation Manager and Resource De-Allocation Manager
(OR)

=>OS is one of Interface Between Program and Computer Hardware(Memory , Processor , I/O Devices ..etc

3. Dynamically Typed

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

1. Static Typed Programming Language.
2. Dynamically Typed Programming Language.

1. Static Typed Programming Language.

=>In Static Typed Programming Language, It mandatory for the programmer To Specify Variable Declaration (Data Type+ Identifiers) Otherwise we get Compile Time Errors

Example Task: Compute Sum of Two Numbers in C, C++, Java

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

=>The Problem of Static Typed Programming Languages is that The Programmer May not be Knowing The Data Type of Value accurately.

=>In Static Typed Programming Languages , It stored Particular Type Value Only But never allows to store Other Types of Values.

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

2. Dynamically Typed Programming Language.

=>In Dynamically Typed Programming Language, Programmers Need not write Variable Declaration. Internally Programming Language Execution Environment will data type of value, which is entered By Programmer.

=>The Advantage of Dynamically Typed Programming Languages, is that

- i) Programmer Need not write Data Type
- ii) Depends type of Value, Execution Environment will assign the Data

type

Examples:

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

```
>>> a=100  
>>> b=1.2  
>>> c=a+b  
>>> print(a,type(a))----- 100 <class 'int'>  
>>> print(b,type(b))----- 1.2 <class 'float'>  
>>> print(c,type(c))----- 101.2 <class 'float'>
```

Examples Languages: PYTHON

4. Interpreted Programming

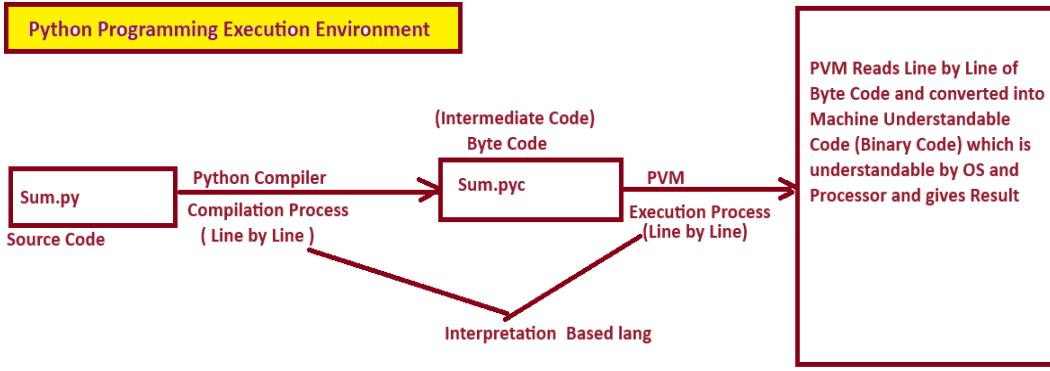
=>When we develop any python program, we must give some file name with an extension .py (File Name.py).

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

- a) Compilation Process
- b) Execution Process.

=>In COMPILEMENT 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) Line by Line and Converted into Machine Understandabel 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 Understandable Code (Executable or binary Code)

DAY-9 23/03/2024

7. Freeware and Open Source

Freeware:

=>If any Programming Language is Freely Downloadable from Official Source(www.python.org from PSF) then that language is called Freeware.

Examples: PYTHON, Java

Open Source:

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

=>Open Source is Nothing but making the Software to be available to all People.

=>Once the Software is Open Source then Some Software Companies came forward and Customized the Python Software for their In-House Tools Like Performance Evaluation, Quality of Testing, Releasing Process...etc.

=>The Customized Softwares of Python are called "Python Distributions".

=>Some of the Python Distributions are

1. JPython OR Jython---->Used for Running Java Based Applications.
2. IronPython OR Ipython--->Used for Runing C#.net Based Applications
3. Micro Python ----->Used for development of Micro Controller Applications.
4. Anaconda Python----->Used for Developing Big Data OR Hadoop Based Applications
5. StackLess Python----->used for developing Concurrency Applications
.....etc

6. High Level Programming

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

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

1. Low Level Programming Languages:

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

Example : a=0b1010101010---Binary Data

b=0xBEE-----Hexa Decimal Data
c=0o23-----Octal Data

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 Language.

Examples:

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

DAY-10 25/03/2024 (MON)

9. Extensible
 10. Embedded
-

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.

6. Robust (Strong)

=>Python Programming Provides a Programming Feature called "Exception Handling".
=>Due to exception handling facility, Python Based Application becomes Robust(Strong).

Definiton of Exception

=>Every Runtime Error of the Program is Called Exception
(Invalid Input---->Runtime Error---->Exception)
=>By default All Exceptions (Runtime Errors) gives Technical Error Message. Which are understandable by Programmers But not by End-Users and This is Not a Recommended Process in software industry.
=>Software industry recommends convert technical Error Message into User Friendly Error Message by using Exception Handling.

Defination of Exception Handling

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

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

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

DAY-11 26/03/2024 (TUE)

Data Representation in Python

Index

- =>What is Data
 - =>Purpose of Data
 - =>Types of Literals OR Values
 - =>Importance of Identifiers OR Variables
 - =>Rules for Using Variables OR Identifiers in Python
-

What is Data

- =>The processed Information is called Data
(OR)
 - =>Studying the insights of Information is called Data
 - =>The purpose of Data is that "To Effective Decisions in Organizations".
-

Types of Literals (OR) Values (OR) Data

In Python Programming, we have Different Types of Literals OR Values OR Data

1. Integer Literals
 2. Float Literals
 3. String Literals
 4. Boolean Literals
 5. Collections Literals
-

Importance of Identifiers OR Variables

- =>In Any Programming Language, Literals Must be Stored in Main Memory by allocating Sufficient Amount of Memory space with the Help Of Data Types.
 - =>We know that All types Literals are stored in main memory by having memory space.
 - =>To Process values which are present in memory space , Programmers must give DISTINCT NAMES to the created memory spaces. These DISTINCT 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.
-

Definition of Variable:

A Variable is an Identifier, whose Value can be Changed OR Varying During the Program Execution.

Rules for Using Variables OR Identifiers in Python

- =>To use Variables in Python Program, we must follow the Rules.
 - =>The Rules for using Variables in Python Program are
-

RULE-1: The Variable Name is a Combination of Alphabets, Digits and Special Symbol Under Score (_).

RULE-2: The First Letter of Variables Name must starts either with Alphabet OR Special Symbol Under Score (_) only.

Examples:

123sal=23-----	Invalid
_sal=45-----	Valid
sal12=34-----	valid
__sal=45-----	valid
_1=12-----	valid

RULE-3: Within the variable name, special symbols are not allowed except under score (_)

Examples

emp_sal=34-----Invalid
emp_sal=45-----valid
emp-sal=45-----invalid
emp\$sal=45----Invalid

RULE-4 :No Keywords to be used as Variable Names (bcoz Keywords are the Reserved Words and gives special

----- Meaning to the Language Compilers)

Examples:

while=45----Invalid
while1=45----Valid
if=34----invalid
_if=45-----Valid
True=56----Invalid
true=56----Valid

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

Examples

```
>>> age=99
>>> AGE=98
>>> Age=97
>>> aGe=96
>>> print(age,AGE,Age,aGe)-----99 98 97 96
```

DAY-12 27/03/2024 (WED)

Data types in Python

=>The purpose of Data Types in Python is that "To allocate Sufficient Amount of Memory Space in Main Memory for storing the Data".

=>In Python Programming, we have 14 Data Types and they are Classified into 6 categories. They are

I. Fundamental Category Data Types

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

II. Sequence Category Data Types

1. str
2. bytes
3. byte array
4. range

III. List Category Data Types (Collections Data Types)

1. list
2. tuple

IV. Set Category Data Types(Collections Data Types)

-
- 1. set
 - 2. frozenset
-

V. Dict Category Data Type (Collection 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 Python Programming, we have 4 Data Types in Fundamental Category . They are

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

1. int

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

=>The purpose of int Data Type is that "To Store Whole Numbers OR Integral Values (Values without Decimal Places)"

such as sno,htno,empno,acno..etc

Examples

Python Instructions	Output
---------------------	--------

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

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

=>By using int Data type, we can also store Different type of Number Systems.

=>In Python Programming, 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

=>This Number System is the Default Number System followed by all Human Beings for Their Day-to-Day Operations

=>This Number System contains the following

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

=>All Base 10 Values are called Decimal Number System Values.

Conversion from Decimal Number System value to Binary System Value	Conversion Binary Number System value to Decimal Number Value
<p>Q1) Convert $(13)_{10} \rightarrow (x)_2$ find $x=1101$</p> <p>Solution:</p> <p>The diagram shows the division of 13 by 2. The quotient is 6 and the remainder is 1. This process is repeated with the quotient 6, resulting in a quotient of 3 and a remainder of 0. Finally, the quotient 3 is divided by 2, resulting in a quotient of 1 and a remainder of 1. The remainders are then read from bottom to top to form the binary number 1101.</p> <p>Hence $(13)_{10} \rightarrow (1101)_2$</p>	<p>Q1) Convert $(1101)_2 \rightarrow (x)_{10}$ find $x=13$</p> <p>Solution:</p> $\begin{array}{r} 1 & 1 & 0 & 1 \\ \Rightarrow & 3 & 2 & 1 & 0 \\ \Rightarrow 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ \Rightarrow 8 + 4 + 0 + 1 \\ \Rightarrow 13 \end{array}$ <p>Hence $(1101)_2 \rightarrow (13)_{10}$</p>

2. Binary Number System

=>This Number System understandable by OS and Processor.

=>This Number System contains the following

Digits: 0 1 -----Total Digits: 2

Base : 2

=>All Base 2 Values are called Binary Number System Values.

=>In Python Programming, to Store Binary Data, The Binary Data must be preceded with a letter 0b OR 0B.

Syntax: varname=0b Binary Value

(OR)

varname=0B Binary Value

=>In Python Programming, even though we store Binary Data, Internally Python Execution Environment automatically OR Implicitly Converts into Decimal Number System.

Examples

```
>>> a=0b1  
>>> print(a,type(a))  
1 <class 'int'>  
>>> a=0b11  
>>> print(a,type(a))  
3 <class 'int'>  
>>> a=0b101  
>>> print(a,type(a))  
5 <class 'int'>  
>>> a=0b111  
>>> print(a,type(a))  
7 <class 'int'>
```

```
>>> a=0b1011  
>>> print(a,type(a))-----11 <class 'int'>  
>>> a=0b1111  
>>> print(a,type(a))-----15 <class 'int'>  
>>> a=0B10000  
>>> print(a,type(a))-----16 <class 'int'>  
>>> a=0b10102-----SyntaxError: invalid digit '2' in binary literal
```

NOTE: To Convert Decimal, Octal, Hexa Decimal Number System Value into Binary Number System Value, we use a

Pre-defined Function called bin()

varname=bin(Decimal / Octal / Hexa Decimal)

Examples

```
>>> bin(11)-----'0b1011'  
>>> bin(15)-----'0b1111'  
>>> bin(13)-----'0b1101'  
>>> bin(1)-----'0b1'  
>>> bin(2)-----'0b10'  
>>> bin(4)-----'0b100'  
>>> bin(5)-----'0b101'  
>>> bin(6)-----'0b110'  
>>> bin(7)-----'0b111'  
>>> bin(8)-----'0b1000'
```

3. Octal Number System

=> This Number System understandable by Micro Processor Kits Like in 8086 OR Assembly Language Programming.

=> This Number System contains the following

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

Base : 8

=> All Base 8 Values are called Octal Number System Values.

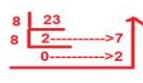
=> In Python Programming, to Store Octal Data, The Octal Data must be preceded with a letter 0o OR 0O.

Syntax: varname=0o Octal Value

(OR)

varname=0O Octal Value

=> In Python Programming, even though we store Octal Data, Internally Python Execution Environment automatically OR Implicitly Converts into Decimal Number System.

Conversion from Decimal Number System to Octal Number System	Conversion from Octal Number System to Decimal Number System
<p>Q1) $(23)_{10} \rightarrow (x)_8$ find x = 27</p> <p>Solution:</p> <p></p> <p>Hence $(23)_{10} \rightarrow (27)_8$</p>	<p>Q2) $(27)_8 \rightarrow (x)_{10}$ find x</p> <p>Solution:</p> <p></p> $\Rightarrow 2 \times 8 + 7 \times 8^0$ $\Rightarrow 2 \times 8 + 7 \times 1$ $\Rightarrow 16 + 7$ $\Rightarrow 23$ <p>Hence $(27)_8 \rightarrow (23)_{10}$</p>

Examples

```
>>> a=0o17  
>>> print(a,type(a))-----15 <class 'int'>
```

```
>>> a=0O123  
>>> print(a,type(a))-----83 <class 'int'>  
>>> a=0o168-----SyntaxError: invalid digit '8' in octal literal
```

NOTE: To Convert Decimal, Binary, Hexa Decimal Number System Value into Octal Number System Value, we use a

Pre-defined Function called oct()

varname=oct(Decimal / Binary / Hexa Decimal)

Examples

```
>>> oct(83)-----'0o123'  
>>> oct(15)-----'0o17'  
>>> oct(23)-----'0o27'
```

DAY-13 28/03/2024 (THU)

4. Hexa Decimal Number System

=>This Number System used in Development of OSes

=>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 : 16
(OR)
a(10) b(11) c(12) d(13) e(14) f(15)

Base: 16

=>All Base 16 Values are called Hexa Decimal Number System Values.

=>In Python Programming, to Store Hexa Decimal, The Hexa Decimal Data must be preceded with a letter 0x OR 0X.

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

varname=0X Hexa Decimal Value

=>In Python Programming, even though we store Hexa Decimal Number System Values, Internally Python Execution Environment automatically OR Implicitly Converts into Decimal Number System.

Examples

```
>>> a=0xac  
>>> print(a,type(a))-----172 <class 'int'>  
>>> a=0Xbee  
>>> print(a,type(a))-----3054 <class 'int'>  
>>> a=0xFACE  
>>> print(a,type(a))-----64206 <class 'int'>  
>>> a=0xFACER-----SyntaxError: invalid hexadecimal literal  
>>> a=0xACE  
>>> print(a,type(a))-----2766 <class 'int'>
```

NOTE: To Convert Decimal, Binary, Octal System Value into Hexa Decimal Number Value, we use a Pre-defined Function called hex()

varname=hex(Decimal / Binary / Octal)

Examples

```
>>> hex(172)-----'0xac'  
>>> hex(3054)-----'0xbe'  
>>> hex(64206)-----'0xface'
```

```
>>> hex(2766)-----'0xace'
```

Conversion from Decimal Number Data to Hexa Decimal Number System	Conversion from Hexa Decimal Number System to Decimal Number System Data
<p>Q1) $(172)_{10} \rightarrow (x)_{16}$ find $x=AC$</p> <p>Solution:</p> <p></p> <p>Hence $(172)_{10} \rightarrow (AC)_{16}$</p>	<p>Q1) $(AC)_{16} \rightarrow (x)_{10}$</p> <p>Solution:</p> <p>$\begin{array}{r} A & C \\ 1 & 0 \\ 16 & 16 \\ \hline \end{array}$</p> <p>$=> A \times 16^1 + C \times 16^0$</p> <p>$=> 10 \times 16 + 12 \times 1$</p> <p>$=> 160 + 12$</p> <p>$=> 172$</p> <p>Hence $(AC)_{16} \rightarrow 172)_{10}$</p>

Base Conversions Techniques in Python

=> In Python Programming, we have 3 types of Base Conversions Functions. They are

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

1. bin()

=> To Convert Decimal, Octal, Hexa Decimal Number System Values into Binary Number System Value, we use a

Pre-defined Function called bin()

varname=bin(Decimal / Octal / Hexa Decimal)

Examples

```
>>> a=10  
>>> bin(a)-----'0b1010' # Dec to Binary
```

```
>>> a=0o17  
>>> bin(a)-----'0b1111' # Octal to Binary
```

```
>>> a=0xB  
>>> bin(a)-----'0b1011' # Hexa Decimal to Binary
```

2. oct()

=> To Convert Decimal, Binary, Hexa Decimal Number System Value into Octal Number System Value, we use a

Pre-defined Function called oct()

varname=oct(Decimal / Binary / Hexa Decimal)

Examples

```
>>> a=15  
>>> oct(a)-----'0o17' # Decimal to Octal
```

```
>>> a=0b10000  
>>> oct(a)-----'0o20' # Binary to Octal
```

```
>>> a=0XC  
>>> oct(a)-----'0o14' # Hexa Dec to Oct
```

3. hex()

NOTE: To Convert Decimal, Binary, Octal Number System Values into Hexa Decimal Number Value, we use a Pre-defined Function called `hex()`

Var name =hex(Decimal / Binary / Octal)

Examples

```
>>> a=15
>>> hex(a)-----'0xf' # Decimal to Hexa Decimal
>>> hex(162)-----'0xa2'
-----
>>> a=0b0011010
>>> hex(a)-----'0x1a' # Binary to Hexa Decimal
-----
>>> a=0o86-----Syntax Error: invalid digit '8' in octal literal
>>> a=0o76
>>> hex(a)-----'0x3e' # Octal to Hexa Decimal
```

DAY-14 29-03-2024 (FRI)

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)".
- =>The float data type is also used for Storing the Floating Point Value in the Scientific Notation. The Syntax of Scientific Notation is " Mantisa e Exponent " OR " Mantisa e -Exponent "
- =>The Eqv Floating Point Value for Scientific Notation if given Below
" Mantisa e Exponent " is Mantisa x 10 to the power of exponent
- =>The Advanatge of Scientific Notation is to Save Memory Space for Strong Big Floating point Values.
- =>The float data type never allows the programmer to specify the Binary , Octal and Hexa decimal values. But it allows only Decimal Number System Values.

Examples

```
>>> print(a,type(a))-----2.5e-26 <class 'float'>
-----
>>> a=0b1010.0b1111-----SyntaxError: invalid decimal literal
>>> a=0o12.0o34-----SyntaxError: invalid decimal literal
>>> a=0xaccc.0b111-----SyntaxError: invalid decimal literal
```

3. bool

=>'bool' is one of pre-defined class treated as Fundamental Data Type.
=>The purpose of bool data type is that "To Store True and False Values(Logical Values) ".
=>Here True and False are Keywords and considered as Values for bool data type.
=>Internally, The True value is Treated as 1 and False is treated as 0.
=>On Bool Values , We can Perform Arithmetic Operations.

Examples

```
>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=False
>>> print(b,type(b))-----False <class 'bool'>
-----
>>> a=true-----NameError: name 'true' is not defined.
>>> b=false-----NameError: name 'false' is not defined.
-----
>>> a=True
>>> b=False
>>> c=a+b
>>> print(c,type(c))-----1 <class 'int'>
>>> print(True+True)----2
>>> print(True+False-True)---0
-----
>>> print(True+2-1)----2
>>> print(3*True+2)---5
-----
>>> print(0b1111+True)----16
>>> print(0xC-True)-----11
>>> print(0o8-True)-----SyntaxError: invalid digit '8' in octal literal
-----
>>> print(True/True)-----1.0
>>> print(True//True)-----1
>>> print(False/True)-----0.0
>>> print(True/False)-----ZeroDivisionError: division by zero (Most Imp)
```

4. complex

=>'complex' is one of the pre-defined data type and treated as Fundamental Data Type.
=>The purpose of complex data type is that "To Store Complex Values and perform operations complex values".

=>The General Notation of Complex Number is Given Below.

a+bj OR a-bj
=>Here 'a' is called Real part
=>Here 'b' is called Imaginary part
=>here the Letter 'j' Represents $\sqrt{-1}$ OR $\text{sqr}(j) = -1$

=>Internally, Real part and Imaginary Part are Considered as float values.

=>To get Real part and Imaginary parts from Complex Object, we use Two Pre-Defined Attributes. they are'

1. real
2. Imag

Syntax1: `complexobj.real`----->Gives Real Part of Complex object

Syntax2: `complexobj.imag`----->Gives Imaginary Part of Complex object

Examples

```
>>> a=2+3j
>>> print(a,type(a))----- (2+3j) <class 'complex'>
>>> b=2-5j
>>> print(b,type(b))----- (2-5j) <class 'complex'>
>>> c=-2+4j
>>> print(c,type(c))----- (-2+4j) <class 'complex'>
>>> d=-2-5j
>>> print(d,type(d))----- (-2-5j) <class 'complex'>
>>> a=2+3J
>>> print(a,type(a))----- (2+3j) <class 'complex'>
-----
>>> a=2.3+3.4j
>>> print(a,type(a))----- (2.3+3.4j) <class 'complex'>
>>> b=-2.5-4.5j
>>> print(b,type(b))----- (-2.5-4.5j) <class 'complex'>
>>> c=-4.5+4.6j
>>> print(c,type(c))----- (-4.5+4.6j) <class 'complex'>
>>> d=3.4-10.2j
>>> print(d,type(d))----- (3.4-10.2j) <class 'complex'>
>>> e=2+4.5j
>>> print(e,type(e))----- (2+4.5j) <class 'complex'>
-----
>>> a=10j
>>> print(a,type(a))----- 10j <class 'complex'>
>>> b=2.3j
>>> print(b,type(b))----- 2.3j <class 'complex'>
>>> c=-3j
>>> print(c,type(c))----- (-0-3j) <class 'complex'>
>>> d=-3.5j
>>> print(d,type(d))----- (-0-3.5j) <class 'complex'>
-----
>>> a=2+3j
>>> print(a,type(a))----- (2+3j) <class 'complex'>
>>> print(a.real)----- 2.0
>>> print(a.imag)----- 3.0
-----
>>> 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=3-4.5j
>>> print(a,type(a))----- (3-4.5j) <class 'complex'>
>>> print(a.real)----- 3.0
>>> print(a.imag)----- -4.5
-----
>>> a=-2.5j
>>> print(a,type(a))----- (-0-2.5j) <class 'complex'>
>>> print(a.real)----- -0.0
>>> print(a.imag)----- -2.5
>>> print(a.imaginary)----- AttributeError: 'complex' object has no attribute 'imaginary'
```

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

Index

=>What is str

=>Definition of str

=>Notations of str

=>Types of strs

=>Syntax for storing str data

=>Memory Management of str data

- a) +Ve Indexing
 - b) -Ve Indexing

=>Operations on str data

- a) Indexing
 - b) Slicing Operation

=>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 triple double quotes and triple single quotes."

=>Def. of str:

=>str is a collection of Characters or Alphanumeric data or numeric data or any type 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:

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

=>With the help of triple double Quotes (" " " " ") and Triple 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
```

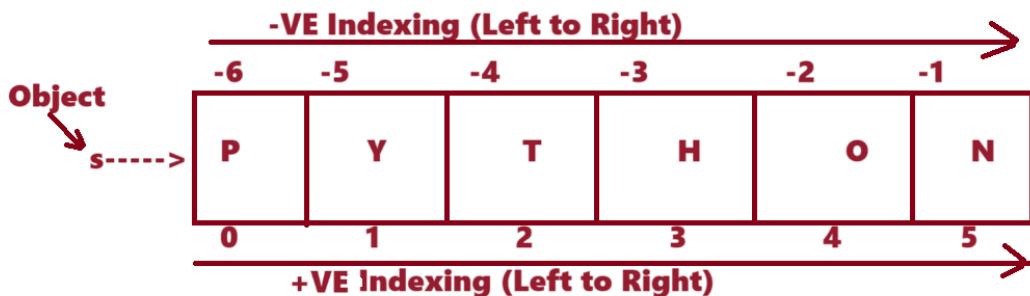
```
>>> s1="""Python Programming"""
>>> print(s1,type(s1))-----Python Programming <class 'str'>
>>> s1="Python Programming"
>>> print(s1,type(s1))-----Python Programming <class 'str'>
>>> s2="""A"""
>>> print(s2,type(s2))-----A <class 'str'>
>>> s2="A"
>>> print(s2,type(s2))-----A <class 'str'>
```

Memory Management of str Data

Consider the following str Data

>>>s="PYTHON"

=>When This Statement is Executed, In the Memory The above str Data Stored as follows



Operations on str data

=>On str Data, we can perform two Types of Operations. They are

1. Indexing Operation
2. Slicing Operations

1. Indexing Operation

=>The purpose of Indexing Operation is that "To get One Value at a time from str object".

=>In other words, The Process of Obtaining One Value from str object by passing valid Index is called Indexing.

=>Syntax: strobj[Index]

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

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

=>If we enter Valid Index then we get Corresponding Value from str object.

=>If we enter Invalid Index then we get IndexError as a Result.

Examples

```
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> print(s[0])-----P
>>> s[0]-----'P'
>>> s[1]-----'Y'
>>> s[2]-----'T'
>>> s[3]-----'H'
>>> s[5]-----'N'
```

```

>>>
>>> s[8]-----Index Error: string index out of range
>>> s[len(s)-1]-----'N'----> Here len(s) gives Number of Chars in 's' object i.e.6
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[-2]-----'O'
>>> s[-3]-----'H'
>>> s[-5]-----'Y'
>>> s[-4]-----'T'
>>> s[-6]-----'P'
>>> s[-1]-----'N'
>>> s[-11]-----IndexError: string index out of range
>>> s[-len(s)]-----'P'
>>> s[len(s)]-----IndexError: string index out of range

```

DAY-16 01-04-2024 _16.14.05_REC LIVE (mon)

2. Slicing Operations---

=>The Process of Obtaining Range Chars OR Values(Sub String) from Main String is called Slicing.
=>To Perform Slicing Operation, we have 5 Types of Syntaxes.

Syntax-1 : strobj[BeginIndex : EndIndex]

=>This Syntax Gives of Range Chars from BEGININDEX to ENDIndex -1 provided BeginIndex<EndIndex
Otherwise we
get Space or '' as Result

Examples

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:4]-----'PYTH'
>>> s[2:5]-----'THO'
>>> s[5:2]-----' '
>>> s[1:5]-----'YTHO'
>>> s[2:6]-----'THON'
>>> s[4:6]-----'ON'
>>> s[3:6]-----'ON'
>>> s[1:6]-----'THON'
>>> s[0:6]-----'PYTHON'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-5:-1]-----'YTHO'
>>> s[-2:-5]-----' '
>>> s[-5:-2]-----'YTH'
>>> s[-6:-1]-----'PYTHO'
>>> s[-3:-1]-----'HO'
>>> s[-5:-3]-----'YT'
>>> s[-6:-3]-----'PYT'

```

Most Imp Point--Sub Point

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[2:-2]-----'TH'
>>> s[1:-1]-----'YTHO'
>>> s[3:-1]-----'HO'

```

```

>>> s[1:-2]-----'YTH'
>>> s[0:-1]-----'PYTHO'
>>> s[2:-1]-----'THO'
>>> s[3:-2]-----'H'
-----
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6:4]-----'PYTH'
>>> s[-5:4]-----'YTH'
>>> s[-4:5]-----'THO'
>>> s[-6:3]-----'PYT'
>>> s[-5:3]-----'YT'
>>> s[-3:1]-----"
>>> s[-4:0]-----"
>>> s[-4:2]-----"
>>> s[-4:3]-----'T'

```

Most PowerFull Point

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:6]-----'PYTHON'
>>> s[0:122]-----'PYTHON'
>>> s[2:150]-----'THON'
>>> s[5:122]-----'N'
>>> s[122:345]-----'

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-122:-5]-----'P'
>>> s[-10:-1]-----'PYTHO'
>>> s[-9:-3]-----'PYT'
>>> s[-10:0]-----''
>>> s[-10:0]-----''

>>> s="PYTHON Prog"
>>> print(s)-----PYTHON Prog
>>> s[-122:456]-----'PYTHON Prog'
>>> s[-333:333]-----'PYTHON Prog'
>>> "JAVA"[-4:3]-----'JAV'
>>> "JAVA"[3:123]-----'A'
>>> "JAVA"[3:-1]-----''
>>> "JAVA"[0:-1]-----'JAV'

>>> "HYDERABAD"[-122:0]-----''
>>> "HYDERABAD"[-122:1]-----'H'
>>> "HYDERABAD"[-122:122]-----'HYDERABAD'
*****
```

Syntax-2: StrObj [BEGINIndex :]

=>In This Syntax, We Specify BEGIN Index and we Don'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 the Range of Chars from BEGIN Index to Last Character.

OR

=>This Syntax gives Range of Chars from BEGIN Index to Last Character.

Examples

```

>>> s="PYTHON"
>>> print(s)-----PYTHON

```

```

>>> s[0:-----'PYTHON'
>>> s[2:-----'THON'
>>> s[3:-----'HON'
>>> s[1:-----'YTHON'
>>> s[4:-----'ON'
-----
>>> s[4:]
'ON'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-122:-----'PYTHON'
>>> s[-2:-----'ON'
>>> s[-6:-----'PYTHON'
>>> s[-5:-----'THON'
>>> s[0:-----PYTHON
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[True:-----'THON'
>>> s[False:-----'THON'
>>> s[False:True]----'P'
>>> s[-0xf:-----'THON'
>>> s[-True:]-----'N'

```

Syntax-3: StrObj [: EndIndex]

- => In This Syntax, We Specify END Index and we Don't Specify BEGIN Index.
- => If we don't specify BEGIN Index Then PVM Takes BEGIN Index as Either 0 (+Ve) or -len(strobj)
- OR
- => If we don't specify BEGIN Index Then PVM Takes the Range of Chars from First Character ENDIndex-1 .
- OR
- => This Syntax gives Range of Chars from Range of Chars from First Character ENDIndex-1.

Examples

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:5]-----'PYTHO'
>>> s[:3]-----'PYT'
>>> s[:4]-----'PYTH'
>>> s[:6]-----'THON'
>>> s[:-3]-----'PYT'
>>> s[:-1]-----'THON'
>>> s[:-2]-----'PYTH'
>>> s[:0]-----'

```

DAY-17 02-04-2024 - LIVE (TUE)

Syntax-4: StrObj [:]-

- => In this Syntax, we didn't specify Begin Index and End Index.
- => If we don't Specify Begin Index then we get First Character Onwards
- => If we don't Specify End Index then we get upto Last Character Onwards
- => Hence This Syntax Gives Range Chars from First Character to Last Character (Complete Str Data)

Examples

```

>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[:]-----'THON'
>>> s[-122:222]-----'THON'

```

```

>>> s[0:-----'PYTHON'
>>> s[-6:-----'PYTHON'
>>> s[:122]-----'PYTHON'
-----
>>> s="JAVA"
>>> print(s)-----JAVA
>>> s[:]-----'JAVA'
>>> s[0:-----'JAVA'
>>> s[-4:-----'JAVA'
>>> s[:4]-----'JAVA'
>>> s[-122:333]-----'JAVA'
>>> s[-122:]-----'JAVA'
>>> s[:333]-----'JAVA'

```

NOTE: ALL THE ABOVE SYNTAXES ARE GIVING RANGE OF VALUES (SUB STRINGS) IN FORWARD DIRECTION WITH

DEFAULT STEP +1

Syntax-5: strobj[Begin :End :Step]

RULE-1: Here BEGIN , END and STEP values can be Either +VE or -VE

RULE-2: If the Value of STEP of +VE then PVM gets the Range of Values from BEGIN to END-1 Index in FORWARD ----- DIRECTION Provided BEGIN<END otherwise we get Space as Result OR '' as Result

Examples--RULE-2

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[::1]-----'PYTHON'
>>> s[::]-----'PYTHON'
>>> s[::2]-----'PTO'
>>> s[::3]-----'PH'
>>> s[::4]-----'PO'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:6:2]-----'PTO'
>>> s[2:6:3]-----'TN'
>>> s[4:2:2]-----' '
>>> s[1:5:3]-----'YO'
>>> s[1:10:2]-----'YHN'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6:-1:2]-----'PTO'
>>> s[-6:-2:4]-----'P'
>>> s[-5:-1:3]-----'YO'
>>> s[-6:-1:3]-----'PH'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0::2]-----'PTO'
>>> s[2::3]-----'TN'
>>> s[3::3]-----'H'
>>> s[-6::2]-----'PTO'
>>> s[-5::1]-----'THON'
>>> s[-5::]-----'THON'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:4:2]-----'PT'
>>> s[:6:3]-----'PH'
>>> s[:3:]-----'PYT'
>>> s[:6:4]-----'PO'
>>> s[:-1:1]-----'PYTHO'
>>> s[:-5:2]-----'P'
>>> s[:-122:1]-----"
>>> s[122::1]-----"

```

RULE-3: If the Value of STEP of -VE then PVM get the Range of Values from BEGIN to END+1 Index in BACKWARD DIRECTION Provided BEGIN>END otherwise we get Space as Result OR '' as Result

Examples:--Rule-3

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[5:1:-1]-----'NOHT'
>>> s[4:0:-1]-----'OHTY'
>>> s[5:0:-2]-----'NHY'
>>> s[0:5:-2]-----' '
>>> s[5:1:-3]-----'NT'
>>> s[::-2]-----'NHY'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6:-1:-1]-----' '
>>> s[-1:-6:-1]-----'NOHTY'
>>> s[-1:-6:-3]-----'NT'
>>> s[-2:-7:-2]-----'OTP'
>>> s[-3:-6:-3]-----'H'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[4::-1]-----'OHTYP'
>>> s[5::-2]-----'NHY'
>>> s[3::-1]-----'HTYP'
>>> s[4::-4]-----'OP'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6::-2]-----'P'
>>> s[-7::-2]-----''
>>> s[-6::-3]-----'P'
>>> s[-4::-2]-----'TP'
>>> s[-1::-5]-----'NP'
>>> s[-4::-3]-----'T'

```

```

>>> s="PYTHON"
>>> s[-6:5:-2]-----"
>>> s[5:-6:1]-----"
>>> s[126:-126:-1]-----'NOHTYP'
>>> s[4:-2:-1]-----"
>>> s[4:-3:-1]-----'O'
>>> s[-1:-5:-2]-----'NH'
>>> s[6:-7:-2]-----'NHY'

```

RULE-4: If FORWARD DIRECTION, if we specify the END INDEX as 0 then we get Space OR '' as Result

Examples: RULE-4:

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

RULE-5: If BACKWARD DIRECTION, if we specify the END INDEX as -1 then we get Space OR '' as Result

Examples: RULE-5:

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

PALINDROME WORDS

```
>>> s="LIRIL"
>>> s==s[::-1]-----True
>>> s="MADAM"
>>> s==s[::-1]-----True
>>> s="RACECAR"
>>> s==s[::-1]-----True
>>> s="MOM"
>>> s==s[::-1]-----True
>>> s="DAD"
>>> s==s[::-1]-----True
>>> s="PYTHON" # NOT a Palindrome Word
>>> s==s[::-1]-----False
>>> s="WOW"
>>> s==s[::-1]-----True
=====
=====
```

DAY-18 => 03-04-2024 (WED)

Type Casting Techniques in Python

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

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

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

1. int()

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

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

Example-1: float type into 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=0.45
>>> print(a,type(a))-----0.45 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-----0 <class 'int'>
*****
*****
```

Example-2: bool type to int type--POSSIBLE

```
*****
*****
```

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

Example-3: complex type to int type--NOT POSSIBLE

```
*****
*****
```

```
>>> a=2+3j
>>> print(a,type(a))-----(2+3j) <class 'complex'>
>>> b=int(a)-----TypeError: int() argument must be a string
*****
*****
```

Example-4: str type to int type

```
*****
*****
```

```
Case-1: str int into int type--POSSIBLE
-----
>>> a="123"
>>> print(a,type(a))-----123 <class 'str'>
>>> b=int(a)
>>> print(b,type(b))-----123 <class 'int'>
```

Case-2 : str float into int type--NOT POSSIBLE

```
-----
>>> a="123.45"
>>> print(a,type(a))-----123.45 <class 'str'>
>>> b=int(a)-----ValueError: invalid literal for int() with base 10: '123.45'
```

Case-3: str bool into int type--NOT POSSIBLE

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

Case-4: str complex into int type--NOT POSSIBLE

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

Case-5: pure str into int type--NOT POSSIBLE

```
-----
>>> a="PYTHON"
```

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

2. float()

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

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

Example-1: int type into 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'>
```

Example-2: bool type to float type--POSSIBLE

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

Example-3: complex type to float type--NOT POSSIBLE

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

Example-4: str type to float type

Case-1: str int into float type--POSSIBLE

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

Case-2 : str float into float type---POSSIBLE

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

Case-3: str bool into float type--NOT POSSIBLE

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

Case-4: str complex into float type--NOT POSSIBLE

```
>>> a="2+3j"
>>> print(a,type(a))-----2+3j <class 'str'>
>>> b=float(a)-----ValueError: could not convert string to float: '2+3j'
```

Case-5: pure str into float type--NOT POSSIBLE

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

3. `bool()`

=>bool() is used for Converting One Type of Possible Value into booleatype Value.

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

=>ALL NON-ZERO VALUES ARE TREATED AS TRUE

=>ALL ZERO VALUES ARE TREATED AS FALSE

Example-1: int type into 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'>
```

Example-2: float type to bool type--POSSIBLE

Example-3: complex type to 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+0j
>>> print(a,type(a))-----0j <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
*****
*****
```

Example-4: str type to bool type

```
*****
*****
```

Case-1: str int into bool type--POSSIBLE

```
-----
>>> a="10"
>>> print(a,type(a))-----10 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a="0"
>>> print(a,type(a))-----0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
```

Case-2 : str float into bool type--POSSIBLE

```
-----
>>> a="0.0"
>>> print(a,type(a))-----0.0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----3
-----
>>> a="1.2"
>>> print(a,type(a))-----1.2 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
```

Case-3: str bool into bool type----POSSIBLE

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

Case-4: str complex into bool type--POSSIBLE

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

Case-5: pure str into bool type--POSSIBLE

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

```
>>> a=""  
>>> print(a,type(a))----- "" <class 'str'>  
>>> b=bool(a)  
>>> print(b,type(b))-----True <class 'bool'>  
>>> len(a)-----4  
>>> a=""  
>>> len(a)-----0  
>>> print(a,type(a))-----"" <class 'str'>  
>>> b=bool(a)  
>>> print(b,type(b))-----False <class 'bool'>
```

DAY-19 04-04-2024 (THU)

4. complex()

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

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

Example-1: int type into complex type--POSSIBLE

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

Example-2: float type to complex type--POSSIBLE

```
>>> a=1.2  
>>> print(a,type(a))-----1.2 <class 'float'>  
>>> b=complex(a)  
>>> print(b,type(b))-----(1.2+0j) <class 'complex'>
```

Example-3: bool type to complex type--POSSIBLE

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

Example-4: str type to int type

Case-1: str int into complex type---POSSIBLE

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

Case-2 : str float into complex type---POSSIBLE

```
>>> a="12.3"  
>>> print(a,type(a))-----12.3 <class 'str'>  
>>> b=complex(a)  
>>> print(b,type(b))-----(12.3+0j) <class 'complex'>
```

Case-3: str bool into complex type---NOT POSSIBLE

```
>>> a="True"
```

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

Case-4: str complex into complex type--POSSIBLE

```
>>> a="2+3j"
>>> print(a,type(a))-----2+3j <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))-----(2+3j) <class 'complex'>
>>> a="2.3+4.6j"
>>> print(a,type(a))-----2.3+4.6j <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))-----(2.3+4.6j) <class 'complex'>
>>> a="2+4i"
>>> print(a,type(a))-----2+4i <class 'str'>
>>> b=complex(a)-----ValueError: complex() arg is a malformed string
```

Case-5: pure str into complex type---NOT POSSIBLE

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

5. str()

=>str() is used for Converting All Types of Values into str type value.

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

Examples

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

2. bytes

=>'bytes' is one of the Pre-Defined Class and Treated Sequence Data Type.

=>The Purpose of bytes data type is that "To Implement End-to-End Encryption" in network based Applications.

=>To Implement End-to-End encryption in network based Applications,bytes data type organizes the Numerical Integer values in range of 0 to 256 i.e It stores 0 to 255 values Only.

=>bytes data type does not contain any symbolic notation for storing bytes data bcoz Programmer will not store bytes data directly in the Program. But We Can convert any other type of values into bytes type by using bytes().

Syntax: varname=bytes(object)

=>An object of bytes maintains Insertion Order (Insertion order is Nothing but Whatever the order we organize in the

same order the data will be displayed)

=>On the Object of bytes, we can Perform Both Indexing and Slicing Operations.

=>An object of bytes belongs to IMMUTABLE bcoz bytes object does not support Item Assignment.

Examples

```
>>> lst=[100,200,10,0,256,45,67]
>>> print(lst,type(lst))-----[100, 200, 10, 0, 256, 45, 67] <class 'list'>
>>> b=bytes(lst)-----ValueError: bytes must be in range(0, 256)
>>> lst=[100,-200,10,0,255,45,67]
>>> print(lst,type(lst))-----[100, -200, 10, 0, 255, 45, 67] <class 'list'>
>>> b=bytes(lst)-----ValueError: bytes must be in range(0, 256)
```

```
>>> lst=[100,200,10,0,255,45,67]
>>> print(lst,type(lst))-----[100, 200, 10, 0, 255, 45, 67] <class 'list'>
>>> b=bytes(lst)
>>> print(b,type(b))-----b'd\xc8\n\x00\xff-C' <class 'bytes'>
```

```
>>> lst=[100,200,10,0,255,45,67]
>>> print(lst,type(lst))
[100, 200, 10, 0, 255, 45, 67] <class 'list'>
>>> b=bytes(lst)
>>> print(b,type(b))-----b'd\xc8\n\x00\xff-C' <class 'bytes'>
>>> for val in b:
...     print(val)
...
100
200
10
0
255
45
67
```

```
>>> b[0]-----100
>>> b[1]-----200
>>> b[-1]-----67
>>> b[1:5]-----b'\xc8\n\x00\xff'
>>> for val in b[1:5]:
...     print(val)
...
200
10
0
255
```

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

```
67
45
255
0
10
```

200
100

```
>>> b[0]=200-----TypeError: 'bytes' object does not support item assignment
```

DAY-20 05-04-2024 (FRI)

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

Mutable

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

Examples:

list, set,dict

Immutable Object

=>An Object is said to Immutable iff It satisfies the following the Properties

- 1) Immutable Objects Value can't be Changed at Same Address
(Immutable Objects Value can be changed and Modified value can be placed at Different Address)
- 2) Immutable Objects never allows us to Perform Item Assignment

Examples

int,float,bool,complex,str,bytes,range,tuple,set,frozenset,NoneType

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

=>'bytearray' is one of the Pre-Defined Class and Treated Sequence Data Type.

=>The Purpose of bytearray data type is that "To Implement End-to-End Encryption" in network based Applications.

=>To Implement End-to-End encryption in network based Applications ,bytearray data type organizes the Numerical Integer values in range of 0 to 256 i.e It stores 0 to 255 values Only.

=>bytearray data type does not contain any symbolic notation for storing bytearray data bcoz Programmer will not store bytearray data directly in the Program. But We Can convert any other type of values into bytearray type by using bytearray().

Syntax: varname=bytearray(object)

=>An object of bytearray maintains Insertion Order (Insertion order is Nothing but Whatever the order we organize in the same order the data will be displayed)

=>On the Object of bytearray, we can Perform Both Indexing and Slicing Operations.

=>An object of bytearray belongs to MUTABLE bcoz bytearray object supports Item Assignment.

NOTE: The functionality of bytearray is exactly similar to bytes But the object of bytes belongs to IMMUTABLE bcoz

bytes object does not support Item Assignment where an object bytearray belongs to MUTABLE bcoz an object of bytearray allows us to perform Item Assignment.

Examples

```
>>> lst=[100,200,256,0,55,166]  
>>> print(lst)-----[100, 200, 256, 0, 55, 166]  
>>> ba=bytearray(lst)----ValueError: byte must be in range(0, 256)  
>>> lst=[100,-200,255,0,55,166]  
>>> print(lst)-----[100, -200, 255, 0, 55, 166]  
>>> ba=bytearray(lst)----ValueError: byte must be in range(0, 256)
```

```

>>> lst=[100,200,255,0,55,166]
>>> print(lst,type(lst))-----[100, 200, 255, 0, 55, 166] <class 'list'>
>>> ba=bytearray(lst)
>>> print(ba,type(ba))-----bytearray(b'd\xc8\xff\x007\x a6') <class 'bytearray'>
>>> for val in ba:
...         print(val)
...
100
200
255
0
55
166
-----
>>> lst=[100,200,255,0,55,166]
>>> ba=bytearray(lst)
>>> print(ba,type(ba))-----byte array(b'd\xc8\xff\x007\x a6') <class 'bytearray'>
>>> ba[0]-----100
>>> ba[-1]-----166
>>> for val in ba[::-1]:
...         print(val)
...
166
55
0
255
200
100
-----
>>> lst=[100,200,255,0,55,166]
>>> ba=bytearray(lst)
>>> print(ba,type(ba),id(ba))---bytearray(b'd\xc8\xff\x007\x a6') <class 'bytearray'> 2233418515568
>>> ba[0]-----100
>>> ba[0]=150 # Item Assigment
>>> print(ba,type(ba),id(ba))---bytearray(b'\x96\xc8\xff\x007\x a6') <class 'bytearray'> 2233418515568
>>> for val in ba:
...         print(val)
...
150
200
255
0
55
166
=====X=====
=====
```

4. range

=>'range' is one of the pre-defined class and treated as Sequence Category Data type.
=>The purpose of range data type is that "To Maintain Sequence of Integer values by maintaining Equal Interval of Value (Step)".
=>On the object of range, we can perform both Indexing and slicing Operations.
=>An object of range belongs to IMMUTABLE bcoz range object does not support Item Assigment.
=>An object of range Maintains Insertion Order.
=>To create an object of range for storing Range OR Sequence of Values, we use 3 Pre-Defined Functions. They are

Syntax1: 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 val in r:
...         print(val)
...
0
1
2
3
4
5
>>> for val in range(6):
...         print(val)
...
0
1
2
3
4
5
```

Syntax2: range(Start,Stop)

=>This Syntax generates range of values from Start to Stop-1

Examples

```
>>> r=range(10,16)
>>> print(r,type(r))-----range(10, 16) <class 'range'>
>>> for val in r:
...         print(val)
...
10
11
12
13
14
15
>>> for val in range(100,106):
...         print(val)
...
100
101
102
103
104
105
```

NOTE: The above TWO Syntaxes,By deafult uses +1 Step

Syntax3: range(Start,Stop,Step)

=>This Syntax generates range of values from Start to stop by maintaining Equal Interval of Value (Step).

Examples

```
>>> r=range(10,21,2)
>>> print(r,type(r))-----range(10, 21, 2) <class 'range'>
>>> for val in r:
```

```
...     print(val)
...
      10
      12
      14
      16
      18
      20
>>> for val in range(100,107,2):
...         print(val)
...
      100
      102
      104
      106
```

Example

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

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

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

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

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

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

```
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 v in range(10,21):
```

```
...
print(v)
...
10
11
12
13
14
15
16
17
18
19
20
>>> for v in range(10,21,1):
...     print(v)
...
10
11
12
13
14
15
16
17
18
19
20
```

Q3) 10 15 20 25 30 35 40 45 50----range(10,51,5)

```
>>> for val in range(10,51,5):
...     print(val)
...
10
15
20
25
30
35
40
45
50
```

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

```
>>> for val in range(-9,0,1):
...     print(val)
...
-9
-8
-7
```

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

Q6) -50 -40 -30 -20 -10 ----range(-50,-9,10)

```
>>> for v in range(-50,-9,10):  
...     print(v)  
...
```

```
-50  
-40  
-30  
-20  
-10
```

```
>>> for v in range(-50,0,10):  
...     print(v)  
...
```

```
-50  
-40  
-30  
-20  
-10
```

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

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

```
-1  
-2  
-3  
-4  
-5  
-6  
-7  
-8  
-9
```

Q8) -100 -120 -140 -160 -180 -200----range(-100,-201,-20)

```
>>> for v in range(-100,-201,-20):  
...     print(v)
```

```
-100  
-120  
-140  
-160  
-180  
-200
```

```
>>> for v in range(-100,-220,-20):
```

```
...     print(v)
...
-100
-120
-140
-160
-180
-200
```

Q9) -5 -4 -3 -2 -1 0 1 2 3 4 5----range(-5,6) OR range(-5,6,1)

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

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

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

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

```
>>>
```

Q10) 1000 1010 1020 1030 1040 1050---range(1000,1051,10)

```
>>> for v in range(1000,1051,10):
...     print(v)
```

```
...
1000
1010
1020
1030
1040
1050
```

```
>>> r=range(1000,1041,10)
>>> r[0]-----1000
>>> r[-1]-----1040
>>> r[0:3]-----range(1000, 1030, 10)
>>> for v in r[0:3]:
```

```
...
print(v)
...
1000
1010
1020
>>> r[0]=120-----TypeError: 'range' object does not support item assignment
```

DAY-21 06/04/2024

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 the Types in single Object with Unique and Duplicates".

=>We have 2 Data Types in List Category. They are

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

FDT----Single Value

SDT----Sequence of Values

My Requirement today

1. I want to store 100 10 120 1000 -456 100 2000 400 10 12.34

Numerical values---Duplicates and Unique Values

2 I want to store 10 "Rossum",45.67,"Python",56.78,True

Numerical vals+Different types of Values+Both type

Store Employee Values in single object eno,ename,sal,dsg,city

1. list

Index

=>Purpose of list

=>Notation for storing list Data

=>Create an object of list

=>Types of Lists

=>Memory Management of List

=>Operations on List

=>Pre-Defined Functions in list

=>Inner List OR Nested List

=>Programming Examples

Properties of List

=>'list' is one of the pre-defined class and treated as List Category Data Type.

=>The purpose of list Data Type is that "To store Multiple Values either of Same Type OR Different Type OR Both the Types in single Object with Unique and Duplicates".

=>To Store the List data, we use Square Brackets. In Otherwords, the elements of list must be enclosed within Square Brackets [] and Values Separated by Comma.

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

=>Here varname is an object of list

=>An object of list Maintains Insertion Order.

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

=>An object of list belongs to MUTABLE bcoz list object allows us to perfom Modification / Updations / Assignments

by using Indexing and Slicing.

=>We can create Two types of list objects. They are

1. Empty list
2. Non-Empty list

Empty List

=>An empty list is one, which does not contain any Elements and whose length is 0.
=>Syntax1: listobj=[]
=>Syntax2: listobj=list()

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=[10,20,30,10,20,30,40]
>>> print(lst1,type(lst1))-----[10, 20, 30, 10, 20, 30, 40] <class 'list'>
>>> lst2=[100,"RS",34.56,True,2+3j]
>>> print(lst2,type(lst2))-----[100, 'RS', 34.56, True, (2+3j)] <class 'list'>
-----
>>> lst2=[100,"RS",34.56,True,2+3j]
>>> print(lst2,type(lst2))-----[100, 'RS', 34.56, True, (2+3j)] <class 'list'>
>>> lst2[0]-----100
>>> lst2[-1]-----(2+3j)
>>> lst2[len(lst2)-1]-----(2+3j)
>>> lst2[-len(lst2)]-----100
>>> lst2[1:4]-----['RS', 34.56, True]
>>> lst2[::2]-----[100, 34.56, (2+3j)]
>>> lst2[::1]-----[(2+3j), True, 34.56, 'RS', 100]
-----
>>> lst2=[100,"RS",34.56,True,2+3j]
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS', 34.56, True, (2+3j)] <class 'list'> 2234809075520
>>> lst2[1]-----'RS'
>>> lst2[1]="Guido Van Rossum" # Indexed Based Assignment
>>> print(lst2,type(lst2),id(lst2))-----[100, 'Guido Van Rossum', 34.56, True, (2+3j)] <class 'list'>
2234809075520
>>> lst2[2:4]-----[34.56, True]
>>> lst2[2:4]=[55.55,False] # Slice Based Assignment
>>> print(lst2,type(lst2),id(lst2))-----[100, 'Guido Van Rossum', 55.55, False, (2+3j)] <class 'list'>
2234809075520
>>> lst2=[100,"RS",34.56,True,2+3j]
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS', 34.56, True, (2+3j)] <class 'list'> 2234809078784
>>> lst2[::2]=[1000,44.44,-4-4.5j]
>>> print(lst2,type(lst2),id(lst2))-----[1000, 'RS', 44.44, True, (-4-4.5j)] <class 'list'> 2234809078784
-----
>>> lst1=[100,"RS"]
>>> print(lst1,type(lst1))-----[100, 'RS'] <class 'list'>
>>> len(lst1)-----2
-----
>>> lst1=[]
>>> print(lst1,type(lst1))-----[] <class 'list'>
>>> len(lst1)-----0
    OR
>>> lst2=list()
>>> print(lst2,type(lst2))-----[] <class 'list'>
>>> len(lst2)-----0
-----
>>> s="MISSISSIPPI"
>>> print(s,type(s))-----MISSISSIPPI <class 'str'>
>>> lst=list(s)
>>> print(lst,type(lst))-----['M', 'I', 'S', 'I', 'S', 'P', 'I', 'P', 'I'] <class 'list'>
```

```

>>> s="ABRAKADABRA"
>>> print(s,type(s))-----ABRAKADABRA <class 'str'>
>>> lst=list(s)
>>> print(lst,type(lst))-----['A', 'B', 'R', 'A', 'K', 'A', 'D', 'A', 'B', 'R', 'A'] <class 'list'>
>>> lst=[10,20,30,40]
>>> print(lst,type(lst))-----[10, 20, 30, 40] <class 'list'>
>>> b=bytes(lst)
>>> print(b,type(b))-----b'\n\x14\x1e(' <class 'bytes'>
>>> lst1=list(b)
>>> print(lst1,type(lst1))-----[10, 20, 30, 40] <class 'list'>
-----
#All Fundamental value are Not Possible to Convert into List bcoz they are all NON-ITERABLE Objects
>>>a=10
>>> lst=list(a)-----TypeError: 'int' object is not iterable
>>> lst=list(1.2)-----TypeError: 'float' object is not iterable
>>> lst=list(True)-----TypeError: 'bool' object is not iterable
>>> lst=list(2+3j)-----TypeError: 'complex' object is not iterable
-----
>>> lst=list([10]) # POSSIBLE
>>> print(lst,type(lst))-----[10] <class 'list'>
>>> lst=list([10.45])
>>> print(lst,type(lst))-----[10.45] <class 'list'>
>>> lst=list([True])
>>> print(lst,type(lst))-----[True] <class 'list'>
-----
>>> s="MISSISSIPPI"
>>> print(s,type(s))-----MISSISSIPPI <class 'str'>
>>> lst=list([s])
>>> print(lst,type(lst))-----['MISSISSIPPI'] <class 'list'>
=====X=====
=====
```

Pre-Defined Functions in list

=>We know that on the object of list, we can perform both Indexing and Slicing Operations
=>With Indexing, we get / Access Single Value and we can update single value of List object
=>With Slicing Operation we get / access multiple values and we can update Multiple Values of list object.
=>Hence with index and slicing Operation on list, we can access and update value(s).
=>To do More Operation on list object along with index and slicing Operation, we use the pre-defined functions present in list object.
=>The pre-defined functions of list are given bellow.

1. append()

=>This function is used for adding the element / value at the end of list (called Appending)
=>Syntax: listobj.append(Value)

Examples

```

>>> lst=[10,"RS"]
>>> print(lst,id(lst))-----[10, 'RS'] 2234811887168
>>> lst.append("Python")
>>> print(lst,id(lst))-----[10, 'RS', 'Python'] 2234811887168
>>> lst.append(34.56)
>>> print(lst,id(lst))-----[10, 'RS', 'Python', 34.56] 2234811887168
-----
>>> lst=[]
>>> print(lst,id(lst))-----[] 2234812410688
>>> lst.append(10)
>>> lst.append(1.2)
>>> lst.append(True)
>>> print(lst,id(lst))-----[10, 1.2, True] 2234812410688

```

2. insert()

=>Syntax: listobj.insert(Index,value)
=>This function is used for Inserting an Element in the Existing List at Specified Index
=>Here the Index Can be either +Ve or -Ve
=>If we Specify Invalid +VE Index then the Value Inserted as last Element in list
=>If we Specify Invalid -VE Index then the Value Inserted as First Element in list

Examples

```
>>> lst=[10,"RS",34.56]
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 34.56] <class 'list'> 2189625840192
>>> lst.insert(2,"Python")
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 'Python', 34.56] <class 'list'> 2189625840192
>>> lst.insert(1,"GUIDO")
>>> print(lst,type(lst),id(lst))-----[10, 'GUIDO', 'RS', 'Python', 34.56] <class 'list'> 2189625840192
-----
>>> lst=[10,"RS",34.56]
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 34.56] <class 'list'> 2189626161728
>>> lst.insert(-1,"Python")
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 'Python', 34.56] <class 'list'> 2189626161728
>>> lst.insert(-2,"HTML")
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 'HTML', 'Python', 34.56] <class 'list'> 2189626161728
-----
>>> lst=[10,"RS",34.56]
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 34.56] <class 'list'> 2189625840192
>>> lst.insert(100,"PYTHON")
>>> print(lst,type(lst),id(lst))-----[10, 'RS', 34.56, 'PYTHON'] <class 'list'> 2189625840192
>>> lst.insert(-100,"HYD")
>>> print(lst,type(lst),id(lst))-----['HYD', 10, 'RS', 34.56, 'PYTHON'] <class 'list'> 2189625840192
```

3. remove()-----Based on Value Removing

=>Syntax: listobj.remove(Value)
=>This function is used for Removing the First Occurrence of Specified Value from List Object.
=>If the Specified value does not exist then we get ValueError

Examples

```
>>> lst=['HYD', 10, 'RS', 34.56, 'PYTHON']
>>> print(lst,id(lst))-----['HYD', 10, 'RS', 34.56, 'PYTHON'] 2189626161728
>>> lst.remove(10)
>>> print(lst,id(lst))-----['HYD', 'RS', 34.56, 'PYTHON'] 2189626161728
>>> lst.remove("HYD")
>>> print(lst,id(lst))-----['RS', 34.56, 'PYTHON'] 2189626161728
>>> lst.remove("PYTHON")
>>> print(lst,id(lst))-----['RS', 34.56] 2189626161728
>>> lst.remove("HYD")-----ValueError: list.remove(x): x not in list
-----
>>> lst=[10,20,20,10,30,40,10,40]
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30, 40, 10, 40] 2189625840192
>>> lst.remove(10)
>>> print(lst,id(lst))-----[20, 20, 20, 30, 40, 10, 40] 2189625840192
>>> lst.remove(20)
>>> print(lst,id(lst))-----[20, 10, 30, 40, 10, 40] 2189625840192
>>> lst.remove(10)
>>> print(lst,id(lst))-----[20, 30, 40, 10, 40] 2189625840192
```

```
>>> lst.remove(100)-----ValueError: list.remove(x): x not in list  
-----  
>>> [].remove(100)-----ValueError: list.remove(x): x not in list  
>>> list().remove("PYTHON")---ValueError: list.remove(x): x not in list
```

4. pop(index)----Based Index Removing

=>Syntax: listobj.pop(index)
=>This Function is used for Removing an Element of List Based on Valid Index
=>Here Index can b either +ve or -ve
=>If we enter Invalid Index then we get IndexError

Examples

```
>>> lst=[10,20,20,10,30,40,10,40]  
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30, 40, 10, 40] 2189628867520  
>>> lst.pop(3)-----10  
>>> print(lst,id(lst))-----[10, 20, 20, 30, 40, 10, 40] 2189628867520  
>>> lst.pop(2)-----20  
>>> print(lst,id(lst))-----[10, 20, 30, 40, 10, 40] 2189628867520  
>>> lst.pop(-2)-----10  
>>> print(lst,id(lst))-----[10, 20, 30, 40, 40] 2189628867520  
>>> lst.pop(-2)-----40  
>>> print(lst,id(lst))-----[10, 20, 30, 40] 2189628867520  
>>> lst.pop(4)-----IndexError: pop index out of range  
  
>>> [].pop(0)-----IndexError: pop from empty list  
>>> list().pop(-1)---IndexError: pop from empty list
```

5. pop()

=>Syntax: listobj.pop()
=>This Function is used for Removing the Last Element of List
=>If we call this function in empty list object then we get IndexError

Examples

```
>>> lst=[10,20,20,10,30,40,10,40]  
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30, 40, 10, 40] 2189628907136  
>>> lst.pop()-----40  
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30, 40, 10] 2189628907136  
>>> lst.pop()-----10  
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30, 40] 2189628907136  
>>> lst.pop()-----40  
>>> print(lst,id(lst))-----[10, 20, 20, 10, 30] 2189628907136  
>>> lst.pop()-----30  
  
>>> lst=[10,20,30]  
>>> print(lst)-----[10, 20, 30]  
>>> lst.pop()-----30  
>>> print(lst)-----[10, 20]  
>>> lst.pop()-----20  
>>> print(lst)-----[10]  
>>> lst.pop()-----10  
>>> print(lst)-----[]  
>>> lst.pop()-----IndexError: pop from empty list  
>>> list().pop()-----IndexError: pop from empty list  
*****
```

NOTE : del operator

Syntax-1: del listobj[Index]----Removes One Element from List Based Indexing

Syntax-2: `del listobj[Begin Index : End Index:Step]`-->Removes Element(s) based on Slicing

Syntax-3: `del listobj----->`Removes All Elements + List Object also

Most IMP: `del` Operator Can Remove Elements of MUTABLE Objects either With Indexing, Slicing and entire Object ----- also where `del` operator can't be used to Elements of IMMUTABLE Objects bUt we can remove Entire Immutable Object (Content +Physical object)

=>Once we Remove The Mutable OR Immutable Object by using `del` Operator, whose memory space collected by Garbage Collector

Examples

```
>>> lst=[10,"RS","PYTHOn",34.56,True,2+3j]
```

```
>>> print(lst,id(lst))---->[10, 'RS', 'PYTHOn', 34.56, True, (2+3j)] 2189628867520
```

```
>>> del lst[-3]
```

```
>>> print(lst,id(lst))---->[10, 'RS', 'PYTHOn', True, (2+3j)] 2189628867520
```

```
>>> del lst[2:4]
```

```
>>> print(lst,id(lst))---->[10, 'RS', (2+3j)] 2189628867520
```

```
>>> del lst[:]
```

```
>>> print(lst,id(lst))---->[] 2189628867520
```

```
>>> lst=[10,"RS","PYTHOn",34.56,True,2+3j]
```

```
>>> print(lst,id(lst))---->[10, 'RS', 'PYTHOn', 34.56, True, (2+3j)] 2189628901568
```

```
>>> del lst
```

```
>>> print(lst,id(lst))---->NameError: name 'lst' is not defined
```

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

```
>>> print(s,type(s))---->PYTHON <class 'str'>----Its an IMMUTABLE
```

```
>>> del s[-2]---->TypeError: 'str' object doesn't support item deletion
```

```
>>> del s[::-2]---->TypeError: 'str' object does not support item deletion
```

```
>>> del s # Immutable Objects can Removed
```

```
>>> print(s,type(s))---->NameError: name 's' is not defined
```

6. clear()

=>Syntax: `listobj.clear()`

=>This Function is used for Removing all Elements from List object

=>If we call this function upon empty list object then we get Space or None as Result.

Examples

```
>>> lst=[10,"RS","PYTHOn",34.56,True,2+3j]
```

```
>>> print(lst,id(lst))---->[10, 'RS', 'PYTHOn', 34.56, True, (2+3j)] 2189628901568
```

```
>>> len(lst)-----6
```

```
>>> lst.clear()
```

```
>>> print(lst,id(lst))---->[] 2189628901568
```

```
>>> len(lst)-----0
```

```
>>> print([] .clear())---->None
```

OR

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

7. index()

=>Syntax: `listobj.index(Value)`

=>This Function is used for Obtaining the Index of First occurrence of Specified Element.

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

Examples

```
>>> lst=[10,20,30,40,10,50,60,10,20,10]
>>> print(lst)-----[10, 20, 30, 40, 10, 50, 60, 10, 20, 10]
>>> lst.index(10)-----0
>>> lst.index(20)-----1
>>> lst.index(200)-----ValueError: 200 is not in list
*****
```

DAY-23 10/04/2024 (WED)

enumerate():

```
*****
```

=>This Function is used for obtaining Index and Value Entries for Every Element of Iterable Object(Contains More than One Value)

=>Syntax: enumerate (IterableObject)

Examples

```
>>> lst=[10,20,30,40,10,50,60,10,20,10]
>>> print(lst,type(lst))-----[10, 20, 30, 40, 10, 50, 60, 10, 20, 10] <class 'list'>
>>> for i,v in enumerate(lst):
...         print(i,"--->",v)
...
    0 ---> 10
    1 ---> 20
    2 ---> 30
    3 ---> 40
    4 ---> 10
    5 ---> 50
    6 ---> 60
    7 ---> 10
    8 ---> 20
    9 ---> 10
>>> for i,v in enumerate(lst):
...         if(v==10):
...             print(i,"--->",v)
...
    0 ---> 10
    4 ---> 10
    7 ---> 10
    9 ---> 10
>>> for i,v in enumerate(lst):
...         if(v==20):
...             print(i,"--->",v)
    1 ---> 20
    8 ---> 20
>>> s="MISSISSIPPI"
>>> for index,value in enumerate(s):
...         print(index,"--->",value)
...
    0 ---> M
    1 ---> I
    2 ---> S
    3 ---> S
    4 ---> I
    5 ---> S
    6 ---> S
    7 ---> I
    8 ---> P
    9 ---> P
```

```
10 ---> I
>>> for index,Value in enumerate(s):
...     if(Value=="I"):
...         print(index,"--->",Value)
...
1 ---> I
4 ---> I
7 ---> I
10 ---> I
>>> for index,Value in enumerate(s):
...     if(Value=="S"):
...         print(index,"--->",Value)
...
2 ---> S
3 ---> S
5 ---> S
6 ---> S
```

8. copy()

=>Syntax: listobj2=listobj1.copy()
=>This Function is used for Copying the content of One List Object to Another List Object (It Implements Shallow Copy).

Examples

```
>>> lst1=[100,"RS"]
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS'] <class 'list'> 1376488641088
>>> lst2=lst1.copy() # Shallow Copy
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS'] <class 'list'> 1376491603904
>>> lst1.append("Python")
>>> lst2.insert(0,"NL")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS', 'Python'] <class 'list'> 1376488641088
>>> print(lst2,type(lst2),id(lst2))-----['NL', 100, 'RS'] <class 'list'> 1376491603904
```

9. count()

Syntax: listobj.count(Value)
=>This Function is used for Counting Number of Occurrences of Specified Value.
=>If the Specified Value does not exist then we get 0.

Examples

```
>>> lst=[10,20,30,40,10,50,60,10,20,10,20,30,40,20]
>>> print(lst,type(lst))-----[10, 20, 30, 40, 10, 50, 60, 10, 20, 10, 20, 30, 40, 20] <class 'list'>
>>> lst.count(10)-----4
>>> lst.count(20)-----4
>>> lst.count(30)-----2
>>> lst.count(40)-----2
>>> lst.count(50)-----1
>>> lst.count(60)-----1
>>> lst.count(160)-----0
>>> [].count(10)-----0
>>> list().count(100)-----0
...
>>> s="MISSISSIPPI"
>>> lst=list(s)
>>> print(lst)-----['M', 'I', 'S', 'I', 'S', 'I', 'P', 'I', 'P', 'I', 'P', 'I']
>>> lst.count("I")-----4
>>> lst.count("S")-----4
```

```
>>> lst.count("M")-----1
>>> lst.count("P")-----2
-----
>>> list("NISSON").count("S")-----2
>>> ["ABRAKADABRA"].count("A")-----0
>>> ["A","B","R","A","K","A","D","A","B","R","A"].count("A")---5
>>> list(["ABRAKADABRA"])[0].count("R")-----2
>>> ["ABRAKADABRA"][0]-----'ABRAKADABRA'
```

10. reverse()

Syntax: listobj.reverse()

=>This Function is used for reversing the elements(Front Elements becomes backelements and Vice-Versa) of listobject and Reversed Elements placed in same listobj itself.

Examples

```
>>> lst=[10,20,15,12,6,17]
>>> print(lst,id(lst))-----[10, 20, 15, 12, 6, 17] 1376491602752
>>> lst1=lst.reverse()
>>> print(lst,id(lst))-----[17, 6, 12, 15, 20, 10] 1376491602752
>>> print(lst1)-----None
>>> lst1=["Python","Java","HTML","C"]
>>> print(lst1,id(lst1))-----[‘Python’, ‘Java’, ‘HTML’, ‘C’] 1376491547520
>>> lst1.reverse()
>>> print(lst1,id(lst1))-----[‘C’, ‘HTML’, ‘Java’, ‘Python’] 1376491547520
```

11. sort()----Most Imp

=>Syntax1: listobj.sort()---->Gives the List Data in ASCENDING ORDER(Here by default reverse is False)

=>Syntax2: listobj.sort(reverse=False)--->Also Gives the List Data in ASCENDING ORDER

=>Syntax3: listobj.sort(reverse=True)---->Gives the List Data in DESCENDING ORDER

=>When we sort the Data, Data must Similar Otherwise we get TypeError

Examples

```
>>> lst1=[10,12,4,-5,0,23,11]
>>> print(lst1,id(lst1))-----[10, 12, 4, -5, 0, 23, 11] 1376491603200
>>> lst1.sort()
>>> print(lst1,id(lst1))-----[ -5, 0, 4, 10, 11, 12, 23] 1376491603200
>>> lst1.reverse()
>>> print(lst1,id(lst1))-----[23, 12, 11, 10, 4, 0, -5] 1376491603200
```

```
>>> lst1=[10,12,4,-5,0,23,11]
>>> print(lst1,id(lst1))-----[10, 12, 4, -5, 0, 23, 11] 1376491547520
>>> lst1.sort(reverse=True)
>>> print(lst1,id(lst1))-----[23, 12, 11, 10, 4, 0, -5] 1376491547520
```

```
>>> lst1=[10,12,4,-5,0,23,11]
>>> print(lst1,id(lst1))-----[10, 12, 4, -5, 0, 23, 11] 1376491603200
>>> lst1.sort(reverse=False)
>>> print(lst1,id(lst1))-----[ -5, 0, 4, 10, 11, 12, 23] 1376491603200
```

```
>>> lst1=[10,"RS",23.45,True,2+3j]
>>> lst1.sort()----TypeError: '<' not supported between instances of 'str' and 'int'
```

12. extend()

=>Syntax: listobj1.extend(listobj2)

=>This function is used for Merging the content of Two List Objects.

=>Here The content of Listobj2 is Merged with Listobj1

OR

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

Examples

```
>>> lst1=[10,20,30,40]
>>> lst2=["RS","TR","SR"]
>>> print(lst1)-----[10, 20, 30, 40]
>>> print(lst2)-----['RS', 'TR', 'SR']
>>> lst1.extend(lst2)
>>> print(lst1)-----[10, 20, 30, 40, 'RS', 'TR', 'SR']
```

OR

```
>>> lst1=[10,20,30,40]
>>> lst2=["RS","TR","SR"]
>>> print(lst1,id(lst1))-----[10, 20, 30, 40] 1376491603200
>>> print(lst2,id(lst2))-----['RS', 'TR', 'SR'] 1376491547520
>>> lst1=lst1+lst2
>>> print(lst1,id(lst1))-----[10, 20, 30, 40, 'RS', 'TR', 'SR'] 1376492918528
```

DAY-24 12-04-2024 FRIDAY

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

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

=>Syntax: `listobj=[Val1,Val2.....[val11,val12,...,Val1n], [Val21,Val22,...,Val2n],...,Val-n]`

Here Val1,Val2....,Val-n are called Outer List Elements

Here val11,val12...val1n are called One inner Elements

Here val21,val22...val2n are called another inner Elements

=>On Inner List, we can perform Both Indexing and Slicing Operations

=>On Inner List, we can apply all types of Pre-defined functions of list

=====
Examples

```
======>>> lst=[10,"RS",[15,17,19],[79,66,80],"OUCET"]
```

```
>>> print(lst)-----[10, 'RS', [15, 17, 19], [79, 66, 80], 'OUCET']
```

```
>>> for val in lst:
```

```
...     print(val,type(val),type(lst))
```

```
...
```

```
    10 <class 'int'> <class 'list'>
    RS <class 'str'> <class 'list'>
    [15, 17, 19] <class 'list'> <class 'list'>
    [79, 66, 80] <class 'list'> <class 'list'>
    OUCET <class 'str'> <class 'list'>
```

```
======>>> lst=[10,"RS",[15,17,19],[79,66,80],"OUCET"]
```

```
>>> print(lst,type(lst))-----[10, 'RS', [15, 17, 19], [79, 66, 80], 'OUCET'] <class 'list'>
```

```
>>> print(lst[2],type(lst[2]))-----[15, 17, 19] <class 'list'>
```

```
>>> print(lst[-2],type(lst[-2]))-----[79, 66, 80] <class 'list'>
```

```
>>> lst[0:3]-----[10, 'RS', [15, 17, 19]]
```

```
>>> lst[2:4]-----[[15, 17, 19], [79, 66, 80]]
```

```
>>> lst[2][::2]-----[15, 19]
```

```
>>> lst[-2][-2]-----66
```

```
>>> lst[-2][-2]=68
```

```
>>> print(lst,type(lst))-----[10, 'RS', [15, 17, 19], [79, 68, 80], 'OUCET'] <class 'list'>
```

```
>>> lst[-2][::2]=[75,76]
```

```

>>> print(lst,type(lst))-----[10, 'RS', [15, 17, 19], [75, 68, 76], 'OUCET'] <class 'list'>
>>> lst[-3].append(14)
>>> print(lst,type(lst))-----[10, 'RS', [15, 17, 19, 14], [75, 68, 76], 'OUCET'] <class 'list'>
>>> lst[-2].insert(-2,65)
>>> print(lst,type(lst))-----[10, 'RS', [15, 17, 19, 14], [75, 65, 68, 76], 'OUCET'] <class 'list'>
>>> lst[2].sort()
>>> print(lst,type(lst))-----[10, 'RS', [14, 15, 17, 19], [75, 65, 68, 76], 'OUCET'] <class 'list'>
>>> lst[-2].sort(reverse=True)
>>> print(lst,type(lst))-----[10, 'RS', [14, 15, 17, 19], [76, 75, 68, 65], 'OUCET'] <class 'list'>
>>> del lst[2][1::2]
>>> print(lst,type(lst))-----[10, 'RS', [14, 17], [76, 75, 68, 65], 'OUCET'] <class 'list'>
>>> lst[3].clear()
>>> print(lst,type(lst))-----[10, 'RS', [14, 17], [], 'OUCET'] <class 'list'>
>>> lst[-2].append(67)
>>> print(lst,type(lst))-----[10, 'RS', [14, 17], [67], 'OUCET'] <class 'list'>
>>> del lst[2]
>>> print(lst,type(lst))-----[10, 'RS', [67], 'OUCET'] <class 'list'>
>>> lst.insert(2,[16,14,18])
>>> print(lst,type(lst))-----[10, 'RS', [16, 14, 18], [67], 'OUCET'] <class 'list'>
>>> lst[3].append(80)
>>> lst[3].append(77)
>>> print(lst,type(lst))-----[10, 'RS', [16, 14, 18], [67, 80, 77], 'OUCET'] <class 'list'>
>>> del lst[2:4]
>>> print(lst,type(lst))-----[10, 'RS', 'OUCET'] <class 'list'>
=====
>>> matrix1=[[10,20,30],[40,50,60],[70,80,90]]
>>> for row in matrix1:
...     print(row)
...
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
>>> matrix2=[[1,2,3],[4,5,6],[7,8,9]]
>>> for row in matrix2:
...     print(row)
...
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
>>> mat3=matrix1+matrix2
>>> for row in mat3:
...     print(row)
...
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
=====
```

2. tuple

Index

- =>Properties of tuple
- =>What is tuple
- =>Notations of tuple
- =>Types of tuple
- =>Operations on tuple
- =>Pre-defined Functions in tuple

=>Nested OR Inner tuple

=>Combination tuples with list

=>Programming Examples

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 of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object . In otherwords tuple used for storing Constant Values of Multiple Values of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object.

=>To store the Elements OR Values in the object of tuple, we use braces () and the Values separated by comma.

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

=>An object of tuple belongs to IMMUTABLE bcoz tuple object does not allows us to perform Item Assignment.

=>An Object of tuple maintains Insertion Order.

=>In Python Programming, we can create Two Types of tuple objects. They are

- 1) Empty tuple
 - 2) Non-Empty tuple
-

1) Empty tuple

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

=>Syntax: varname=()

(OR)

varname=tuple()

2) Non-Empty tuple

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

=>Syntax: varname=(Val1,Val2,.....,Val-n)

OR

varname=tuple(object)

OR

varname=Val1,Val2,.....,Val-n

OR

varname=(Val1,)

NOTE: The Functionality of tuple is exactly similar to list, But an Object list belongs to MUTABLE where an object of tuple belongs to IMMUTABLE.

Examples

```
>>> t1=(10,20,30,10,40,20-23,2.3)
```

```
>>> print(t1,type(t1))-----(10, 20, 30, 10, 40, -3, 2.3) <class 'tuple'>
```

```
>>> t2=(10,"RS",34.56,True,2+3j)
```

```
>>> print(t2,type(t2))-----(10, 'RS', 34.56, True, (2+3j)) <class 'tuple'>
```

```
>>> t3=10,"TR",45.67,True,"NL"
```

```
>>> print(t3,type(t3))-----(10, 'TR', 45.67, True, 'NL') <class 'tuple'>
```

```
>>> t4=(10)
```

```
>>> print(t4,type(t4))-----10 <class 'int'>
```

```
>>> t4=(10,)
```

```
>>> print(t4,type(t4))-----(10,) <class 'tuple'>
```

```
>>> t4=10,
```

```
>>> print(t4,type(t4))-----(10,) <class 'tuple'>
```

```
>>> t1=(10,20,30,10,40,20-23,2.3)
```

```

>>> print(t1,type(t1))----- (10, 20, 30, 10, 40, -3, 2.3) <class 'tuple'>
>>> t1[0]-----10
>>> t1[-1]-----2.3
>>> t1[2]-----30
>>> t1[2:]----- (30, 10, 40, -3, 2.3)
>>> t1[::-2]----- (10, 30, 40, 2.3)
>>> t1[::-1]----- (2.3, -3, 40, 10, 30, 20, 10)
-----
>>> t1=(10,20,30,10,40,20-23,2.3)
>>> print(t1,type(t1),id(t1))----- (10, 20, 30, 10, 40, -3, 2.3) <class 'tuple'> 1785575627680
>>> t1[0]=25-----TypeError: 'tuple' object does not support item assignment---IMMUTABLE
-----
>>> s="MISSISSIPPI"
>>> print(s,type(s))-----MISSISSIPPI <class 'str'>
>>> t=tuple(s)
>>> print(t,type(t))-----('M', 'I', 'S', 'I', 'S', 'I', 'P', 'P', 'I') <class 'tuple'>
>>> l1=[10,20,30,40]
>>> print(l1,type(l1))-----[10, 20, 30, 40] <class 'list'>
>>> t1=tuple(l1)
>>> print(t1,type(t1))----- (10, 20, 30, 40) <class 'tuple'>
>>> t1=tuple(range(10,20,2))
>>> print(t1,type(t1))----- (10, 12, 14, 16, 18) <class 'tuple'>
-----
```

MOST IMP

```

>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> t=tuple(a)-----TypeError: 'int' object is not iterable
>>> t=tuple(12.34)-----TypeError: 'float' object is not iterable
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> t=tuple(a,)-----TypeError: 'int' object is not iterable
>>> t=tuple([a]) # Possible
>>> print(t,type(t))----- (10,) <class 'tuple'>
>>> t=tuple((a))-----TypeError: 'int' object is not iterable
=====X=====
```

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 present in tuple.

- 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
```

```
>>> t1=(10,20,30,40,50,10)
>>> print(t1,id(t1),type(t1))----(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>
>>> t2=t1 # Deep Copy Possible but Not Shallow Copy
>>> print(t2,id(t2),type(t2))----(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>
>>> t3=t1 # Deep Copy Possible but Not Shallow Copy
>>> print(t3,id(t3),type(t3))----(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>
```

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 bcoz tuple object belongs to Immutable but we can delete entire tuple 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'>
>>> 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'>
```

X

Types of Copy Techniques in Python

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

1. Shallow Copy
 2. Deep Copy
-

1. Shallow Copy

=>The Properties of Shallow Copy are

- a) The Initial Content of Both the Objects are Same.
- b) The Memory Address of Both the Objects are Different.
- c) The Modifications are Independent.

(Whatever the Changes we do on one Object, Those Changes are not Reflecting to another object bcoz Memory address are Different)

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

=>Syntax: Object2=Object1.copy()

Examples

```
>>> lst1=[100,"RS"]
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS'] <class 'list'> 1376488641088
>>> lst2=lst1.copy() # Shallow Copy
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS'] <class 'list'> 1376491603904
>>> lst1.append("Python")
>>> lst2.insert(0,"NL")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS', 'Python'] <class 'list'> 1376488641088
>>> print(lst2,type(lst2),id(lst2))-----['NL', 100, 'RS'] <class 'list'> 1376491603904
*****
```

2. Deep Copy

=>The Properties of Deep Copy are

- a) The Initial Content of Both the Objects are Same.
- b) The Memory Address of Both the Objects are Same.
- c) The Modifications are Dependent.

(Whatever the Changes we do on one Object, Those Changes are Reflecting to another object bcoz Memory address are Same)

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

=>Syntax: Object2 = Object1

Examples

```
>>> lst1=[100,"RS"]
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS'] <class 'list'> 1376492918528
>>> lst2=lst1 # Deep Copy
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS'] <class 'list'> 1376492918528
>>> lst1.append("PYTHON")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'RS', 'PYTHON'] <class 'list'> 1376492918528
>>> print(lst2,type(lst2),id(lst2))-----[100, 'RS', 'PYTHON'] <class 'list'> 1376492918528
>>> lst1.insert(1,"Guido")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'Guido', 'RS', 'PYTHON'] <class 'list'> 1376492918528
>>> print(lst2,type(lst2),id(lst2))-----[100, 'Guido', 'RS', 'PYTHON'] <class 'list'> 1376492918528
>>> lst2.remove("RS")
>>> print(lst1,type(lst1),id(lst1))-----[100, 'Guido', 'PYTHON'] <class 'list'> 1376492918528
>>> print(lst2,type(lst2),id(lst2))-----[100, 'Guido', 'PYTHON'] <class 'list'> 1376492918528
=====
```

Nested OR Inner tuple

=>The Process of Defining One tuple Inside of Another tuple is called Inner OR Nested tuple.
=>Syntax: tplobj=(Val1,Val2.....(Val11,Val12....Val-1n), (Val21,Val22,...Val-2n),....Val-n)
=>Here (Val1,Val2.....,Val-n) is called Outer tuple Eleemnts
=>Here (Val11,Val12....Val-1n) is called Inner tuple Elements
=>Here (Val21,Val22,...Val-2n) is also another Inner tuple Elements.
=>On the Inner tuple Objects, we can also Perform Both Indexing and Slicing Operations.
=>On the Objects Inner tuple, we can apply all the Pre-defined Functions of tuple (index(), count())

Examples

```
>>> t1=(10,"RS", (16,18,17),(67,80,78),"OUCET")
>>> print(t1,type(t1))-----(10, 'RS', (16, 18, 17), (67, 80, 78), 'OUCET') <class 'tuple'>
>>> for val in t1:
...     print(val,type(val),type(t1))
...
10 <class 'int'> <class 'tuple'>
RS <class 'str'> <class 'tuple'>
(16, 18, 17) <class 'tuple'> <class 'tuple'>
(67, 80, 78) <class 'tuple'> <class 'tuple'>
OUCET <class 'str'> <class 'tuple'>

>>> t1=(10,"Rossum", (17,16,18),(77,78,66),"OUCET")
>>> print(t1,type(t1))-----(10, 'Rossum', (17, 16, 18), (77, 78, 66), 'OUCET') <class 'tuple'>
>>> t1[0]-----10
>>> t1[1]-----'Rossum'
>>> t1[2]-----(17, 16, 18)
>>> t1[3]-----(77, 78, 66)
>>> t1[2][1]-----16
>>> t1[-2][-1]-----66
```

Possibility 1: List in Tuple

=>Syntax: tplobj=(Val1,Val2.....[Val11,Val12....Val-1n], [Val21,Val22,...Val-2n],....Val-n)

=>Here (Val1,Val2.....,Val-n) is called Outer tuple Eleemnts
=>Here [Val11,Val12....Val-1n] is called Inner list Elements
=>Here [Val21,Val22,...Val-2n] is also another Inner list Elements.

Examples

```
>>> t1=(10,"Rossum", [17,16,18],[77,78,66],"OUCET")
>>> print(t1,type(t1))-----(10, 'Rossum', [17, 16, 18], [77, 78, 66], 'OUCET') <class 'tuple'>
>>> print(t1[2],type(t1[2]))-----[17, 16, 18] <class 'list'>
>>> print(t1[3],type(t1[3]))-----[77, 78, 66] <class 'list'>
>>> t1[2].sort()
>>> print(t1,type(t1))-----(10, 'Rossum', [16, 17, 18], [77, 78, 66], 'OUCET') <class 'tuple'>
>>> t1[3].sort(reverse=True)
>>> print(t1,type(t1))-----(10, 'Rossum', [16, 17, 18], [78, 77, 66], 'OUCET') <class 'tuple'>
```

Possibility 2: tuple in list

=>Syntax: listobj=[Val1,Val2.....(Val11,Val12....Val-1n), (Val21,Val22,...Val-2n),....Val-n]

=>Here [Val1,Val2.....,Val-n] is called Outer list Eleemnts
=>Here (Val11,Val12....Val-1n) is called Inner tuple Elements
=>Here (Val21,Val22,...Val-2n) is also another Inner tuple Elements.

Examples

```
>>> l1=[10,"Rossum",(17,16,18),(77,78,66),"OUCET"]
>>> print(l1,type(l1))-----[10, 'Rossum', (17, 16, 18), (77, 78, 66), 'OUCET'] <class 'list'>
>>> l1[1]-----'Rossum'
>>> print(l1[2],type(l1[2]))-----(17, 16, 18) <class 'tuple'>
>>> print(l1[3],type(l1[3]))-----(77, 78, 66) <class 'tuple'>
=====x=====
```

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)

Set Category Data Types

=>The purpose of Set Category Data Types is that "To store Multiple Values either of Same Type OR Different Type OR Both the Types in single Object with Unique Elements Only (Duplicates are not allowed)".
=>We have 2 Data Types in Set Category. They are

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

set

Index

- =>Properties of set
- =>What is set
- =>Notations of set
- =>Types of sets
- =>Operations on sets
- =>Pre-defined Functions in sets
- =>Nested OR Inner sets
- =>Combination set with tuples with list
- =>Programming Examples

=>'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 the

Types in single Object with Unique Elements Only (Duplicates are not allowed)".

=>The values of set must stored within Curly Braces { } and Values must be separated by comma.

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

=>Set Object never maintains Insertion Order bcoz PVM Displays any of the Possibility of the elements of set.

=>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 Mutable bcoz doing changes at Same Address and Immutable bcoz It never allows us to perform item assignment

=>In Python Programming, we can create Two types of Set Objects. they are

- a) Empty Set
- b) Non-Empty Set

a) Empty-Set

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

=>Syntax: setobj=set()

b) Non-Empty-Set

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

=>Syntax: setobj={Val1,Val2,.....,val-n}
OR
setobj=set(object)

Examples

```
>>> s1={10,20,30,40,50,10,20}
>>> print(s1,type(s1))-----{50, 20, 40, 10, 30} <class 'set'>
>>> s2={"Python","HTML","CSS",23,45.78,True}
>>> print(s2,type(s2))-----{'CSS', True, 'HTML', 23, 45.78, 'Python'} <class 'set'>
-----
>>> s2={"Python","HTML","CSS",23,45.78,True}
>>> print(s2,type(s2))-----{'CSS', True, 'HTML', 23, 45.78, 'Python'} <class 'set'>
>>> s2[0]-----TypeError: 'set' object is not subscriptable
>>> s2[0:4]-----TypeError: 'set' object is not subscriptable
-----
>>> s1={10,"RS",34.56,True}
>>> print(s1,type(s1),id(s1))-----{True, 10, 'RS', 34.56} <class 'set'> 1796075992480
>>> s1[0]=False-----TypeError: 'set' object does not support item assignment--Immutable
>>> s1.add(2+3j)
>>> print(s1,type(s1),id(s1))-----{True, 'RS', 34.56, 10, (2+3j)} <class 'set'> 1796075992480--Mutable
-----
>>> s1=set()
>>> print(s1,type(s1))-----set() <class 'set'>
>>> len(s1)-----0
>>> s2={10,20,30,40,20}
>>> print(s2,type(s2))-----{40, 10, 20, 30} <class 'set'>
>>> len(s2)-----4
-----
>>> lst1=[10,20,30,10,20,10]
>>> print(lst1,type(lst1))-----[10, 20, 30, 10, 20, 10] <class 'list'>
>>> s1=set(lst1)
>>> print(s1,type(s1))-----{10, 20, 30} <class 'set'>
>>> s="MISSISSIPPI"
>>> print(s,type(s))-----MISSISSIPPI <class 'str'>
>>> s1=set(s)
>>> print(s1,type(s1))-----{'I', 'P', 'M', 'S'} <class 'set'>
```

Pre-defined Functions in sets

=>We know that on the object of set, we can't perform Indexing and slicing Operations bcoz set never maintains Insertion Order.
=>But On set objects we can perform different operations by using Pre-Defined Functions which are present in set object. They are

1. add()

=>Syntax: setobj.add(Value)

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

Examples

```
>>> s1={10,"Rossum",34.56}
>>> print(s1,type(s1),id(s1))-----{10, 'Rossum', 34.56} <class 'set'> 1796075985760
>>> s1.add("Python")
>>> print(s1,type(s1),id(s1))-----{10, 'Rossum', 34.56, 'Python'} <class 'set'> 1796075985760
>>> s1.add(34.67)
>>> print(s1,type(s1),id(s1))-----{34.56, 34.67, 10, 'Python', 'Rossum'} <class 'set'> 1796075985760
```

```
>>> s1=set()
>>> print(s1,type(s1),id(s1))-----set() <class 'set'> 1796075990240
>>> s1.add(100)
>>> s1.add("Travis")
>>> s1.add(True)
>>> s1.add(34.67)
>>> print(s1,type(s1),id(s1))-----{True, 34.67, 100, 'Travis'} <class 'set'> 1796075990240
*****
```

DAY-26 15/04/2024 (MONDAY)

2) remove()

Syntax: setobj.remove(value)

=>This Function is used for Removing the Value from set object

=>If the value does not exist in set object and if try to remove then we get KeyError

Examples

```
>>> s1={10,"RS",34.56,True,2+3j}
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 'RS', 10, (2+3j)} <class 'set'> 2000798133472
>>> s1.remove(10)
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 'RS', (2+3j)} <class 'set'> 2000798133472
>>> s1.remove(34.56)
>>> print(s1,type(s1),id(s1))-----{True, 'RS', (2+3j)} <class 'set'> 2000798133472
>>> s1.remove(100)-----KeyError: 100
```

```
>>> s=set()
>>> s.remove(100)-----KeyError: 100
>>> set().remove(12.34)----KeyError: 12.34
```

3) discard()

Syntax: setobj.discard(Value)

=>This Function is used for Removing the Value from set object

=>If the value does not exist in set object and if try to discard then we never get KeyError

Examples

```
>>> s1={10,"RS",34.56,True,2+3j}
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 'RS', 10, (2+3j)} <class 'set'> 2000798135712
>>> s1.discard(10)
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 'RS', (2+3j)} <class 'set'> 2000798135712
>>> s1.discard(34.56)
>>> print(s1,type(s1),id(s1))-----{True, 'RS', (2+3j)} <class 'set'> 2000798135712
>>> s1.discard(100)-----No KeyError will come
>>> s1.remove(100)-----KeyError: 100
```

```
>>> set().discard(100)
      OR
```

```
>>> print(set().discard(100))-----None
>>> set().remove(100)-----KeyError: 100
```

4) pop()

=>Syntax: setobj.pop()

=>This Function is used for Removing any ARBITRARY ELEMENT from set Object Provided NO ORDER OF DISPLAY

=>This Function is used for Removing always FIRST ELEMENT from set Object Provided ORDER OF DISPLAY is shown

Examples

```
>>> s1={10,"RS",34.56,True,2+3j} # NO Order of Display
>>> s1.pop()-----True
>>> s1.pop()-----34.56
>>> s1.pop()-----'RS'
>>> s1.pop()-----10
>>> s1.pop()-----(2+3j)
>>> s1.pop()-----KeyError: 'pop from an empty set'
>>> set().pop()-----KeyError: 'pop from an empty set'
-----
>>> s2={10,20,30,40,50,50,60,70}
>>> print(s2,type(s2),id(s2))---{50, 20, 70, 40, 10, 60, 30} <class 'set'> 2000798135712---Order of Display
>>> s2.pop()-----50
>>> print(s2,type(s2),id(s2))---{20, 70, 40, 10, 60, 30} <class 'set'> 2000798135712
>>> s2.pop()-----20
>>> print(s2,type(s2),id(s2))---{70, 40, 10, 60, 30} <class 'set'> 2000798135712
>>> s2.pop()----70
>>> s2.pop()----40
>>> s2.pop()----10
*****
```

5) clear()

=>Syntax: setobj.clear()
=>This function is used for Removing all the Elements from non-empty set object.
=>If we call this Function empty set object then we get none as a result

Examples

```
>>> s1={10,"RS",34.56,True,2+3j}
>>> print(s1,type(s1),id(s1))-----{True, 34.56, 'RS', 10, (2+3j)} <class 'set'> 2000798128992
>>> len(s1)-----5
>>> print(s1,type(s1),id(s1))-----set() <class 'set'> 2000798128992
>>> len(s1)-----0
>>> print(s1.clear())-----None
>>> print(set().clear())-----None
```

6) isdisjoint()

=>Syntax: setobj1.isdisjoint(setobj2)
=>This Function returns True provided There is no common element present in setobj1 and setobj2(called Disjoint Sets)
=>This Function returns False provided There is atleast one common element present in setobj1 and setobj2(called Non-Disjoint Sets)

Examples

```
>>> s1={10,20,30,40}
>>> s2={15,25,35}
>>> s3={15,10,56}
>>> s1.isdisjoint(s2)-----True
>>> s1.isdisjoint(s3)-----False
>>> s2.isdisjoint(s3)-----False
>>> s2.isdisjoint(s1)-----True
-----
>>> s1=set()
```

```
>>> s2=set()
>>> s1.isdisjoint(s2)-----True
>>> set().isdisjoint(set())----True
>>> set().isdisjoint({10,20,30})---True
*****
```

7) issuperset()

```
=>Syntax: setobj1.issuperset(setobj2)
=>This Function returns True provided setobj1 contains all the elements of setobj2 Otherwise It returns False.
```

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))-----{10, 20} <class 'set'>
>>> s1.issuperset(s2)-----True
>>> s2.issuperset(s1)-----False
```

```
>>> s1={10,20,30,40}
>>> s2={10,20,35,45}
>>> s1.issuperset(s2)-----False
>>> set().issuperset(set())---True
>>> set().issubset({10,20,30})---True
*****
```

8) issubset()

```
Syntax: setobj1.issubset(setobj2)
=>This Function returns True provided all the Elements of setobj1 are present in setobj2. Otherwise It returns False.
```

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,30,40,50,60,70}
>>> print(s1)-----{40, 10, 20, 30}
>>> print(s2)-----{50, 20, 70, 40, 10, 60, 30}
>>> s1.issubset(s2)---True
>>> s3={10,20,35,45,67}
>>> s1.issubset(s3)-----False
>>> s2.issubset(s1)-----False
>>> set().issubset(set())---True
>>> {10,20,30,40}.issubset(set())---False
*****
```

DAY-27 16/04/2024 (TUESDAY)

9) union()

```
Syntax: setobj3=setobj1.union(setobj2)
=>This Function is used for Obtaining Unique Values of setobj1 and setobj2 and Place them in setobj3.
```

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,25,35}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))-----{25, 10, 35, 20} <class 'set'>
>>> s3=s1.union(s2)
>>> print(s3,type(s3))-----{35, 40, 10, 20, 25, 30} <class 'set'>
```

```
>>> x={"Python","HTML","Django"}  
>>> y={"CSS","RestAPI","MySQL"}  
>>> z=x.union(y)  
>>> print(z,type(z))----{'CSS', 'Django', 'RestAPI', 'Python', 'HTML', 'MySQL'} <class 'set'>  
-----  
>>> s1={10,20,30,40}  
>>> s2={10,20,25,35}  
>>> s3={1,2,"Python"}  
>>> s4=s1.union(s2,s3)  
>>> print(s4,type(s4))-----{1, 2, 35, 40, 10, 20, 25, 'Python', 30} <class 'set'>  
>>>
```

10) intersection()

Syntax: setobj3=setobj1.intersection(setobj2)
=>This Function is used for Obtaining Common Elements from Both Setobj1 and setobj2

Examples

```
>>> s1={10,20,30,40}  
>>> s2={10,20,25,35}  
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>  
>>> print(s2,type(s2))-----{25, 10, 35, 20} <class 'set'>  
>>> s3=s1.intersection(s2)  
>>> print(s3,type(s3))-----{10, 20} <class 'set'>  
-----  
>>> x={"Python","HTML","Django"}  
>>> y={"CSS","RestAPI","MySQL"}  
>>> z=x.intersection(y)  
>>> print(z,type(z))-----set() <class 'set'>  
-----  
>>> s1={10,20,30,40}  
>>> s2={10,20,25,35}  
>>> s3={1,2,"Python"}  
>>> s4=s1.intersection(s2,s3)  
>>> print(s4,type(s4))-----set() <class 'set'>
```

11) difference()

=>Syntax: setobj3=setobj1.difference(setobj2)
=>This Function Removes the Common Elements from Both setobj1 and setobj2 and Takes Remaining Elements from Setobj1 and place them in setobj3.

Examples

```
>>> s1={10,20,30,40}  
>>> s2={10,20,25,35}  
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>  
>>> print(s2,type(s2))-----{25, 10, 35, 20} <class 'set'>  
>>> s3=s1.difference(s2)  
>>> print(s3,type(s3))-----{40, 30} <class 'set'>  
>>> s4=s2.difference(s1)  
>>> print(s4,type(s4))-----{25, 35} <class 'set'>  
-----  
>>> x={"Python","HTML","Django"}  
>>> y={"CSS","RestAPI","MySQL"}  
>>> z=x.difference(y)  
>>> print(z,type(z))-----{'Python', 'HTML', 'Django'} <class 'set'>
```

```
>>> z=y.difference(x)
>>> print(z,type(z))-----{'RestAPI', 'MySQL', 'CSS'} <class 'set'>
-----
```

```
>>> k={10,20,30}
>>> v={10,20,30}
>>> r=k.difference(v)
>>> print(r,type(r))-----set() <class 'set'>
*****
```

12) **symmetric_difference()**-----In Maths , we call this Operation as Delta

Syntax: setobj3=setobj1.symmetric_difference(setobj2)
=>This Function Removes the Common Elements from Both setobj1 and setobj2 and Takes Remaining Elements from Setobj1 and setobj2 and place them in setobj3.

OR

Syntax: setobj3=setobj1.union(setobj2).difference(setobj1.intersection(setobj2))

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,25,35}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))-----{25, 10, 35, 20} <class 'set'>
>>> s3=s1.symmetric_difference(s2)
>>> print(s3,type(s3))-----{35, 40, 25, 30} <class 'set'>
```

```
>>> x={"Python", "HTML", "Django"}
>>> y={"CSS", "RestAPI", "MySQL"}
>>> z=x.symmetric_difference(y)
>>> print(z,type(z))-----{'Django', 'CSS', 'HTML', 'RestAPI', 'Python', 'MySQL'} <class 'set'>
>>> k={10,20,30}
>>> v={10,20,30}
>>> r=k.symmetric_difference(v)
>>> print(r,type(r))-----set() <class 'set'>
```

-----By Formula-----

```
>>> s1={10,20,30,40}
>>> s2={10,20,25,35}
>>> s3=(s1.union(s2)).difference(s1.intersection(s2))
>>> print(s3,type(s3))-----{40, 25, 35, 30} <class 'set'>
```

13) **symmetric_difference_update()**

=>Syntax: setobj1.symmetric_difference_update(setobj2)
=>This Function Removes the Common Elements from Both setobj1 and setobj2 and Takes Remaining Elements from

Setobj1 and setobj2 and place them in setobj1 itself.

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,25,35}
>>> s1.symmetric_difference_update(s2)
>>> print(s1,type(s1))-----{35, 40, 25, 30} <class 'set'>
```

14) **update()**

Syntax: setobj1.update(setobj2)
=>This Function is used for Adding OR Merging of setobj2 Elements with setobj1

Examples

```
>>> s1={10,20,30,40}
>>> s2={10,20,25,35}
>>> s3=s1.update(s2)
>>> print(s1,type(s1))-----{35, 40, 10, 20, 25, 30} <class 'set'>
>>> print(s3,type(s3))-----None <class 'NoneType'>
```

```
>>> x={10,20}
>>> y={10,20}
>>> x.update(y)
>>> print(x,type(x))-----{10, 20} <class 'set'>
```

NOTE: del operator

```
>>> s1={10,20,30,40,50,60,70}
>>> print(s1,type(s1))-----{50, 20, 70, 40, 10, 60, 30} <class 'set'>
>>> del s1[1:4]-----TypeError: 'set' object does not support item deletion
>>> del s1[0]-----TypeError: 'set' object doesn't support item deletion
>>> del s1 # Possible
```

Solve the Following Problem By using Sets with Set Functions

Consider the following

```
Set Cricket Players={"Rohit","Kohli","Rossum"}
Set Tennis Players={"Travis","Kinney","Rossum"}
```

Write the Code for the following

- Find the names of all the players who are playing all types of Games
- Find the names of the players who are playing Both Cricket and Tennis
- Find the names of the players who are playing Only Cricket But not Tennis
- Find the names of the players who are playing Only Tennis But not Cricket
- Find the names of the players who are Exclusively playing Cricket and Tennis

Solutions

a) Find the names of all the players who are playing all types of Games-----union()

ANS:

```
>>> cp={"Rohit","Kohli","Rossum"}
>>> tp={"Travis","Kinney","Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> allcptp=cp.union(tp)
>>> print(allcptp,type(allcptp))----{'Kinney', 'Rossum', 'Rohit', 'Travis', 'Kohli'} <class 'set'>
```

b) Find the names of the players who are playing Both Cricket and Tennis--intersection()

ANS:

```
>>> cp={"Rohit","Kohli","Rossum"}
>>> tp={"Travis","Kinney","Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> bothcptp=cp.intersection(tp)
>>> print(bothcptp,type(bothcptp))-----{'Rossum'} <class 'set'>
```

c) Find the names of the players who are playing Only Cricket But not Tennis---difference()

ANS:

```
>>> cp={"Rohit","Kohli","Rossum"}
>>> tp={"Travis","Kinney","Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
```

```
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> onlycp=cp.difference(tp)
>>> print(onlycp,type(onlycp))-----{'Kohli', 'Rohit'} <class 'set'>
=====
```

d) Find the names of the players who are playing Only Tennis But not Cricket---difference()

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}
>>> tp={"Travis", "Kinney", "Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> onlycp=cp.difference(tp)
>>> print(onlycp,type(onlycp))-----{'Kohli', 'Rohit'} <class 'set'>
>>> onlytp=tp.difference(cp)
>>> print(onlytp,type(onlytp))-----{'Kinney', 'Travis'} <class 'set'>
=====
```

e) Find the names of the players who are Exclusively playing Cricket and Tennis--symmetric_difference()

ANS

```
>>> cp={"Rohit", "Kohli", "Rossum"}
>>> tp={"Travis", "Kinney", "Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> exclcptp=cp.symmetric_difference(tp)
>>> print(exclcptp,type(exclcptp))-----{'Kinney', 'Travis', 'Kohli', 'Rohit'} <class 'set'>
=====
```

Solve the Following Problem By using Sets without using set Functions
Here we are solving the following Problem with BITWISE OPERATORS

Consider the following

Set Cricket Players={"Rohit", "Kohli", "Rossum"}

Set Tennis Players={"Travis", "Kinney", "Rossum"}

Write the Code for the following

- Find the names of all the players who are playing all types of Games
- Find the names of the players who are playing Both Cricket and Tennis
- Find the names of the players who are playing Only Cricket But not Tennis
- Find the names of the players who are playing Only Tennis But not Cricket
- Find the names of the players who are Exclusively playing Cricket and Tennis

Solutions

a) Find the names of all the players who are playing all types of Games-----union() OR Bitwise OR (|)

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}
>>> tp={"Travis", "Kinney", "Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> allcptp=cp|tp # Bitwise OR ( | )
>>> print(allcptp,type(allcptp))----{'Kinney', 'Rossum', 'Rohit', 'Travis', 'Kohli'} <class 'set'>
=====
```

b) Find the names of the players who are playing Both Cricket and Tennis--intersection() OR Bitwise AND (&)

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}
>>> tp={"Travis", "Kinney", "Rossum"}
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>
>>> bothcptp=cp&tp # Bitwise AND ( & )
>>> print(bothcptp,type(bothcptp))-----{'Rossum'} <class 'set'>
```

c) Find the names of the players who are playing Only Cricket But not Tennis---difference() OR Arithmetic Minus (-)

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}  
>>> tp={"Travis", "Kinney", "Rossum"}  
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>  
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>  
>>> onlycp=cp-tp # Arithmetic Minus ( - )  
>>> print(onlycp,type(onlycp))-----{'Kohli', 'Rohit'} <class 'set'>
```

d) Find the names of the players who are playing Only Tennis But not Cricket---difference() OR Arithmetic Minus (-)

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}  
>>> tp={"Travis", "Kinney", "Rossum"}  
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>  
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>  
>>> onlycp=cp.difference(tp)  
>>> print(onlycp,type(onlycp))-----{'Kohli', 'Rohit'} <class 'set'>  
>>> onlytp=tp-cp # Arithmetic Minus ( - )  
>>> print(onlytp,type(onlytp))-----{'Kinney', 'Travis'} <class 'set'>
```

e) Find the names of the players who are Exclusively playing Cricket and Tennis -- symmetric_difference() OR

Bitwise XOR (^)

ANS:

```
>>> cp={"Rohit", "Kohli", "Rossum"}  
>>> tp={"Travis", "Kinney", "Rossum"}  
>>> print(cp,type(cp))-----{'Rossum', 'Kohli', 'Rohit'} <class 'set'>  
>>> print(tp,type(tp))-----{'Kinney', 'Travis', 'Rossum'} <class 'set'>  
>>> exclcptp=cp^tp # Bitwise XOR ( ^ )  
>>> print(exclcptp,type(exclcptp))-----{'Kinney', 'Travis', 'Kohli', 'Rohit'} <class 'set'>
```

DAY-28 17/04/2024 (WEDNESDAY)

Combination set with set,list and tuple

OR

Nested OR Inner Sets

Case-1 : It is Not Possible to Define One Set in Another Set bcoz sets are unhashable type(Not Possible to apply Indexing + Not Possible to Modify)

Examples:

```
>>> s1={10,"Rossum",{16,19,18},{77,76,75}, "OUCET"}-----TypeError: unhashable type: 'set'
```

Case-2: It is Not Possible to Define One List in Another Set bcoz sets are unhashable type(Not Possible to apply Indexing + Not Possible to Modify)

Examples

```
>>> s1={10,"Rossum", [16,19,18], [77,76,75], "OUCET"}-----TypeError: unhashable type: 'list'
```

Case-3: It is Possible to Define One Tuple in Another Set bcoz tuples are Immutable

Examples

```
>>> s1={10,"Rossum",(16,19,18),(77,76,75),"OUCET"}  
>>> print(s1,type(s1))----{(77, 76, 75), (16, 19, 18), 'OUCET', 'Rossum', 10} <class 'set'>  
>>> for val in s1:  
...     print(val,type(val))  
...  
(77, 76, 75) <class 'tuple'>  
(16, 19, 18) <class 'tuple'>  
OUCET <class 'str'>  
Rossum <class 'str'>  
10 <class 'int'>
```

Case-4: It is Possible to Define One Set in Another List bcoz Lists are mutable and allows us to locate set objects by using Indices.

Examples

```
>>> lst=[10,"Rossum",{16,19,18},{77,76,75}, "OUCET"]  
>>> print(lst,type(lst))----[10, 'Rossum', {16, 18, 19}, {75, 76, 77}, 'OUCET'] <class 'list'>  
>>> print(lst[2],type(lst[2]))----{16, 18, 19} <class 'set'>  
>>> lst[2].add(15)  
>>> print(lst)----[10, 'Rossum', {16, 18, 19, 15}, {75, 76, 77}, 'OUCET']  
>>> lst[-2].add(66)  
>>> print(lst)----[10, 'Rossum', {16, 18, 19, 15}, {66, 75, 76, 77}, 'OUCET']
```

Case-5: It is Possible to Define One Set in Another tuple bcoz Tuples are Immutable and allows us to locate set objects by using Indices.

Examples

```
>>> tpl=(10,"Rossum",{16,19,18},{77,76,75}, "OUCET")  
>>> print(tpl,type(tpl))----(10, 'Rossum', {16, 18, 19}, {75, 76, 77}, 'OUCET') <class 'tuple'>  
>>> print(tpl[2],type(tpl[2]))----{16, 18, 19} <class 'set'>  
>>> tpl[2].add(15)  
>>> print(tpl,type(tpl))----(10, 'Rossum', {16, 18, 19, 15}, {75, 76, 77}, 'OUCET') <class 'tuple'>  
>>> tpl[-2].remove(76)  
>>> print(tpl,type(tpl))----(10, 'Rossum', {16, 18, 19, 15}, {75, 77}, 'OUCET') <class 'tuple'>
```

=====

frozense

- =>'frozense' is one of the pre-defined class and treated as set data type.
=>The purpose of frozense data type is that "To store Multiple Values either Simiar Type or Different Type or Both the Types in Single Object with Unique Values".
=>The elements of frozense must be obtained from different objects like set , tuple and list..etc.
 Syntax: frozenseobj=frozense(set/list/tuple/str/bytes/bytarray/range)
=>An Object of frozense never maintains Insertion Order bcoz PVM can display any one of the possibility of elements of frozense object.
=>On the object of frozense, we can't perform Indexing and Slicing Operations bcoz frozense object never maintains Insertion Order.
=>An object of frozense belongs to Immutable bcoz frozense' object does not support item assignment and not possible to modify / Change / add.
=>we can create two types of frozense objects. They are
 a) Empty frozense
 b) Non-Empty frozense
-

a) Empty frozense:

- =>An Empty frozense 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 an object of set belongs to both Mutable(add,remove, pop, discard...etc) and also Immutable in the case of Item assigment Whereas frozenset object belongs to Immutable bcoz neither Possible to perform add,remove, pop, discard...etc nor possible to perform Item assigment.

Examples

```
>>> s1={10,20,30,10,20,60,70}
```

```
>>> print(s1,type(s1))-----{20, 70, 10, 60, 30} <class 'set'>
```

```
>>> fs1=frozenset(s1)
```

```
>>> print(fs1,type(fs1))-----frozenset({20, 70, 10, 60, 30}) <class 'frozenset'>
```

```
>>> s1={10,"RS",33.33,True}
```

```
>>> print(s1,type(s1))-----{33.33, 10, True, 'RS'} <class 'set'>
```

```
>>> fs2=frozenset(s1)
```

```
>>> print(fs2,type(fs2))-----frozenset({33.33, 10, True, 'RS'}) <class 'frozenset'>
```

```
>>> len(fs2)-----4
```

```
>>> fs3=frozenset()
```

```
>>> print(fs3,type(fs3))-----frozenset() <class 'frozenset'>
```

```
>>> len(fs3)-----0
```

```
>>> s1={10,"RS",33.33,True}
```

```
>>> print(s1,type(s1))-----{33.33, 10, True, 'RS'} <class 'set'>
```

```
>>> fs2=frozenset(s1)
```

```
>>> print(fs2,type(fs2))-----frozenset({33.33, 10, True, 'RS'}) <class 'frozenset'>
```

```
>>> fs2[0]-----TypeError: 'frozenset' object is not subscriptable
```

```
>>> fs2[0:3]-----TypeError: 'frozenset' object is not subscriptable
```

```
>>> fs2[0]=23-----TypeError: 'frozenset' object does not support item assignment
```

```
>>> del fs2[0]-----TypeError: 'frozenset' object doesn't support item deletion
```

```
>>> del fs2[0:2]-----TypeError: 'frozenset' object does not support item deletion
```

```
>>> del fs2 # Possible
```

```
>>> print(fs2)-----NameError: name 'fs2' is not found
```

Pre-Defined Functions in frozenset

=>frozenset contains the following Functions

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

NOTE:

```
>>> fs1=frozenset({10,20,30,409})
```

```
>>> print(fs1,type(fs1),id(fs1))-----frozenset({409, 10, 20, 30}) <class 'frozenset'> 2068835340960
```

```
>>> fs2=fs1.copy() # Deep Copy
```

```
>>> print(fs2,type(fs2),id(fs2))-----frozenset({409, 10, 20, 30}) <class 'frozenset'> 2068835340960
```

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

=>frozenset does not contain the following Functions

- a) clear()
- b) add()
- c) remove()
- d) discard()
- e) pop()
- f) update()
- g) symmetric_difference_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([10,20,50,60])) 
          frozenset({40, 50, 60, 30})
>>> fs1|fs2-----frozenset({100, 70, 40, 200, 10, 300, 50, 20, 60, 30})
=====X=====
```

DAY-29 18/04/2024 (THURSDAY)

Dict Category Data Type

=>The data Type in Dict Category is 'dict'.

=>dict is one of the pre-defined class

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

=>In (Key,value), the Values of Key Represents unique and Values of Value Represents May or May not be Unique.

=>To store (Key,value) in dict, we use curly braces and (Key,value) separated by comma.

=>Syntax: varname={Key1:Val1,Key2:Val2,.....,Key-n:Val-n}
 here Key1,Key2...Key-n Represents of Values of Key
 Val1,Val2,...Val-n Represents of Values of Values

=>An objct of dict maintains Insertion Order

=>An object of dict never Contains Indices. So that we can't perform Both Index and Slicing Operations.

=>An object of dict belongs to MUTABLE. In detail, the values of Key belongs to IMMUTABLE and Values of Value belongs to MUTABLE.

=>In Python Programming, we can create Two Types of Dict Objects. They are

a) Empty Dict

b) Non-Empty Dict

a) Empty Dict

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

=>Syntax: dictobj={}

(OR)
dictobj=dict()

b) Non-Empty Dict

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

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

=>Syntax Adding (Key,value) to Empty / Non-Empty Dict

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

dictobj[Key-n]=Val-n

=>here Key1,Key2...Key-n Represents of Values of Key
Val1,Val2,...Val-n Represents of Values of Values

=====

Examples

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

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

```
>>> d2={"Python":1,"C":2,"Java":3,"C++":4}
```

```
>>> print(d2,type(d2))-----{'Python': 1, 'C': 2, 'Java': 3, 'C++': 4} <class 'dict'>
```

```
>>> len(d2)-----4
```

```
>>> d3={100:1.2,200:1.3,300:1.2,400:1.3}
```

```
>>> print(d3,type(d3))-----{100: 1.2, 200: 1.3, 300: 1.2, 400: 1.3} <class 'dict'>
```

```
>>> d1={10:1.2,10:2.3,10:4.5,10:0.2}
```

```
>>> print(d1)-----{10: 0.2}
```

```
>>> d1={10: 1.2, 20: 2.3, 10: 1.9, 80 : 2.9}
```

```
>>> print(d1)-----{10: 1.9, 20: 2.3, 80: 2.9}
```

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

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

```
>>> d1[0]-----KeyError: 0
```

```
>>> d1[10]-----'Apple'
```

```
>>> d1[20]-----'Mango'
```

```
>>> d1[30]-----'Kiwi'
```

```
>>> d1[40]-----'Sberry'
```

```
>>> d1[50]-----KeyError: 50
```

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

```
>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry'} <class 'dict'>
```

```
1978790674944
```

```
>>> d1[10]—"Guava"
```

```
>>> print(d1,type(d1),id(d1))----{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry'} <class 'dict'>
```

```
1978790674944
```

```
>>> d1={}-----
```

```

>>> print(d1,type(d1))-----{} <class 'dict'>
>>> len(d1)-----0
    OR
>>> d2=dict()
>>> print(d2,type(d2))-----{} <class 'dict'>
>>> len(d2)-----0
-----
>>> d1={}
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'> 1978790785984
>>> len(d1)-----0
>>> d1[100]=1.2 # Inserted Entry
>>> d1[200]=2.2 # Inserted Entry
>>> d1[300]=1.2 # Inserted Entry
>>> d1[400]=4.2 # Inserted Entry
>>> print(d1,type(d1),id(d1))---{100: 1.2, 200: 2.2, 300: 1.2, 400: 4.2} <class 'dict'> 1978790785984
>>> len(d1)-----4
>>> d1[500]=5.5 # Inserted Entry
>>> print(d1,type(d1),id(d1))---{100: 1.2, 200: 2.2, 300: 1.2, 400: 4.2, 500: 5.5} <class 'dict'>
1978790785984
>>> d1[300]=0.2 # Modified Entry bcoz the key 300 already exist in d1
>>> print(d1,type(d1),id(d1))---{100: 1.2, 200: 2.2, 300: 0.2, 400: 4.2, 500: 5.5} <class 'dict'>
1978790785984
=====
```

Pre-Defined Functions in dict

=>We know that on dict object, we learned How to Insert the (Key,value) and How Modify the value of Value by passing Value of Key.Along with these Operations, we can also Perform different Operations by using Pre-defined Function of dict object.

1) clear()

=>Syntax: dictobj.clear()
=>This Function is used for Removing all the (Key,Value) from dict object
=>When we call this Function w.r.t empty dict then we get None as Result

Examples

```

>>> d1={10:1.2,20:3.4,30:1.2,40:4.5,50:3.4}
>>> print(d1,type(d1),id(d1))---{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5, 50: 3.4} <class 'dict'> 3078526298944
>>> len(d1)-----5
>>> d1.clear()
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'> 3078526298944
>>> len(d1)-----0
>>> print(d1.clear())-----None
>>> print({}.clear())-----None
>>> print(dict().clear())----None
=====
```

2) 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 then we get KeyError

Examples

```

>>> d1={10:1.2,20:3.4,30:1.2,40:4.5,50:3.4}
>>> print(d1,type(d1),id(d1))---{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5, 50: 3.4} <class 'dict'> 3078526259136
>>> d1.pop(30)-----1.2
>>> print(d1,type(d1),id(d1))---{10: 1.2, 20: 3.4, 40: 4.5, 50: 3.4} <class 'dict'> 3078526259136
>>> d1.pop(20)-----3.4
>>> print(d1,type(d1),id(d1))---{10: 1.2, 40: 4.5, 50: 3.4} <class 'dict'> 3078526259136
=====
```

```
>>> d1.pop(50)-----3.4
>>> print(d1,type(d1),id(d1))----{10: 1.2, 40: 4.5} <class 'dict'> 3078526259136
>>> d1.pop(120)-----KeyError: 120
>>> {}.pop(10)-----KeyError: 10
>>> dict().pop(20)-----KeyError: 20
```

3) popitem()

Syntax: dictobj.popitem()

=>This Function is used for Removing Last (Key,value) from Non-empty dict object

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

Examples

```
>>> d1={10:1.2,20:3.4,30:1.2,40:4.5,50:3.4}
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5, 50: 3.4} <class 'dict'> 3078526259392
>>> d1.popitem()-----(50, 3.4)
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5} <class 'dict'> 3078526259392
>>> d1.popitem()-----(40, 4.5)
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4, 30: 1.2} <class 'dict'> 3078526259392
>>> d1.popitem()-----(30, 1.2)
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4} <class 'dict'> 3078526259392
>>> d1.popitem()-----(20, 3.4)
>>> print(d1,type(d1),id(d1))----{10: 1.2} <class 'dict'> 3078526259392
>>> d1.popitem()-----(10, 1.2)
>>> print(d1,type(d1),id(d1))-----{} <class 'dict'> 3078526259392
>>> d1.popitem()-----KeyError: 'popitem(): dictionary is empty'
>>> {}.popitem()-----KeyError: 'popitem(): dictionary is empty'
>>> dict().popitem()-----KeyError: 'popitem(): dictionary is empty'
```

4) copy()

Syntax: dictobj2=dictobj1.copy()

=>This Function is used for Copying the content of One Dict object to another dict object(Implementation of Shallow Copy)

Examples

```
>>> d1={10:1.2,20:3.4,30:1.2}
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4, 30: 1.2} <class 'dict'> 3078527577088
>>> d2=d1.copy() # Shallow Copy
>>> print(d2,type(d2),id(d2))----{10: 1.2, 20: 3.4, 30: 1.2} <class 'dict'> 3078526259392
>>> d1[40]=4.5
>>> d2[25]=1.2
>>> print(d1,type(d1),id(d1))----{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5} <class 'dict'> 3078527577088
>>> print(d2,type(d2),id(d2))----{10: 1.2, 20: 3.4, 30: 1.2, 25: 1.2} <class 'dict'> 3078526259392
```

5) get()--Most Imp

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

=>This Function is used for Obtaining Value of Value by passing the value of Key

=>If the Value of Key does not exist then we get None as 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={10:1.2,20:3.4,30:1.2,40:4.5,50:3.4}
>>> print(d1,type(d1))----{10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5, 50: 3.4} <class 'dict'>
```

```

>>> val=d1.get(10)
>>> print(val)-----1.2
>>> val=d1.get(20)
>>> print(val)-----3.4
>>> val=d1.get(100)
>>> print(val)-----None
>>> #-----OR-----
>>> d1={10:1.2,20:3.4,30:1.2,40:4.5,50:3.4}
>>> print(d1,type(d1))----{(10: 1.2, 20: 3.4, 30: 1.2, 40: 4.5, 50: 3.4) <class 'dict'>}
>>> d1[10]-----1.2
>>> d1[20]-----3.4
>>> d1[100]-----KeyError: 100
*****

```

DAY- 30 (19/04/2024 FRIDAY)

6) keys()

Syntax: varname=dictobj.keys()
=>This Function is used for Obtainng Values of Key and placed in varname and whose type
<class,'dict_keys'>

Examples

```

>>> d1={10:"Python",20:"Django",30:"C++",40:"C"}
>>> print(d1,type(d1))----{(10: 'Python', 20: 'Django', 30: 'C++', 40: 'C') <class 'dict'>}
>>> ks=d1.keys()
>>> print(ks,type(ks))----dict_keys([10, 20, 30, 40]) <class 'dict_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 Obtainng Values of Value and placed in varname and whose type
<class,'dict_values'>

Examples

```

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

```

>>> for v in d1.values():
...         print(v)
...
Python
Django
C++
C
*****

```

8) items()

Syntax: varname=dictobj.items()

=>This Function is used for Obtainng (Key,value) from dict object and placed in varname and whose type <class,'dict_items'>

Examples

```

>>> d1={10:"Python",20:"Django",30:"C++",40:"C"}
>>> print(d1,type(d1))----{10: 'Python', 20: 'Django', 30: 'C++', 40: 'C'} <class 'dict'>
>>> dit=d1.items()
>>> print(dit,type(dit))---dict_items([(10, 'Python'), (20, 'Django'), (30, 'C++'), (40, 'C')]) <class 'dict_items'>
>>> for its in dit:
...         print(its,type(its))
...
(10, 'Python') <class 'tuple'>
(20, 'Django') <class 'tuple'>
(30, 'C++') <class 'tuple'>
(40, 'C') <class 'tuple'>
>>> for kv in d1.items():
...         print(kv,type(kv))
...
(10, 'Python') <class 'tuple'>
(20, 'Django') <class 'tuple'>
(30, 'C++') <class 'tuple'>
(40, 'C') <class 'tuple'>
>>>
>>> for k,v in d1.items():
...         print(k,"--->",v)
...
10 ---> Python
20 ---> Django
30 ---> C++
40 ---> C
=====
```

Combination of dict with dict,set,tuple and list

Case1: dict in dict --Nested Dict Possible to Define

Examples

```

>>>
d1={"sno":10,"sname":"RS","IntMarks":{"cm":17,"cpp":16,"os":19},"ExtMarks":{"cm":78,"cpp":77,"os":79},"cname":"CUCET"}
>>> print(d1,type(d1))
{'sno': 10, 'sname': 'RS', 'IntMarks': {'cm': 17, 'cpp': 16, 'os': 19}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'os': 79}, 'cname': 'CUCET'} <class 'dict'>
>>> for its in d1.items():
...         print(its)
...
('sno', 10)
('sname', 'RS')

```

```

('IntMarks', {'cm': 17, 'cpp': 16, 'os': 19})
('ExtMarks', {'cm': 78, 'cpp': 77, 'os': 79})
('cname', 'CUCET')
>>> for k,v in d1.items():
...     print(k,"--->",v)
...
    sno ---> 10
    sname ---> RS
    IntMarks ---> {'cm': 17, 'cpp': 16, 'os': 19}
    ExtMarks ---> {'cm': 78, 'cpp': 77, 'os': 79}
    cname ---> CUCET
>>> for k,v in d1.items():
...     print(k,"--->",v,"--->",type(v),type(d1))
...
    sno ---> 10 --> <class 'int'> <class 'dict'>
    sname ---> RS --> <class 'str'> <class 'dict'>
    IntMarks ---> {'cm': 17, 'cpp': 16, 'os': 19} --> <class 'dict'> <class 'dict'>
    ExtMarks ---> {'cm': 78, 'cpp': 77, 'os': 79} --> <class 'dict'> <class 'dict'>
    cname ---> CUCET --> <class 'str'> <class 'dict'>
>>> d1["IntMarks"]-----{'cm': 17, 'cpp': 16, 'os': 19}
>>> d1["IntMarks"]["DBMS"]=16
>>> d1["ExtMarks"]["DBMS"]=74
>>> print(d1,type(d1))
    {'sno': 10, 'sname': 'RS', 'IntMarks': {'cm': 17, 'cpp': 16, 'os': 19, 'DBMS': 16}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'os': 79, 'DBMS': 74}, 'cname': 'CUCET'} <class 'dict'>
>>> d1["IntMarks"].pop("cm")-----17
>>> d1["ExtMarks"].pop("os")-----79
>>> print(d1,type(d1))
    {'sno': 10, 'sname': 'RS', 'IntMarks': {'cpp': 16, 'os': 19, 'DBMS': 16}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'DBMS': 74}, 'cname': 'CUCET'} <class 'dict'>
-----
>>> d1={"sno":100,"name":"RS","IntMarks":{"cm":17,"c++":16},"ExtMarks":{"cm":70,"c++":67},"cname":"OU"}
>>> print(d1,type(d1))
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 17, 'c++': 16}, 'ExtMarks': {'cm': 70, 'c++': 67}, 'cname': 'OU'} <class 'dict'>
>>> for k in d1.keys():
...     print(k)
...
sno
name
IntMarks
ExtMarks
cname
>>> for v in d1.values():
...     print(v)
...
100
RS
{'cm': 17, 'c++': 16}
{'cm': 70, 'c++': 67}
OU
>>> for it in d1.items():
...     print(it)
...
('sno', 100)
('name', 'RS')
('IntMarks', {'cm': 17, 'c++': 16})
('ExtMarks', {'cm': 70, 'c++': 67})
('cname', 'OU')
>>> for gudu,sahu in d1.items():
...     print(gudu,"--->",sahu)

```

```

...
sno ---> 100
name ---> RS
IntMarks ---> {'cm': 17, 'c++': 16}
ExtMarks ---> {'cm': 70, 'c++': 67}
cname ---> OU
>>> for gudu,sahu in d1.items():
...     print(gudu,"--->",sahu,"--->",type(sahu),"--->",type(d1))
...
sno ---> 100 ---> <class 'int'> ---> <class 'dict'>
name ---> RS ---> <class 'str'> ---> <class 'dict'>
IntMarks ---> {'cm': 17, 'c++': 16} ---> <class 'dict'> ---> <class 'dict'>
ExtMarks ---> {'cm': 70, 'c++': 67} ---> <class 'dict'> ---> <class 'dict'>
cname ---> OU ---> <class 'str'> ---> <class 'dict'>
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 17, 'c++': 16}, 'ExtMarks': {'cm': 70, 'c++': 67}, 'cname': 'OU'}
>>> d1.get("IntMarks")
{'cm': 17, 'c++': 16}
>>> d1.get("ExtMarks")
{'cm': 70, 'c++': 67}
>>> for it in d1.get("IntMarks").items():
...     print(it)
...
('cm', 17)
('c++', 16)
>>> for k,v in d1.get("IntMarks").items():
...     print(k,"--->",v)
...
cm ---> 17
c++ ---> 16
>>> for k,v in d1.get("ExtMarks").items():
...     print(k,"--->",v)
...
cm ---> 70
c++ ---> 67
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 17, 'c++': 16}, 'ExtMarks': {'cm': 70, 'c++': 67}, 'cname': 'OU'}
>>> for k,v in d1["IntMarks"].items():
...     print(k,"--->",v)
...
cm ---> 17
c++ ---> 16
>>> for k,v in d1["ExtMarks"].items():
...     print(k,"--->",v)
...
cm ---> 70
c++ ---> 67
>>>
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 17, 'c++': 16}, 'ExtMarks': {'cm': 70, 'c++': 67}, 'cname': 'OU'}
>>> d1["IntMarks"]["cm"]
17
>>> d1["IntMarks"]["cm"]=18
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 18, 'c++': 16}, 'ExtMarks': {'cm': 70, 'c++': 67}, 'cname': 'OU'}
>>> d1["ExtMarks"]["cm"]=77
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 18, 'c++': 16}, 'ExtMarks': {'cm': 77, 'c++': 67}, 'cname': 'OU'}
>>> d1["ExtMarks"]["cm"]=d1["ExtMarks"].get("cm")+2
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 18, 'c++': 16}, 'ExtMarks': {'cm': 79, 'c++': 67}, 'cname': 'OU'}

```

```

>>> d1["ExtMarks"]["c++"] = d1["ExtMarks"]["c++"] + 2
>>>
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 18, 'c++': 16}, 'ExtMarks': {'cm': 79, 'c++': 69}, 'cname': 'OU'}
>>> d1["IntMarks"].pop("c++")
16
>>> print(d1)
{'sno': 100, 'name': 'RS', 'IntMarks': {'cm': 18}, 'ExtMarks': {'cm': 79, 'c++': 69}, 'cname': 'OU'}
>>> d1.pop("IntMarks")
{'cm': 18}
>>> print(d1)
{'sno': 100, 'name': 'RS', 'ExtMarks': {'cm': 79, 'c++': 69}, 'cname': 'OU'}
>>> d1["IntMarks"] = {"Java": 15, "Python": 16}
>>> print(d1)
{'sno': 100, 'name': 'RS', 'ExtMarks': {'cm': 79, 'c++': 69}, 'cname': 'OU', 'IntMarks': {'Java': 15, 'Python': 16}}
>>> d1["ExtMarks"]["os"] = 80
>>> print(d1)
{'sno': 100, 'name': 'RS', 'ExtMarks': {'cm': 79, 'c++': 69, 'os': 80}, 'cname': 'OU', 'IntMarks': {'Java': 15, 'Python': 16}}
>>> d1["ExtMarks"]["Bonus"] = {"CB": 1, "CPPB": 2}
>>> print(d1)
{'sno': 100, 'name': 'RS', 'ExtMarks': {'cm': 79, 'c++': 69, 'os': 80, 'Bonus': {'CB': 1, 'CPPB': 2}}, 'cname': 'OU', 'IntMarks': {'Java': 15, 'Python': 16}}
>>> for k, v in d1["ExtMarks"]["Bonus"].items():
...     print(k, "-->", v)
...
CB --> 1
CPPB --> 2
>>> d1.get("ExtMarks").pop("Bonus")
{'CB': 1, 'CPPB': 2}
>>> print(d1)
{'sno': 100, 'name': 'RS', 'ExtMarks': {'cm': 79, 'c++': 69, 'os': 80}, 'cname': 'OU', 'IntMarks': {'Java': 15, 'Python': 16}}

```

Case2: set in dict--Possible

```

>>> d1 = {"sno": 10, "sname": "RS", "IntMarks": {17, 19, 16}, "ExtMarks": {67, 77, 78}, "cname": "CUCET"}
>>> print(d1, type(d1)) --- {'sno': 10, 'sname': 'RS', 'IntMarks': {16, 17, 19}, 'ExtMarks': {67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>
>>> for k, v in d1.items():
...     print(k, "-->", v, "-->", type(v), type(d1))
...
sno --> 10 --> <class 'int'> <class 'dict'>
sname --> RS --> <class 'str'> <class 'dict'>
IntMarks --> {16, 17, 19} --> <class 'set'> <class 'dict'>
ExtMarks --> {67, 77, 78} --> <class 'set'> <class 'dict'>
cname --> CUCET --> <class 'str'> <class 'dict'>
>>> d1["IntMarks"] ----- {16, 17, 19}
>>> d1["IntMarks"].add(15)
>>> d1["ExtMarks"].add(64)
>>> print(d1, type(d1)) --- {'sno': 10, 'sname': 'RS', 'IntMarks': {16, 17, 19, 15}, 'ExtMarks': {64, 67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>
>>> d1.pop("IntMarks") ----- {16, 17, 19, 15}
>>> print(d1, type(d1)) ----- {'sno': 10, 'sname': 'RS', 'ExtMarks': {64, 67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>
>>> d1["ExtMarks"].clear() ----- >>> print(d1, type(d1))
{'sno': 10, 'sname': 'RS', 'ExtMarks': set(), 'cname': 'CUCET'} <class 'dict'>
>>> d1.pop("ExtMarks") ----- set()
>>> print(d1, type(d1)) --- {'sno': 10, 'sname': 'RS', 'cname': 'CUCET'} <class 'dict'>
>>> d1["IntMarks"] = {17, 16, 15}

```

```

>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'cname': 'CUCET', 'IntMarks': {16, 17, 15}} <class 'dict'>
=====
Case3: tuple in dict--Possible
=====
>>> d1={"sno":10,"sname":"RS","IntMarks":(17,19,16),"ExtMarks":(67,77,78),"cname":"CUCET"}
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': (17, 19, 16), 'ExtMarks': (67, 77, 78), 'cname': 'CUCET'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"--->",v,"--->",type(v),type(d1))
...
sno ---> 10 ---> <class 'int'> <class 'dict'>
sname ---> RS ---> <class 'str'> <class 'dict'>
IntMarks ---> (17, 19, 16) ---> <class 'tuple'> <class 'dict'>
ExtMarks ---> (67, 77, 78) ---> <class 'tuple'> <class 'dict'>
cname ---> CUCET ---> <class 'str'> <class 'dict'>
>>> d1["IntMarks"]-----(17, 19, 16)
>>> d1["IntMarks"][:2]-----(17, 16)
>>> d1["ExtMarks"]-----(67, 77, 78)
>>> d1["ExtMarks"]=tuple(sorted(d1["ExtMarks"])[::-1])
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': (17, 19, 16), 'ExtMarks': (78, 77, 67), 'cname': 'CUCET'} <class 'dict'>
=====
```

Case4: list in dict--Possible

```

>>> d1={"sno":10,"sname":"RS","IntMarks":[17,19,16],"ExtMarks": [67,77,78],"cname":"CUCET"}
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': [17, 19, 16], 'ExtMarks': [67, 77, 78], 'cname': 'CUCET'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"--->",v,"--->",type(v),type(d1))
...
sno ---> 10 ---> <class 'int'> <class 'dict'>
sname ---> RS ---> <class 'str'> <class 'dict'>
IntMarks ---> [17, 19, 16] ---> <class 'list'> <class 'dict'>
ExtMarks ---> [67, 77, 78] ---> <class 'list'> <class 'dict'>
cname ---> CUCET ---> <class 'str'> <class 'dict'>
>>> d1["IntMarks"].insert(1,15)
>>> d1["ExtMarks"].insert(-1,55)
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': [17, 15, 19, 16], 'ExtMarks': [67, 77, 55, 78], 'cname': 'CUCET'} <class 'dict'>
>>> d1["IntMarks"].sort()
>>> d1["ExtMarks"].sort(reverse=True)
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, 17, 19], 'ExtMarks': [78, 77, 67, 55], 'cname': 'CUCET'} <class 'dict'>
>>> d1["ExtMarks"].insert(-1,[1.2,2.3])
>>> print(d1,type(d1))--{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, 17, 19], 'ExtMarks': [78, 77, 67, [1.2, 2.3], 55], 'cname': 'CUCET'} <class 'dict'>
>>> d1["IntMarks"].insert(2,(1.2,2.3))
>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, (1.2, 2.3), 17, 19], 'ExtMarks': [78, 77, 67, [1.2, 2.3], 55], 'cname': 'CUCET'} <class 'dict'>
=====
```

Case5: dict in set-----Not Possible

```

=====
```

```
>>> s1={10,"RS",{"C":15,"CPP":14,"OS":17}, "OUCET"}-----TypeError: unhashable type: 'dict'
```

Case6: dict in tuple--Possible

```

=====
```

```
>>> tpl=(10,"RS",{"C":15,"CPP":14,"OS":17}, "OUCET")
>>> print(tpl,type(tpl))----(10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17}, 'OUCET') <class 'tuple'>
>>> for val in tpl:
...     print(val,"--->",type(val),type(tpl))
```

```

...
    10 ---> <class 'int'> <class 'tuple'>
    RS ---> <class 'str'> <class 'tuple'>
    {'C': 15, 'CPP': 14, 'OS': 17} ---> <class 'dict'> <class 'tuple'>
    OUCET ---> <class 'str'> <class 'tuple'>
>>> tpl[2]["DBMS"]=16
>>> print(tpl,type(tpl))----(10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17, 'DBMS': 16}, 'OUCET') <class 'tuple'>
>>> for k,v in tpl[2].items():
...     print(k,"-->",v)
...
    C --> 15
    CPP --> 14
    OS --> 17
    DBMS --> 16
>>> del tpl[2]-----TypeError: 'tuple' object doesn't support item deletion
>>> tpl[2].pop("OS")----17
>>> print(tpl,type(tpl))----(10, 'RS', {'C': 15, 'CPP': 14, 'DBMS': 16}, 'OUCET') <class 'tuple'>
=====
```

Case7: dict in list---Possible

```

>>> lst=[10,"RS", {"C":15,"CPP":14,"OS":17}, "OUCET"]
>>> print(lst,type(lst))----[10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17}, 'OUCET'] <class 'list'>
>>> for val in lst:
...     print(val,"--->",type(val),type(lst))
...
    10 ---> <class 'int'> <class 'list'>
    RS ---> <class 'str'> <class 'list'>
    {'C': 15, 'CPP': 14, 'OS': 17} ---> <class 'dict'> <class 'list'>
    OUCET ---> <class 'str'> <class 'list'>
>>> lst.pop(2)----{'C': 15, 'CPP': 14, 'OS': 17}
>>> print(lst,type(lst))----[10, 'RS', 'OUCET'] <class 'list'>
>>> lst.insert(-1,{"C":15,"CPP":14})
>>> print(lst,type(lst))----[10, 'RS', {'C': 15, 'CPP': 14}, 'OUCET'] <class 'list'>
>>> lst[2]["OS"]=15
>>> print(lst,type(lst))----[10, 'RS', {'C': 15, 'CPP': 14, 'OS': 15}, 'OUCET'] <class 'list'>
=====
```

NoneType Category Data Type

```

=>" NoneType" is one of the pre-defined class
=>None is the Keyword and Treated as Value of NoneType Data Type
=>None is not False, Space, 0 and None is Nothing
=>We can't create an object of NoneType bcoz It contains Single Value
=====
```

Examples

```

>>> a=None
>>> print(a,type(a))----None <class 'NoneType'>
>>> None==0-----False
>>> None=="-----False
>>> None==False-----False
>>> None==None-----True
>>> n=NoneType()-----NameError: name 'NoneType' is not defined
=====
```

DAY 31 20/04/2024 (SATURDAY)

Definition of Program

=>Set of Optimized Instructions is called Program.

=>Optimized Instructions of Program takes Less Memory Space(Space Complexity) and Less Execution(Time Complexity).

Need of Writing the Program

=>The Need of writing the Program is that To Implement Certain Task OR To develop any Real Time Applications

=>Programmatically, In Python, we define OR develop set of Optimized Instructions and Saved on Some Name with an extension called .py (Called File Name--->FileName.py)

=>In Python Programming, 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 Interactive Approach, Python Programmer can Issue One Instruction at a time and gets an output at a time.

=> Interactive Approach is useful to test One Instruction at a time But Not Useful to develop Big Applications/Programs

bcoz This Environment never allows us to Save the Instructions and More Over Never allows us to re-use those Instructions in other part of Program.

=>In Order to write Programs in real Time, we always go for Batch Mode Approach.

Example Software:

- a. Python Command
 - b. Python IDLE Shell
-

2. By Using Batch Mode Approach

=>In Batch Mode Approach, we define set of Optimized Instructions and Saved on Some File Name with an extension called .py (FileName.py).

=>To develop any real time application, we always go for Batch Mode.

Examples Softwares:

- a. Python IDLE Shell
-

By using IDEs(Integrated Development Environment)

PyCharm
Jupiter Note Book
Spider
VS Code
Google Clab.....etc

=>To Run the python Program from Windows Command Prompt, we use the Following Syntax

=>Syntax: python FileName.py

OR

py FileName.py

Here "python" and "py" are tools used for Running the python Program from Windows Command Prompt.

Examples:

```
E:\KVR-PYTHON-4PM\BATCH MODE>python SumOpEx3.py
```

```
Enter First Value:3
```

```
Enter Second Value:6.7
```

```
=====
```

```
Val of a= 3.0
```

```
Val of b= 6.7
```

```
Sum= 9.7
```

```
=====
```

OR

```
E:\KVR-PYTHON-4PM\BATCH MODE>py SumOpEx3.py
```

```
Enter First Value:3
```

```
Enter Second Value:6.7
```

```
=====
```

```
Val of a= 3.0
```

```
Val of b= 6.7
```

```
Sum= 9.7
```

```
=====
```

#Program for adding of Two Numbers

```
a=10
```

```
b=20
```

```
c=a+b
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

```
=====
```

#Program for adding two numbers

```
a=10
```

```
b=20
```

```
c=a+b
```

```
print("-----")
```

```
print("Val of a=",a)
```

```
print("Val of b=",b)
```

```
print("Sum=",c)
```

```
print("-----")
```

DAY-32 22/04/2024 (MONDAY)

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 other words, print() is a pre-defined Function used for Display the Result of Python Program on the Console.

=>print() can be used with the following Syntaxes

```
*****
```

Syntax-1:

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

```
*****
```

=>This Syntax displays either Single Value or Multiple Values.

```
=====
```

Examples

```
>>> a=10
```

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

```
>>> a=10
```

```
>>> b=20
```

```
>>> c=a+b
```

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

Syntax-2: print(Msg1)

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

=>Here Msg1,Msg2,...,Msg-n are called Messages of type <class,'str'>

=>Here This Syntax display String Messages on the Console

Examples

```
>>> print("Hello Python")-----Hello Python
```

```
>>> print('Hello Python')-----Hello Python
```

```
>>> print("Hello Python")-----Hello Python
```

```
>>> print("""Hello Python""")-----Hello Python
```

```
>>> print("Hello","Python")----Hello Python
```

```
>>> print('Hello','Python')----Hello Python
```

```
>>> print("hello"+ "python")-----hellogpython
```

```
>>> print("Hello" + " "+ "Python")-----Hello Python
```

```
>>> print("Python"+3.12)-----TypeError: can only concatenate str (not "float") to str
```

```
>>> print("Python"+str(3.12))-----Python3.12
```

```
>>> print("Python"+"3.12")-----Python3.12
```

```
>>> print("Python"+str(2+3))-----Python5
```

```
>>> print("Python"+str("2+3"))----Python2+3
```

Syntax-3: print(Message Cum Value)

```
    print(Value Cum Message)
```

=>This Syntax displays Message Cum Values OR Values Cum Messages.

Examples

```
>>> print("Val of a=",a)-----Val of a= 10
```

```
>>> print(a,"is the val of a")----10 is the val of a
```

```
>>> a=10
```

```
>>> print("Val of a="+str(a))----Val of a=10
```

```
>>> print(str(a)+" is the val of a")---10 is the val of a
```

```
>>> a=10
```

```
>>> b=20
```

```
>>> c=a+b
```

```
>>> print("sum=",c)-----sum= 30
```

```
>>> print("sum="+str(c))----sum=30
```

```
>>> print(c," is the sum")---30 is the sum
```

```
>>> print(str(c)+" is the sum")---30 is the sum
```

```
>>> print("Sum of",a," and ",b,"=",c)---Sum of 10 and 20 = 30
```

```
>>> print("sum of "+str(a)+" and "+str(b)+"="+str(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
```

```
>>> print("Sum of "+str(a)+" , "+str(b)+" and "+str(c)+"="+str(d))---Sum of 10,20 and 30=60
```

Syntax-4: print(Message Cum Values with format())

```
*****
```

=>This Function display the Messages cum values with format()

Examples

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

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

```
-----  
>>> a=10
```

```
>>> b=20
```

```
>>> c=a+b
```

```
>>> print("Sum of {} and {}={}".format(a,b,c))---Sum of 10 and 20=30
```

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

```
-----  
>>> print("sum({},{})={}".format(a,b,c))-----sum(10,20)=30
```

```
>>> print("{}+{}={}".format(a,b,c))-----10+20=30
```

```
-----  
>>> sno=10
```

```
>>> sname="Rossum"
```

```
>>> print("My Number is {} and Name is {}".format(sno,sname))---My Number is 10 and Name is Rossum
```

```
-----  
>>> sno=10
```

```
>>> sname="Rossum"
```

```
>>> print("My Number is {} and Name is '{}' ".format(sno,sname))---My Number is 10 and Name is 'Rossum'
```

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

=>In Python Programming, we have the following Format Specifiers

=>Here %d Represents int Data

=>Here %f Represents float Data

=>Here %s Represents str Data

Examples

```
>>> a=10
```

```
>>> b=2.3
```

```
>>> c=a+b
```

```
>>> print("Sum of %d and %f =%f" %(a,b,c))-----Sum of 10 and 2.300000 =12.300000
```

```
>>> print("Sum of %d and %0.2f =%0.2f" %(a,b,c))---Sum of 10 and 2.30 =12.30
```

```
-----  
>>> sno=10
```

```
>>> sname="Rs"
```

```
>>> marks=975.67
```

```
>>> print("My Number is %d and Name is '%s' and Marks is %f" %(sno,sname,marks))
```

My Number is 10 and Name is 'Rs' and

Marks is 975.670000

```
>>> print("My Number is %d and Name is '%s' and Marks is %0.2f" %(sno,sname,marks))
```

My Number is 10 and Name is 'Rs' and

Marks is 975.67

```
-----  
>>> print("Content of lst=%s" %str(lst))-----Content of lst=[100, 'RS', 23.45]
```

```
*****  
Syntax-6: print(value,end=" ")
```

=>This syntax display the the result of Python program in same line

Examples

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

```
...         print(val,end=" ")----10 11 12 13 14 15 16 17 18 19 20
```

```
>>> for val in range(10,21):
...     print(val,end="-->")--- 10-->11-->12-->13-->14-->15-->16-->17-->18-->19-->20-->
*****
```

DAY 33 23/04/2024 (TUESDAY)

Reading the Data OR Input Dynamically From Key Board

=>To Read the Data from KeyBoard, we have 2 Pre-Defined Functions. They are

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

1. `input()`

=>Syntax: `varname=input()`

=>here `input()` is used for Reading Any type of Value from Key Board and Stored in LHS Variable in the form of str.

=>Here str type data can be converted into any type of Data based Type Conversion Functions.

2. `input(Message)`

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

=>Here Message Represents User-Prompting Message and It is str type.

=>here `input(Message)` is used for Reading Any type of Value from Key Board and Stored in LHS Variable in the form of str and Additionally This function also Gives User-Prompting Message.

=>Here str type data can be converted into any type of Data based Type Conversion Functions.

```
#DataReadEx1.py
```

```
print("Enter Any Value") # User-Prompting Message
```

```
x=input()
```

```
print(x,type(x))
```

```
#DataRead2.py
```

```
x=input("Enter Any Value:")
```

```
print(x,type(x))
```

```
#Program for Multiplying Two Numbers
```

```
#DataRead3.py
```

```
print("Enter First Value:")
```

```
x=input()
```

```
print("Enter Second Value:")
```

```
y=input()
```

```
#Convert str int into float or int type
```

```
a=float(x)
```

```
b=float(y)
```

```
c=a*b
```

```
print("First Value={}".format(a))
```

```
print("Second Value={}".format(b))
```

```
print("Mul({},{})={}".format(a,b,c))
```

```
#Program for Multiplying Two Numbers
```

```
#DataRead4.py
```

```
print("Enter Two Values:")
```

```
#Reading the Data from KBD and Convert
```

```
a=float(input())
```

```
b=float(input())
```

```
c=a*b
```

```
print("First Value={}".format(a))
```

```
print("Second Value={}".format(b))
```

```
print("Mul({},{}]={})".format(a,b,c))
```

```
#Program for Multiplying Two Numbers  
#DataRead4.py  
print("Enter Two Values:")  
#Reading the Data from KBD and Convert  
a=float(input())  
b=float(input())  
print("First Value={}".format(a))  
print("Second Value={}".format(b))  
print("Mul({},{}]={})".format(a,b,a*b))
```

```
#Program for Multiplying Two Numbers  
#DataRead6.py  
a=input("Enter First Value:")  
b=input("Enter Second Value:")  
x=float(a)  
y=float(b)  
z=x*y  
print("Mul({},{}]={})".format(x,y,z))
```

```
#Program for Multiplying Two Numbers  
#DataRead7.py  
print("Mul={}".format(float(input("Enter First Value:"))*float(input("Enter Second Value:"))))
```

```
#Program for Multiplying Two Numbers  
#DataRead8.py  
x=float(input("Enter First Value:"))  
y=float(input("Enter Second Value:"))  
z=x*y  
print("Mul({},{}]={})".format(x,y,z))  
print("=====OR=====")  
print("Mul({},{}]={})".format(x,y,x*y))
```

```
#Program for Calculating Area of Circle  
r=float(input("Enter Radius:"))  
area=3.14*r*r  
print("Radius={}".format(r))  
print("Area of Circle={}".format(area))  
print("-----OR-----")  
print("Area of Circle=%0.3f" %area)  
print("-----OR-----")  
print("Area of Circle={}".format(round(area,3)))
```

```
#Program for Calculating Area and Circumference of Rect  
length=float(input("Enter Length:"))  
breadth=float(input("Enter Breadth:"))  
#Cal Area  
ar=length*breadth  
#cal Circumference  
cr=2*(length+breadth)  
print("=*50")  
print("Length={}".format(length))  
print("Breadth={}".format(breadth))  
print("Area of Rect={}".format(ar))  
print("Circumference of Rect={}".format(cr))  
print("****50")
```

Operators and Expressions in Python

- =>An Operator is a Symbol which will perform an Operation on Data OR Values OR Objects.
=>If Two or More Objects OR Variables OR Values Connected with an Operator then It is called Expression.
=>In Other words, An Expression is a Collection of Objects OR Variables OR Values Connected with Operator(s).
=>In Python Programming, we have 7 Types of Operators. They are

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

NOTE-1: Python Programming does not Support Unary Operators (++) and (- -)

NOTE-2: Python Programming does not Support Ternary Operator of C,C++,Java (? :)

NOTE-3: Python Programming Supports Short Hand Operators

NOTE-4: Python Programming is Having Its Own Ternary Operator (if..else Operator)

1. Arithmetic Operators

- =>The purpose of Arithmetic Operators is that "To perform Arithmetic Operations such as Addition, Subtraction , Multiplication ..etc".
=>If two or More Objects or Values Connected with Arithmetic Operators then It is Called Arithmetic Expression.
=>In Python Programming, we have 7 Types of Arithmetic Operators. They are given in the following Table.
-
-

SLNO	SYMBOL	MEANING	EXAMPLE : a=10 b=3
1.	+	Addition	print(a+b)----->13 here a+b is called Arithmetic Expression
2.	-	Subtraction	print(a-b)----->7
3.	*	Multiplication	print(a*b)----->30
4.	/	Division >3.333333333333	print(a/b)-----
5.	//	Floor Division	print(a//b)----->3

(Integer Quotient)

6.	%	Modulo Division (Remainder)	print(a%b)---->1
7.	**	Exponentiation	print(a**b)---->1000.0

(Power of Operator)

NOTE:

```
>>> 10/3-----3.333333333333335
>>>10//3-----3
-----
>>> 10.0/3.0-----3.333333333333335
>>> 10.0//3.0-----3.0
-----
>>> 10.0/3-----3.333333333333335>>> 10.0//3-----3.0
-----
>>> 10/3.0-----3.333333333333335
>>> 10//3.0-----3.0
```

=====

```
#ArithmeticOperatorEx1.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("****50)
print("\t\tResults of Arithmetic Operators")
print("****50)
print("\t\tSum({},{})={}".format(a,b,a+b))
print("\t\tSub({},{})={}".format(a,b,a-b))
print("\t\tMul({},{})={}".format(a,b,a*b))
print("\t\tDiv({},{})={}".format(a,b,a/b))
print("\t\tFloor Div({},{})={}".format(a,b,a//b))

print("\t\tModDiv({},{})={}".format(a,b,a%b))
print("\t\tpower({},{})={}".format(a,b,a**b))
print("****50)
#Here a+b , a-b, a*b, a/b , a//b, a%b , a**b are called Arithmetic Expressions.
```

=====

```
#Program for Demonstrating Square Root of given Number without using sqrt() of math module
#ArithmeticOperatorEx2.py
```

```
n=float(input("Enter a Number for cal Square Root of a Number:"))
res=n**0.5 # OR res=n**(1/2)
print("sqrt({})={}".format(n,res))
```

=====

```
#Program for Demonstrating Cube Root of given Number without using sqrt() of math module
#ArithmeticOperatorEx3.py
```

```
n=float(input("Enter a Number for cal Square Root of a Number:"))
res=n**(1/3)
print("cube({})={}".format(n,round(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
```

X

3. Relational Operators

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

=>If Two Objects 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 called Test Condition(Simple Test Condition).

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

SLNO	SYMBOL	MEANING	EXAMPLE
1.	>	Greater Than	print(10>2)-----True print(10>40)----False
2.	<	Less Than	print(10<5)-----False

		print(10<20)----True
3.	==	Equality (Double Equal to)
4.	!=	Not Equal to
5.	>=	Greater than or equal to
6.	<=	Less than or equal to

Note-1: Complex Data Type Values Can't be Compared with Relational Operators bcoz complex number is the composition of Numerical values with special symbol 'j'

Note-2:

```
>>> "india"=="INDIA"-----False
>>> "abc">>"acb"-----False
>>> "abc">>="acb"-----False
>>> "acb">>="abc"-----True
>>> "india">>="indai"-----True
>>> "just"><="jsut"-----False
>>> "this">>="thsi"-----False
>>> "wrong"><="wrogn"-----False
```

NOTE: To find Unicode value of any Symbol or Alphabet, we use ord()

Syntax: ord("Symbol / Alphabet")

```
>>> ord('p')-----112
>>> ord('P')-----80
>>> ord('a')-----97
>>> ord('A')-----65
>>> ord("#")-----35
>>> ord("$")-----36
>>> "#">"$"-----False
```

NOTE: chr() is used for Obtaining Char Value for Numerical Integer Values

Syntax: chr(Integer Value)

Examples

```
>>> chr(112)-----'p'
>>> chr(65)-----'A'
>>> chr(97)-----'a'
>>> chr(36)-----'$'
>>> chr(35)-----'#'
```

```
>>> for val in range(65,91):
...     print(val,"-->",chr(val))
```

```
...
65 --> A
66 --> B
67 --> C
68 --> D
69 --> E
70 --> F
71 --> G
72 --> H
```

```
73 --> I
74 --> J
75 --> K
76 --> L
77 --> M
78 --> N
79 --> O
80 --> P
81 --> Q
82 --> R
83 --> S
84 --> T
85 --> U
86 --> V
87 --> W
88 --> X
89 --> Y
90 --> Z
>>>
>>>
>>> for val in range(97,123):
... print(val,"-->",chr(val))
...
97 --> a
98 --> b
99 --> c
100 --> d
101 --> e
102 --> f
103 --> g
104 --> h
105 --> i
106 --> j
107 --> k
108 --> l
109 --> m
110 --> n
111 --> o
112 --> p
113 --> q
114 --> r
115 --> s
116 --> t
117 --> u
118 --> v
119 --> w
120 --> x
121 --> y
122 --> z
```

#Program for Swapping / interchanging of Two Values #AssignmentOpEx1.py--Logic-1

```
a=input("Enter Value of a:")
b=input("Enter Value of b:")
print("****50)
print("\tOriginal Value of a:{}".format(a))
print("\tOriginal Value of b:{}".format(b))
print("****50)
#Swapping Logic
a,b=b,a # OR b,a=a,b
print("\tSwapped Value of a:{}".format(a))
```

```
print("\tSwapped Value of b:{} ".format(b))
print("****50)
=====
```

#Program for Swapping / interchanging of Two Values #AssignmentOpEx2.py--Logic-2

```
a,b=input("Enter Value of a:"),input("Enter Value of b:")
print("****50)
print("\tOriginal Value of a:{} ".format(a))
print("\tOriginal Value of b:{} ".format(b))
print("****50)
#Swapping Logic
x=a # Here x is called Temp Variable
a=b
b=x
print("\tSwapped Value of a:{} ".format(a))
print("\tSwapped Value of b:{} ".format(b))
print("****50)
```

#Program for Swapping / interchanging of Two Integer Values #AssignmentOpEx3.py--Logic-3

```
a,b=int(input("Enter Integer Value for a:")),int(input("Enter Integer Value for b:"))
print("****50)
print("\tOriginal Value of a:{} ".format(a))
print("\tOriginal Value of b:{} ".format(b))
print("****50)
#Swapping Logic
a=a+b
b=a-b
a=a-b
print("\tSwapped Value of a:{} ".format(a))
print("\tSwapped Value of b:{} ".format(b))
print("****50)
```

#Program for Swapping / interchanging of Two Integer Values

#AssignmentOpEx4.py--Logic-4

```
a,b=int(input("Enter Integer Value for a:")),int(input("Enter Integer Value for b:"))
print("****50)
print("\tOriginal Value of a:{} ".format(a))
print("\tOriginal Value of b:{} ".format(b))
print("****50)
#Swapping Logic
a=a*b
b=a//b
a=a//b
print("\tSwapped Value of a:{} ".format(a))
print("\tSwapped Value of b:{} ".format(b))
print("****50)
```

=====

DAY- 36 26/04/2024 (FRIDAY)

=====

4. Logical 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 It is Called Logical Expression.
- =>The Result of Logical Expression is either True or False.
- =>The Logical Expression is called Compund Test Condition (Multiple Relational Expressions)
- =>In Python Programming, we have 3 Types of Logical Operators. They are Given in the following Table

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

1. and Operators

=>The functionality of "and" Operator is described in the following Truth table

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

Example1

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

Example2

```
>>> 10>2 and 10>30-----False----Full Length Evaluation
>>> 10>20 and 20>10-----False----Short Circuit Evaluation
>>> 10>20 and 20>30 and 10>4----False----Short Circuit Evaluation
>>> 10>2 and 30>20 and 10>2-----True----Full Length Evaluation
```

Short Circuit Evaluation of "and" Operator

If Two or More Relational Expressions Connected with "and" Operator and If the Result of First Relational Expression is False then PVM will not evaluate Second and Sub-Sequent Relational Expressions and Total Result of Entire Expressions (Logical Expression) is False. This Process of Evaluation is called Short Circuit Evaluation.

2. or Operator

=>The functionality of "or" Operator is described in the following Truth table

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

Example-1

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

Example-2

```
>>> 10>2 or 20>3-----True-----Short Circuit Evaluation  
>>> 10>20 or 20>30-----False-----Full Length Evaluation  
>>> 10>2 or 20>10 or 20>30----True---Short Circuit Evaluation  
>>> 100>20 or 200>300 or 400>500----True---Short Circuit Evaluation
```

Short Circuit Evaluation of "or" Operator

=>If Two or More Relational Expressions Connected with "or" Operator and If the Result of First Relational Expression is True then PVM will not evaluate Second and Sub-Sequent Relational Expressions and Total Result of Entire Expressions (Logical Expression) is True. This Process of Evaluation is called Short Circuit Evaluation.

3. not Operator

=>The functionality of "not" Operator is described in the following Truth table

RelExpr1	not RelExpr1
False	True
True	False

Example-1

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

Example-2--Special Points of "not" Operator

```
>>> not 123-----False  
>>> not -345-----False  
>>> not 0-----True  
>>> not "Python"----False  
>>> not "not"-----False  
>>> not "false"-----False  
>>> not ""-----True  
>>> not "10-10"----False  
>>> not (10-10)----True
```

Special Points on "and" Operator

```
>>> 100 and 200-----200  
>>> 100 and 0-----0  
>>> -24 and -34-----34  
>>> 100 and 10 and 1---- 1  
>>> 100 and 0 and 200---- 0  
>>> 100 and True and -25-----25  
>>> "Python" and "java" and "HTML"-----'HTML'  
>>> True and True and "Python"-----'Python'
```

Special Points on "or" Operator

```
>>> 100 or 200-----100  
>>> 100 or 0-----100  
>>> 0 or 100-----100  
>>> 0 or True or 200----True  
>>> 100 or 200 or 0-----100  
>>> "Python" or "Java" or "C" or "HTML"-----'Python'
```

Special Points on "and" and 'or' Operators

```

>>> 100 or 200 and 300 or 400-----100
>>> 10 and 20 or 40 and 50 or 40 and 80----20
>>> "java" and "Python" and "HTML" or "Django" and 20---'HTML'
>>> 0 and 30 or -123 and 0 and "Python" or True----True
=====
```

#RelationalOperatorsEx.py

```

a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
print("****50)
print("Result of Relational Expression")
print("****50)
print("\t{} > {}= {}".format(a,b,a>b))
print("\t{} < {}= {}".format(a,b,a<b))
print("\t{}=={} = {}".format(a,b,a==b))
print("\t{}!={} = {}".format(a,b,a!=b))
print("\t{} >= {}= {}".format(a,b,a>=b))
print("\t{} <= {}= {}".format(a,b,a<=b))
print("****50)
```

DAY- 37 27/04/2024 (SATURDAY)

5. Bitwise Operators-----Most Imp

- =>The Bitwise Operators are used performing Operations on Integer Data in the form of Bit by Bit.
- =>Bitwise Operators are Applicable on Integer Values only But Not applicable on Floating Point Values bcoz Floating Point Values does not contain Certainty where as Integer Data Contains Certainty.
- =>The Execution Process of Bitwise Operators is shown bellow
 - a) Bitwise Operators Converts the Integer Data into Binary Bits
 - b) Bitwise Operators applied on Binary Bits and Peforms Operation depends on Type of Bitwise Operator.
 - c) Finally the Result of Bitwise Operators displays the Python Execution Environment in the form Decimal Number System.
- =>Since Bitwise Operators performs the Operations on Binary Bits 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 Given in the following Table.

SLNO	SYMBOL	MEANING
1.	<<	Bitwise Left Shift Operator
2.	>>	Bitwise Right Shift Operator
3.	&	Bitwise AND Operator
4.		Bitwise OR Operator
5.	~	Bitwise Complement Operator (Tilde)
6.	^	Bitwise XOR Operator

1. Bitwise Left Shift Operator (<<)

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

Explanation:

The Execution Process of Bitwise LeftShift Operator (<<) is that "It Moves Number of Bits Towards Left Side By Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at

Right

Bitwise Left Shift Operator (<<)

varname= Given Number << No. of Bits

Examples:

>>>a=10



>>>b = a<<3



>>>print(b)----80



b-----> 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0

Result is 80

Formula for Bitwise Left shift operator

varname= Given Number << No. of Bits

varname= Given Number $\times 2^{\text{No. of Bits}}$

>>>print(2<<3)----16

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

>>>print(4<<2)----16

Side.

Examples

>>>a=10

>>>b=a<<3

>>>print(b)-----80

>>>print(4<<3)-----32

>>>print(9<<2)-----36

>>>print(10<<0)-----10

>>>print(10.3<<2)-----TypeError: unsupported operand type(s) for <<: 'float' and 'int'

2. Bitwise RightShift Operator (>>)

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

Explanation:

The Execution Process of Bitwise Right Shift Operator (>>) is that "It Moves Number of Bits Towards Right Side By Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at Left Side

2. Bitwise Right Operator (>>)

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

Examples:

>>>a=10-----> 0 0 0 0 0 0 0 0 0 1 0 1 0

a=10 -16-Bit Register

>>>b=a>>3-----> 0 0 0 0 0 0 0 0 0 1 0 1 0

Flipped-off

a=10 -16-Bit Register

b-----> 0 0 0 0 0 0 0 0 0 0 0 0 1

Result=1

>>>print(b)----> 1

Formula

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

result = $\frac{\text{Given Data}}{2^{\text{No. of Bits}}}$

In Python---> Given Data // 2^{No. of Bits}

Examples:

>>>a=10

>>>b=a>>3

>>>print(b)-----1

>>>print(16>>2)---4

>>>print(32>>3)---4

>>>print(32>>2)---8

>>>print(32>>0)---32

Examples

>>>a=10

>>>b=a>>3

>>>print(b)-----1

>>>print(16>>2)---4

>>>print(32>>3)---4

```
>>> print(32>>2)---8  
>>> print(32>>0)---32  
>>> print(80.5<<4)-----TypeError: unsupported operand type(s) for <<: 'float' and 'int'
```

Bitwise AND Operator (&)

=> Syntax: Value1 & Value2

=> The Functionality of Bitwise AND Operator (&) is described with following truth table.

Value1	Value2	Value1 & Value2
0	1	0
1	0	0
0	0	0
1	1	1

Example-1

```
>>> 0 & 1-----0  
>>> 1 & 0-----0  
>>> 0 & 0-----0  
>>> 1 & 1-----1
```

Example-2

```
>>>a=10----- 0000 1010  
>>>b=4----- 0000 0100  
  
>>>c=a&b----- 0000 0000  
  
>>>print(c)-----0  
>>> 10 and 4-----4  
>>> 4 and 10-----10  
>>> True and True-----True  
>>> True & True-----True  
>>> 10.2 & 2-----TypeError: unsupported operand type(s) for &: 'float' and 'int'  
>>> 10.2 and 2-----2  
>>> "Apple" and "mango"-----'mango'  
>>> "Apple" & "mango"-----TypeError: unsupported operand type(s) for &: 'str' and 'str'  
>>> print(5&4)-----4  
>>> print(4&5)-----4  
>>> 5 and 4-----4  
>>> 4 and 5-----5
```

Example-3

```
>>> s1={10,20,30}  
>>> s2={15,20,35}  
>>> s3=s1.intersection(s2)  
>>> print(s3,type(s3))-----{20} <class 'set'>  
-----OR-----  
>>> s1={10,20,30}  
>>> s2={15,20,35}  
>>> s3=s1&s2 # Bitwise & (AND) Operator  
>>> print(s3,type(s3))-----{20} <class 'set'>  
>>> s1 and s2-----{35, 20, 15}
```

```
>>> s1={"Apple","mango","Kiwi"}  
>>> s2={"Guava","Orango","mango"}  
>>> s3=s1.intersection(s2)
```

```

>>> print(s3,type(s3))-----{'mango'} <class 'set'>
-----
>>> s1={"Apple","mango","Kiwi"}
>>> s2={"Guava","Orango","mango"}
>>> s3=s1&s2
>>> print(s3,type(s3))-----{'mango'} <class 'set'>
-----
>>> s1={1.2,2.3,4.5}
>>> s2={2.3,3.3,4.4}
>>> s3=s1.intersection(s2)
>>> print(s3,type(s3))-----{2.3} <class 'set'>
>>> s1={1.2,2.3,4.5}
>>> s2={2.3,3.3,4.4}
>>> s3=s1&s2---
>>> print(s3,type(s3))-----{2.3} <class 'set'>
>>> 2.5&3.4-----TypeError: unsupported operand type(s) for &: 'float' and 'float'
-----
>>> lst=[10,20,30,40]
>>> tpl=(10,15,25,56)
>>> lst&tpl-----TypeError: unsupported operand type(s) for &: 'list' and 'tuple'
=====
```

DAY-38 29/04/2024 (MONDAY)

Bitwise OR Operator (|)

=>Syntax:Value1 | Value2
=>The Functionality of Bitwise OR Operator (|) is described with following truth table.

	Value1	Value2	Value1 Value2
	0	1	1
	1	0	1
	0	0	0
	1	1	1

Example-1

```

>>> 0 | 1-----1
>>> 1 | 0-----1
>>> 0 | 0-----0
>>> 1 | 1-----1
=====
```

Example-2

```

a=10----->0000 1010
b=4----->0000 0100
=====
c=a|b----->0000 1110
print(c)---->14
=====
```

```

>>> a=10
>>> b=10
>>> print(a|b)----10
=====
```

Example-3

```

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

```

>>> s2={15,10,25}
>>> s3=s1.union(s2)
>>> print(s3,type(s3))-----{20, 25, 10, 30, 15} <class 'set'>
-----OR-----
>>> s1={10,20,30}
>>> s2={15,10,25}
>>> s3=s1|s2
>>> print(s3,type(s3))-----{20, 25, 10, 30, 15} <class 'set'>
>>> s1={1.2,2.3,4.5}
>>> s2={1.2,2.3,4.6}
>>> s3=s1|s2
>>> print(s3,type(s3))-----{1.2, 2.3, 4.5, 4.6} <class 'set'>
>>> 1.2|2.3-----TypeError: unsupported operand type(s) for |: 'float' and 'float'
-----
>>> s1={"Python","Django"}
>>> s2={"C","HTML","C++"}
>>> s3=s1|s2
>>> print(s3,type(s3))-----{'C++', 'HTML', 'C', 'Python', 'Django'} <class 'set'>
>>> "Python"|"Java"-----TypeError: unsupported operand type(s) for |: 'str' and 'str'
=====X=====

```

5. Bitwise Complement Operator (~)

=> Syntax: varname= ~Given Number
 => The execution process of Bitwise Complement Operator (~) is that " It Inverts the given bits".
 => Inverting the bits is nothing but 1 becomes 0 and 0 becomes 1
 => The formula for Bitwise Complement Operator (~) is given below

$$\sim \text{Given Number} = -(\text{Given Number} + 1)$$

Example1:

```

>>> a=10
>>> print(~a)----- -11
>>> a=100
>>> print(~a)----- -101
-----
>>> a=-123
>>> print(~a)----- 122
=====X=====

```

Q) Prove that $\sim 10 = -11$

Proof : Given $\sim 10 = -11$
 Here -11 is the Opposite counter part of 11
 => Given Number 10 and Whose Binary Part is 1010
 $\Rightarrow \sim 10 = \sim(1010) = 1101$ (Which is the Binary form of -11 which is the 2's complement of 11)
 NOTE: All the Negative Numbers are 2's Complement of Their +Ve Numbers
 [Example: 10 whose Counter part is -10--which is 2's complement of 10]

\Rightarrow Here -11 is the Opposite counter part of 11 (2's complement of 11)
 \Rightarrow All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

\Rightarrow Here we Take 11 and whose Binary form is 1 0 1 1
 (1's Complement of any Number= 1 becomes 0 and 0 becomes 1)
 \Rightarrow 1's Complement of 11 is-----0 1 0 0
 \Rightarrow 2's Complement of 11 is-----1's Complement of 11 + 1

$\Rightarrow 0100$
 $\Rightarrow 0001$ Binary

Addition Rules (0+0=0, 1+0=1, 0+1=1 , 1+1= 0 with carry 1)

0101---which is 2's

Complement of 11 --result is -11

=====

Q) Prove that ~ 16 is -17

=====

Proof : Given $\sim 16 = -17$

Here -17 is the Opposite counter part of 17

=>Given Number 16 and Whose Binary Part is 0001 0000

=> $\sim 16 = \sim(0001 0000) = 1110 1111$ (Which is the Binary form of -17)

=====

=>Here -17 is the Opposite counter part of 17

=>All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

=====

=>Here we Take 17 and whose Binary form is 0001 0001

(1's Complement of any Number= 1 becomes 0 and 0 becomes 1)

=>1's Complement of 17 is-----1110 1110

=>2's Complement of 17 is-----1's Complement of 17 + 1

$$\Rightarrow \begin{array}{r} 1110 1110 \\ 0000 0001 \end{array} \text{Binary}$$

Addition Rules ((0+0=0, 1+0=1, 0+1=1, 1+1= 0 with carry 1)

$$1110 1111 \quad (\text{Which})$$

is the Binary form of -17

=====

Q) Prove that ~ 15 is -16

=====

Proof : Given $\sim 15 = -16$

Here -16 is the Opposite counter part of 16

=>Given Number 15 and Whose Binary Part is 0000 1111

=> $\sim 15 = \sim(0000 1111) = 1111 0000$ (Which is the Binary form of -16)

=====

=>Here -16 is the Opposite counter part of 16

=>All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

=====

=>Here we Take 16 and whose Binary form is 0001 0000

(1's Complement of any Number= 1 becomes 0 and 0 becomes 1)

=>1's Complement of 16 is-----1110 1111

=>2's Complement of 16 is-----1's Complement of 16 + 1

$$\Rightarrow \begin{array}{r} 1110 1111 \\ 0000 0001 \end{array} \text{Binary}$$

Addition ((0+0=0, 1+0=1, 0+1=1 , 1+1= 0 with carry 1)

$$1111 0000 \quad (\text{Which})$$

is the Binary form of -16

Bitwise XOR Operator (^)

=>Syntax: Varname= Value1 ^ Value2

=>The Functionality of Bitwise XOR Operator (^) is expressed with the following Truth Table

Value1	Value2	Value1^Value2
1	0	1
0	1	1
1	1	0

Examples1

```
>>> 1^0-----1
>>> 0^1-----1
>>> 1^1-----0
>>> 0^0-----0
```

Examples2

```
>>> print(2^3)-----1
>>> print(10^15)----5
```

Special Points

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

```
>>> s1={10,20,30}
>>> s2={15,20,25}
>>> s3=s1^s2 # Bitwise XOR Operator (^)
>>> print(s3,type(s3))-----{10, 15, 25, 30} <class 'set'>
>>> s1={"apple","mango","kiwi"}
>>> s2={"Sberry","mango","guava"}
>>> s3=s1^s2 # Bitwise XOR Operator (^)
>>> print(s3,type(s3))-----{'guava', 'apple', 'kiwi', 'Sberry'} <class 'set'>
>>> {1.2,2.3,3.4}^{1.2,2.3,4.5}-----{3.4, 4.5}
```

```
>>> 1.2^2.3-----TypeError: unsupported operand type(s) for ^: 'float' and 'float'
>>> "apple"^1.2-----TypeError: unsupported operand type(s) for ^: 'str' and 'float'
```

Imp Logic---Swapping of Two Integer values

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

DAY-39 30/04/2024 (TUESDAY)

Membership Operators

=>The purpose of Membership Operators is that "To Check the existence the Specified Value in Iterable Object".

=>An Iterable Object is One, which contains More than One Value--such as sequence type(str,bytes,bytearray,range), list type (list,tuple) , set type(set,frozenset) and dict type.

=>A Non-Iterable Object is One which contains Only One Value such as int,float,bool and complex and None

=>In Python Programming, we have 2 Membership Operators. They are

1. in
 2. not in
-

1) in Operator

Syntax: Value in Iterable-Object
=>Here "in" Operator Returns True provided "Value" Present in Iterable-Object.
=>Here "in" Operator Returns False provided "Value" Not Present in Iterable-Object.

2) not in Operator

Syntax: Value not in Iterable-Object
=>Here "not in" Operator Returns True provided "Value" Not Present in Iterable-Object.
=>Here "not in" Operator Returns False provided "Value" Present in Iterable-Object.

Examples

```
>>> s="PYTHON"
>>> "P" in s-----True
>>> "K" in s-----False
>>> "p" not in s-----True
>>> "P" not in s-----False
-----
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> "PYT" in s-----True
>>> "PYT" not in s-----False
>>> "HON" not in s-----False
>>> "NOH" not in s-----True
>>> "NOH" not in s[::-1]---False
>>> "NOH" in s[::-1]-----True
>>> "PTO" in s-----False
>>> "PTO" in s[::-2]-----True
-----
>>> lst=[10,"Rossum",23.45,2+3j]
>>> print(lst)-----[10, 'Rossum', 23.45, (2+3j)]
>>> 10 in lst-----True
>>> "Rossum" in lst-----True
>>> 23.45 not in lst-----False
>>> "Ros" in lst-----False
>>> "Ros" in lst[1]-----True
>>> "Rsu" in lst[1][::-2]-----True
>>> "Rsu"[::-1] in lst[1][::-2][::-1]---True
-----
>>> lst=[10,"Rossum",23.45,2+3j]
>>> print(lst)-----[10, 'Rossum', 23.45, (2+3j)]
>>> 2+3j in lst[-1]-----TypeError: argument of type 'complex' is not iterable
>>> 2+3j in lst[-1].real-----TypeError: argument of type 'float' is not iterable
-----
>>> lst=[10,"Rossum",23.45,2+3j]
>>> "mus" not in lst[-3]-----True
>>> "mus"[::-1] not in lst[-3]-----False
>>> "mus"[::-1] not in lst[-3][:-1]---True
-----
>>> d={10:"Apple",20:"Mango",30:"Sberry"}
>>> print(d)-----{10: 'Apple', 20: 'Mango', 30: 'Sberry'}
>>> "Apple" not in d-----True
>>> 10 in d-----True
>>> "Apple" in d[10]-----True
>>> "App" in d[10][::-1]---False
>>> "App"[::-1] not in d[10][::-1]---False
>>> "Apple" in d.values()-----True
>>> 10 in d.values()-----False
```

```
>>> 12 in 123-----TypeError: argument of type 'int' is not iterable
>>> 12 in "123"-----TypeError: 'in <string>' requires string as left operand, not int
>>> "12" in "123"-----True
>>> "13" not in "123"-----True
>>> "13" not in "123"[:2]----False
>>> "13" in "123"[:2]-----True
```

7. Identity Operators (Applicable on Python Command Only)

=>The purpose of Identity Operators is that "To compare the Memory Address of Two Objects".
=>In Python Programming, we have 2 Identity Operators. They are

1. is Operator
 2. is not Operator
-

1. is Operator

Syntax: Object1 is Object2

=>Here "is" Operator Returns True Provided the Memory Address of Object1 and Object2 are Same.

=>Here "is" Operator Returns False Provided the Memory Address of Object1 and Object2 are Different.

2. is not Operator

Syntax: Object1 is not Object2

=>Here "is not" Operator Returns True Provided the Memory Address of Object1 and Object2 are Different.

=>Here "is not" Operator Returns False Provided the Memory Address of Object1 and Object2 are Same.

Examples:

NOTE1: All Deep Copy Objects with "is" Operator returns True

NOTE2: All Deep Copy Objects with "is not" Operator returns False

NOTE3: All Shallow Copy Objects with "is" Operator returns False

NOTE4: All Shallow Copy Objects with "is not" Operator returns True

```
>>> lst1=[10,"RS"]
>>> lst2=lst1 # Deep Copy
>>> print(lst1,id(lst1))-----[10, 'RS'] 2153932973440
>>> print(lst2,id(lst2))-----[10, 'RS'] 2153932973440
>>> lst1 is lst2-----True
>>> lst1 is not lst2-----False
```

```
>>> lst1=[10,"RS"]
>>> lst2=lst1.copy() # Shallow Copy
>>> print(lst1,id(lst1))-----[10, 'RS'] 2153931658112
>>> print(lst2,id(lst2))-----[10, 'RS'] 2153931508480
>>> lst1 is lst2-----False
>>> lst1 is not lst2-----True
```

Examples

```
>>> a=None
>>> b=None
>>> print(a,id(a))-----None 140718855683792
>>> print(b,id(b))-----None 140718855683792
>>> a is b-----True
>>> a is not b-----False
```

```
>>> d1={10:"Apple",20:"Mango"}
>>> d2={10:"Apple",20:"Mango"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango'} 2153931585920
>>> print(d2,id(d2))-----{10: 'Apple', 20: 'Mango'} 2153932971072
```

```

>>> d1 is d2-----False
>>> d1 is not d2-----True
-----
>>> s1={10,20,30,40}
>>> s2={10,20,30,40}
>>> print(s1,id(s1))-----{40, 10, 20, 30} 2153931553728
>>> print(s2,id(s2))-----{40, 10, 20, 30} 2153931547232
>>> s1 is s2-----False
>>> s1 is not s2-----True
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s2)
>>> print(fs1,id(fs1))-----frozenset({40, 10, 20, 30}) 2153931553952
>>> print(fs2,id(fs2))-----frozenset({40, 10, 20, 30}) 2153932207744
>>> fs1 is fs2-----False
>>> fs1 is not fs2-----True
-----
>>> t1=(10,20,30,40)
>>> t2=(10,20,30,40)
>>> print(t1,id(t1))-----(10, 20, 30, 40) 2153931710736
>>> print(t2,id(t2))-----(10, 20, 30, 40) 2153931711536
>>> t1 is t2-----False
>>> t1 is not t2-----True
>>> lst1=[10,20,30,1.2]
>>> lst2=[10,20,30,1.2]
>>> print(lst1, id(lst1))-----[10, 20, 30, 1.2] 2153932973440
>>> print(lst2, id(lst2))-----[10, 20, 30, 1.2] 2153931658112
>>> lst1 is lst2-----False
>>> lst1 is not lst2-----True
-----
>>> r1=range(10,20)
>>> r2=range(10,20)
>>> print(r1,id(r1))-----range(10, 20) 2153933009136
>>> print(r2,id(r2))-----range(10, 20) 2153933008032
>>> r1 is r2-----False
>>> r1 is not r2-----True
-----
>>> ba1=bytearray([10,20,30])
>>> ba2=bytearray([10,20,30])
>>> print(ba1,id(ba1))-----bytearray(b'\n\x14\x1e') 2153931656816
>>> print(ba2,id(ba2))-----bytearray(b'\n\x14\x1e') 2153931506096
>>> ba1 is ba2-----False
>>> ba1 is not ba2-----True
>>> ba1=bytes([10,20,30])
>>> ba2=bytes([10,20,30])
>>> print(ba1,id(ba1))-----b'\n\x14\x1e' 2153933008128
>>> print(ba2,id(ba2))-----b'\n\x14\x1e' 2153933009520
>>> ba1 is ba2-----False
>>> ba1 is not ba2-----True
-----
```

If String Objects Contains Data with Same Meaning,Same Case and Same Order then Both String Obejcts contains Same Address otherwise Contains Different Address.

```

>>> s1="Rossum"
>>> s2="Rossum"
>>> print(s1,id(s1))-----Rossum 2153931722208
>>> print(s2,id(s2))-----Rossum 2153931722208
>>> s1 is s2-----True
>>> s1 is not s2-----False
>>> s1="INDIA"
>>> s2="INDIA"
>>> s1 is s2-----True
```

```
>>> s1 is not s2-----False
>>> s1="INDIA"
>>> s2="INDAI"
>>> print(s1,id(s1))-----INDIA 2153933008512
>>> print(s2,id(s2))-----INDAI 2153933009616
>>> s1 is s2-----False
>>> s1 is not s2-----True
-----
>>> a=2+3j
>>> b=2+3j
>>> print(a,id(a))-----(2+3j) 2153931621264
>>> print(b,id(b))-----(2+3j) 2153931635728
>>> a is b-----False
>>> a is not b-----True
-----
>>> a=True
>>> b=True
>>> print(a,id(a))-----True 140718855683728
>>> print(b,id(b))-----True 140718855683728
>>> a is b-----True
>>> a is not b-----False
-----
>>> a=1.2
>>> b=1.2
>>> print(a,id(a))-----1.2 2153931634608
>>> print(b,id(b))-----1.2 2153931635728
>>> a is b-----False
>>> a is not b-----True
-----
>>> a=10
>>> b=10
>>> print(a,id(a))-----10 140718856477400
>>> print(b,id(b))-----10 140718856477400
>>> a is b-----True
>>> a is not b-----False
>>> a=256
>>> b=256
>>> print(a,id(a))-----256 140718856485272
>>> print(b,id(b))-----256 140718856485272
>>> a is b-----True
>>> a is not b-----False
>>> a=0
>>> b=0
>>> print(a,id(a))-----0 140718856477080
>>> print(b,id(b))-----0 140718856477080
>>> a is b-----True
>>> a is not b-----False
-----
>>> a=257
>>> b=257
>>> print(a,id(a))-----257 2153931635952
>>> print(b,id(b))-----257 2153931621040
>>> a is b-----False
>>> a is not b-----True
-----
>>> a=-1
>>> b=-1
>>> print(a,id(a))-----1 140718856477048
>>> print(b,id(b))-----1 140718856477048
>>> a is b-----True
>>> a is not b-----False
```

```
>>> a=-5
>>> b=-5
>>> print(a,id(a))----- -5 140718856476920
>>> print(b,id(b))----- -5 140718856476920
>>> a is b-----True
>>> a is not b-----False
>>> a=-6
>>> b=-6
>>> print(a,id(a))----- -6 2153931621040
>>> print(b,id(b))----- -6 2153931635952
>>> a is b----- False
>>> a is not b----- True
```

Special Point

```
>>> a,b=2000,2000
>>> print(a,id(a))-----2000 2153931635984
>>> print(b,id(b))-----2000 2153931635984
>>> a is b-----True
>>> a is not b-----False
```

```
>>> a,b=1.2,1.2
>>> print(a,id(a))-----1.2 2153931634608
>>> print(b,id(b))-----1.2 2153931634608
>>> a is b-----True
>>> a is not b-----False
```

```
>>> lst1,lst2=[10,"RS"],[10,"RS"]
>>> print(lst1,id(lst1))-----[10, 'RS'] 2153931508480
>>> print(lst2,id(lst2))-----[10, 'RS'] 2153932973568
```

=====

DAY-40 01/05/2024(WEDNESDAY)

=====

Python Ternary Operator

=>The Name of Python Ternary Operator is "if..else " Operator.

Syntax: varname = Expression-1 if Test Condition else Expression-2

=>Here Test Condition can be either Relational OR Logical Expression and whose Result can be either True or False.

=>If Test Condition Result is True then PVM Executes Expression-1 whose Result Placed in LHS Varname.
=>If Test Condition Result is False then PVM goes to else part and Executes Expression-2 whose Result Placed in LHS Varname.

=>Hence in Python Ternary Operator, at any Point of Time PVM Executes either Expression-1 OR Expression-2 and whose Result placed in LHS Varname.

#Program for accepting Three Numerical Values and Find Biggest among them

#BigThreeEx1.py

```
a=int(input("Enter Value of a:")) # a=10
```

```
b=int(input("Enter Value of b:")) # b=10
```

```
c=int(input("Enter Value of c:"))# c=3
```

```
res=a if a>=b and a>c else b if b>a and b>=c else c if c>=a and c>b else "All are Equal"
```

```

print("Big({}, {}, {})={}".format(a,b,c,res))
=====

#Program for accepting Three Numerical Values and Find Biggest among them
#BigThreeEx1.py
a=int(input("Enter Value of a:")) # a=10
b=int(input("Enter Value of b:")) # b=10
c=int(input("Enter Value of c:"))# c=3
res=a if b<=a>c else b if a<b>=c else c if a<=c>b else "All are Equal"
print("Big({}, {}, {})={}".format(a,b,c,res))
=====

#Program for finding Biggest of Two Numbers
#BigTwoEx1.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
#Find Big of a and b
bv = a if a>b else b # Python Ternary Operator
print("Big({}, {})={}".format(a,b,bv))
=====

#Program for finding Biggest of Two Numbers and check for equality
#BigTwoEx2.py
a=int(input("Enter Value of a:")) # 10
b=int(input("Enter Value of b:")) # 10
res=a if a>b else b if b>a else "equal"
print("Big({}, {})={}".format(a,b,res))
=====

#Program for accepting a word / Value and decide weather It is Palindrome or not
#PalindromeEx.py
value=input("Enter a Value:") # LIRIL
res="Palindrome" if value==value[::-1] else "Not Palindrome"
print("{} is {}".format(value,res))
=====

#Program for accepting a word and Decide whether It is Vowel Word or not
#VowelWordEx.py
#Example Apple---Vowel Word try---Not Vowel Word
word=input("Enter Any Word:") # Apple
res="Vowel Word" if 'a' in word or 'e' in word or 'i' in word or 'o' in word or 'u' in word else "Not Vowel Word"
print("{} is {}".format(word,res))
=====
```

DAY-41 02/05/2024(THURSDAY)

Flow Control Statements in Python

(OR)

Control Structures in Python---10 Days

Index

=>Purpose of Flow Control Statements

=>Types of Flow Control Statements

I. Conditional OR Selection OR Branching Statements

1. simple if statement
2. if..else statement
3. if..elif..else statement
4. match case statement

=>Programming Examples

II. Looping OR Iterating OR Repetative Statements

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

=>Programming Examples

III. Transfer Flow Statements

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

=>Programming Examples

=>Combined Programming Examples in Conditional, Looping and Transfer Flow Statements

=>Inner OR Nested Loops

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

=>Programming Examples

Flow Control Statements in Python

=>The purpose of Flow Control Statements in Python is that " To Perform Certain Operation(X-Operation in the case of True OR Y-Operation in the Case of False) Only Once OR To 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.

1. Conditional OR Selection OR Branching Statements
 2. Looping OR Iterating OR Repetative 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 X-Operation in the Case of True OR Y-Operation in the Case of False Only Once."

=>In Python Programming, we have 4 Tuples 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=="yes"):
    print("Enter into theater")
    print("Watch the Moviee")
    print("Eat Pop Corn!!!")
print("Goto Home")
```

#Program for Accepting Two Numerical value and Decide Biggest and Check for equality also
#SimpleIFStmtEx1.py

```
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
if(a>b):
    print("Big({},{})={}".format(a,b,a))
if(b>a):
    print("Big({},{})={}".format(a,b,b))
```

```

if(a==b):
    print("Both the Value are Equal")
print("Program Execution completed!!!")
=====
#Program for Accepting a Number and Decide whether It is +ve or -ve or Zero
#SimpleIFStmtEx2.py
n=float(input("Enter Any 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))
=====

#SimpleIFStmtEx3.py
n=float(input("Enter any Number:"))
if(n<=0):
    print("{} is Invalid Input".format(n))
if(n%2==0) and (n>0):
    print("{} is EVEN".format(n))
if(n%2!=0) and (n>0):
    print("{} is ODD".format(n))
=====
```

DAY-42 03/05/2024 (FRIDAY)

```

#Program for Accepting Two Numerical value and Decide Biggest and Check for equality also
#IfElifElseStmtEx1.py
a=float(input("Enter First Value:")) # 10
b=float(input("Enter Second Value:")) # 3
if(a>b):
    print("Big({}, {})={}".format(a,b,a))

elif(b>a):
    print("Big({}, {})={}".format(a,b,b))
elif(a==b):
    print("Both the Value are Equal")
else:
    print("i am from else Part")
print("Program Execution completed!!!")
=====
```

```

##Program for accepting a Digit and display Digit Name
#IfElifElseStmtEx2.py
d=int(input("Enter a Digit:")) # d=0 1 2 3 4 5 6 7 8 9
if(d==0):
    print("{} is Zero".format(d))
elif(d==1):
    print("{} is One".format(d))
elif(d==2):
    print("{} is Two".format(d))
elif(d==3):
    print("{} is Three".format(d))
elif(d==4):
    print("{} is Four".format(d))
elif(d==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 range(-1,-10,-1)):
    print("{} is -Ve Digit".format(d))
elif(d<0):
    print("{} is -ve Number".format(d))
else:
    print("{} is +Ve Number".format(d))
print("Program Execution Completed")
=====
```

```

#Program for accepting a Digit and display Digit Name
#IfElifElseStmtEx3.py
d=int(input("Enter Digit:"))
dobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}
res=dobj.get(d) if dobj.get(d)!=None else "-Ve Digit" if d in range(-1,-10,-1) else "+Ve Number" if d>0 else "-Ve Number"
print("{} is {}".format(d,res))
=====
```

```

#Program for accepting a Digit and display Digit Name
#IfElifElseStmtE3A.py
d=int(input("Enter Digit:"))
dobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}
res=dobj.get(d) if dobj.get(d)!=None else "-Ve Digit" if d in range(-1,-10,-1) else "+Ve Number" if d>0 else "-Ve Number"
print("{} is {}".format(d,res))
=====
```

```

#Program for Accepting Two Numerical value and Decide Biggest
#IfElseStmtEx1.py
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
if(a>b):
    print("Big({}, {}) = {}".format(a,b,a))
else:
    print("Big({}, {}) = {}".format(a, b, b))
print("Program Execution Completed")
=====
```

```

#Program for Accepting Two Numerical value and Decide Biggest and Check for equality also
#IfElseStmtEx2.py
a=float(input("Enter First Value:"))#a=10
b=float(input("Enter Second Value:"))#b=10
if(a>b): # Here Line No: 5 to 6 Represent outer if..else
    print("Big({}, {}) = {}".format(a,b,a))
else:
    if(b>a): # Here Line No: 8 to 10 Represent Inner if..else
        print("Big({}, {}) = {}".format(a, b, b))
    else:
        print("Both Values are Equal")
    print("other stmts--inner if..else stmt")
print("other stmts--outer if..else stmt")
=====
```

```

#Program for accepting a Digit and display Digit NametEx2.py
#IfElseStmt3.py
```

```

d=int(input("Enter a Digit:")) # d=0 1 2 3 4 5 6 7 8 9
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:
                                        print("{} is not Digit".format(d))
=====

```

```

#Program for accepting a Digit and display Digit Name
#SimpleIFStmtEx3.py
d=int(input("Enter a Digit:")) # d=0 1 2 3 4 5 6 7 8 9
if(d==0):
    print("{} is Zero".format(d))
if(d==1):
    print("{} is One".format(d))
if(d==2):
    print("{} is Two".format(d))
if(d==3):
    print("{} is Three".format(d))
if(d==4):
    print("{} is Four".format(d))
if(d==5):
    print("{} is Five".format(d))
if(d==6):
    print("{} is Six".format(d))
if(d==7):
    print("{} is Seven".format(d))
if(d==8):
    print("{} is Eight".format(d))
if(d==9):
    print("{} is Nine".format(d))
=====
```

match case statement (Python3.10 Version onwards)

=>match case statement is one of conditional statement available from Python3.10 Version onwards.
=>The purpose of match case statement is that "To Deal with Pre-Designed Conditions OR Menu Driven Applications".
=>The Menu Driven Applications contains Pre-Designed 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 Stements-n
    case _:
        # Default Case Block
        default Block of Statements
```

Other Statements in Program

Explanation:

=>here "match" and "case" are the keywords(proposed)
=>"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 to 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 (|) .

Examples

Arithmetic Operations

1. Addition
 2. Substraction
 3. Multiplication
 4. Division
 5. Modulo Division
 6. Exponentiation
 7. Exit
-

Enter Ur Choice:

Area of Different Figures

```
*****
```

- R. Rectangle
 - S. Square
 - C. Circle
 - T. Triangle
 - E. Exit
- ```

```

Enter Ur Choice:

```

```

Home Work---student must do

```

```

## Temprature Conversion Calculator

```

```

- 1. F to C
  - 2. F to K
  - 3. C to F
  - 4. C to K
  - 5. K to F
  - 6. K to C
  - 7. Exit
- ```
*****
```

Enter Ur Choice: 1

```
*****
```

Fahrenheit to Celcius: $C = (F-32) \cdot \frac{5}{9}$

Fahrenheit to Kelvin: $K = (F-32) \cdot \frac{5}{9} + 273.15$

Celsius to Fahrenheit: $F = C \cdot \frac{9}{5} + 32$

Celsius to Kelvin: $K = C + 273.15$

Kelvin to Celcius: $C = K - 273.15$

Kelvin to Fahrenheit: $F = (K-273.15) \cdot \frac{9}{5} + 32$

```
=====
```

```
#MatchCaseEx1.py
```

```
import sys
```

```
s="*****
```

Temprature Conversion Calculator

```
*****
```

- 1. F to C
 - 2. F to K
 - 3. C to F
 - 4. C to K
 - 5. K to F
 - 6. K to C
 - 7. Exit
- ```

```

```
print(s)
```

```
ch=int(input("Enter Ur Choice:"))
```

```
print("*****")
```

```
match(ch):
```

```
 case 1:
```

```
 F=float(input("Enter Temp. Value in Trems of F to get Temp in terms of C:"))
 C = (F - 32)*(5 / 9)
 print("Temp in C=",C)
```

```
 case 2:
```

```
 F = float(input("Enter Temp. Value in Trems of F to get Temp in terms of K:"))
 K = (F - 32)*(5 / 9) + 273.15
 print("Temp in K=", K)
```

```
 case 3:
```

```
 C = float(input("Enter Temp. Value in Trems of C to get Temp in terms of F:"))
 F = C*(9 / 5) + 32
```

```

print("Temp in F=",F)
case 4:
 C = float(input("Enter Temp. Value in Terms of C to get Temp in terms of K:"))
 K = C + 273.15
 print("Temp in K=", K)
case 5:
 K = float(input("Enter Temp. Value in Terms of K to get Temp in terms of F:"))
 F = (K - 273.15)*(9 / 5) + 32
 print("Temp in F=",F)
case 6:
 K = float(input("Enter Temp. Value in Terms of K to get Temp in terms of C:"))
 C = K - 273.15
 print("Temp in C=", C)
case 7:
 print("Thx for using Program")
 sys.exit()
case _:
 print("Ur Selection of Operation Wrong!!!--try again")
print("Program Execution Completed")

```

---

**DAY-44 06/05/2024 MONDAY**

---

## String Handling Part-2

---

=>We know that, on str data we can perform Both Indexing and Slicing Operations.  
=>By using Indexing Concept, we can get Single Value from str object.  
=>By using Slicing Concept, we can get Range of Values from str object.  
=>Along with Indexing and Slicing Operations, we can perform Various Operations on str data 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 Alphabets OR Numerics or Alpha-Numerics only otherwise It returns False.

=>Syntax: strobj. isalphanum()

-----  
=>Examples:

```
>>> s="python310"
>>> s.isalnum()-----True
>>> s="python"
>>> s.isalnum()-----True
>>> s="310"
>>> s.isalnum()-----True
>>> s="$python310"
>>> s.isalnum()-----False
>>> s="python 310"
>>> s.isalnum()-----False
>>> s="$python3.10"
>>> s.isalnum()-----False
>>> s="python3.10"
>>> s.isalnum()-----False
```

## 11) isspace()

=>This Function returns True provided str obj contains purely space otherwise it returns False.  
=>Syntax: strobj.isspace()

Examples:

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

## 12) split()

=>This Function is used for splitting the given str object data into different words base specified delimiter (-  
\_ # % ^ ^ , ; ....etc)  
=>The default delimiter is space  
=>The Function returns Splitting data in the form of list object  
=>Syntax: strobj.split("Delimiter")

(OR)  
strobj.split()  
(OR)  
listobj= strobj.split("Delimiter")  
(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="PYThon"
>>> s.swapcase()-----'pytHON'
>>> s="PYTHON"
>>> s.swapcase()-----'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')-----ValueError: substring not found
>>> s.index('2')-----ValueError: substring not found
>>> s="python"
>>> s.index('thon')-----2
>>> s.index('khon')-----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')-----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)-----'applemangokiwiguava'
>>> print(k)

>>> lst=["apple","mango","kiwi","guava"]
>>> print(lst,type(lst))-----['apple', 'mango', 'kiwi', 'guava'] <class 'list'>
>>> k=""
>>> k=k.join(lst)
>>> print(k)-----applemangokiwiguava
>>> k-----'applemangokiwiguava'
>>> 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
```

=Examples window on String handling-2

---

capitalize() and title()

---

```
>>> s="python"
>>> print(s)-----python
>>> s.capitalize()-----'Python'
>>> print(s)-----python
>>> s="python"
>>> print(s)-----python
>>> s=s.capitalize()
>>> print(s)-----Python
>>> s="python"
>>> print(s)-----python
>>> s.title()-----'Python'
>>> s="python is an oop lang"
>>> print(s)-----python is an oop lang
>>> s.capitalize()-----'Python is an oop lang'
>>> print(s)-----python is an oop lang
>>> s.title()-----'Python Is An Oop Lang'
>>> s="Python IS AN OOP LANG"
>>> print(s)-----Python IS AN OOP LANG
>>> s.capitalize()-----'Python is an oop lang'
>>> s="Python IS AN OOP LANG"
>>> print(s)-----Python IS AN OOP LANG
>>> s.title()-----'Python Is An Oop Lang'
>>> s="python developed guio van rossum.python is an oop lang"
>>> print(s)-----python developed guio van rossum.python is an oop lang
>>> s.capitalize()-----'Python developed guio van rossum.python is an oop lang'
>>> print(s)-----python developed guio van rossum.python is an oop lang
>>> s.title()-----'Python Developed Guio Van Rossum.Python Is An Oop Lang'
```

---

count()

---

```
>>> s="MISSISSIPPI"
>>> print(s)-----MISSISSIPPI
>>> s.count("M")-----1
>>> s.count("S")-----4
>>> s.count("P")-----2
>>> s.count("I")-----4
>>> s=123321456234123412
>>> print(s,type(s))-----123321456234123412 <class 'int'>
>>> s1=str(s)
>>> print(s1,type(s1))-----123321456234123412 <class 'str'>
>>> s1.count(1)-----TypeError: must be str, not int
>>> s1.count('1')-----4
>>> s1.count('2')-----5
>>> s1.count(str(3))-----4
>>> len("MISSISSIPPI")-----11
>>> "MISSISSIPPI".count("I")-----4
>>> s="123321456234123412"
>>> s.count("1")-----4
>>> len(s)-----18
```

---

swapcase()

---

```
>>> s="PyThOn"
```

```
>>> print(s)-----PyThOn
>>> s.swapcase()-----'pYtHoN'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s.swapcase()-----'python'
>>> s="python"
>>> print(s)-----python
>>> s.swapcase()-----'PYTHON'
>>> s="PYThon"
>>> print(s)-----PYThon
>>> s.swapcase()-----'pyTHON'
>>> s="12345"
>>> print(s)-----12345
>>> s.swapcase()-----'12345'
>>> s="PyTHOn3.12"
>>> print(s)-----PyTHOn3.12
>>> s.swapcase()-----'pYthoN3.12'
```

---

upper() and lower()

---

```
>>> s="python"
>>> print(s)-----python
>>> s.upper()-----'PYTHON'
>>> s="PYThon"
>>> print(s)-----PYThon
>>> s.upper()-----'PYTHON'
>>> print(s)-----PYThon
>>> s.swapcase()-----'pyTHON'
>>> s="python"
>>> print(s)-----python
>>> s.swapcase()-----'PYTHON'
>>> print(s)-----python
>>> s.upper()-----'PYTHON'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s.lower()-----'python'
>>> print(s)-----PYTHON
>>> s.swapcase()-----'python'
>>> s="PYThon"
>>> print(s)-----PYThon
>>> s.lower()-----'python'
>>> print(s)-----PYThon
>>> s.swapcase()-----'pytHON'
```

---

isupper() and islower()

---

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s.isupper()-----True
>>> s.islower()-----False
>>> s="python"
>>> print(s)-----python
>>> s.isupper()-----False
>>> s.islower()-----True
>>> s="PYThon"
>>> print(s)-----PYThon
>>> s.isupper()-----False
>>> s.islower()-----False
>>> s="123#$%%"
>>> print(s)-----123#$%%
>>> s.isupper()-----False
```

```
>>> s.islower()-----False
>>> s="P1234"
>>> print(s)-----P1234
>>> s.isupper()-----True
>>> s.islower()-----False
>>> s="123p456"
>>> s.islower()-----True
>>> s.isupper()-----False
```

---

isdigit()

---

```
>>> s="123"
>>> print(s)-----123
>>> s.isdigit()-----True
>>> s="Python123"
>>> s.isdigit()-----False
>>> s="12.34"
>>> s.isdigit()-----False
```

---

isalpha()

---

```
>>> s="python"
>>> print(s)-----python
>>> s.isalpha()-----True
>>> s="python3.10"
>>> s.isalpha()-----False
>>> s="1234"
>>> s.isalpha()-----False
>>> s="Python Prog"
>>> s.isalpha()-----False
>>> s="Guido Van Rossum"
>>> s.isalpha()-----False
```

---

isalnum()

---

```
>>> s="Python312"
>>> s.isalnum()-----True
>>> s="123Java"
>>> s.isalnum()-----True
>>> s="Python"
>>> s.isalnum()-----True
>>> s="123"
>>> s.isalnum()-----True
>>> s="Python3.12"
>>> s.isalnum()-----False
>>> s="Python 312"
>>> s.isalnum()-----False
```

---

isspace()

---

```
>>> s=" "
>>> s.isspace()-----True
>>> s=""
>>> s.isspace()-----False
>>> s="python 123"
>>> s.isspace()-----False
>>> s="123 456 789"
>>> s.isspace()-----False
```

---

index()

---

```
>>> s="PYTHON"
>>> s.index("P")-----0
>>> s="RADAR"
>>> s.index("R")-----0
>>> s="MISSISSIPPI"
>>> s.index("I")-----1
>>> for ind,ch in enumerate(s):
... print(ind,"-->",ch)
0 --> M
1 --> I
2 --> S
3 --> S
4 --> I
5 --> S
6 --> S
7 --> I
8 --> P
9 --> P
10 --> I
>>> for ind,ch in enumerate(s):
... if(ch=="I"):
... print(ind,"-->",ch)
...
1 --> I
4 --> I
7 --> I
10 --> I
```

---

#### split() and split("delimiter")

---

```
>>> s="Python is an oop lang"
>>> x=s.split()
>>> print(x,type(x))----,['Python', 'is', 'an', 'oop', 'lang'] <class 'list'>
>>> len(x)-----5
>>> len(s)----21
>>> s="06-05-2024"
>>> x=s.split()
>>> print(x)----['06-05-2024']
>>> x=s.split("-")
>>> print(x)----[06, 05, 2024]
>>> print("DD=",x[0])-----DD= 06
>>> print("MM=",x[1])-----MM= 05
>>> print("YY=",x[2])-----YY= 2024
>>> s="Apple#Mango-Kiwi#Guava"
>>> x=s.split("#")
>>> print(x)----,['Apple', 'Mango-Kiwi', 'Guava']
>>> y=x[1].split("-")
>>> print(y)----['Mango', 'Kiwi']
>>> x[1]=y[0]
>>> print(x)----,['Apple', 'Mango', 'Guava']
>>> x.insert(-1,y[1])
>>> print(x)----,['Apple', 'Mango', 'Kiwi', 'Guava']
```

---

#### join()

---

```
>>> lst=['PYTHON', 'IS', 'AN', 'FUN', 'PROG', 'LANG']
>>> print(lst,type(lst))----,['PYTHON', 'IS', 'AN', 'FUN', 'PROG', 'LANG'] <class 'list'>
>>> s=" "
>>> s.join(lst)----'PYTHON IS AN FUN PROG LANG'
>>> print(s)
```

```

>>> s-----
>>> lst=['PYTHON', 'IS', 'AN', 'FUN', 'PROG', 'LANG']
>>> print(lst,type(lst))-----['PYTHON', 'IS', 'AN', 'FUN', 'PROG', 'LANG'] <class 'list'>
>>> s=" "
>>> s=s.join(lst)
>>> print(s)-----PYTHON IS AN FUN PROG LANG

```

lstrip() , rstrip() strip()

```

>>> s=" Python"
>>> print(s,len(s))-----Python 10
>>> s=s.lstrip()
>>> print(s,len(s))-----Python 6
>>> s="Python "
>>> print(s,len(s))-----Python 9
>>> s=s.rstrip()
>>> print(s,len(s))-----Python 6
>>> s=" Python "
>>> print(s,len(s))----- Python 13
>>> s=s.strip()
>>> print(s,len(s))-----Python 6

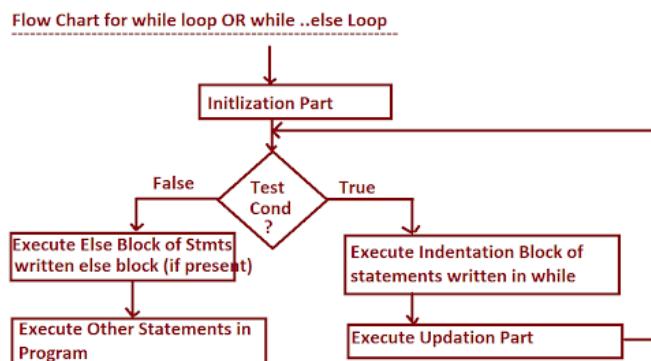
```

```

>>> s="Nirkar Behra "
>>> print(s,len(s))-----Nirkar Behra 18
>>> s=s.strip()
>>> print(s,len(s))-----Nirkar Behra 14
>>> " ".join(s.split())-----'Nirkar Behra'
>>> s="++++++PYTHON+++++"
>>> s.strip("+")-----'PYTHON'

```

DAY-45 07/05/2024 (TUESDAY)



### #Program for Generating 1 to n Numbers

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

```
#WhileLoopEx1.py
```

```
n=int(input("Enter How Many Numbers u want to Generate:"))
```

```
if(n<=0):
```

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

```
else:
```

```
 i=1 # InitLization Part
```

```
 while(i<=n): # Cond Part
```

```
 print("{}".format(i))
```

```
i=i+1 # Updation Part
else:
 print("I am from else part of while loop")
print("Other statements of while Loop Statements")
print("Other statements of if..else Statements")
=====
```

### #Program for Generating n to 1 numbers

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

```
#WhileLoopEx2.py
```

```
n=int(input("Enter How Many Number u want to generate:"))
if(n<=0):
 print("{} is Invalid input".format(n))
else:
 print("-"*50)
 print("Numbers within {} to 1".format(n))
 print("-" * 50)
 i=n
 while(i>=1):
 print("\t\t{}".format(i))
 i=i-1
 else:
 print("-" * 50)
=====
```

### #Program for Generating odd Numbers within in n

```
#WhileLoopEx3.py
```

```
n=int(input("Enter How Many Odd Number u want to generate within range:"))
if(n<=0):
 print("{} is Invalid input".format(n))
else:
 print("-"*50)
 print("Odd Numbers within 1 to n".format(n))
 print("-" * 50)
 i=1
 while(i<=n):
 print("\t\t{}".format(i))
 i=i+2
 else:
 print("-" * 50)
=====
```

### #Program for Generating Even Numbers within n in Decresing Order

```
#WhileLoopEx4.py
```

```
n=int(input("Enter How Many Even Number u want to generate within range:"))
if(n<=0):
 print("{} is Invalid input".format(n))
else:
 print("-"*50)
 print("Even Numbers within {} to 2".format(n))
 print("-" * 50)
 n= n if(n%2==0) else n-1
 while(n>=2):
 print("\t\t{}".format(n))
 n=n-2
 else:
 print("-" * 50)
=====
```

### #Program for generating Mul Table for a given Number

```
#WhileLoopEx5.py
```

```
n=int(input("Enter a Number for Generating Mul Table:"))
```

```

if(n<=0):
 print("{} is Invalid Input".format(n))
else:
 print("*50)
 print("Mul Table for :{}".format(n))
 print(" * 50)
 i=1 # Initialization Part
 while(i<=10): # Cond Part
 print("\t{} x {}={}".format(n,i,n*i))
 i=i+1 # Updation Part
 else:
 print(" * 50)
=====
```

### #Program for Generating a chars from word or line of Text

#### #WhileLoopEx6.py

```

word=input("Enter a word OR Line of Text:")
print("Given Word={}".format(word))
i=0
while(i<len(word)):
 print("\t{}".format(word[i]))
 i=i+1
print("-----OR-----")
i=len(word)-1
while(i>=0):
 print("\t{}".format(word[i]))
 i=i-1
print("-----OR-----")
i=0
word=word[::-1]
while(i<len(word)):
 print("\t{}".format(word[i]))
 i=i+1
print("-----OR-----")
word=word[::-1]
i=-1
while(i>=(-len(word))):
 print("\t{}".format(word[i]))
 i=i-1
print("-----OR-----")
i=-len(word)
while(i<=-1):
 print("\t{}".format(word[i]))
 i=i+1
=====
```

**DAY-46 08/05/2024(WEDNESSDAY)**

---

### 2. for loop or for ...else loop

---

Syntax1:-

---

for varname in Iterable\_object:

-----  
Indentation block of stmts  
-----

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

Syntax2:

```

for varname in Iterable_object:

 Indentation block of stmts

else:

 else block of statements

 Other statements in Program

```

### Explanation:

- =>Here 'for' , "in" and 'else' are keywords
  - =>Here Iterable\_object can be Sequence(bytes,bytearray,range,str),  
list(list,tuple),set(set,frozenset) and dict.
  - =>The execution process of for loop is that " Each of Element of Iterable\_object selected , placed in varname and executes Indentation block of statements".This Process will be repeated until all elements of Iterable\_object completed.
  - => when all the elements of Iterable Object completed then PVM comes out of for loop and executes else block of statements which are written under else block
  - =>Writing else block is optional.
- 

#Program for generating 1 to n Numbers where n is +ve

#ForLoopEx1.py

```

n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
 print("{} is Invalid".format(n))
else:
 print("-" * 50)
 print("Numbers within {}".format(n))
 print("-" * 50)
 for i in range(1,n+1):
 print("\t{}".format(i))
 else:
 print("-"*50)

```

---

#Program for generating n to 1 Numbers where n is +ve

#ForLoopEx2.py

```

n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
 print("{} is Invalid".format(n))
else:
 print("-" * 50)
 print("Numbers within {}".format(n))
 print("-" * 50)
 for i in range(n,0,-1):
 print("\t{}".format(i))
 else:
 print("-"*50)

```

---

#Program for generating Mul Table for a given Number

#ForLoopEx3.py

```

n=int(input("Enter a Number for Generating Mul Table:"))
if(n<=0):
 print("{} is Invalid Input".format(n))
else:
 print("*"*50)

```





```
line=input("Enter Line of Text:")
print("Given line:{}".format(line))
for ch in line:
 if(ch.isalpha()):
 print("{}".format(ch))
```

---

```
#Program for Obtaining Only Special Symbols from given line of Text / word
#and also Find Number of Special Symbols
```

```
#Input: P6&yt$ho@N
#Expected Output: $ @
```

```
#GetSpecialSymbols.py
```

```
line=input("Enter Line of Text:")
nosp=0
print("Given Line=",line)
for ch in line:
 if(not ch.isalnum()):
 nosp=nosp+1
 print("{}".format(ch))
else:
 print("Number of Special Symbols=",nosp)
```

---

```
#Program for Obtaining Only Special Symbols from given line of Text / word
#and also Find Number of Special Symbols
```

```
#GetSpecialSymbolsWithoutSpaces.py
```

```
line=input("Enter Line of Text:")
nosp=0
print("Given Line=",line)
for ch in line:
 if((not ch.isalnum()) and (not ch.isspace())):
 nosp=nosp+1
 print("{}".format(ch))
else:
 print("Number of Special Symbols=",nosp)
```

---

```
#Program for Obtaining Only Special Symbols from given line of Text / word
#and also Find Number of Special Symbols
```

```
#GetSpecialSymbolsWithoutSpacesnewlgic.py
```

```
line=input("Enter Line of Text:")
nosp=0
print("Given Line=",line)
words=line.split() # words= ['Python', 'is', 'an', 'oop$', 'La#ng']
for word in words:
 for ch in word:
 if(not ch.isalnum()):
 print("\t{}".format(ch))
 nosp=nosp+1
else:
 print("Number of Special Symbols=",nosp)
```

---

```
#Program for Reading List of Values from Key Board and Display those Values.
```

```
#ReadValuesEx1.py--Logic-1
```

```
nov=int(input("How Many Values u want to Enter:"))
```

```
if(nov<=0):
```

```
 print("{} is Invalid Input".format(nov))
```

```
else:
```

```
 lst=[] # create an empty list for adding the values
```

```
 for i in range(1,nov+1):
```

```
 val=float(input("Enter {} Value:".format(i)))
```

```
Ist.append(val)
else:
 print("List of Values={}".format(Ist))
```

---

#Program for Reading List of Values from Key Board and Display those Values.

#ReadValuesEx2.py--Logic-2

```
import sys
print("Enter Number of Values and Press @ to stop")
Ist=[]
while(True):
 value=input()
 if(value=="@"):
 if(len(Ist)==0):
 print("list is empty")
 else:
 print("Content of Lst=", Ist)
 sys.exit()
 else:
 Ist.append(float(value))
```

---

**DAY-48 10/05/2024 (FRIDAY)**

---

### Transfer Flow Statements

---

=>The purpose of Transfer Flow Statements is that "To transfer the control of PVM from One Part of the Program to

Another Part of the Program"

=>In Python Programming, we have 4 types of Transfer Flow Statements. They are

1. break
  2. continue
  3. pass
  4. return (In Functions )
- 

#### **1. break statement**

---

=>break is a key word

=>break keyword must be used always inside of loops otherwise we get Syntax error

=>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

```

---

## #Program for Demonstrating break key word

### #BreakStmtEx1.py

```
s="PYTHON"
print("By using For Loop")
for ch in s:
 print("\t{}".format(ch))
else:
 print("I am from else part of for loop")
print("-----")
#My Requirment is to dispaly 'PYTH' without using Indexing and slicing
for ch in s: # s="PYTHON"
 if(ch=="O"):
 break
 else:
 print(ch,end="")
else:
 print("I am from else part of for loop")
print("\nOther statements in Program")
```

## #Program for Demonstrating break key word

### #BreakStmtEx2.py

```
s="PYTHON"
print("By using while Loop")
i=0
while(i<len(s)):
 print("\t{}".format(s[i]))
 i=i+1
else:
 print("I am from else part of while loop")
print("-----")
#My Requirment is to dispaly 'PYTH' without using Indexing and slicing
i=0
while(i<len(s)):
 if(s[i]=="O"):
 break
 print(s[i],end="")
 i=i+1
else:
 print("I am from else part of while loop")
print("\nOther statements in Program")
```

## #Program for Demonstrating break key word

### #BreakStmtEx3.py

```
s="MISSISSIPPI"
print("By using For Loop")
for ch in s:
 print("\t{}".format(ch))
else:
 print("I am from else part of for loop")
print("-----")
#My Requirment is to dispaly 'MISS' without using Indexing and slicing
ni=0
i=0 # s="MISSISSIPPI"
while(i<len(s)):
 if(s[i]=="I"):
 ni=ni+1
 if(ni==2):
 break
 print(s[i],end="")
 i=i+1
```

```
else:
 print("I am from else part of for loop")
print("\nOther statements in Program")
=====
```

```
#Program for Demonstrating break key word
#BreakStmtEx3.py
s="MISSISSIPPI"
print("By using For Loop")
for ch in s:
 print("\t{}".format(ch))
else:
 print("I am from else part of for loop")
print("-----")
#My Requirment is to dispaly 'MISS' without using Indexing and slicing
ni=0
s="MISSISSIPPI"
for i in range(len(s)):
 if(s[i]=="I"):
 ni=ni+1
 if(ni==2):
 break
 print(s[i],end="")
else:
 print("I am from else part of for loop")
print("\nOther statements in Program")
=====
```

```
#Program for deciding whether the Number is Prime or Not
#BreakStmtEx5.py
n=int(input("Enter a Number to decide Prime or Not:"))
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
 print("{} is {}".format(n,res))
=====
```

```
#Program for deciding whether the Number is Prime or Not
#BreakStmtEx6.py
n=int(input("Enter a Number to decide Prime or Not:"))
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))
=====
```

```
#Program for deciding whether the Number is Prime or Not
#BreakStmtEx6.py
n=int(input("Enter a Number to decide Prime or Not:"))
```

```

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
 res="NOT PRIME" if res else "PRIME"
 print("{} is {}".format(n,res))

```

---

#Program for Deciding whether the given word is Vowel or Not

#BreakStmtEx8.py

```

word=input("Enter a word:") #HYDERABAD
res="NOT VOWEL WORD"
for ch in word:
 if ch.lower() in ['a','e','i','o','u']:
 res="VOWEL WORD"
 break
print("{} is {}".format(word,res))

```

**DAY-48 11/05/2024 (SATURDAY)**

## 2. 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

```

#Program for Demonstrating Continue Statement

#ContinueStmtEx1.py

s="PYTHON"

for ch in s:

print("{}".format(ch))

else:

```

print("i am else part of for loop")
print("-----")
#My Requirement is to Display: PTON
for ch in s: # S="PYTHON"
 if(ch=="H") or (ch=="Y"):
 continue
 print(ch,end="")
else:
 print("\ni am else part of for loop")
print("-----")
=====
```

### #Program for Demonstrating Continue Statement

#### #ContinueStmtEx2.py

```

s="PYTHON"
i=0
while(i<len(s)):
 print("{}".format(s[i]))
 i=i+1
else:
 print("i am from else part of while loop")
print("-----")
#My Requirement is to Display: PYTON
i=0
while(i<len(s)):
 if s[i] in ["Y","H"]:
 i = i + 1
 continue
 print("{}".format(s[i]),end="")
 i=i+1
else:
 print("\ni am from else part of while loop")
print("-----")
```

---

### #Program for accepting List of Values and separate them with +ve and -ve vals

#### #ContinueStmtEx3.py

```

nov=int(input("How Many Values u want to Enter:"))
if(nov<=0):
 print("{} is Invalid Input".format(nov))
else:
 lst=[] # create an empty list for adding the values
 for i in range(1,nov+1):
 val=float(input("Enter {} Value:".format(i)))
 lst.append(val)
 else:
 print("List of Values={}".format(lst)) # [10.0, -20.0, 30.0, -40.0, -50.0]
 #Code for Obtaining List of +Ve vals
 pslist=[]
 for val in lst:
 if(val<=0):
 continue
 pslist.append(val)
 else:
 print("List of +VE Values=",pslist)
 print("Number of +Ve Values=",len(pslist))
 print("-----")
 #Code for Obtaining List of -Ve vals
 nvlist=list()
 for val in lst:
 if(val>=0):
 continue
 nvlist.append(val)
 else:
 print("List of -VE Values=",nvlist)
 print("Number of -Ve Values=",len(nvlist))
 print("-----")
```

```
 continue
 nvlist.append(val)
else:
 print("List of -VE Values=", nvlist)
 print("Number of -Ve Values=", len(nvlist))
 print("-----")
```

---

### #Program for accepting List of words and obtain those words whose length is either 3 or 5 #ContinueStmtEx4.py

```
nov=int(input("How Many Words u want to Enter:"))
if(nov<=0):
 print("{} is Invalid Input".format(nov))
else:
 lst=[] # create an empty list for adding the values
 for i in range(1,nov+1):
 val=input("Enter {} Value:".format(i))
 lst.append(val)
 else:
 print("List of Words={}".format(lst))
 #['Why', 'Apple', 'Elephant', 'Python', 'while']
 #Code for Obtaining those words whose length is either 3 or 5
 word35=list() # for storing 3 or 5 length word
 for word in lst:
 if(len(word) not in [3,5]):
 word35.append(word)
 else:
 print("List of words with 3 or 5 in Length")
 print(word35)
```

---

### #Program for accepting List fo words and Obtains only Palindrome Words #ContinueStmtEx5.py

```
print("Enter Number of Words and Press any special to stop")
lst=[]
while(True):
 value=input()
 if(not value.isalnum()):
 break
 else:
 lst.append(value)
print("List of Words=",lst)
#Code for Obtaining Palindrome words
plist=list() # For Storing Palindrome Words
for word in lst:
 if(word!=word[::-1]):
 continue
 plist.append(word)
else:
 print("List of Palindrome Words")
 print(plist)
```

---

### #Program for accepting List of Values and separate them with +ve and -ve vals #ContinueStmtEx3.py

```
nov=int(input("How Many Values u want to Enter:"))
if(nov<=0):
 print("{} is Invalid Input".format(nov))
else:
 lst=[] # create an empty list for adding the values
 for i in range(1,nov+1):
 val=float(input("Enter {} Value:".format(i)))
```

```

lst.append(val)
else:
 print("List of Values={}".format(lst)) # [10.0, -20.0, 30.0, -40.0, -50.0]
#Code for Obtaining List of +Ve vals
pslist=[]
for val in lst:
 if(val<=0):pass
 else:
 pslist.append(val)
else:
 print("List of +VE Values=",pslist)
 print("Number of +Ve Values=",len(pslist))
 print("-----")
#Code for Obtaining List of -Ve vals
nvlist=list()
for val in lst:
 if(val>=0):pass
 else:
 nvlist.append(val)
else:
 print("List of -VE Values=", nvlist)
 print("Number of -Ve Values=", len(nvlist))
 print("-----")

```

---

### #Program for accepting List of Text and find length of each word

#### #WordsLengthEx.py

```

line=input("Enter Line of Text:")
print("Given Line=",line)
words=line.split()
print("-----")
for word in words:
 print("{}-->{}".format(word,len(word)))
print("-----OR-----")
wl=dict()
for word in words:
 wl[word]=len(word)
else:
 for k,v in wl.items():
 print("\t{}---->{}".format(k,v))
print("-----OR-----")
wt=list()
for word in words:
 wt.append((word,len(word)))
else:
 for tpl in wt:
 print(tpl)

```

---

**DAY-49 14/05/2024 (TUESDAY)**

---

## 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 Loops is that "For Every Value of Outer Loop Inner Loop Repeated Finite Number of Times Until Inner loop Test Cond becomes False".
- =>In General , Outer Loop Defined for 'n' Times to repeat and Corresponding Inner Loop Defined for 'm' times the total Number of times Both the loop repeats is ' $n*m$ ' times.
- =>In Python Programming, we can Define Inner Loop in 4 ways. They are

1. for loop in for loop.
  2. while loop in while loop.
  3. for loop in while loop.
  4. while loop in for loop.
- 

Syntax-1: 1. for loop in for loop.

---

```
for varname1 in iterable-object: # Outer Loop

 for varname2 in iterable-object: # Inner Loop

 else:

 else:

```

---

Syntax-2: while loop in while loop

---

```
while(Test Cond1): # Outer Loop

 while(Test Cond2): # Inner Loop

 else:

 else:

```

---

Syntax-3: for loop in while loop.

---

```
while(Test Cond1): # Outer Loop

 for varname in Iterfable-Object: # Inner Loop

 else:

 else:

```

---

Syntax-4: while loop in for loop.

---

```
for varname1 in iterable-object: # Outer Loop

 while(Test Cond2): # Inner Loop

 else:

 else:

```

---

#InnerLoopEx1.py---for loop in for loop  
for i in range(1,6):#Outer Loop

```
print("val of i-outer loop={}".format(i))
for j in range(1,4):#Inner loop
 print("\tval of j-inner loop={}".format(j))
else:
 print("\tOut-off inner loop")
else:
 print("Out-off outer loop")
```

---

```
#InnerLoopEx2.py---while loop in while loop
i=1
while(i<=6):#Outer Loop
 print("val of i-outer loop={}".format(i))
 j=1
 while(j<=3):# Inner loop
 print("\tval of j-inner loop={}".format(j))
 j=j+1
 else:
 print("\tOut-off inner loop")
 i = i + 1
else:
 print("Out-off outer loop")
```

---

```
#InnerLoopEx3.py---for loop in while loop
i=6
while(i>=1): # outer loop
 print("val of i-outer loop={}".format(i))
 for j in range(3,0,-1): # Inner loop
 print("\tval of j-inner loop={}".format(j))
 else:
 print("\tOut-off inner loop")
 i = i-1
else:
 print("Out-off outer loop")
```

---

```
#InnerLoopEx4.py---while loop in for loop
for i in range(6,0,-1): # outer loop
 print("val of i-outer loop={}".format(i))
 j=1
 while(j<=3): # Inner loop
 print("\tval of j-inner loop={}".format(j))
 j=j+1
 else:
 print("\tOut-off inner loop")
else:
 print("Out-off outer loop")
```

---

```
#InnerLoopEx5.py---for in for in for
for i in range(1,4): # outer Loop
 print("\ti\tj\tk")
 print("-----")
 for j in range(1,3):# Inner Loop

 for k in range(1,3):#Inner-Inner loop
 print("\t{}\t{}\t{}".format(i,j,k))
 else:pass
 else:
 print("-----")
else:
```

```
print("=====")

```
#InnerLoopEx6.py---Mul tables for 1 to n Numbers
n=int(input("Enter How Many Mul Tables u want display:"))
if(n<=0):
    print("{} is Invalid input".format(n))
else:
    for num in range(1,n+1): # Outer For loop supply the number
        print("-----")
        print("Mul Table for {}".format(num))
        print("-----")
        for i in range(1,11): # Inner loop--generates mul table
            print("\t{} x {}={}".format(num,i,num*i))
        else:
            print("-----")
```

```

```
#Mul Tables for random Numbers
#InnerLoopEx7.py
n=int(input("Enter How Many Mul Tables u want display:"))
if(n<=0):
 print("{} is Invalid input".format(n))
else:
 lst=list()
 for i in range(1,n+1):
 val=int(input("Enter {} Value for Generating Mul Table:".format(i)))
 lst.append(val)
 else:
 print("Given List of Values:{}".format(lst))# [3, 19, -9, 0, 12]
 #Code for Mul tables for Random Numbers
 for num in lst: # Outer For loop supply the number from list
 if(num<=0):
 print("{} is Invalid input".format(num))
 else:
 print("-----")
 print("Mul Table for:{}".format(num))
 print("-----")
 for i in range(1,11):#Inner Loop--generates mul Table
 print("\t{} x {}={}".format(num,i,num*i))
 else:
 print("-----")
```

---

```
#Program for Generating Prime Numbers within the Given Range
#InnerLoopEx8.py
n=int(input("Enter Range in which u want Prime Numbers:"))
if(n<=2):
 print("{} is Invalid input".format(n))
else:
 print("Prime Numbers within:{}".format(n))
 nop=0
 for num in range(2,n+1): # Outer Loop--Supply the Numbers
 res=True
 for i in range(2,num):#Inner loop--Decides weather the Num is Prime or not
 if(num%i==0):
 res=False
 break
 if(res):
 print("\t{}".format(num))
 nop = nop + 1
 else:
```

```
print("Numbers Primes within {}={}".format(n, nop))
```

---

```
#Given Input lst=[123,45,34,24,89]
Expected Output: sumlist= [6,9,7,6,17]
#InnerLoopEx10.py
n=int(input("Enter How Values u want to List:"))
if(n<=0):
 print("{} is Invalid input".format(n))
else:
 lst=list()
 for i in range(1,n+1):
 val=int(input("Enter {} Number :".format(i)))
 lst.append(val)
 else:
 sumlist=list()
 print("Given List of Values:{}".format(lst))#[123, 45, 2345, 12, 67]
 for num in lst:## Outer Loop--Supply the Numbers
 s=0
 for v in str(num):
 s=s+int(v)
 else:
 sumlist.append(s)
 else:
 print("Given list=",lst)
 print("Sum List=",sumlist)
```

---

## DAY-50 15/05/2024 (WEDNESDAY)

---

### Functions in Python----7 days+1

---

=====  
Index

---

=>Types of Languages  
    i) Un-Structured Programming Languages  
    ii) Structured Programming Languages  
=>Purpose of Functions  
=>Advantages of Functions  
=>Parts of Functions  
=>Phases in Functions  
=>Syntax for Defining Functions in Python  
=>Number of Approaches to Define Functions.  
=>Programming Examples

---

=>Parameters and Arguments  
=>Types of Arguments  
    i) Positional Arguments  
    ii) Default Arguments  
    iii) Keyword Arguments  
    iv) Variable Length Arguments  
    v) Keyword Variable Length Arguments

=>Programming Examples

---

=>Local and Global Variables  
=>Programming Examples  
=>global Keyword  
=>Programming Examples  
=>globals()  
=>Programming Examples

---

=>Comprehension Techniques

- i) List Comprehension
- ii) set Comprehension
- iii) Dict Comprehension
- iv) tuple Comprehension (Not there)

=>Programming Examples

---

=>Anonymous Functions OR Lambda Functions

=>Implementation and Syntax for Anonymous Functions OR Lambda Functions

=>Programming Examples

---

=>Special Functions Python

- i) filter() with Normal and Anonymous Functions
- ii) map() with Normal and Anonymous Functions
- iii) reduce() with Normal and Anonymous Functions

=>Programming Examples

---

---

## Types of Languages

---

=>In Industry , we have Two Types of Programming Langauges. They are

- 1. Un-Structured Programming Languages.
- 2. Structured Programming Languages.

1. Un-Structured Programming Languages

---

=>The Un-Structured Programming Languages DOES NOT CONTAIN FUNCTIONS Concept. So that whose related Applications has the Following Limitations.

- 1. Application Development time is More
- 2. Application Memory Space is More
- 3. Application Execution Time is More
- 4. Application Performnace is Degraded
- 5. Redundency(Duplication or Replication) of the Code is More

Examples: GW-BASIC

---

2. Structured Programming Languages.

---

=>The Structured Programming Languages CONTAINS FUNCTIONS Concept. So that whose related Applications has the Following Advatnages.

- 1. Application Development time is Less.
- 2. Application Memory Space is Less.
- 3. Application Execution Time is Less.
- 4. Application Performnace is Enhanced (Improved)
- 5. Redundency(Duplication or Replication) of the Code is Minimized.

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

---

## 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.

#### 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 is 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 OR Func Proc 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)

=====X=====

## 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 Definition 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

**DAY-51 16/05/2024 (THURSDAY )**

===== Number of Approaches to Define Functions. =====

=>In Python Programming, To Define any Function, we have 4 Approaches. They are

### Approach-1

#INPUT: Taking from Function call  
#PROCESS: Taken Place in Function Body  
#Output: Given to Function call

### Approach-2

#INPUT: Taking inSide of Function Body  
#PROCESS: Taken in Function Body  
#Output: Display Inside of Function Body

---

### Approach-3

---

#INPUT: Taking from Function call  
#PROCESS: Taken Place in Function Body  
#Output: Display Inside of Function Body

---

### Approach-4

---

#INPUT: Taking inSide of Function Body  
#PROCESS: Taken in Function Body  
#Output: Given to Function call

---

#Define a Function for Addition of Two Numbers.c=a+b  
#Approach1.py

```
def addop(a,b): # Here a, b are called Formal Parameters
 print("I am in Function Def")
 c=a+b # Here c is called Local Variable
 return c
#Main Program
print("I am from Main Program")
print("Type of addop=",type(addop))
res=addop(10,20) # Function Call
print("sum=",res)
print("-----")
c=addop(100,200) # Function Call
print("Sum=",c)
print("-----")
c=addop(-100,-200) # Function Call
print("Sum=",c)
```

---

#Define a Function for Addition of Two Numbers.c=a+b

#ApproachEx1.py

#INPUT: Taking from Function call  
#PROCESS: Taken Place in Function Body  
#Output: Given to Function call

```
def addop(x,y): # here x,y are called formal Parameters
 z=x+y # Here z is called Local Var
 return z
#Main Program
print("-----")
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
res=addop(a,b) # Function Call
print("Sum({},{})={}".format(a,b,res))
print("-----")
x=float(input("Enter First Value:"))
y=float(input("Enter Second Value:"))
z=addop(x,y) # Function Call
print("Sum({},{})={}".format(x,y,z))
print("-----")
```

---

#Define a Function for Addition of Two Numbers.c=a+b

#ApproachEx2.py

#INPUT: Taking inSide of Function Body  
#PROCESS: Taken in Function Body  
#Output: Display Inside of Function Body

```
def addop():
 #Input: Taking inSide of Function Body
```

```
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
#Process: Taken in Function Body
c=a+b
#Output: Display Inside of Function Body
print("sum({},{})={}".format(a,b,c))
```

```
#Main Program
addop() # Function Call
```

---

```
#Define a Function for Addition of Two Numbers.c=a+b
#ApproachEx3.py
#INPUT: Taking from Function call
#PROCESS: Taken Place in Function Body
#Output: Display Inside of Function Body
def addop(a,b):
 #Process
 c=a+b
 #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 Addition of Two Numbers.c=a+b
#ApproachEx4.py--Most Imp
#INPUT: Taking inSide of Function Body
#PROCESS: Taken in Function Body
#Output: Given to Function call
def addop():
 #Taking Input
 k=float(input("Enter First Value:"))
 v=float(input("Enter Second Value:"))
 #Process
 r=k+v
 #return the value
 return k,v,r # return stmt can return one or More Number of values
#Main Program
x,y,res=addop() # Function Call with Multiline Assigment
print("sum({},{})={}".format(x,y,res))
print("-----OR-----")
hyd=addop() # Function Call with Single Line Assigment
#here hyd is an object of <class,tuple>
print("sum({},{})={}".format(hyd[0],hyd[1],hyd[2]))
print("-----OR-----")
print("sum({},{})={}".format(hyd[-3],hyd[-2],hyd[-1]))
```

---

---

DAY-52 17/05/2024 THURSDAY

---

```
#Program for accepting list of words and find max length words
#MaxlengthWordsEx1.py
def getwords():
 print("Enter List of Words and Press @ to stop:")
 lst=[]
 while(True):
 val=input()
 if(val=="@"):
 break
 else:
```

```

 lst.append(val)
 return lst
def find maxlenword(words):#['Python', 'java', 'HTML', 'django', 'Hyderabad', 'bang']
 if(len(words)==0):
 print("List is empty--can't find max length word")
 else:
 d=dict()
 for word in words:
 d[word]=len(word)
 else:
 lst=[]
 print(d) # {'python': 6, 'java': 4, 'banglore': 8,'C': 1, 'C++': 3 }
 ml=max(list(d.values())) # ml=8
 for word,length in d.items():
 if(length==ml):
 lst.append(word)
 else:
 print("Max Length Words")
 for word in lst:
 print("\t{}".format(word))
 else:
 print("Number of Max Length Words=",len(lst))

```

---

```

#Program for accepting list of words and find max length words
#MaxlengthWordsEx1.py
def getwords():
 print("Enter List of Words and Press @ to stop:")
 lst=[]
 while(True):
 val=input()
 if(val=="@"):
 break
 else:
 lst.append(val)
 return lst
def find maxlenword(words):#['Python', 'java', 'HTML', 'django', 'Hyderabad', 'bang']
 if(len(words)==0):
 print("List is empty--can't find max length word")
 else:
 d=dict()
 for word in words:
 d[word]=len(word)
 else:
 lst=[]
 print(d) # {'python': 6, 'java': 4, 'banglore': 8,'C': 1, 'C++': 3 }
 list1=list(d.values()) # ml=8
 #Code for Find Max Value from Dict of Values
 ml=list1[0]
 for val in list1:
 if val>ml:
 ml=val
 else:
 for word,length in d.items():
 if(length==ml):
 lst.append(word)
 else:
 print("Max Length Words")
 for word in lst:
 print("\t{}".format(word))

```

```
else:
 print("Number of Max Length Words=",len(lst))
#Main program
words=getwords()#['Python', 'java', 'HTML', 'django', 'Hyderabad', 'bang']
findmaxlenword(words)
```

---

```
#SimpleIntEx1.py
def simpleint():
 p=float(input("Enter Principle Amount:"))
 t=float(input("Enter Time:"))
 r=float(input("Enter Rate of Interest:"))
 #cal si and totamt to pay
 si=(p*t*r)/100
 totamt=p+si
 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)
```

```
#Main Program
simpleint() # Function Call
```

---

```
#SimpleIntEx2.py
def takevalues():
 p = float(input("Enter Principle Amount:"))
 t = float(input("Enter Time:"))
 r = float(input("Enter Rate of Interest:"))
 return p,t,r
def calsimpleint():
 p,t,r=takevalues() # Function with Multiline assigment
 si=(p*t*r)/100
 totamt=p+si
 return p,t,r,si,totamt
def displayresult():
 p,t,r,si,totamt=calsimpleint()# Function with Multiline assigment
 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)
```

```
#Main Program
displayresult() # Function call
```

```
#NOTE: displayresult()--->calsimpleint()--->takevalues() is called Function Chaining
```

---

```
#Program for accepting List of Values and find their sum and avg
#SumAvgEx1.py
def readvalues():
 nov=int(input("Enter How Many Value u want:"))
 if(nov<=0):
 return []
 else:
 lst=[]
 for i in range(1,nov+1):
 val=float(input("Enter {} Vakue:".format(i)))
```

```

 lst.append(val)
 return lst
def findsumavg(lst): # lst=[10,20,30,40]
 if(len(lst)==0):
 print("List is empty-can't find sum and avg")
 else:
 s=0
 for val in lst:
 print("\t{}".format(val))
 s=s+val
 else:
 print("Sum={}".format(s))
 print("Avg={}".format(s/len(lst)))
#Main Program
lst=readvalues() # Function Call
findsumavg(lst) # Function call

```

---

**DAY-53 18/05/2024 SATURDAY**

### ===== Parameters and Arguments =====

#### Parameters

=>In Python Parameters are classified into Two Types. They are

1. Formal Parameters / Variables
2. Local Variables / Parameters

#### 1. Formal Parameters / Variables

=>Formal Parameters / Variables are those which are used in Function Heading.

=>The Purpose of Formal Parameters is that "To store OR Hold the Inputs coming From Function Calls".

=>The values of Formal Parameters can be accessed within in Corresponding Function Definition But not Possible to

Access in Other Part of Function Definition OR In other Part of the Program.

#### Local Variables / Parameters

=>Local Variables / Parameters are those which are used as a part of Function Body.

=>The purpose of Local Variables is that "To store the Result of Function Processing Logic"

=>The values of Local Parameters can be accessed within in Corresponding Function Definition But not Possible to

Access in Other Part of Function Definition OR In other Part of the Program.

#### Examples

```
def sumop(a,b,c): # Here a,b,c are called Formal Parameters
```

```
d=a+b+c # Here d is called local Variable
```

#### Arguments

=>Arguments are also called Variables

=>Arguments are Variables used inside of Function Call.

#### Examples:

```
sumop(10,20,30) # Function call--here 10 20 30 are called Argument Values
OR
```

```
x,y,z=10,20,30
```

```
sumop(x,y,z) # Function call--- here x,y,z are called Arguments
```

=>The Relationship between Arguments and Parameters is that Every Value of Argument Must pass to Formal Parameters

### Types of Arguments and Parameters

=The Relationship between Arguments and Parameters is that Every Value of Argument Must pass to Formal Parameters.

=>Based on Passing the Values of Arguments to Formal Parameters, Arguments are Classified into 5 types. They are

1. Positional Arguments (OR) Parameters
2. Default Arguments (OR) Parameters
3. Keyword Arguments (OR) Parameters
4. Variable Length Arguments (OR) Parameters
5. Keyword Variable Length Arguments (OR) Parameters

#### 1. Posstional Arguments

=>Posstional Arguments Mechanism is the default arguments passing mechanism used by PVM in Functions for Passing the values of Arguments of Function Call to Formal Parameters of Function Defintion.

=>Posstional Arguments concept says that Every Argument Value Passing to Every Formal Parameter Based on their Posstion by maintaining Order and Meaning for Higher Accuracy. In Otherwords The number of arguments must be equal to Number of Formal Parameters.

=>Posstional Arguments concept always used for Passing Specific Data from Function calls to Function Definitions.

=>PVM gives High Priority for Posstional Arguments

Syntax: def functionname(Param1,Param2,...Param-n): # Function Definition

-----  
-----  
Block of Statements--perform Operation  
-----  
-----

Syntax: functionname(arg1,arg2,.....,arg-n) # Function call

=>Here the values of arg1,arg2,.....,arg-n of Function call are passing to Param1,Param2,...Param-n of Function Definition 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 arguments OR 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 "Posstional parameters".

Rule:- When we use default parameters in the function definition, They must be used as last Parameter(s) otherwise we get Error( SyntaxError: parameter without a default follows parameter with a default)

#Program for Demonstrating Default Arguments OR Parameters

```
#DefaultArgsEx1.py
```

```
def studinfo(sno,sname,marks,crs="PYTHON"): # Here sno,sname,marks are called Posstional Parameters and crs is called Default Parameter
```

```
print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))
```

```
#Main Program
print("*50)
print("\tSTNO\tNAME\tMARKS\tCOURSE")
print("*50)
studinfo(100,"RS",34.56) # function call--here 100,"RS",34.56 are called Possitional Arguments
studinfo(200,"TR",45.67) # Function call
studinfo(300,"DR",35.67) # Function call
studinfo(400,"SS",11.11) # Function call
studinfo(500,"SR",34.56,"Java") # Function call
studinfo(600,"XX",21.11) # Function call
print("*50)
```

---

```
#Program for Demonstrating Default Arguments OR Parameters
#DefaultArgsEx2.py
def studinfo(crs="PYTHON",city="HYD",sno,sname,marks): # here crs and city are called Default Arguments
 print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,city))
#Main Program
print("*50)
print("\tSTNO\tNAME\tMARKS\tCOURSE\tCITY")
print("*50)
studinfo(100,"RS",34.56) # function call--here 100,"RS",34.56 are called Possitional Arguments
studinfo(200,"TR",45.67) # Function call
studinfo(300,"DR",35.67) # Function call
studinfo(400,"SS",11.11) # Function call
studinfo(500,"SR",34.56,"Java") # Function call
studinfo(600,"XX",21.11) # Function call
studinfo(700,"DT",37.11,city="USA")
studinfo(900,"PT",27.11,city="RSA",crs="Java")

print("*50)
```

---

```
#Program for Demonstrating Possitional Arguments
#PossArgsEx1.py
def studinfo(sno,sname,marks):
 print("\t{}\t{}\t{}".format(sno,sname,marks))
#Main Program
print("*50)
print("\tSTNO\tNAME\tMARKS")
print("*50)
studinfo(100,"RS",34.56) # function call
studinfo(200,"TR",45.67) # Function call
studinfo(300,"DR",35.67) # Function call
studinfo(400,"SS",11.11) # Function call
print("*50)
```

---

```
#Program for Demonstrating Possitional Arguments
#PossArgsEx2.py
def studinfo(sno,sname,marks,crs): # Here sno,sname,marks are called Possitional Parameters
 print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))
#Main Program
print("*50)
print("\tSTNO\tNAME\tMARKS\tCOURSE")
print("*50)
studinfo(100,"RS",34.56,"PYTHON") # function call--here 100,"RS",34.56,PYTHON are called Possitional Arguments
studinfo(200,"TR",45.67,"PYTHON") # Function call
studinfo(300,"DR",35.67,"PYTHON") # Function call
studinfo(400,"SS",11.11,"PYTHON") # Function call
```

```
print("*50)
```

**DAY-54 20/05/2024 (MONDAY)**

---

### **3) Keyword Parameters (or) arguments**

---

=>In some of the circumstances, we know the function name and formal parameter names and we don't know the order of formal Parameter names and to pass the data / values accurately we must use the concept of Keyword Parameters (or) arguments.

=>The implementation of Keyword Parameters (or) arguments says that all the formal parameter names used as arguments in Function call(s) as keys.

---

Syntax for function definition:-

---

```
def functionname(param1,param2...param-n):
```

---

Syntax for function call:-

---

```
functionname(param-n=val-n,param1=val1,param-n-1=val-n-1,.....)
```

Here param-n=val-n,param1=val1,param-n-1=val-n-1,..... are called Keywords arguments

=>When we specify Keyword arguments before Posstional Arguments in Function Calls(s) then we get SyntaxError: positional argument follows keyword argument bcoz PVM gives First Priority for positional arguments.

---

=====X=====

---

### **4) Variables Length Parameters (or) arguments**

---

=>When we have familiy of multiple Similar function calls with Variable number of values / arguments then with normal python programming, we must define mutiple function defintions. This process leads to more development time.

=>To overcome this process, we must use the concept of Variable length Parameters .

=>To Implement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called astrisk (\* param) and the formal parameter with astrisk symbol is called Variable length Parameters and whose purpose is to hold / store any number of values coming from similar function calls and whose type is <class, 'tuple'>.

---

Syntax for function definition with Variables Length Parameters:

---

```
def functionname(list of 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) .

---

=====X=====

```
#Program for Demonstrating Keyword Arguments
```

```
#KeyWordsArgsEx1.py
```

```
def disp(A,B,C,D):
```

```
 print("\t{}\t{}\t{}\t{}".format(A,B,C,D))
```

```
#Main Program
```

```
print("-----")
```

```
print("\tA\tB\tC\tD")
```

```
print("-----")
```

```
disp(10,20,30,40) # Function Call with Posstional args
```

```
disp(B=20,A=10,D=40,C=30) # Function Call with Keyword args
```

```
disp(D=40,C=30,B=20,A=10) # Function Call with Keyword args
```

```

disp(10,20,D=40,C=30) # Function Call with Possitional args and Keyword args
disp(10,C=30,B=20,D=40) # Function Call with Possitional args and Keyword args
#disp(C=30,B=20,D=40,10) -----SyntaxError: positional argument follows keyword argument
print("-----")

#Program for Demonstrating Keyword Arguments
#KeyWordsArgsEx2.py
def dispstudinfo(sno,sname,marks,crs="PYTHON",cnt="INDIA"):
 print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))

#Main Program
print("-"*50)
print("\tSNO\tNAME\tMARKS\tCOURSE\tCOUNTRY")
print("-"*50)
dispstudinfo(100,"RS",34.56) # Function Call with Possitional Args
dispstudinfo(sname="TR",sno=200,marks=88.88) # Function Call with Keyword Args
dispstudinfo(crs="Java",sno=300,sname="DR",marks=55.55,cnt="USA") # Function Call with Keyword
Args
dispstudinfo(400,"SR",cnt="RSA",marks=67.88,crs="HTML") # Function Call with Keyword Args
print("-"*50)

#Program for Demonstrating Variable Length Arguments (OR) Parameters
#VariableArgsLenEx1.py
#This Program will not execute as it is bcoz PVM Remembers Latest Function Definition Only due to its
Interpretation Process.
def dispvals(a,b,c,d,e): # Function Def-1
 print(a,b,c,d,e)
def dispvals(a,b,c,d): # Function Def-2
 print(a,b,c,d)
def dispvals(a,b,c): # Function Def-3
 print(a,b,c)
def dispvals(a,b): # Function Def-4
 print(a,b)
def dispvals(a): # Function Def-5
 print(a)

#Main Program
dispvals(10,20,30,40,50) # Function Call-1 with 5 Pos Args
dispvals(10,20,30,40) # Function Call-2 with 4 Pos Args
dispvals(10,20,30) # Function Call-3 with 3 Pos Args
dispvals(10,20) # Function Call-4 with 2 Pos Args
dispvals(10) # Function Call-5 with 1 Pos Args

#Program for Demonstrating Variable Length Arguments (OR) Parameters
#VariableArgsLenEx2.py
#This Program will execute as it is
def dispvals(a,b,c,d,e): # Function Def-1
 print(a,b,c,d,e)
dispvals(10,20,30,40,50) # Function Call-1 with 5 Pos Args
#-----
def dispvals(a,b,c,d): # Function Def-2
 print(a,b,c,d)
dispvals(10,20,30,40) # Function Call-2 with 4 Pos Args
#-----
def dispvals(a,b,c): # Function Def-3
 print(a,b,c)
dispvals(10,20,30) # Function Call-3 with 3 Pos Args
#-----
def dispvals(a,b): # Function Def-4

```

```

print(a,b)

dispvals(10,20) # Function Call-4 with 2 Pos Args
#-----
def dispvals(a): # Function Def-5
 print(a)

dispvals(10) # Function Call-5 with 1 Pos Args
#-----
#Here we have 5 fun calls---->5 fun defs
#In general we have n-fun calls----->n-Fun Def Required--Waste of time
We are Expecting--- n-fun calls---Define One Function Def....
---Program for Demonstrating Variable Length Arguments (OR) Parameters
#PureVariableArgsLenEx1.py
def dispvals(*a):# here *a is called called Variable Length Parameters and whose type is tuple
 print("-"*50)
 for v in a:
 print("{}".format(v),end=" ")
 print()

#Main Program
dispvals(10,20,30,40,50) # Function Call-1 with 5 Pos Args
dispvals(10,20,30,40) # Function Call-2 with 4 Pos Args
dispvals(10,20,30) # Function Call-3 with 3 Pos Args
dispvals(10,20) # Function Call-4 with 2 Pos Args
dispvals(10) # Function Call-5 with 1 Pos Args
dispvals() # Function Call-6 with 0 Pos Args

#Program for Demonstrating Variable Length Arguments (OR) Parameters
#PureVariableArgsLenEx2.py
def dispvals(sno,sname, *a):# here *a is called called Variable Length Parameters and whose type is tuple
 print("-"*50)
 print("Student Number=",sno)
 print("Student Name=",sname)
 s=0
 for v in a:
 print("{}".format(v),end=" ")
 s=s+v
 print()
 print("Sum=",s)

#Main Program
dispvals(100,"RS",10,20,30,40,50) # Function Call-1 with 5 Pos Args
dispvals(200,"DR",10,20,30,40) # Function Call-2 with 4 Pos Args
dispvals(300,"TR",10,20,30) # Function Call-3 with 3 Pos Args
dispvals(400,"SS",10,20) # Function Call-4 with 2 Pos Args
dispvals(500,"KV",10) # Function Call-5 with 1 Pos Args
dispvals(600,"SQ") # Function Call-6 with 0 Pos Args

#Program for Demonstrating Variable Length Arguments (OR) Parameters
#PureVariableArgsLenEx3.py
def dispvals(sno,sname,*a, city="HYD"):# here *a is called called Variable Length Parameters and whose
type is tuple
 print("-"*50)
 print("Student Number=",sno)
 print("Student Name=",sname)
 print("Student Living City=",city)
 s=0
 for v in a:

```

```

 print("{}".format(v),end=" ")
 s=s+v
 print()
 print("Sum=",s)

#Main Program
dispvals(100,"RS",10,20,30,40,50) # Function Call-1 with 5 Pos Args
dispvals(200,"DR",10,20,30,40) # Function Call-2 with 4 Pos Args
dispvals(300,"TR",10,20,30) # Function Call-3 with 3 Pos Args
dispvals(400,"SS",10,20) # Function Call-4 with 2 Pos Args
dispvals(500,"KV",10) # Function Call-5 with 1 Pos Args
dispvals(600,"SQ") # Function Call-6 with 0 Pos Args
dispvals(600,"SQ",1.2,3.4,4.5,city="BANG") # Function Call-6 with 0 Pos Args
#dispvals(city="AP",sname="RA",sno=700,1.5,2.5,3.5)--SyntaxError: positional argument follows keyword
argument

```

---

### DAY-55 TUESDAY(21/05/2024)

#### 5) Key Word Variables Length Parameters (or) arguments

- =>When we have family of multiple function calls with Key Word Variable number of values / arguments then with normal python programming, we must define multiple function definitions. This process leads to more development time.
- =>To overcome this process, we must use the concept of Keyword Variable length Parameters .
- =>To Implement, Keyword Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called double astrisk ( \*\* param) and the formal parameter with double astrisk symbol is called Keyword Variable length Parameters and whose purpose is to hold / store any number of (Key,Value) coming from similar function calls and whose type is <class, 'dict'>.

Syntax for function definition with Keyword Variables Length Parameters:

```
def functionname(list of formal params, **param) :
```

- =>Here \*\*param is called Keyword Variable Length parameter and it can hold any number of Key word argument values (or) Keyword variable number of argument values and \*\*param type is <class,'dict'>
- =>Rule:- The \*\*param must always written at last part of Function Heading and it must be only one (but not multiple)

Final Syntax for defining a Function

```
def funcname(PosFormal parms, *Varlenparam, default params, **kwdvarlenparam):
```

```
#Program for Demonstrating Keyword Variable Length Arguments
```

```
#KeywordVarLenArgsEx1.py
```

#This Program will not execute as it is bcoz PVM remembers Latest Function Definition due to its Interpretation Process.

```
def dispvalues(sno,sname,marks): # Function Def-1
 print(sno,sname,marks)
```

```
def dispvalues(eno,ename,sal,cname): # Function Def-2
 print(eno,ename,sal,cname)
```

```

def dispvalues(sid,stname,hb1,hb2,hb3,hb4): # Function Def-3
 print(sid,stname,hb1,hb2,hb3,hb4)

def dispvalues(tno,tname,sub1,sub2,sub3): # Function Def-4
 print(tno,tname,sub1,sub2,sub3)

#Main Program
dispvalues(sno=10,sname="RS",marks=23.45)# Function Call-1 with 3 Keyword args
dispvalues(eno=20,ename="DR",sal=3.4,cname="TCS") # Function Call-2 with 4 Keyword args
dispvalues(sid=30,stname="RA",hb1="eating",hb2="sleeping",hb3="chatting",hb4="roaming") # Function
Call-3 with 6 Keyword args
dispvalues(tno=40,tname="KV",sub1="Python",sub2="Java",sub3="Numpy-Pandas-Matplotlib") # Function
Call-4 with 5 Keyword args

```

```

#Program for Demonstrating Keyword Variable Length Arguments
#KeywordVarLenArgsEx2.py
#This Program will execute as it is
def dispvalues(sno,sname,marks): # Function Def-1
 print(sno,sname,marks)
dispvalues(sno=10,sname="RS",marks=23.45)# Function Call-1 with 3 Keyword args
#-----
def dispvalues(eno,ename,sal,cname): # Function Def-2
 print(eno,ename,sal,cname)
dispvalues(eno=20,ename="DR",sal=3.4,cname="TCS") # Function Call-2 with 4 Keyword args
#-----
def dispvalues(sid,stname,hb1,hb2,hb3,hb4): # Function Def-3
 print(sid,stname,hb1,hb2,hb3,hb4)
dispvalues(sid=30,stname="RA",hb1="eating",hb2="sleeping",hb3="chatting",hb4="roaming") # Function
Call-3 with 6 Keyword args
#-----
def dispvalues(tno,tname,sub1,sub2,sub3): # Function Def-4
 print(tno,tname,sub1,sub2,sub3)
dispvalues(tno=40,tname="KV",sub1="Python",sub2="Java",sub3="Numpy-Pandas-Matplotlib") # Function
Call-4 with 5 Keyword args
#-----
```

#Here we have 4 Function Calls---having 4 Function defs  
#In General If we have n-Function Calls---u Must Define n-Function Definitions--waste of time

---

```

#Program for Demonstrating Keyword Variable Length Arguments
#PureKeywordVarLenArgsEx1.py
def dispvalues(**kvr):#here **param is Kwd Var length param whose type dict
 print(kvr,type(kvr))

#Main Program
dispvalues(sno=10,sname="RS",marks=23.45)# Function Call-1 with 3 Keyword args
dispvalues(eno=20,ename="DR",sal=3.4,cname="TCS") # Function Call-2 with 4 Keyword args
dispvalues(tno=40,tname="KV",sub1="Python",sub2="Java",sub3="Numpy-Pandas-Matplotlib") # Function
Call-4 with 5 Keyword args
dispvalues(sid=30,stname="RA",hb1="eating",hb2="sleeping",hb3="chatting",hb4="roaming") # Function
Call-3 with 6 Keyword args
dispvalues(CID=100,CNAME="TRAVIS")
dispvalues()
```

---

#Program for Demonstrating Keyword Variable Length Arguments  
#PureKeywordVarLenArgsEx2.py

```

def dispvalues(**kvr):#here **param is Kwd Var length param whose type dict
 print("-----")
 print("Number of Values=",len(kvr))
 for k,v in kvr.items():
 print("\t{}-->{}".format(k,v))
 print("-----")

#Main Program
dispvalues(sno=10,sname="RS",marks=23.45)# Function Call-1 with 3 Keyword args
dispvalues(eno=20,ename="DR",sal=3.4,cname="TCS") # Function Call-2 with 4 Keyword args
dispvalues(tno=40,tname="KV",sub1="Python",sub2="Java",sub3="Numpy-Pandas-Matplotlib") # Function
Call-4 with 5 Keyword args
dispvalues(sid=30,stname="RA",hb1="eating",hb2="sleeping",hb3="chatting",hb4="roaming") # Function
Call-3 with 6 Keyword args
dispvalues(CID=100,CNAME="TRAVIS")
dispvalues()

```

#Program for caculating Total Marks obtained by Different Student who are studying Different Class of Different Subjects.

#PureKeywordVarLenArgsEx3.py

```

def findtotalmarks(sno,sname,cls,**submarks):
 print("-----")
 print("Student Number={}".format(sno))
 print("Student Name={}".format(sname))
 print("Student Class={}".format(cls))
 if(len(submarks)!=0):
 print("-----")
 print("\tSubject\tMarks")
 print("-----")
 totmarks=0
 for subject,marks in submarks.items():
 print("\t{}\t{}".format(subject,marks))
 totmarks=totmarks+marks
 print("\tTOTAL MARKS={}".format(totmarks))

 print("=====")
```

#Main program

```

findtotalmarks(100,"Rajesh","X",Eng=70,Hindi=60,Telugu=65,Maths=98,Science=89,Social=88)
findtotalmarks(200,"Rakesh","XII",English=90,Sanskrit=99,Mathematics=75,Physics=60,Chemistry=57)
findtotalmarks(300,"Ramesh","B.Tech",OS=35,DBMS=30,NW=35,CLab=25)
findtotalmarks(400,"Rossum","Research")
findtotalmarks(cls="Scientist",sno=500,sname="Travis",numpy=50,pandas=70)

```

#Program for caculating Total Marks obtained by Different Student who are studying Different Class of Different Subjects.

#PureKeywordVarLenArgsEx4.py

```

def findtotalmarks(sno,sname,cls,city="HYD",**submarks):
 print("-----")
 print("Student Number={}".format(sno))
 print("Student Name={}".format(sname))
 print("Student Class={}".format(cls))
 print("Student Living City={}".format(city))
 if(len(submarks)!=0):
 print("-----")
 print("\tSubject\tMarks")
 print("-----")
 totmarks=0
 for subject,marks in submarks.items():
```

```

 print("\t{}\t{}\t{}".format(subject,marks))
 totmarks=totmarks+marks
 print("\tTOTAL MARKS={}".format(totmarks))

 print("====")
#Main program
findtotalmarks(100,"Rajesh","X",Eng=70,Hindi=60,Telugu=65,Maths=98,Science=89,Social=88)
findtotalmarks(200,"Rakesh","XII",English=90,Sanskrit=99,Mathematics=75,Physics=60,Chemistry=57)
findtotalmarks(300,"Ramesh","B.Tech",OS=35,DBMS=30,NW=35,CLab=25)
findtotalmarks(400,"Rossum","Research",city="NL")
findtotalmarks(cls="Scientist",sno=500,sname="Travis",numpy=50,pandas=70,city="UK")

#Program for caculating Total Marks obtained by Different Student who are studying Different Class of
Different Subjects.
#PureKeywordVarLenArgsEx5.py
def findtotalmarks(sno,sname,cls,*vals,city="HYD",**submarks):
 print("-----")
 print("Variable Number args={}".format(vals))
 print("-----")
 print("Student Number={}".format(sno))
 print("Student Name={}".format(sname))
 print("Student Class={}".format(cls))
 print("Student Living City={}".format(city))
 if(len(submarks)!=0):
 print("-----")
 print("\tSubject\t\tMarks")
 print("-----")
 totmarks=0
 for subject,marks in submarks.items():
 print("\t{}\t\t{}".format(subject,marks))
 totmarks=totmarks+marks
 print("\tTOTAL MARKS={}".format(totmarks))

 print("====")
#Main program
findtotalmarks(100,"Rajesh","X",10,20,30,40,Eng=70,Hindi=60,Telugu=65,Maths=98,Science=89,Social=88)
findtotalmarks(200,"Rakesh","XII",100,200,300,English=90,Sanskrit=99,Mathematics=75,Physics=60,Chemistry=57)
findtotalmarks(300,"Ramesh","B.Tech",1.2,2.3,4.5,6.7,7.8,OS=35,DBMS=30,NW=35,CLab=25)
findtotalmarks(400,"Rossum","Research",12,13,14,city="NL")
findtotalmarks(cls="Scientist",sno=500,sname="Travis",numpy=50,pandas=70,city="UK")

```

---

### DAY-56 22/05/2024 WEDNESDAY

---

=====

#### Local and Global Variables

=====

#### Local variables

=====

- =>Local variables are used in Function Body
  - =>The Purpose of Local Variables is that "To store Temporary Results after Function Processing".
  - =>Local Variables can be accessed inside of Corresponding function definition only but not possible to access other part of the program
- =====

#### Global Variables

=====

- =>Global Variables are those which are used for Providing Common Value for all Different Function Definitions.

=>Global Variables to be defined before all the function calls. So that we can access global variable values in all those

Function definitions.

=>Syntax:

```

 def fun1():

 def fun2():

 def fun-n():


```

#Main Program

Var1=Val1

Var2=val2

-----  
Var\_n=Val\_n # Here Var1,Var2,.....Var-n are called Global Variables

fun1() # Function Call-1

fun2() # Function Call-2

-----  
fun-n() # Function Call-n

Hence Var1,Var2,.....Var-n are the Global Variables defined before Function Definitions. so that We can access those values inside of Function Definitions.

---

### ===== 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

```

**NOTE:** To MODIFY Global variable Values inside of Function Definition, we use global Keyword (Mandatory to write)

**NOTE:** To ACCESS Global variable Values inside of Function Definition, we Don't use global Keyword

---

#Program for Demonstrating local and Global Variables

#LocalGlobalVarEx1.py

def learnAI():

```
sub1="AI" # Here sub1 is local variable
print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub1,lang))
#print(sub2,sub3)--can't access bcoz sub2 and sub3 are local variables in other Funs
def learnML():
 sub2="ML" # Here sub2 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub2,lang))
 # print(sub1,sub3)--can't access bcoz sub1 and sub3 are local variables in other Funs
def learnDL():
 sub3="DL" # Here sub3 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub3,lang))
 # print(sub1,sub2)--can't access bcoz sub1 and sub2 are local variables in other Funs
#Main Program
lang="PYTHON" # Here lang is called Global Variable
learnAI()
learnML()
learnDL()
```

---

```
#Program for Demonstrating local and Global Variables
#LocalGlobalVarEx2.py
def learnAI():
 sub1="AI" # Here sub1 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub1,lang))
 #print(sub2,sub3)--can't access bcoz sub2 and sub3 are local variables in other Funs
def learnML():
 sub2="ML" # Here sub2 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub2,lang))
 # print(sub1,sub3)--can't access bcoz sub1 and sub3 are local variables in other Funs
def learnDL():
 sub3="DL" # Here sub3 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub3,lang))
 # print(sub1,sub2)--can't access bcoz sub1 and sub2 are local variables in other Funs
#Main Program
#learnAI()---we can't access global variables lang in learnAI() bcoz it was defined after Its Function Call
lang="PYTHON" # Here lang is called Global Variable
learnML()
learnDL()
```

---

```
#Program for Demonstrating local and Global Variables
#LocalGlobalVarEx3.py
lang="PYTHON" # Here lang is called Global Variable
def learnAI():
 sub1="AI" # Here sub1 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub1,lang))
 #print(sub2,sub3)--can't access bcoz sub2 and sub3 are local variables in other Funs
def learnML():
 sub2="ML" # Here sub2 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub2,lang))
 # print(sub1,sub3)--can't access bcoz sub1 and sub3 are local variables in other Funs
def learnDL():
 sub3="DL" # Here sub3 is local variable
 print("To develop '{}' Applications, we use '{}' Prog Lang".format(sub3,lang))
 # print(sub1,sub2)--can't access bcoz sub1 and sub2 are local variables in other Funs
#Main Program
learnAI()
learnML()
learnDL()
```

---

```
#Program for Demonstrating global keyword
#GlbalKwdEx1.py
```

```

def incr():
 global a
 a=a+1

def update():
 global a
 a=a*2

#Main program
a=10 # here a is called global Variable
print("Val of a in Main Program before incr()={}".format(a)) # 10
incr() # Function call
print("Val of a in Main Program after incr()={}".format(a)) # 11
update()
print("Val of a in Main Program after update()={}".format(a)) # 22
-----\

#Program for Demonstrating global keyword
#GlbalKwdEx2.py
def modify1():
 global a,b
 a=a+1
 b=b+1
def modify2():
 global a,b
 a=a*2
 b=b*3

#Main program
a,b=10,20 # Here a,b are called Global Variables
print("In main Prog before modify1()--> a={} b={}".format(a,b)) # a=10 b=20
modify1()
print("In main Prog after modify1()--> a={} b={}".format(a,b)) # a=11 b=21
modify2()
print("In main Prog after modify2()--> a={} b={}".format(a,b)) # a=22 b=63

#Program for Demonstrating global keyword
#GlbalKwdEx3.py
def modify1():
 global a,b
 a=a+1
 b=b+1
def getvals():
 # There is no need to write global kwd bcoz are not modifying global variable vals but just we are
 # accessing
 c=a*2
 d=b*3 # here c and d are called local Variable
 print("Inside of getvals()--> c={} d={}".format(c,d))

#Main program
a,b=10,20 # Here a,b are called Global Variables
print("In main Prog before modify1()--> a={} b={}".format(a,b)) # a=10 b=20
modify1()
print("In main Prog after modify1()--> a={} b={}".format(a,b)) # a=11 b=21
getvals()
print("In main Prog after getvals()--> a={} b={}".format(a,b)) # a=11 b=21

```

## =====

## 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
Extarct the global variables values
dictobj=globals()

globalval1=dictobj['var1'] # or dictobj.get("var1") or globals()['var1'] or
global().get('var1')
globalval2=dictobj['var2'] # or dictobj.get("var2") or globals()['var2']
```

-----

-----

## =====

## comprehension

## =====

=>Python is famous for allowing you to write code that's elegant, easy to write, and almost as easy to read as plain English. One of the language's most distinctive features is the list comprehension, which you can use to create powerful functionality within a single line of code rather than writing legacy lines of Code.  
=>The purpose of List comprehension is that to read the values dynamically from key board separated by a delimiter ( space, comma, colon..etc)  
=>List comprehension is the most effective way for reading the data for list instead traditional reading the data.

=>Syntax:- `listobj=[ expression for varname in Iterable_object if test Cond ]`

=>here expression represents either type casting or mathematical expression

Examples:

```
print("Enter List of values separated by space:") # [10 2 22 50 10 4 55 -3 0 22]
lst= [float(val) for val in input().split()]
print("content of lst",lst)
```

-----

Examples:

```
lst=[4,3,7,-2,6,3]
newlst=[val*2 for val in lst]
print("new list=",newlst) # [8, 6, 14,-4,12,6]
```

-----

```
#Program for donstrating globals()
#globalsfunEx1.py
a=10
b=20
c=30
d=40 # here a,b,c,d are called global variables
def operation():
 x=100
```

```

y=200
z=300
k=400 # here x,y,z,k are called local variables
res=x+y+z+k+a+b+c+d
print("sum=",res)

#Main program
operation()

#Program for demonstrating globals()
#globalsfunEx2.py
a=10
b=20
c=30
d=40 # here a,b,c,d are called global variables
def operation():
 a=100
 b=200
 c=300
 d=400 # here a,b,c,d are called local variables
 res=a+b+c+d+globals()['a']+globals()['b']+globals()['c']+globals()['d']
 print("sum=",res)

#Main program
operation()

#Program for Demonstrating globals()
#globalsfunEx3.py
a=10
b=20 # here a,b are called global variables
def getglobalvals():
 a=100
 b=200 # here a,b are called local variables
 d=globals() # here d is an object of <class, dict>
 print("-----")
 print("Invisible and Programmer-Defined Global Variables")
 print("-----")
 for gvn,gvv in d.items():
 print("\t{}-->{}".format(gvn,gvv))
 print("-----")
 print("Programmer-Define Global Variables--Way-1")
 print("-----")
 print("Global Var-a=",d['a'])
 print("Global Var-b=",d['b'])
 print("-----")
 print("Programmer-Define Global Variables--Way-2")
 print("-----")
 print("Global Var-a=",d.get('a'))
 print("Global Var-b=",d.get('b'))
 print("-----")
 print("Programmer-Define Global Variables--Way-3")
 print("-----")
 print("Global Var-a=",globals().get('a'))
 print("Global Var-b=",globals().get('b'))
 print("-----")
 print("Programmer-Define Global Variables--Way-4")
 print("-----")
 print("Global Var-a=",globals()['a'])
 print("Global Var-b=",globals()['b'])

```

```

print("-----")
#Main program
getglobalvals() # Function Call

#Program for Reading the Values from KBD and find squares by using Dict Comprehension
#DictComprhenEx1.py
print("Enter List of Values separated by comma:")
d={int(val):int(val)**2 for val in input().split(",") } # here d is called dict type
print("-----")
print("\tNumber\tSquares")
print("-----")
for n,sn in d.items():
 print("\t{}---->{}".format(n,sn))
print("-----")
=====
#Program for Reading the +Values from KBD from Mixed Numerical values
#ListComprhenEx2.py
print("Enter List of Values separated by comma:") # 10 -20 -30 40 30 -15 23 -56
pslist=[float(val) for val in input().split(",") if float(val)>0]
print("Enter List of Values separated by comma:")
nslist=[float(val) for val in input().split(",") if float(val)<0]
print("List of +ve Values=",pslist)
print("List of -ve Values=",nslist)

#Program for Reading the words whose length is either 2 or 4 from list of words
#ListComprhenEx3.py
print("Enter Line of Text:")
word24=[str(word) for word in input().split() if len(word) in [2,4]]
print("Words with 2 or 4 in Length=",word24)
=====
#Program for Reading the Values from KBD by using Set Comprehension
#SetComprhenEx1.py
print("Enter List of Values separated by space:")
st= { int(val) for val in input().split()}
print("Content of st=",st) # here st is called set type
=====
#Program for Reading the Values from KBD by using Set Comprehension
#TupleComprhenEx1.py
print("Enter List of Values separated by space:")
x= (int(val) for val in input().split())
print("type of x=",type(x)) # here x is called --generator
print("Content of x=",x)
print("-----")
#Convert generator object into tuple
tpl=tuple(x)
print("Content of tpl=",tpl)
print("Type of tpl=",type(tpl))
=====
```

---

**DAY-58 24/05/2024 (FRIDAY)**

---

### Anonymous Functions OR Lambda Functions

---

=>Anonymous Functions are those which does not have any Name Function Name Explicitly.  
 =>The purpose of Anonymous Function is that "To Perform Instant Operations".

- =>Instant Operations are those which are used at that point of time only But no longer Interested to use in other part of the Application / Project
  - =>To define Anonymous Functions, we use 'lambda' keyword and hence Anonymous Functions are called Lambda Functions.
  - =>Since Anonymous Functions perform Instant Operations and They contains single executable statement only (But not Multiple).
  - =>Anonymous Functions returns the Result Automatically / Implicitly (There is no need to use return statement).
- 

### Syntax for Defining Anonymous Function

---

varname = lambda Params-List : Expression

---

### Explanation

---

- =>varname Represents an object of <class,'function'> and It can be treated as Indirectly as Function Call
  - =>lambda Represents a Keyword and It is used for Defining Anonymous OR Lambda Function
  - =>Params-List Represents List for Formal Parameters which are used for Holding OR Storing the Inputs Coming from Function calls.
  - =>Expression Represents Single Executable Statement and It is the Solution OR Logic for Instant Operations.
  - =>Anonymous Functions returns the Result Automatically / Implicitly by executing Single Executable Statement. There is no need to use return statement.
- 
- 

Question: Define a Function for adding Two Numbers

---

|                                                               |                                                                           |
|---------------------------------------------------------------|---------------------------------------------------------------------------|
| 1. By using Normal Functions<br>Functions                     | 1. By using Anonymous<br>Functions                                        |
| <pre>def addop(a,b):   c=a+b   return c</pre>                 | <pre>sumop = lambda a,b : a+b #</pre>                                     |
| #Main Program                                                 | #Main Program                                                             |
| <pre>res=addop(10,20) # Function Call print("Sum=",res)</pre> | <pre>res=sumop(100,200) # Anonymous Function Call print("Sum=",res)</pre> |

---



---

|                                                                         |  |
|-------------------------------------------------------------------------|--|
| #AnonymousFunEx1.py                                                     |  |
| def addop(a,b): # Normal Function                                       |  |
| c=a+b                                                                   |  |
| return c                                                                |  |
| sumop = lambda a,b : a+b # Anonymous Function                           |  |
| #Main program                                                           |  |
| print("Type of addop=",type(addop)) # type of addop= <class 'function'> |  |
| x=addop(10,20)                                                          |  |
| print("sum=",x)                                                         |  |
| print("-----")                                                          |  |
| print("Type of sumop=",type(sumop))                                     |  |
| res=sumop(200,300) #Anonymous Function Call                             |  |
| print("sum=",res)                                                       |  |

```
#Program for finding Biggest of Two Numbers
#AnonymousFunEx2.py
findmax=lambda a,b: a if a>b else b if b>a else "Both Vals are Equal"

#Main program
a,b=float(input("Enter First Value:")),float(input("Enter Second Value:"))
res=findmax(a,b)
print("Big({},{}]={}" .format(a,b,res))
```

```
#Program for Deciding whether Number is Palindrome or Not
#AnonymousFunEx3.py
palindrome=lambda x: "Palindrome" if x==x[::-1] else "Not Palindrome"
```

```
#Main program
n=input("Enter a Number / Word:")
res= palindrome(n) # Anonymous Fun Call
print("{} is {}".format(n,res))
```

```
#Program for Finding Max and Min of List of Numbers
#AnonymousFunEx4.py
findmax = lambda values: max(values)
findmin = lambda values: min(values)
```

```
#main program
print("Enter List of Values separated by space:")
vals=[int(val) for val in input().split()] # List Comprehension
print('Content of list=',vals)
bv=findmax(vals) # Anonymous Function Call
sv=findmin(vals) # Anonymous Function Call
print("Max({})={}" .format(vals,bv))
print("Min({})={}" .format(vals,sv))
```

```
#Program for Finding Max and Min of List of Numbers
#AnonymousFunEx5.py
#Normal functions
def kvrmax(values): # values=[10,2,20,12,34]
 maxv=values[0]
 for val in values:
 if(val>maxv):
 maxv=val
 return maxv
def kvrmin(values): # values=[10,2,20,12,34]
 minv=values[0]
 for val in values:
 if(val<minv):
 minv=val
 return minv
```

```
#Anonymous Functions
findmax = lambda values: kvrmax(values)
findmin = lambda values: kvrmin(values)
```

```
#main program
#main program
print("Enter List of Values separated by space:")
vals=[int(val) for val in input().split()] # List Comprehension
print('Content of list=',vals)
```

```
bv=findmax(vals) # Anonymous Function Call
sv=findmin(vals) # Anonymous Function Call
print("Max({})={}".format(vals,bv))
print("Min({})={}".format(vals,sv))
```

---

## DAY-59 25/05/2024 (SATURDAY)

---

### Special Functions 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 +Ve and -Ve Values from List of Values

```
#FilterEx1.py
def pos(val):
 if(val>0):
 return True
 else:
 return False
def neg(val):
 if(val<0):
 return True
 else:
 return False
```

#Main Program

```
lst=[10,-20,30,0,-40,-50,60,70,-80]
ps=filter(pos,lst)
ns=filter(neg,lst)
print("Type of ps=",type(ps)) #<class, filter>
print("Type of ns=",type(ns)) #<class, filter>
#Convert Filter object ps into List object
pslst=list(ps)
#Convert Filter object ns into tuple object
nslst=tuple(ns)
print("Given List:{}".format(lst))
print("Positive List:{}".format(pslst))
print("Negative List:{}".format(nslst))
```

---

#Program for Filtering +Ve and -Ve Values from List of Values

```
#FilterEx2.py
pos=lambda val: val>0 # Anonymous function Def-1
```

```
neg=lambda val: val<0 # Anonymous function Def-2
#Main Program
lst=[10,-20,30,0,-40,-50,60,70,-80]
ps=list(filter(pos,lst))
ns=tuple(filter(neg,lst))
print("Given List:{}".format(lst))
print("Positive List:{}".format(ps))
print("Negative List:{}".format(ns))
```

---

```
#Program for Filtering +Ve and -Ve Values from List of Values
#FilterEx3.py
print("Enter List of Values Separated by Space:")
lst=[int(val) for val in input().split()]
print("Content of List=",lst)
#Filter the +ve and -ve Values from lst
ps=list(filter(lambda val : val>0,lst))
ns=list(filter(lambda val: val<0,lst))
print("List of +VE Values={}".format(ps))
print("List of -VE Values={}".format(ns))
```

---

```
#Program for Filtering +Ve Even Numbers from List of Numerical Values
#FilterEx4.py
print("Enter List of Values Separated by Space:")
lst=[int(val) for val in input().split()]
print("Content of List=",lst)
psevenlst=list(filter(lambda val: val>0 and val%2==0, lst))
nsevenlst=tuple(filter(lambda val: val<0 and val%2==0,lst))
print("List of +ve Even Numbers={}".format(psevenlst))
print("List of -ve Even Numbers={}".format(nsevenlst))
```

---

```
#Program for Obtaining alphabets, symbols and digits
#FilterEx5.py
value = "P2yt$1H4#o*3N" # Here Value of <class,str> and is Iterabel object
alphas=list(filter(lambda ch:ch.isalpha() , value))
upperalphas=list(filter(lambda ch:ch.isalpha() and ch.isupper() , value))
loweralphas=list(filter(lambda ch:ch.isalpha() and ch.islower() , value))
digits=tuple(filter(lambda ch: ch.isdigit(), value))
symbols=list(filter(lambda ch: not ch.isalnum(),value))
print("-----")
print("Given Data=",value)
print("Alphabets={}".format("".join(alphas)))
print("\t\tUpper Alphabets={}".format("".join(upperalphas)))
print("\t\tLower Alphabets={}".format("".join(loweralphas)))
print("Digits={}".format("".join(digits)))
print("Special Symbols={}".format("".join(symbols)))
print("-----")
```

---

```
#Program for Obtaining alphabets, symbols and digits
#FilterEx5.py
value = input("Enter a word with Mixed Values:") # Here Value of <class,str> and is Iterabel object
alphas=list(filter(lambda ch:ch.isalpha() , value))
upperalphas=list(filter(lambda ch:ch.isalpha() and ch.isupper() , value))
loweralphas=list(filter(lambda ch:ch.isalpha() and ch.islower() , value))
digits=tuple(filter(lambda ch: ch.isdigit(), value))
symbols=list(filter(lambda ch: not ch.isalnum(),value))
print("-----")
print("Given Data=",value)
print("Alphabets={}".format("".join(alphas)))
```

```
print("\t\tUpper Alphabets={}".format("".join(upperalphas)))
print("\t\tLower Alphabets={}".format("".join(loweralphas)))
print("Digits={}".format("".join(digits)))
print("Special Symbols={}".format("".join(symbols)))
print("-----")
```

---

## DAY-60 27/05/2024 (MONDAY)

---

### 2) map()

=>map() is used for obtaining new Iterable object from existing iterable object by applying old iterable elements to the function.  
=>In otherwords, map() is used for obtaining new list of elements from existing list of elements by applying old list elements to the function.

=>Syntax:- varname=map(functionName,Iterable\_object)

=>here 'varname' is an object of type <class,map> and we can convert into any iteratable object by using type casting functions.

=>"functionName" represents either Normal function or anonymous functions.

=>"Iterable\_object" represents Sequence, List, set and dict types.

=>The execution process of map() is that " map() sends every element of iterable object to the specified function, process it and returns the modified value (result) and new list of elements will be obtained". This process will be continued until all elements of Iterable\_object completed.

---

#Program for Reading Old List of Salaries and Obtains New List of Salaries by giving 20% Incrment

```
#MapFunEx1.py
def incrment(sal):
 sal=sal+sal*(20/100)
 return sal
#Main Program
print("Enter List of Old Salaries:")
oldsals=[float(sal) for sal in input().split()]
nsl=map(incrment,oldsals) # here nsl is called an object of <class,map>
#Convert map object into list
newsal=list(nsl)
print("*50)
print("Old Slary List\t\tNew Salary List")
print("*50)
for osl,newsl in zip(oldsals,newsal):
 print("\t{}\t\t\t{}".format(osl,newsl))
print("*50)
```

---

#Program for Reading Old List of Salaries and Obtains New List of Salaries by giving 20% Incrment

```
#MapFunEx2.py
incrment=lambda sal:sal+sal*20/100 # Anonymous Function
#Main Program
print("Enter List of Old Salaries:")
oldsals=[float(sal) for sal in input().split()]
nsl=map(incrment,oldsals) # here nsl is called an object of <class,map>
#Convert map object into list
newsal=list(nsl)
print("*50)
print("Old Slary List\t\tNew Salary List")
print("*50)
for osl,newsl in zip(oldsals,newsal):
 print("\t{}\t\t\t{}".format(osl,newsl))
print("*50)
```

---

```
#Program for Reading Old List of Salaries and Obtains New List of Salaries by giving 20% Increment
#MapFunEx3.py
print("Enter List of Old Salaries:")
oldsals=[float(sal) for sal in input().split()]
newsal=list(map(lambda sal:sal+sal*20/100,oldsals)) # here nsal is called an object of <class, map>
print("*50)
print("Old Slary List\t\tNew Salary List")
print("*50)
for osal,newsal in zip(oldsals,newsal):
 print("\t{}\t\t\t{}".format(osal,newsal))
print("*50)
```

#Program for Reading Old List of values and Obtains squares and square roots

```
#MapFunEx4.py
print("Enter List of Numerical Values:")
values=[float(val) for val in input().split()]
#Compute Square Roots
sqroots=list(map(lambda val:val**0.5,values))
#Compute Squares
squares=list(map(lambda val:val**2,values))
print("*50")
print("Original Value\t\tSquare Roots")
print("*50")
for on,sqt in zip(values,sqroots):
 print("\t{}\t\t{}".format(on,sqt))
print("*50")
print("Original Value\t\tSquares")
print("*50")
for on,sq in zip(values,squares):
 print("\t{}\t\t\t{}".format(on,sq))
print("*50")
```

```
#MapFunEx5.py
sallist=[100,200,300,400]
commelist=[10,20,30,40]
totalsal=list(map(lambda sal,comm:sal+comm, sallist,commelist))
print("=*50)
print("Salary\t\tCommision\t\tTotal Salary")
print("=*50)
for sal,comm,tosal in zip(sallist,commelist,totsal):
 print("\t{}\t\t{}\t\t{}".format(sal,comm,tosal))
print("=*50)
```

```
#MapFunEx6.py
print("Enter List of Salaries separated by space: ")
sallist=[float(sal) for sal in input().split()]
print("Enter List of Commision Values separated by space:")
commlist=[float(comm) for comm in input().split()]
#print("Sal List=",sallist) # [100,200,300,400]
#print("Comm List=",commlist) [10,20]
if(len(sallist)>len(commlist)):
 for i in range(len(sallist)-len(commlist)):
 commlist.append(0)
elif(len(commlist)>len(sallist)):
 for i in range(len(commlist)-len(sallist)):
 sallist.append(0)
#Compute Total Salary
totalsal=list(map(lambda sal,comm:sal+comm,sallist,commlist))
print("=*50")
```

```

print("Salary\tCommision\tTotal Salary")
print("*50)
for sal,comm,tSal in zip(salList,commList,totSal):
 print("\t{}\t{}\t{}".format(sal,comm,tSal))
print("*50)

#MapFunEx7.py
Animal=["Cat","Dog","Human","Tiger"]
Sound=["Mew","Bow","aww","roar"]
animal_sound=tuple(map(lambda ani,snd:ani+'-'+snd, Animal,Sound))
print("*50)
print("Animal\tSound\tResult")
print("*50)
for an,sn,ansn in zip(Animal,Sound,animal_sound):
 print("\t{}\t{}\t{}".format(an,sn,ansn))
print("*50)

```

---

## DAY-61 28/05/2024 (TUESDAY)

---

=====

### 3. reduce()

=====

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

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

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

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

-----  
Internal Flow of reduce()

Step-1:- Initially, reduce() selects First Two values of Iterable object and place them in First var and Second var .

Step-2:- The function-name(lambda or normal function) utilizes the values of First var and Second var and applied to the specified logic and obtains the result.

Step-3:- reduce () places the result of function-name in First variable and reduce()  
selects the succeeding element of Iterable object and places in second variable.

Step-4: Repeat Step-2 and Step-3 until all elements completed in  
Iterable object and returns the result of First Variable.

-----  
#Program for accepting List of Values find their sum by using reduce()  
#ReduceEx1.py  
import functools as fc  
print("Enter List of Values Separated by Space:")  
lst=[float(val) for val in input().split()]  
res=fc.reduce(lambda k,v:k+v, lst )  
print("Sum=",res)

-----  
#Program for accepting List of Values find their sum by using reduce()  
#ReduceEx2.py  
import functools as fc  
def sumop(k,v):  
 return (k+v)  
#main program  
print("Enter List of Values Separated by Space:")  
lst=[float(val) for val in input().split()]  
res=fc.reduce(sumop, lst )  
print("Sum=",res)

```
#Program for accepting List of Values find max value by using reduce()
#ReduceEx3.py
import functools
print("Enter List of Values Separated by Space:")
lst=[float(val) for val in input().split()] # [10,20,4,25,24,30,5]
maxv=functools.reduce(lambda k,v: k if k>v else v, lst)
print("Max({})={}".format(lst,maxv))

```

```
#Program for accepting List of Values find max value by using reduce()
#ReduceEx4.py
import functools
def findmax(k,v):
 if k>v:
 return k
 else:
 return v

print("Enter List of Values Separated by Space:")
lst=[float(val) for val in input().split()] # [10,20,4,25,24,30,5]
maxv=functools.reduce(findmax, lst)
print("Max({})={}".format(lst,maxv))

```

```
#Program for accepting List of Values find max value by using reduce()
#ReduceEx5.py
import functools
def findmax(k,v):
 return(k if k>v else v)
#main Program
print("Enter List of Values Separated by Space:")
lst=[float(val) for val in input().split()] # [10,20,4,25,24,30,5]
maxv=functools.reduce(findmax, lst)
print("Max({})={}".format(lst,maxv))

```

```
#Program for accepting List of words obtain a single line of text by using reduce()
#ReduceEx6.py
import functools
print("Enter List of words Separated by comma:")
lst=[str(val) for val in input().split(",")] # ["Python","is","Invented","By","Rossum"]
res=functools.reduce(lambda k,v :k+ " "+v,lst)
print("Line of Text")
print(res)

```

```
#Program for accepting List of Numerical Values obtain sum of +ve and -Ve values
#ReduceEx7.py
import functools
print("Enter List of words Separated by comma:")
lst=[float(val) for val in input().split(",")] # [10,-20,30,-34.5,-56.7,35,0]
posvals=list(filter(lambda val: val>0,lst))
negvals=list(filter(lambda val: val<0,lst))
psum=functools.reduce(lambda k,v: k+v,posvals)
nsum=functools.reduce(lambda k,v: k+v,negvals)
print("PosSum({})={}".format(posvals,psum))
print("NegSum({})={}".format(negvals,nsum))

```

```
#FilterMapReduceEx.py
import functools
print("Enter List of Salaries separated by space:")
```

```

sals=[float(sal) for sal in input().split() if 0<=float(sal)<=1000]
#Obtain salaries ranges from 0 to 501
sal0_500=list(filter(lambda sal: 0<=sal<=500,sals))
#Obtain salaries ranges from 0 to 501
sal501_1000=list(filter(lambda sal: 501<=sal<=1000, sals))
#Give 10% Hike to those employee whose sal ranges from 0 to 500
hksal0_500=list(map(lambda sal:sal+sal*10/100,sal0_500))
#Give 20% Hike to those employee whose sal ranges from 501 to 1000
hksal501_1000=list(map(lambda sal:sal+sal*20/100,sal501_1000))
#Get the Total of those employee whose sal ranges from 0 to 500 before and after hike
totalsal0_500=functools.reduce(lambda sal1,sal2:sal1+sal2,sal0_500)
hktotsal0_500=functools.reduce(lambda sal1,sal2:sal1+sal2,hksal0_500)
#Get the Total of those employee whose sal ranges from 501 to 1000 before and after hike
totalsal501_1000=functools.reduce(lambda sal1,sal2:sal1+sal2,sal501_1000)
hktotsal501_1000=functools.reduce(lambda sal1,sal2:sal1+sal2,hksal501_1000)
print("*50")
print("Sal0_500\t\tHiked Sal0-500")
print("*50)
for osl,nsl in zip(sal0_500,hksal0_500):
 print("\t{}\t\t{}".format(osl,nsl))
print("-----")
print("\t{}\t\t{}".format(totalsal0_500,hktotsal0_500))
print("*50)
print("Sal501_1000\t\tHiked Sal501_1000")
print("*50)
for osl,nsl in zip(sal501_1000,hksal501_1000):
 print("\t{}\t\t{}".format(osl,nsl))
print("-----")
print("\t{}\t\t{}".format(totalsal501_1000,hktotsal501_1000))
print("*50)
gtotsal0_1000=totalsal0_500+totalsal501_1000
ghktotsal0_1000=hktotsal0_500+hktotsal501_1000
print("Grand Total paid by Company Before Hike:{}".format(gtotsal0_1000))
print("Grand Total paid by Company after Hike:{}".format(ghktotsal0_1000))
print("*50)

```

## DAY-63 29/05/2024 (WEDNESDAY)

### Modules in Python

#### Index

=>Purpose of Modules

=>Definition of Module

=>Types of Modules

- a) Pre-Defined OR Built-In Modules
- b) Programmer OR User OR Custom Defined Modules

=>Examples

=>Steps for Developing Programmer OR User OR Custom Defined Modules

=>Programming Examples

=>What \_\_pycache\_\_ Folder

=>Number of Approaches for Re-Using Modules

- a) By using import statement

Syntax-1  
Syntax-2  
Syntax-3  
Syntax-4

- b) By using "from...import statement"

Syntax-1  
Syntax-2

### Syntax-3

=>Programming Examples  
=>Re-Loading Modules  
=>Implementation of Re-Loading Modules  
    a) By using imp Module (What is Deprecation)  
    b) By using importlib Module  
=>Programming Examples  
=>Case Studies

---

## 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 contains Global Var Names, function Names and Class names 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-312.pyc ".

Examples:

\_\_pycache\_\_ <----Folder Name

Aop.cpython-312.pyc <-----Module Name

mathsinfo.cpython-312.pyc<-----Module Name  
icici.cpython-312.pyc<-----Module Name

### Number of approaches to re-use Modules

=>We know that A Module is a collection of variables, Functions and Classes.

=>To re-use the features(Variable Names, Function Names and Class Names ) of module, we have 2 approaches.They are

- 1) By using import statement
- 2) By using from.... import statement.

#### 1) By using import statement:

=>'import' is a keyword

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

=>we can use import statement in 4 ways.

=>Syntax-1: import module name

=>This syntax imports single module

Example: import icici  
              import aop  
                        import mathsinfo

=>Syntax-2: import module name1, module name2....Module name-n

=>This syntax imports multiple modules

Example: import icici , aop, mathsinfo

=>Syntax-3: import module name as alias name

=>This syntax imports single module and aliased with another unique names

Example: import icici as i  
              import aop as a  
                        import mathsinfo as m

=>Syntax-4: import module name1 as alias name, module name2 as alias name.....module name-n as alias name

=>This syntax imports multiple modules and aliased with another unique names

Example: import icici as i, aop as a , mathsinfo as m

=>Hence after importing module name(s) by using "import statement", all the variable names, Function names and class names must access variable names, Function names and class names w.r.t Module Names or alias names.

Module Name.Variable Name  
Module Name.Function Name  
Module Name.Class Name  
(OR)  
Alias Name.Variable Name  
Alias Name.Function Name  
Alias Name.Class Name

#### 2) By using from.... import statement.

=>Here "from" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names of any modules in the context of current program directly without writing module name OR alias name of Module name."

=> we can use from.... import statement in 3 ways.

Syntax-1: from module name import Variable Names,Function Names, Class Names

=>This syntax imports the Variable Names,Function Names, Class Names of a module.

Example: from calendar import month  
from aop import addop,subop  
from mathinfo import pi,e  
from icici import bname,addr, simpleint

Syntax-2: from module name import Variable Names as alias name,Function Names as alias name , Class Names as alias names.

=>This syntax imports the Variable Names,Function Names, Class Names of a module with Unique alias Names

Example: from calendar import month as m  
from aop import addop as a,subop as s, mulop as m  
from mathinfo import pi as p ,e as k  
from icici import bname as b,addr as n , simpleint as si

Syntax-3: from module name import \*

=>This syntax imports ALL Variable Names,Function Names, Class Names of a module.

=>This syntax is not recommended to use bcoz it imports required Features of Module and also import uninterested features also imported and leads more main memory space.

Example: from calendar import \*  
from aop import \*  
from mathsinfo import \*

=>Hence after importing all the variable names, Function names and class names by using "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.

Variable Name  
Function Name  
Class Name

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

=====X=====

#Aop.py<---File Name and Acts as Module name

```
def sumop(a,b):
 print("sum({},{})={}".format(a,b,a+b))
```

```
def subop(a,b):
 print("sub({},{})={}".format(a,b,a-b))
```

```
def mulop(a,b):
 print("mul({},{})={}".format(a,b,a*b))
```

#icici.py<---File Name and Module Name

bname="ICICI"

addr="HYDERABAD" # here bname and addr are called Global variables

```
def simpleint(): # Function Def....
```

```
 p = float(input("Enter principle Amount:"))
 t = float(input("Enter Time:"))
```

```
r = float(input("Enter Rate of Interest:"))
cal si and totamt
si = (p * t * r) / 100
totamt = p + si
display the Result
print("*" * 50)
print("\tResults of Simple Interest")
print("*" * 50)
print("\tPrinciple Amount:{}\n".format(p))
print("\tTime:{}\n".format(t))
print("\tRate of Interest:{}\n".format(r))
print("\tSimple Interest:{}\n".format(si))
print("\tTOTAL AMOUNT TO PAY:{}\n".format(totamt))
print("*" * 50)
```

---

```
#MathsInfo.py<----File Name acts as Module Name
PI=3.14
E=2.71 # here PI and E are called Global Variables
```

---

```
#SE1.py---I will come back
import MathsInfo
print("Val of PI=",MathsInfo.PI)
print("Val of E=",MathsInfo.E)
```

---

```
#SE2.py
import Aop
Aop.sumop(100,200) # function call
Aop.subop(30,40) # Function call
Aop.mulop(4,5) # Function call
```

---

```
#SE3.py
import icici
import Aop
print("Bank Name:{}\n".format(icici.bname))
print("Bank Address:{}\n".format(icici.addr))
icici.simpleint() #Function Call
print("-----")
Aop.sumop(10,20)
```

---

```
#ImportStmtSyntax-1.py
import icici
import Aop
print("Bank Name:{}\n".format(icici.bname))
print("Bank Address:{}\n".format(icici.addr))
icici.simpleint() #Function Call
print("-----")
Aop.sumop(10,20)
```

---

```
#ImportStmtSyntax-2.py
import icici,Aop
print("Bank Name:{}\n".format(icici.bname))
print("Bank Address:{}\n".format(icici.addr))
icici.simpleint() #Function Call
print("-----")
Aop.sumop(10,20)
```

---

```
#ImportStmtSyntax-3.py
import icici as ic
import Aop as ap
print("Bank Name:{}".format(ic.bname))
print("Bank Address:{}".format(ic.addr))
ic.simpleint() #Function Call
print("-----")
ap.sumop(10,20)
```

---

```
#ImportStmtSyntax-2.py
import icici as i,Aop as a
print("Bank Name:{}".format(i.bname))
print("Bank Address:{}".format(i.addr))
i.simpleint() #Function Call
print("-----")
a.sumop(10,20)
```

---

## DAY-64 30/05/2024 THURSDAY

---

### Number of approaches to re-use Modules

---

=>We know that A Module is a collection of variables, Functions and Classes.  
=>To re-use the features(Variable Names, Function Names and Class Names ) of module, we have 2 approaches.They are

- 1) By using import statement
  - 2) By using from.... import statement.
- 

#### 1) By using import statement:

---

=>'import' is a keyword  
=>The purpose of import statement is that "To refer or access the variable names, function names and class names of module in the context of current program"  
=>we can use import statement in 4 ways.

---

=>Syntax-1:            import module name

---

=>This syntax imports single module

---

Example:            import icici  
                      import aop  
                      import mathsinfo

---

=>Syntax-2:        import module name1, module name2....Module name-n

---

=>This syntax imports multiple modules

---

Example:            import icici , aop, mathsinfo

---

=>Syntax-3:        import module name as alias name

---

=>This syntax imports single module and aliased with another unique names

---

Example:            import icici as i  
                      import aop as a  
                      import mathsinfo as m

---

=>Syntax-4: import module name1 as alias name, module name2 as alias name.....module name-n as alias name

=>This syntax imports multiple modules and aliased with another unique names

Example: import icici as i, aop as a , mathsinfo as m

=>Hence after importing module name(s) by using "import statement", all the variable names, Function names and class names must access variable names, Function names and class names w.r.t Module Names or alias names.

Module Name.Variable Name  
Module Name.Function Name  
Module Name.Class Name  
(OR)  
Alias Name.Variable Name  
Alias Name.Function Name  
Alias Name.Class Name

## 2) By using from.... import statement.

=>Here "from" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names of any modules in the context of current program directly without writing module name OR alias name of Module name."

=> we can use from.... import statement in 3 ways.

Syntax-1: from module name import Variable Names,Function Names, Class Names

=>This syntax imports the Variable Names,Function Names, Class Names of a module.

Example: from calendar import month  
from aop import addop,subop  
from mathinfo import pi,e  
from icici import bname,addr, simpleint

Syntax-2: from module name import Variable Names as alias name,Function Names as alias name ,  
Class Names as alias names.

=>This syntax imports the Variable Names,Function Names, Class Names of a module with Unique alias Names

Example: from calendar import month as m  
from aop import addop as a,subop as s, mulop as mp  
from mathinfo import pi as p ,e as k  
from icici import bname as b,addr as n , simpleint as si

Syntax-3: from module name import \*

=>This syntax imports ALL Variable Names,Function Names, Class Names of a module.

=>This syntax is not recommended to use bcoz it imports required Features of Module and also import un-interested features also imported and leads more main memory space.

Example: from calendar import \*  
from aop import \*  
from mathsinfo import \*

=>Hence after importing all the variable names, Function names and class names by using "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.

Variable Name  
Function Name  
Class Name

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

=====X=====

### reloading a modules in Python

=>To reload a module in python , we use a pre-defined function called reload(), which is present in importlib module.

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

=>Purpose / Situation:

=>reload() reloads a previously imported module.

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

=====X=====

#shares.py---file and treated as module name

```
def sharesinfo():
 d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}
 return d
```

#main program

#sharesdemo.py

import shares

import time

import importlib

def disp(d):

```
 print("-"*50)
 print("\tShare Name\tValue")
 print("-"*50)
 for sn,sv in d.items():
 print("\t{}\t{}".format(sn,sv))
 else:
 print("-"*50)
```

#main program

d=shares.sharesinfo()

disp(d)

time.sleep(15)

importlib.reload(shares) # reloading previously imported module

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

disp(d)

#Faculty12.py

import Faculty1

import Faculty2

print(Faculty1.name,Faculty1.sub)

print(Faculty2.name,Faculty2.sub)

Faculty1.teach()

Faculty2.teach()

```
#Faculty1.py<---File Name and Module Name
name="KVR"
sub="PYTHON"
def teach():
 print("{} Teaches {}".format(name,sub))

#Faculty2.py<---File Name and Module Name
name="KV"
sub="JAVA"
def teach():
 print("{} Teaches {}".format(name,sub))

#Faculty12.py
from Faculty1 import name as n1,sub as s1,teach as t1
from Faculty2 import name as n2,sub as s2,teach as t2
print(n1,s1)
t1()
print("-----")
print(n2,s2)
t2()

#FromImportStmtSyntax-1.py
from icici import bname,addr,simpleint
print("Bank Name:{}".format(bname))
print("Bank Address:{}".format(addr))
simpleint() #Function Call

#FromImportStmtSyntax-2.py
from icici import bname as bn, addr as ad,simpleint as si
print("Bank Name:{}".format(bn))
print("Bank Address:{}".format(ad))
si() #Function Call

#FromImportStmtSyntax-3.py
from icici import *
from Aop import *
print("Bank Name:{}".format(bname))
print("Bank Address:{}".format(addr))
simpleint() #Function Call
sumop(10,20)
subop(30,40)
mulop(4,5)

#Shares.py<----File Name and Module Name
def shareinfo():
 d={"IT":11,"FIN":12,"Auto":31,"Pharma":11222}
 return d

import Shares
import time,importlib
def dispshares(d):
 print("-----")
 print("ShareName\tShareValue")
 print("-----")
 for sn,sv in d.items():
 print("\t{}\t{}".format(sn,sv))
```

```
print("-----")
#Main Program
d=Shares.shareinfo() # Function Call
dispshares(d)
print("I am going to sleep for 10 Secs")
time.sleep(20)
print("I am coming out-off sleep after 10 Secs")
#Reloading the Previuosly imported module- we use reload()--importlib(deprecated from imp module)
importlib.reload(Shares)
d=Shares.shareinfo() # Function Call
dispshares(d)
print("I am going to sleep for 10 Secs")
time.sleep(20)
print("I am coming out-off sleep after 10 Secs")
#Reloading the Previuosly imported module- we use reload()--importlib(deprecated from imp module)
importlib.reload(Shares)
d=Shares.shareinfo() # Function Call
dispshares(d)

```

```
#AopDemo.py<---Main Program
from menu import menu
import sys
from Operations import addop,subop,mulop,divop,modop,expop
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 program")
 sys.exit()
 case _:
 print("Ur Selection of Operation is wrong-try again")

```

```
#menu.py<--File Name and Module Name
def menu():
 print("*50)
 print("Arithmetric Operations")
 print("= * 50)
 print("\t1.Addition")
 print("\t2.Substration")
 print("\t3.Multiplication")
 print("\t4.Division")
 print("\t5.Modulo Division")
 print("\t6. Exponentiation")
 print("\t7. Exit")
 print("= * 50)

```

```
#Operations.py<--File Name and Module Name
def addop():
 print("Enter Two Values for addition:")
 a,b=float(input()),float(input())
 print("Sum({},{})={}".format(a,b,a+b))
def subop():

```

```

print("Enter Two Values for Subtraction:")
a, b = float(input()), float(input())
print("Sub({},{})={}".format(a, b, a - b))
def mulop():
 print("Enter Two Values for Multiplication:")
 a, b = float(input()), float(input())
 print("Mul({},{})={}".format(a, b, a * b))
def divop():
 print("Enter Two Values for Division:")
 a, b = float(input()), float(input())
 print("Div({},{})={}".format(a, b, a / b))
 print("Floor_Div({},{})={}".format(a, b, a // b))
def modop():
 print("Enter Two Values for Mod Div:")
 a, b = float(input()), float(input())
 print("Mod({},{})={}".format(a, b, a % b))
def expop():
 a, b = float(input("Enter Base:")), float(input("Enter Power:"))
 print("Pow({},{})={}".format(a, b, a ** b))

```

---

## DAY-65 31/05/2024 (Friday)

---

### Difference between Function, Module and Package

---

#### **Function**

---

- =>A Function is a Sub Program, which is used to perform Certain Operation and Provides Code Re-Usability
  - =>A Function related Code can be accessed within Same Program But Not Possible to Access Across the Program
- 

#### **Module:**

---

- =>A Module is a Collection Global Variables , Function Names and Class Names
  - =>The purpose of Module is that to Re-Use the Global Variables , Function Names and Class Names of Module either within the Same or Across the Programs provided Module and Its Corresponding Main Program Must Present in Same Folder But Not Possible Access across the Folders OR Environments OR Networks.
- 

#### **Package**

---

- =>A Package is a Collection Modules .
  - =>The Purpose of Package is that to Re-Use Modules(Global Variables , Function Names and Class Names) either within the Same Folder OR Different Folder OR Environments OR Networks.
- 

NOTE: The concepts of Functions, Modules and Packages are called Re-Usable Techniques in Functional Programming.

---

### **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 the 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
 icici1.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.path.append()`
- ii) by using `PYTHONPATH` Environmental Variable Name

---

i) By using `sys.path.append()`

---

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-11am\\PACKAGES\\BANK")
 (or)
sys.path.append("D:\\KVR-PYTHON-11am\\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

---

### AOP FOLDER:

#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))
```

---

3 folders

1.AOP=>C:\\Users\\prashanth\\Downloads\\AOP

2.BANK=>C:\\Users\\prashanth\\Downloads\\BNAK\\BNAK

3.P3=>C:\\Users\\prashanth\\Downloads\\P3\\P3

---

**DAY-66 1/06/2024 (SATURDAY)**

---

### Decorators in Python

---

=>A Decorator is one of the Function which will provide Additional Functionality to the Normal Function.

=>A Decorator always takes Normal Function as Parameter.

---

=>Syntax for Defining Decorator

---

```
def Function-Name1(Formal Parameter for Normal Function):
 def Function_Name2():
```

---

```


Block of Statements
provides Addl. Functioality to Normal Function


```

---

=>Here Function-Name1 is called Decorator

=>Here Function-Name2 is called Inner Function.

---

#Program for Demonstrating Decorators

#DecEx1.py

```
def getval():
```

```
 return float(input("Enter a Number:"))
```

def square(kvr): # Here square is called Decorator and operation is called inner Function

```
 def operation():
```

```
 n=kvr()
```

```
 res=n**2
```

```
 return res
```

```
 return operation
```

```
#Main Program
op=square(getval) # Here Function call is Taking another function and It is called Decorator Function
result=op()
print("Result=",result)

#Program for Demonstrating Decorators
#DecEx2.py
def getval():
 return float(input("Enter a Number:"))

def square(kvr): # Here square is called Decorator
 def operation(): # Here operation is called inner Function
 n=kvr()
 res=n**2
 return n,res
 return operation

#Main Program
op=square(getval) # Here Function call is Taking another function and It is called Decorator Function
n,sqres=op()
print("square({})={}".format(n,sqres))

```

```
#Program for Demonstrating Decorators
#DecEx3.py
def square(gv):
 def calculation():
 n=gv()
 res=n**2
 return n,res
 return calculation

@square
def getval():
 return float(input("Enter a Number:"))

#Main Program
x,y=getval() # Normal Function call
print("square({})={}".format(x,y))

```

```
#Program for Demonstrating Decorators
#DecEx3.py
def cube(kvr):
 def operation():
 n,sqres=kvr()
 cbres=n**3
 return n,sqres,cbres
 return operation

def square(gv):
 def calculation():
 n=gv()
 res=n**2
 return n,res
 return calculation

@cube
@square
def getval():
 return float(input("Enter a Number:"))

```

```
#Main Program
x,y,z=getval() # Normal Function call
```

```
print("square({})={}".format(x,y))
print("cube({})={}".format(x,z))

#DecEx5.py
def kvrlower(getln):
 def textconvert():
 line,us=getln()
 ls=line.lower()
 return line,us,ls
 return textconvert

def kvrupper(getln):
 def converttext():
 line=getln()
 us=line.upper()
 return line,us
 return converttext

@kvrlower
@kvrupper
def getlinetext():
 return input("Enter a Line of Text:")

#Main Program
line,upr,lwr=getlinetext()
print("Given Line:",line)
print("Upper Text:",upr)
print("Lower Text:",lwr)

#Non-DecEx.py
def getval(): # Defined by KVR
 return float(input("Enter a Number:"))

def square():
 n=getval()
 res=n**2
 print("Square({})={}".format(n,res))

def cube():
 n=getval()
 res=n**3
 print("Cube({})={}".format(n,res))

def squareroot():
 n=getval()
 res=n**0.5
 print("sqrt({})={}".format(n,res))

#Main Program
square()
cube()
squareroot()
```

---

### All students are requested to solve the following Questions

1. Write a Python program to sum all the items in a list.
2. Write a Python program to multiply all the items in a list.
3. Write a Python program to get the largest number from a list.
4. Write a Python program to get the smallest number from a list.

5. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

Sample List : ['abc', 'xyz', 'aba', '1221']

Expected Result : 2

6. Write a Python program to get a list, sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

Sample List : [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

Expected Result : [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]

7. Write a Python program to remove duplicates from a list.

8. Write a Python program to check a list is empty or not.

9. Write a Python program to clone or copy a list.

10. Write a Python program to find the list of words that are longer than n from a given list of words.

11. Write a Python function that takes two lists and returns True if they have at least one common member.

12. Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

Sample List : ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']

Expected Output : ['Green', 'White', 'Black']

13. Write a Python program to generate a 3\*4\*6 3D array whose each element is \*.

14. Write a Python program to print the numbers of a specified list after removing even numbers from it.

15. Write a Python program to shuffle and print a specified list.

16. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).

17. Write a Python program to generate and print a list except for the first 5 elements, where the values are square of numbers between 1 and 30 (both included).

18. Write a Python program to generate all permutations of a list in Python.

19. Write a Python program to get the difference between the two lists.

20. Write a Python program access the index of a list.

21. Write a Python program to convert a list of characters into a string.

22. Write a Python program to find the index of an item in a specified list.

23. Write a Python program to flatten a shallow list.

24. Write a Python program to append a list to the second list.

25. Write a Python program to select an item randomly from a list.

26. Write a python program to check whether two lists are circularly identical.

27. Write a Python program to find the second smallest number in a list.

28. Write a Python program to find the second largest number in a list.

29. Write a Python program to get unique values from a list.

30. Write a Python program to get the frequency of the elements in a list.
31. Write a Python program to count the number of elements in a list within a specified range.
32. Write a Python program to check whether a list contains a sublist.
33. Write a Python program to generate all sublists of a list.

35. Write a Python program to create a list by concatenating a given list which range goes from 1 to n.

Sample list : ['p', 'q']

n =5

Sample Output : ['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4', 'p5', 'q5']

36. Write a Python program to get variable unique identification number or string.

37. Write a Python program to find common items from two lists.

38. Write a Python program to change the position of every n-th value with the (n+1)th in a list.

Sample list: [0,1,2,3,4,5]

Expected Output: [1, 0, 3, 2, 5, 4]

39. Write a Python program to convert a list of multiple integers into a single integer.

Sample list: [11, 33, 50]

Expected Output: 113350

40. Write a Python program to split a list based on first character of word.

41. Write a Python program to create multiple lists.

42. Write a Python program to find missing and additional values in two lists.

Sample data : Missing values in second list: b,a,c

Additional values in second list: g,h

43. Write a Python program to split a list into different variables.

44. Write a Python program to generate groups of five consecutive numbers in a list.

45. Write a Python program to convert a pair of values into a sorted unique array.

46. Write a Python program to select the odd items of a list.

47. Write a Python program to insert an element before each element of a list.

48. Write a Python program to print a nested lists (each list on a new line) using the print() function.

49. Write a Python program to convert list to list of dictionaries.

Sample lists: ["Black", "Red", "Maroon", "Yellow"], ["#000000", "#FF0000", "#800000", "#FFFF00"]

Expected Output: [{"color\_name": 'Black', 'color\_code': '#000000'}, {"color\_name": 'Red', 'color\_code': '#FF0000'}, {"color\_name": 'Maroon', 'color\_code': '#800000'}, {"color\_name": 'Yellow', 'color\_code': '#FFFF00"}]

50. Write a Python program to sort a list of nested dictionaries.

```
my_collection = {
 'KEY1':{'name':'foo','data':1351,'completed':100},
 'KEY2':{'name':'bar','data':1541,'completed':12},
 'KEY3':{'name':'baz','data':58413,'completed':18}
}
sorted_keys = sorted(my_collection, key=lambda x: (my_collection[x]['completed']))
print(sorted_keys)
```

51. Write a Python program to split a list every Nth element.

Sample list: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n']

Expected Output: [['a', 'd', 'g', 'j', 'm'], ['b', 'e', 'h', 'k', 'n'], ['c', 'f', 'i', 'l']]

52. Write a Python program to compute the difference between two lists.

Sample data: ["red", "orange", "green", "blue", "white"], ["black", "yellow", "green", "blue"]

Expected Output:

Color1-Color2: ['white', 'orange', 'red']

Color2-Color1: ['black', 'yellow']

53. Write a Python program to create a list with infinite elements.

54. Write a Python program to concatenate elements of a list.

55. Write a Python program to remove key values pairs from a list of dictionaries.

56. Write a Python program to convert a string to a list.

57. Write a Python program to check if all items of a given list of strings is equal to a given string.

58. Write a Python program to replace the last element in a list with another list.

Sample data : [1, 3, 5, 7, 9, 10], [2, 4, 6, 8]

Expected Output: [1, 3, 5, 7, 9, 2, 4, 6, 8]

59. Write a Python program to check whether the n-th element exists in a given list.

60. Write a Python program to find a tuple, the smallest second index value from a list of tuples.

61. Write a Python program to create a list of empty dictionaries.

62. Write a Python program to print a list of space-separated elements.

63. Write a Python program to insert a given string at the beginning of all items in a list.

Sample list : [1,2,3,4], string : emp

Expected output : ['emp1', 'emp2', 'emp3', 'emp4']

64. Write a Python program to iterate over two lists simultaneously.

65. Write a Python program to move all zero digits to end of a given list of numbers.

Expected output:

Original list:

[3, 4, 0, 0, 0, 6, 2, 0, 6, 7, 6, 0, 0, 0, 9, 10, 7, 4, 4, 5, 3, 0, 0, 2, 9, 7, 1]

Move all zero digits to end of the said list of numbers:

[3, 4, 6, 2, 6, 7, 6, 9, 10, 7, 4, 4, 5, 3, 2, 9, 7, 1, 0, 0, 0, 0, 0, 0, 0]

66. Write a Python program to find the list in a list of lists whose sum of elements is the highest.

Sample lists: [1,2,3], [4,5,6], [10,11,12], [7,8,9]

Expected Output: [10, 11, 12]

67. Write a Python program to find all the values in a list are greater than a specified number.

68. Write a Python program to extend a list without append.

Sample data: [10, 20, 30]

[40, 50, 60]

Expected output : [40, 50, 60, 10, 20, 30]

69. Write a Python program to remove duplicates from a list of lists.

Sample list : [[10, 20], [40], [30, 56, 25], [10, 20], [33], [40]]

New List : [[10, 20], [30, 56, 25], [33], [40]]

70. Write a Python program to find the items starts with specific character from a given list.

Expected Output:

Original list:

['abcd', 'abc', 'bcd', 'bkie', 'cder', 'cdsw', 'sdfsd', 'dagfa', 'acjd']

Items start with a from the said list:

['abcd', 'abc', 'acjd']

Items start with d from the said list:

['dagfa']

Items start with w from the said list:

[]

71. Write a Python program to check whether all dictionaries in a list are empty or not.

Sample list : [{}, {}, {}]

Return value : True

Sample list : [{1,2}, {}, {}]

Return value : False

72. Write a Python program to flatten a given nested list structure.

Original list: [0, 10, [20, 30], 40, 50, [60, 70, 80], [90, 100, 110, 120]]

Flatten list:

[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]

73. Write a Python program to remove consecutive duplicates of a given list.

Original list:

[0, 0, 1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 8, 9, 4, 4]

After removing consecutive duplicates:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 4]

74. Write a Python program to pack consecutive duplicates of a given list elements into sublists.

Original list:

[0, 0, 1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 8, 9, 4, 4]

After packing consecutive duplicates of the said list elements into sublists:

[[0, 0], [1], [2], [3], [4, 4], [5], [6, 6, 6], [7], [8], [9], [4, 4]]

75. Write a Python program to create a list reflecting the run-length encoding from a given list of integers or a given list of characters.

Original list:

[1, 1, 2, 3, 4, 4.3, 5, 1]

List reflecting the run-length encoding from the said list:

[[2, 1], [1, 2], [1, 3], [1, 4], [1, 4.3], [1, 5], [1, 1]]

Original String:

automatically

List reflecting the run-length encoding from the said string:

[[1, 'a'], [1, 'u'], [1, 't'], [1, 'o'], [1, 'm'], [1, 'a'], [1, 't'], [1, 'i'], [1, 'c'], [1, 'a'], [2, 'l'], [1, 'y']]

76. Write a Python program to create a list reflecting the modified run-length encoding from a given list of integers or a given list of characters.

Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

List reflecting the modified run-length encoding from the said list:

[[2, 1], 2, 3, [2, 4], 5, 1]

Original String:

aabcccccccaaaaaaa

List reflecting the modified run-length encoding from the said string:

[[2, 'a'], 'b', 'c', [4, 'd'], 'a', 'd', 'n', [2, 's']]

77. Write a Python program to decode a run-length encoded given list.

Original encoded list:

[[2, 1], 2, 3, [2, 4], 5, 1]

Decode a run-length encoded said list:

[1, 1, 2, 3, 4, 4, 5, 1]

78. Write a Python program to split a given list into two parts where the length of the first part of the list is given.

Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

Length of the first part of the list: 3

Split the said list into two parts:

([1, 1, 2], [3, 4, 4, 5, 1])

79. Write a Python program to remove the K'th element from a given list, print the new list. Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

After removing an element at the kth position of the said list:

[1, 1, 3, 4, 4, 5, 1]

80. Write a Python program to insert an element at a specified position into a given list.

Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

After inserting an element at kth position in the said list:

[1, 1, 12, 2, 3, 4, 4, 5, 1]

81. Write a Python program to extract a given number of randomly selected elements from a given list.

Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

Selected 3 random numbers of the above list:

[4, 4, 1]

82. Write a Python program to generate the combinations of n distinct objects taken from the elements of a given list.

HINT

Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9] Combinations of 2 distinct objects: [1, 2] [1, 3] [1, 4] [1, 5] .... [7, 8] [7, 9] [8, 9]

83. Write a Python program to round every number of a given list of numbers and print the total sum multiplied by the length of the list.

Original list: [22.4, 4.0, -16.22, -9.1, 11.0, -12.22, 14.2, -5.2, 17.5]

Result:

243

84. Write a Python program to round the numbers of a given list, print the minimum and maximum numbers and multiply the numbers by 5. Print the unique numbers in ascending order separated by space.

Original list: [22.4, 4.0, 16.22, 9.1, 11.0, 12.22, 14.2, 5.2, 17.5]

Minimum value: 4

Maximum value: 22

Result:

20 25 45 55 60 70 80 90 110

85. Write a Python program to create a multidimensional list (lists of lists) with zeros.

Multidimensional list: [[0, 0], [0, 0], [0, 0]]

86. Write a Python program to create a 3X3 grid with numbers.

3X3 grid with numbers:

[[1, 2, 3], [1, 2, 3], [1, 2, 3]]

87. Write a Python program to read a matrix from console and print the sum for each column. Accept matrix rows, columns and elements for each column separated with a space(for every row) as input from the user.

Input rows: 2

Input columns: 2

Input number of elements in a row (1, 2, 3):

1 2

3 4

sum for each column:

4 6

88. Write a Python program to read a square matrix from console and print the sum of matrix primary diagonal. Accept the size of the square matrix and elements for each column separated with a space (for every row) as input from the user.

Input the size of the matrix: 3

2 3 4

4 5 6

3 4 7

Sum of matrix primary diagonal:

14

89. Write a Python program to Zip two given lists of lists.

Original lists:

[[1, 3], [5, 7], [9, 11]]

[[2, 4], [6, 8], [10, 12, 14]]

Zipped list:

[[1, 3, 2, 4], [5, 7, 6, 8], [9, 11, 10, 12, 14]]

90. Write a Python program to count number of lists in a given list of lists.

Original list:  
[[1, 3], [5, 7], [9, 11], [13, 15, 17]]  
Number of lists in said list of lists:  
4  
Original list:  
[[2, 4], [[6, 8], [4, 5, 8]], [10, 12, 14]]  
Number of lists in said list of lists:  
3

91. Write a Python program to find the list with maximum and minimum length.

Original list:  
[[0], [1, 3], [5, 7], [9, 11], [13, 15, 17]]  
List with maximum length of lists:  
(3, [13, 15, 17])  
List with minimum length of lists:  
(1, [0])  
Original list:  
[[0], [1, 3], [5, 7], [9, 11], [3, 5, 7]]  
List with maximum length of lists:  
(3, [3, 5, 7])  
List with minimum length of lists:  
(1, [0])  
Original list:  
[[12], [1, 3], [1, 34, 5, 7], [9, 11], [3, 5, 7]]  
List with maximum length of lists:  
(4, [1, 34, 5, 7])  
List with minimum length of lists:  
(1, [12])

92. Write a Python program to check if a nested list is a subset of another nested list.

Original list:  
[[1, 3], [5, 7], [9, 11], [13, 15, 17]]  
[[1, 3], [13, 15, 17]]  
If the one of the said list is a subset of another.:  
True  
Original list:  
[[[1, 2], [2, 3]], [[3, 4], [5, 6]]]  
[[[3, 4], [5, 6]]]  
If the one of the said list is a subset of another.:  
True  
Original list:  
[[[1, 2], [2, 3]], [[3, 4], [5, 7]]]  
[[[3, 4], [5, 6]]]  
If the one of the said list is a subset of another.:  
False

93. Write a Python program to count the number of sublists contain a particular element.

Original list:  
[[1, 3], [5, 7], [1, 11], [1, 15, 7]]  
Count 1 in the said list:  
3  
Count 7 in the said list:  
2  
Original list:  
[['A', 'B'], ['A', 'C'], ['A', 'D', 'E'], ['B', 'C', 'D']]  
Count 'A' in the said list:  
3  
Count 'E' in the said list:  
1

94. Write a Python program to count number of unique sublists within a given list.

Original list:  
[[1, 3], [5, 7], [1, 3], [13, 15, 17], [5, 7], [9, 11]]  
Number of unique lists of the said list:

```
{(1, 3): 2, (5, 7): 2, (13, 15, 17): 1, (9, 11): 1}
Original list:
[['green', 'orange'], ['black'], ['green', 'orange'], ['white']]
Number of unique lists of the said list:
{('green', 'orange'): 2, ('black',): 1, ('white',): 1}
```

95. Write a Python program to sort each sublist of strings in a given list of lists.

```
Original list:
[[2], [0], [1, 3], [0, 7], [9, 11], [13, 15, 17]]
Sort the list of lists by length and value:
[[0], [2], [0, 7], [1, 3], [9, 11], [13, 15, 17]]
```

96. Write a Python program to sort a given list of lists by length and value.

```
Original list:
[[2], [0], [1, 3], [0, 7], [9, 11], [13, 15, 17]]
Sort the list of lists by length and value:
[[0], [2], [0, 7], [1, 3], [9, 11], [13, 15, 17]]
```

97. Write a Python program to remove sublists from a given list of lists, which contains an element outside a given range.

```
Original list:
[[2], [0], [1, 2, 3], [0, 1, 2, 3, 6, 7], [9, 11], [13, 14, 15, 17]]
After removing sublists from a given list of lists, which contains an element outside the given range:
[[13, 14, 15, 17]]
```

98. Write a Python program to scramble the letters of string in a given list.

```
Original list:
['Python', 'list', 'exercises', 'practice', 'solution']
After scrambling the letters of the strings of the said list:
['tnPhyo', 'tlis', 'ecrsseiex', 'ccpitear', 'noiltuos']
```

99. Write a Python program to find the maximum and minimum values in a given heterogeneous list.

```
Original list:
['Python', 3, 2, 4, 5, 'version']
Maximum and Minimum values in the said list:
(5, 2)
```

100. Write a Python program to extract common index elements from more than one given list.

```
Original lists:
[1, 1, 3, 4, 5, 6, 7]
[0, 1, 2, 3, 4, 5, 7]
[0, 1, 2, 3, 4, 5, 7]
Common index elements of the said lists:
[1, 7]
```

---

## DAY-67 03/06/2024 MONDAY

---

### Exception Handling in Python---Most Imp

---

#### Index

---

=>Purpose of Exception Handling

=>Types of Errors

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

=>Building Points in Exception Handling

=>Types of Exceptions

- a) Pre-Defined OR Built-in Exceptions
- b) Programmer OR User OR Custom Defined Exceptions

=>Programming Examples

=>Key words for Dealing with Exception Handling

- a) try
- b) except
- c) else
- d) finally
- e) raise

=>Syntax for Handling Exceptions

=>Explanation for the Key words of Exception Handling

=>Programming Examples

=>Various Forms of Exception Handling

=>Programming Examples

=>Development of Programmer OR User OR Custom Defined Exceptions

=>Programming Examples

=>AMT Case Study With Exception Handling

=====X=====

## Types of Errors in Python

=>The purpose of Exception Handling is that " To Develop OR Build Robust (Strong) Applications ".

In real time to develop any project, we choose a lang. By using that lang we develop, compile and execute Various Programs. During this Process, we get 3 types of Errors. They are

1. Compile Time Error
2. Logical Error
3. Runtime Error

### 1. Compile Time Error

=>These Errors occurs during Compilation Time (.py----->.pyc)

=>These errors occurs due to Syntaxes are not followed during Program Development.

=>These errors are solved by Programmers During Development Time

### 2. Logical Error

=>These Errors occurs during Runtime OR Execution Time

=>These errors occurs due to Wrong Representation of Logic

=>Logical Error always gives Wrong Result

=>These errors are solved by Programmers During Development Time

### 3. Runtime Error

=>These Errors occurs during Runtime OR Execution Time

=>These errors occurs due to WRONG INPUT / INVALID INPUT Entered by Users OR Application Users.

=>When Runtime Errors occurs By Default all the Languages gives Technical Error Messages, which are understandable by Programmers but not by End-Users. This is not a Recommended Process in Real Time.

=>According to Industry Standards, It recommends to display always User-Friendly Error Messages for making the Application Robust by using Exception Handling

NOTE:

=>When the End User enters Valid Input to the Project then the Project gives Successful Result.

=>When the End User enters Invalid Input to the Project then the Project Must display User-Friendly Error Messages

by using Exception Handling

## Building Points in Exception Handling

1. When the Application User Enters Wrong OR Invalid Input then we get Runtime Error.

- ( Invalid Input----->Runtime Error)
2. All Runtime Errors by default 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)  
All invalid Inputs are considered as Exceptions
4. All Exceptions by default gives Technical Error Messages.
5. Definition of Exception Handling:- The Process of Converting Technical Error Messages into User-Friendly Error Messages is called Exception Handling.
6. When the exception occurs in Python Program. The Steps Takes Place Internally.
- i) Program Execution Abnormally Terminated
  - ii) PVM Comes out-of Program flow
  - iii) PVM by default generates Technical Error Message.
7. To do the above Step-(i),Step-(ii) and Step-(iii), PVM CREATES AN OBJECT w.r.t An EXCEPTION CLASS.
8. When an Exception occurs in Python Program then PVM creates an Object of appropriate EXCEPTION CLASS
9. Hence All Exceptions are Considered as objects of EXCEPTION CLASSES.
- 
- 

**DAY-68 04/06/2024 TUESDAY**

---

---

### **Types of Exceptions in Python**

---

---

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

- 1. Pre-Defined OR Built-In Exceptions
  - 2. Programmer OR User OR Custom Defined Exceptions
- 

#### **1. Pre-Defined OR Built-In Exceptions**

---

=>Pre-Defined OR Built-In Exceptions are those which are already developed in Python Language Developers and available in Python Software and They are used by Python Lang Programmers for Dealing with Universal Problems.

=>Some of the Universal Problems are

- i) Invalid Index ( IndexError )
  - ii) Invalid Conversions ( ValueError )
  - iii) Invalid Number of Arguments OR Operations ( TypeError )
  - iv) Passing Invalid Key ( KeyError )
  - v) Division by zero problems ( ZeroDivisionError: ) .....etc
- 

#### **2. Programmer OR User OR Custom Defined Exceptions**

---

=>Programmer OR User OR Custom Defined Exceptions are those which are developed by Python Programmers and they are available in Python Project and They are used by Other Team Members of Same Project and They are always deals with Common Problems occurring in the project.

=>Some of the Common Problems are

- i) Attempting to enter Invalid User Name and Password
  - ii) Attempting to enter Invalid PIN in ATM Applications
  - iii) Attempting to Withdraw More Amount than Existing Bal of Account
  - iv) Attempting Invalid Names for P
- 

Handling the Exceptions in Python

OR

Key words for Dealing with Exception Handling

---

=>Handling the Exceptions in Python is nothing but Converting Technical Error Messages into User-Friendly Error

Messages. To do this Process, In Python Programming, we have 5 Key Words. They are

1. try

2. except
3. else
4. finally
5. raise

---

## Syntax for Handling the Exceptions:

---

try:

-----  
Block of Statements  
Generating Exceptions  
-----

except <exception-class-name-1>:

-----  
Block of Statements  
Generating User-Friendly Error Message  
-----

except <exception-class-name-2>:

-----  
Block of Statements  
Generating User-Friendly Error Message  
-----

except <exception-class-name-n>:

-----  
Block of Statements  
Generating User-Friendly Error Message  
-----

else:

-----  
Block of statements  
generates Results of the Program  
-----

finally:

-----  
Block of Statements  
Executes Compulsorily  
-----

---

## Explanation for the keywords used in Syntax of Handling Exception

---

### 1.try

=>It is the block in which we write block of statements generating exceptions. In otherwords what are all the statements are generating exceptions, those statements must be written within try block and hence try block is called Exception monitoring block.

=>When an exception occurs in try block then PVM comes out of try block and executes appropriate except block.

=>After executing appropriate except block, PVM never goes to try block for executing rest of the statements in try block.

=>Every try block must be immediately followed by except block ( Otherwise we get SyntaxError)

=>Every try block must contain atleast one except block . It is recommended to write multiple except blocks for generating User-Friendly error messages.

### 2.exception

=>It is the block in which we write block of statements generates User-Friendly Error Friendly Messages. In Otherwords except block suppreses Technical error messages and generates User-Freindly Error Messages and hence except block is called Exception Processing Block.

Note: Handling exception= try block + except block

=>except block will execute when there is an exception occurs in try block.

=>Even we write multiple except blocks , PVM executes Appropriate except block(Single Block) depends on type of exception occurs in try block.

=>The place for writing except block is that after try block and before else block.

---

### 3.else

---

=>It is the block in which we write block of statements will display results of the program and hence else block is called Result Generated Block.

=>else block will execute when there is no exception occurs in try block.

=>Writing else block is optional

=>The place of writing else block is that after except block and before finally block (if it present).

---

### 4.finally

---

=>It is the block, in which we write block of statements will relinquish (release / close / give-up/clean-up) the resources ( Files, Database softwares) which are obtained in try block and finally block is called Resources relinquishing Block.

=>finally block will execute compulsorily.

=>finally block is optional to write

=>The place of writing finally block is that after else block ( if else block present)

---

=====X=====

#Program for Cal div of Two Numbers

```
#Div1.py
print("Program Execution Started")
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
#Convert s1 and s2 into int type
a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
print("First Value=",a)
print("Second Value=",b)
c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
print("Div=",c)
print("Program Execution Ended")
```

---

#program for Cal Div of Two Numebrs

```
#Div2.py
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
except ZeroDivisionError:
 print("\tDON'T ENTER ZERO FOR DEN....")
except ValueError:
 print("\tDON'T ENTER ALNUMS,STRS AND SYMBOLS")
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
 print("-----")
finally:
 print("Program Execution Completed")
```

**DAY-69 05/06/2024 WEDNESSDAY**

---

### Various Forms of "except" Blocks

---

=>The except block can be used in the following formats

---

Format-1: Single except block can handle one specific exception at a time

---

```
try:

 Block of statements--generates exception

except exception-class-name-1:

 Block of statemenets--generates User-Freindly Error Message

except exception-class-name-2:

 Block of statemenets--generates User-Freindly Error Message

except exception-class-name-n:

 Block of statemenets--generates User-Freindly Error Message

```

---

Format-2: Single except block can handle Multiple specific exception at a time---This Type Facility is called Multi Exception handling Block.

---

```
try:

 Block of statements--generates exception

except (exception-class-name-1,exception-class-name-2,...exception-class-name-n):

 Block of statements--generates User-Freindly Error Message
 which coresponds to the type of exception

```

---

Format-3: Single except block can handle the exception with alias name---Here alias can capture type of Error Message occured due to type of exception in other software products like oracle, mysql...etc

---

```
try:

 Block of statements--generates exception

except exception-class-name-1 as alias name:

 print(alias name)

except exception-class-name-2 as alias name:

 print(alias name)

except exception-class-name-n as alias name:

 print(alias name)

```

---

Format-4: Single except block can handle ALL TYPES of Exceptions--This except block is called default except block

and It must be always written at last otherwise we get SyntaxError

---

NOTE : Only bellow Syntax --Not Recommended

```
try:

 Block of statements--generates exception
```

```

except :

 print("Ooops some thing went wrong")

```

## Final Syntax for Handling the Exceptions

```

try:

 Block of statements--generates exception

except exception-class-name-1:

 Block of statemenets--generates User-Freindly Error Message

except exception-class-name-2:

 Block of statemenets--generates User-Freindly Error Message

except exception-class-name-n:

 Block of statemenets--generates User-Freindly Error Message

except : # default except block at last

 print("Ooops some thing went wrong")

```

```
else:

 Block of statements
 generates Results of the Program

finally:

 Block of Statements
 Executes Compulsorily

```

(OR)

```
try:

 Block of statements--generates exception

except (exception-class-name-1,exception-class-name-2,...exception-class-name-n):

 Block of statements--generates User-Freindly Error Message
 which coresponds to the type of exception

except : # default except block at last

 print("Ooops some thing went wrong")

else:

 Block of statements
 generates Results of the Program

finally:

 Block of Statements
 Executes Compulsorily

```

```

#program for Cal Div of Two Numebrs handling all types specific exceptions at a time
#Div3.py
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
except (ZeroDivisionError,ValueError): # multi exception handling block
 print("\tDON'T ENTER ZERO FOR DEN....")
 print("\tDON'T ENTER ALNUMS,STRS AND SYMBOLS")
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
 print("-----")
finally:
 print("Program Execution Completed")
```

---

```
#program for Cal Div of Two Numebrs
#Div4.py
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
except ZeroDivisionError as z:
 print(z)
except ValueError as kvr:
 print(kvr)
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
 print("-----")
finally:
 print("Program Execution Completed")
```

---

```
#program for Cal Div of Two Numebrs
#Div5.py<----This is not a Recommended Program
#Writing only default except is not Recommended
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
except : # default except block
 print("oooooops some went wrong!!!")
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
```

```
print("-----")
finally:
 print("Program Execution Completed")

#program for Cal Div of Two Numebrs
#Div6.py---This Program Developed by KVR on 05-06-2024. At this point of time only 2 exceptions found
With Forecasting Knowledge of KVR , In the Future after 5 years, there is chance Modifying this Code by a new
Programmer
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
 s="PYTHON"
 print(s[10])
except ZeroDivisionError:
 print("\tDON'T ENTER ZERO FOR DEN....")
except ValueError:
 print("\tDON'T ENTER ALNUMS,STRS AND SYMBOLS")
except IndexError:
 print("Invalid Index--check Index Properly")
except: # Default except block
 print("oooops some thing went wrong")
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
 print("-----")
finally:
 print("Program Execution Completed")

#program for Cal Div of Two Numebrs
#Div7.py---This Program Developed by KVR on 05-06-2024. At this point of time only 2 exceptions found
With Forecasting Knowledge of KVR , In the Future after 5 years, there is chance Modifying this Code by a new
Programmer
try:
 print("Program Execution Started")
 s1=input("Enter First Value:")
 s2=input("Enter Second Value:")
 #Convert s1 and s2 into int type
 a=int(s1) # Problematic Stmt---Exception generated stmt--ValueError
 b=int(s2) # Problematic Stmt---Exception generated stmt--ValueError
 c=a/b # Problematic Stmt---Exception generated stmt--ZeroDivisionError
 s="PYTHON"
 print(s[10])
except Exception:
 print("oooops some thing went wrong")
else:
 print("-----else-----")
 print("First Value=",a)
 print("Second Value=",b)
 print("Div=",c)
 print("-----")
finally:
 print("Program Execution Completed")
```

**DAY 70 06/06/2024 THURSDAY**

---

=====

**raise key word**

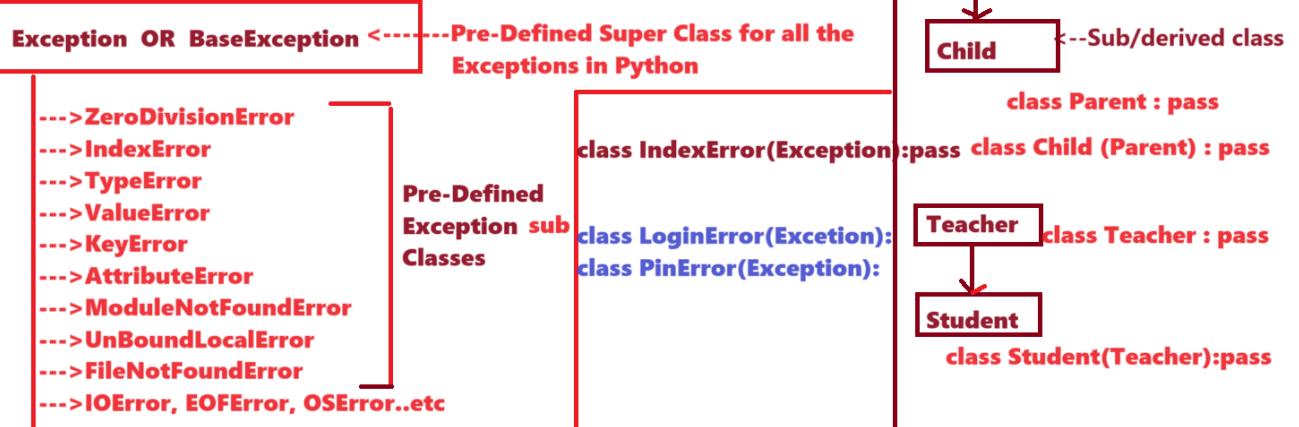
=====

=>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>  
-----  
=====

### Exception Handling Hierarchy Chart



### Development of Programmer OR User OR Custom Defined Exceptions

=>Programmer OR User OR Custom Defined Exceptions are those which are developed by Python Programmers and they are available in Python Project and They are used by Other Team Members of Same Project and They are always deals with Common Problems occurring in the project.

=>Some of the Common Problems are

- i) Attempting to enter Invalid User Name and Password
- ii) Attempting to enter Invalid PIN in ATM Applications
- iii) Attempting to Withdraw More Amount than Existing Bal of Account
- iv) Attempting Invalid Names for People, Places , Product..etc

=>To Develop Programmer OR User OR Custom Defined Exceptions, we use the following steps.

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.Hence the Programmer-Defined Class is called Programmer-Defined Exception Sub Class .

Step-3: Save the Step-1 and Step-2 on some file name with an extension .py (FileName.py)

-- Examples: Attempting to enter Invalid User Name and Password

```
class LoginError(Exception):pass
```

Examples: Attempting to enter Invalid PIN in ATM Applications  

```
class PINError(Exception):pass
```

**DAY 71 07/06/2024 FRIDAY**

---

```
#MulTableExcept.py<---File Name and Module Name
class NegNumError(Exception):pass
class ZeroError(BaseException):pass
#MulTableOperation.py<--File Name and Module Name
from MulTableExcept import NegNumError,ZeroError
def table(num): # here the value num is str, which is Received from Main Program
 n=int(num) # Here Implcily there Possibility of Raising ValueError
 if(n<0):
 raise NegNumError
 elif(n==0):
 raise ZeroError
 else:
 print("-"*50)
 print("Mul Table for :{}".format(n))
 print("-"*50)
 for i in range(1,11):
 print("\t{} x {} = {}".format(n,i,n*i))
 print("-"*50)
#MulTableOperationDemo.py<---Main Program
from MulTableOperation import table
from MulTableExcept import NegNumError,ZeroError
while(True):
 try:
 n=input("Enter a Number for Generating Mul table:")
 table(n) # Funtion call---either Result OR exception
 except ValueError:
 print("\tDon't Enter alnums,strs,floats and special symbols")
 except NegNumError:
 print("\tDon't enter -VE Number for Mul table:")
 except ZeroError:
 print("\tDon't enter Zero for Mul table:")
 else:
 print("Thx for using program")
 break
#AllAtOnceMutable.py
class NegNumError(Exception):pass
class ZeroError(BaseException):pass
#-----
def table(num): # here the value num is str, which is Received from Main Program
 n=int(num) # Here Implcily there Possibility of Raising ValueError
 if(n<0):
 raise NegNumError
 elif(n==0):
 raise ZeroError
 else:
 print("-"*50)
 print("Mul Table for :{}".format(n))
 print("-"*50)
 for i in range(1,11):
 print("\t{} x {} = {}".format(n,i,n*i))
 print("-"*50)
#-----
#Main Program
while(True):
 try:
 n=input("Enter a Number for Generating Mul table:")
 table(n) # Funtion call---either Result OR exception
 except ValueError:
 print("\tDon't Enter alnums,strs,floats and special symbols")
 except NegNumError:
 print("\tDon't enter -VE Number for Mul table:")
```

```
except ZeroError:
 print("\tDon't enter Zero for Mul table:")
else:
 print("Thx for using program")
 break
```

---

```
#ATMExcept.py<----File Name and Module name
```

```
class DepositError(Exception):pass
```

```
class WithdrawError(BaseException):pass
```

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

---

```
#ATMMenu.py<----File Name and Module Name
```

```
def menu():
 print("-"*50)
 print("\tATM Operations OR FundsTransfer")
 print("-"*50)
 print("\t1. Deposit")
 print("\t2. Withdraw")
 print("\t3. Bal Enq")
 print("\t4. Exit")
 print("-"*50)
```

---

```
#ATMOperations.py<----File Name and Module Name
```

```
from ATMExcept import DepositError,WithdrawError,InSuffFundError
```

```
bal=500.00 # Global Variable
```

```
def deposit():
```

```
 damt=float(input("Enter Ur Deposit Amount:")) # Implicitly ValueError Raises when value is alnum or
 symbols or pure str
```

```
 if(damt<=0):
```

```
 raise DepositError
```

```
 else:
```

```
 global bal
```

```
 bal=bal+damt
```

```
 print("Ur Account xxxxxxxx123 Credited with INR:{}".format(damt))
```

```
 print("Now Ur Account xxxxxxxx123 Bal INR:{}".format(bal))
```

```
def withdraw():
```

```
 wamt=float(input("Enter Ur Withdraw Amount:"))# Implicitly ValueError Raises when value is alnum or
 symbols or
```

```
 pure str
```

```
 global bal
```

```
 if(wamt<=0):
```

```
 raise WithdrawError
```

```
 elif((wamt+500)>bal):
```

```
 raise InSuffFundError
```

```
 else:
```

```
 bal=bal-wamt
```

```
 print("Ur Account xxxxxxxx123 Debitted with INR:{}".format(wamt))
```

```
 print("Now Ur Account xxxxxxxx123 Bal INR:{}".format(bal))
```

```
def balenq():
```

```
 print("Ur Account xxxxxxxx123 Bal INR:{}".format(bal))
```

---

```
#ATMOperationsDemo.py
```

```
from ATMMenu import menu
```

```
from ATMOperations import deposit,withdraw,balenq
```

```
from ATMExcept import DepositError,WithdrawError,InSuffFundError
```

```
while(True):
```

```
 try:
```

```
 menu()
```

```

ch=int(input("Enter Ur Choice:"))
match(ch):
 case 1:
 try:
 deposit()
 except ValueError:
 print("\tDon't enter alnums,strs and symbols for Depositing the
amt:")
 except DepositError:
 print("\tDon't try to Deposit -Ve Amount OR Zero in the account")
 case 2:
 try:
 withdraw()
 except ValueError:
 print("\tDon't enter alnums,strs and symbols for withdraw the
amount:")
 except WithdrawError:
 print("\tDon't try to withdraw -Ve Amount OR Zero from the
account")
 except InsuffFundError:
 print("\tUr Account does not have Suff Funds---Read Python
Notes")
 case 3:
 balenq()
 case 4:
 print("Thx for Using Program")
 break
 case _:
 print("Ur Selection of Operation is wrong--try again")
except ValueError:
 print("Don't Enter alnums,strs and symbols for Choice--try again")

```

---

**DAY-72 08/06/2024 SATURDAY**

---

## ===== 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.

=>On the object of generator, we can't perform Indexing and Slicing Operations bcoz They supply the value only on demand.

=====X

```

#Program which demonstrates the need of Generator
#GenEx1.py
def kvrrange(val):
 i=0
 while(i<val):
 yield i
 i=i+1

```

#Main Program  
r=kvrrange(10) # Function call---creates an object of <class, generator>

```
#Get First Value from Generator Object
print(next(r))
print(next(r))
print(next(r))
while(True):
 try:
 print(next(r))
 except StopIteration:
 break
print("-----OR-----")
x=kvrrange(6) # Function call---creates an object of <class, generator>
for val in x: # here x is an object <class, generator>
 print(val)

#Program for demonstrating Generator
#GenEx2.py
def kvrrange(start,stop,step=1):
 while(start<=stop):
 yield start
 start=start+step

#Main Program
go=kvrrange(10,50,10) # function call----creates an object of <class, generator>
print(next(go))
print(next(go))
while(True):
 try:
 print(next(go))
 except StopIteration:
 break
print("-----OR-----")
go1=kvrrange(100,110,3)
for val in go1:
 print(val)
print("-----OR-----")
go2=kvrrange(50,60)
for val in go2:
 print(val)
```

---

```
#GenEx3.py
def getcourses():
 yield "C"
 yield "C++"
 yield "PYTHON"
 yield "HTML"
```

---

```
#Main Program
crs=getcourses()
print(next(crs))
print(next(crs))
print(next(crs))
print(next(crs))
```

---

## ===== 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 / Function. It uses the next() method for iteration.  
=>Here iter() is used for converting Iterable object into Iterator object.  
=>next() is used for obtaining next element of iterator object and if no next element then we get an exception called StopIteration.  
=>On the object of Iterator, we can't perform Indexing and Slicing Operations bcoz They supply the value on demand .

---

Examples:

```
s = 'Python'
itobj = iter(s)
while True:
 try:
 item = next(s) # Iterate by calling next
 print(item)
 except StopIteration: # exception will happen when iteration will over
 break
```

---

```
#Program for demonstrating Iterator
#IterObjectEx1.py
x=[10,"RS",34.56,2+3j,"Python"]
print(type(x)) # <class 'list'>
#Convert Iterable object x into Iterator object type
itobj=iter(x)
print("type of itobj=",type(itobj)) # <class 'list_iterator'>
print(next(itobj))
print(next(itobj))
while(True):
 try:
 print(next(itobj))
 except StopIteration:
 break
```

---

```
#Program for demonstrating Iterator
#IterObjectEx2.py
x=(10,"RS",34.56,2+3j,"Python")
print(type(x)) # <class 'tuple'>
#Convert Iterable object x into Iterator object type
itobj=iter(x)
print("type of itobj=",type(itobj)) # <class 'tuple_iterator'>
print(next(itobj))
print(next(itobj))
for val in itobj:
 print(val)
```

---

```
#Program for demonstrating Iterator
#IterObjectEx3.py
x={10,"RS",34.56,2+3j,"Python"}
print(type(x)) # <class 'set'>
#Convert Iterable object x into Iterator object type
itobj=iter(x)
print("type of itobj=",type(itobj)) # <class 'set_iterator'>
print(next(itobj))
print(next(itobj))
for val in itobj:
 print(val)
```

---

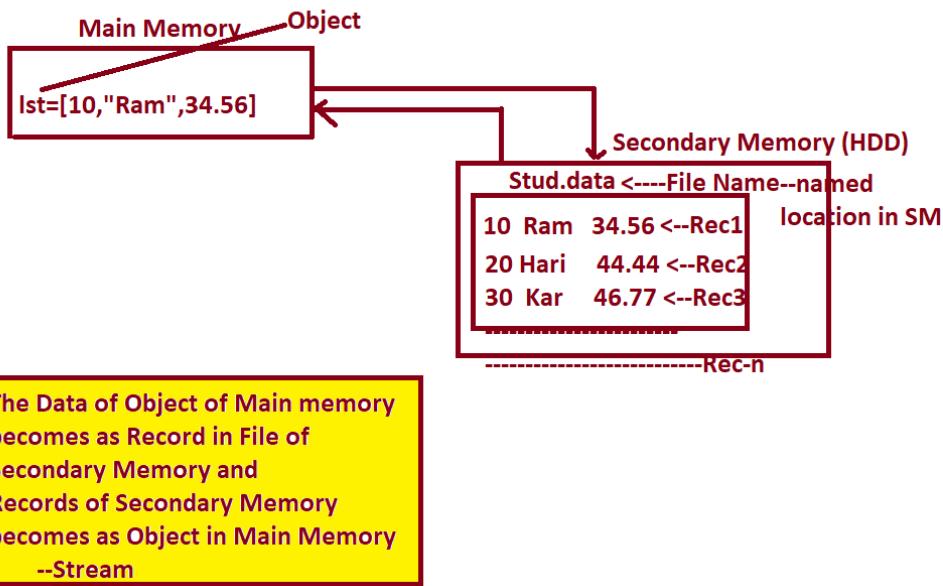
```
#Program for demonstrating Iterator
#IterObjectEx4.py
x={10:"Python",20:"Java",30:"C",40:"C++"}
print(type(x)) # <class 'dict'>
#Convert Iterable object x into Iterator object type
itobj=iter(x)
print("type of itobj=",type(itobj)) # <class 'dict_key_iterator'>
for val in itobj:
 print("{}-->{}".format(val,x.get(val)))
```

---

#Program for demonstrating Iterator

```
#IterObjectEx5.py
x="PYTHON"
print(type(x)) # <class 'str'>
#Convert Iterable object x into Iterator object type
itobj=iter(x)
print("type of itobj=",type(itobj)) # <class 'str_ascii_Iterator'>
for ch in itobj:
 print(ch)
```

**DAY-73 10/06/2024 MONDAY**



## Files OR Streams

### Index

=>Purpose of Files

=>Types of Applications

- i) Non-Persistent Applications
- ii) Persistent Applications

=>Definition of File and Stream

=>Operations on Files

- i) Write Operations
- ii) Read Operations

=>Types of Files

- i) Text Files
- ii) Binary Files

=>File Opening Modes

- |        |        |          |
|--------|--------|----------|
| i) r   | iv) r+ | vii) x   |
| ii) w  | v) w+  | viii) x+ |
| iii) a | vi) a+ |          |

=>Syntax for Opening the Files

- i) By using `open()`
- ii) By using " with `open()` as "

=>Programming Examples

=>Writing the Data to The Files

- i) `write()`
- ii) `writelines()`

=>Programming Examples

=>Reading the Data from Files

- i) read()
- ii) readlines()

=>Programming Examples

=>Random Access Files

- i) seek()
- ii) tell()

=>Programming Examples

---

=>Pickling (Object Serialization) and Un-Pickling(Object De-Serialization)

=>Implementation of Pickling and Un-Pickling Operations

=>Pickle Module

- i) dump()
- ii) load()

=>Programming Examples

---

=>Working With CSV Files

=>Dealing with csv Modules

=>Operations on CSV Files with csv Module

- i) csv.reader()
- ii) csv.writer()
- iii) csv.DictReader()
- iv) csv.DictWriter()

=>Programming Examples

---

=>Working with JSON(Java Script Object Notation) Files

=>json module

- i) json.loads()
- ii) json.load()
- iii) json.dump()

=>Programming Examples

---

=>Working OS Based Operations

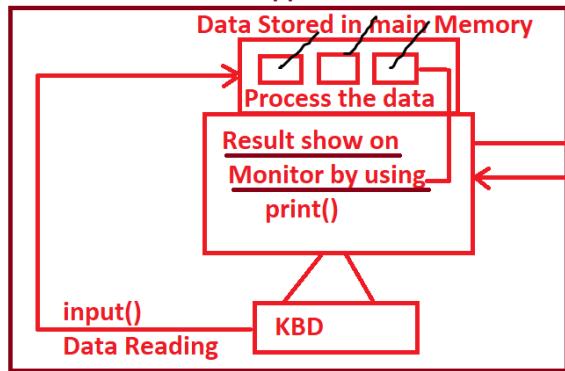
=>os Module

- i) os.mkdir()
- ii) os.makedirs()
- iii) os.rmdir()
- iv) os.removedirs()
- v) os.rename() with File and Folder
- vi) os.remove()
- v) os.listdir()

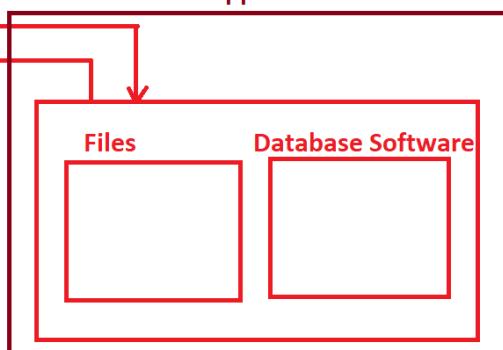
=>Programming Examples

---

### Non-Persistent Applications



### Persistent Applications



## **Types of Application in Files**

=>The purpose of Files in any programming language is that " To maintain Data Persistency".

=>The Process of storing the data permanently is called Data Persistency.

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

    1) Non-Persistent Applications

    2) Persistent Applications

=>In Non-Persistent Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form of objects, processed and whose results displayed on Monitor.

Examples: ALL our previous examples comes under Non-Persistent Applications.

=>We know that Data stored in Main Memory(RAM) is temporary.

=>In Persistent Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form 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 Persistence by Using Files of Python**

Def. of File:

=>A File is a collection of Records.

=>Files Reside in Secondary Memory(HDD).

=>Technically, A File Name is considered as a named Location in Secondary Memory.

=>The purpose of Files is that "To get Data Persistence".

=>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) FileNotFoundError

### **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) FileNotFoundError
- 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)

=>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 with Images

**DAY-74 11/06/2024 TUESDAY**

## **File Opening Modes**

=>The purpose of File Opening Modes is that In which Mode we are Opening the File for Performing an Operation on the File.

=>In Python Programming, we have 8 File Opening Modes. They are

- 1. r
- 2. w
- 3. a
- 4. r+
- 5. w+
- 6. a+
- 7. x
- 8. x+

1. r

=>This Mode is used for Opening in the File Read Mode and we can perform Read Operation

=>If we don't Specify any File Opening Mode then PVM by default Takes as "r" mode (default file mode).

=>If we open the file name in "r" mode and if that file name does not exist then we get FileNotFoundError.

2. w

=>This Mode is used for Creating the File and Open that File in Write Mode and we can write Operation.  
=>When we Choose NEW FILE and If we open in "w" mode then NEW FILE Created and Opened in write Mode and we  
can perform Write Operations.  
=>When we Choose EXISTING FILE and If we open in "w" mode then EXISTING FILE Opened in write Mode and  
Existing Data OVERLAPPED with New Data

---

3. a

=>This Mode is used for Creating the File and Open that File in Write Mode and we can write Operation.  
=>When we Choose NEW FILE and If we open in "a" mode then NEW FILE Created and Opened in write Mode and we  
can perform Write Operations.  
=>When we Choose EXISTING FILE and If we open in "a" mode then EXISTING FILE Opened in write Mode and  
Existing Data APPENDED with New Data.

---

---

### Syntax for Opening the Files

---

=>In Python Programming, we can Open the file in 2 Ways. They are

1. By using `open()`
  2. By using "`with open()` as "
- 

#### 1. By using `open()`

---

Syntax: `varname=open("File Name","File Mode")`

=>Here `open()` is a pre-defined function, which is used opening the specified file name in specified file mode.  
=>Here File Name Represents Name of the File  
=>File Mode Represents either `r,w,a,r+,w+,a+,x,x+`  
=>Here varname represents an object called File Pointer which is pointing to the file and whose type is `<class, _io.TextIOWrapper>`

=>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--- there is no concept of Auto-Closeability).

---

#### 2. By using "`with open() as` "

---

Syntax: `with open("File Name","File Mode") as varname:`

-----  
-----  
Block of Statements Performs  
Operations on File  
-----  
-----

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

#### Explanation

---

=>Here "with" and "as" are the Keywords.  
=>Here Varname Represents File pointer which will point to the Opened File and whose type is `<class, _io.TextIOWrapper>`.  
=>`open()` is used for Opening the File in Specified File Mode  
=>FileName Represents Name of the File to be specified by the Programmer  
=>File Mode Represents any one of the File Opening Modes ( `r,w,a,r+,w+,a+,x,x+`)  
=>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 out 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")
except FileNotFoundError:
 print("File does not Exist")
else:
 print("File Opened in Read Mode Sucessfully")
 print("Type of fp=", type(fp)) # <class '_io.TextIOWrapper'>
finally:
 print("I am from finally block")
 print("Is File Closed=",fp.closed)
 fp.close() # Close the file manually
 print("Is File Closed=", fp.closed)
```

---

```
#FileOpenEx2.py
fp=open("kvr.data","w")
print("File Created and Opened in write Mode")
print("Type of fp=", type(fp))
```

---

```
#FileOpenEx3.py
with open("kvr.data","r") as fp:
 print("-----")
 print("File Opened in Read Mode Sucessfully")
 print("Type of fp=", type(fp)) # <class '_io.TextIOWrapper'>
 print("-----")
 print("Is File Closed within 'with open()' as='=", fp.closed)
print("\n")
print("Is File Closed after 'with open() as' Indentation =", fp.closed)
```

---

```
#FileOpenEx4.py
with open("kvr1.data","w") as fp:
 print("File Created and Opened in Write Mode")
 print("File Name:",fp.name)
 print("File Mode:",fp.mode)
 print("Is File Readable=",fp.readable())
 print("Is File Writeable=",fp.writable())
 print("Is File Closed=",fp.closed)
```

---

```
#FileOpenEx5.py
try:
 with open("kvr1.data","r+") as fp:
 print("File Opened in Read Mode")
 print("File Name:",fp.name)
 print("File Mode:",fp.mode)
 print("Is File Readable=",fp.readable())
 print("Is File Writeable=",fp.writable())
 print("Is File Closed=",fp.closed)
except FileNotFoundError:
 print("File Does not Exist")
```

---

```
#FileOpenEx6.py
with open("kvr2.data","a+") as fp:
 print("File Created and Opened in Write Mode")
 print("File Name:",fp.name)
 print("File Mode:",fp.mode)
 print("Is File Readable=",fp.readable())
 print("Is File Writeable=",fp.writable())
 print("Is File Closed=",fp.closed)
print("Is File Closed=",fp.closed)
```

---

```
#FileOpenEx7.py
try:
 with open("kv.py","x+") as fp:
 print("File Created and Opened in Write Mode")
 print("File Name:",fp.name)
 print("File Mode:",fp.mode)
 print("Is File Readable=",fp.readable())
 print("Is File Writeable=",fp.writable())
 print("Is File Closed=",fp.closed)
 print("Is File Closed=",fp.closed)
```

```
except FileExistsError:
 print("File already created")
```

**DAY-75 12/06/2024 WEDNESDAY**

## Writing the Data to the File

=>To write the Data to the File, we have Two Pre-Defined Functions present in the object of File Pointer Object. They are

1. write()
2. writelines()

### 1. write()

Syntax: FilePointerobj.write(str data)

=>This function is used for writing any type of Data to the File in the form of str.

=>If we have non-str data then we must convert into str and we must write that str to the file.

### 2. writelines()

Syntax: FilePointerobj.writelines(str[Iterable-object])

=>This function is used for writing any Iterable Object type to the File in the form of str.

NOTE: The above Two Functions writes the Data to the file in the form of Value by Value (It is one of the Limitation--we want all the values to write at a time to File)

## Reading the Data from the File

=>To read the Data from the File, we have 2 Pre-Defined Functions present in File Pointer object. They are

1. read()
2. readlines()

### 1. read()

Syntax: varname=FilePointerObj.read()

=>This Function is used for reading entire Data of the File and Placed in LHS Varname in the form str.

### 2. readlines()

Syntax: varname=FilePointerObj.readlines()

=>This Function is used for reading the entire data of the file Placed in LHS Varname in the form of list

NOTE: The above Two Functions Reads the Data from the file in the form of Value by Value (It is one of the Limitation--we want all the values to read at a time from File)

#Program for Copying the Content of One File into Another File

#FileCopy.py

srcfile=input("Enter Source File Name:")

try:

```
with open(srcfile,"rt") as rp: # Opening SRC File in read Mode
 dstfile=input("Enter Destination File:")
 with open(dstfile,"at") as wp: # Opening DEST File in Write Mode
 srcfiledata=rp.read() # reading entire data from src file
 wp.write(srcfiledata) # write srcfiledata to dest file
 print("File Copied--Verify")
```

except FileNotFoundError:

```
 print("Source File Does not Exist")
```

#program for Demonstrating Reading the Data from the file

#FileReadEx1.py

try:

```
 with open("hyd.info") as fp:
 filedata=fp.read()
 print("-----")
 print(filedata)
```

```
print("-----")
except FileNotFoundError:
 print("File does not Exist")

#program for displaying the content of any file name
#FileReadEx2.py
try:
 filename=input("Enter Any File Name to View Its Content:")
 with open(filename,"rt") as fp:
 filedata=fp.read()
 print("-----")
 print(filedata)
 print("-----")
except FileNotFoundError:
 print('File Does not Exist')
```

```
#program for Demonstrating Reading the Data from the file
#FileReadEx3.py
try:
 with open("kvr.data") as fp:
 filedata=fp.readlines()
 print("-----")
 for line in filedata:
 print(line,end="")
 print()
 print("-----")
except FileNotFoundError:
 print("File does not Exist")

```

```
#program for displaying the content of any file name
#FileReadEx4.py
```

```
try:
 filename=input("Enter Any File Name to View Its Content:")
 with open(filename,"rt") as fp:
 filedata=fp.readlines()
 print("-----")
 for line in filedata:
 print(line,end="")
 print()
 print("-----")
except FileNotFoundError:
 print('File Does not Exist')
```

```
#Program for Demonstrating Writing the Data to the file
```

```
#FileWriteEx1.py
sno=100
sname="Dennis"
marks=22.88
#Perform write Operation
with open("student.data","a") as fp:
 fp.write(str(sno)+"\t")
 fp.write(sname+"\t")
 fp.write(str(marks)+"\n")
 print("Data Written to the file")
```

```
#Program for Demonstrating Writing the Data to the file
```

```
#FileWriteEx2.py
sno=int(input("Enter Student Number:"))
sname=input("Enter Student Name:")
marks=float(input("Enter Student Marks:"))
#Perform write Operation
with open("E:\\KVR-PYTHON-4PM\\FILES\\student.data","a") as fp:
 fp.write(str(sno)+"\t")
 fp.write(sname+"\t")
 fp.write(str(marks)+"\n")
 print("Data Written to the file")
```

```
#Program for Demonstrating Writing the Data to the file
```

```
#FileWriteEx3.py
with open("student.py","a") as fp:
 fp.write(str("print(Hello Python World)+"\t"))
 fp.write("print(Read Notes)+"\t")
```

```
print("Data Written to the file")
#Program for Demonstrating Writing the Data to the file
#FileWriteEx4.py
x={10:1.2,20:2.3,30:4.5,40:5.6}
with open("stud.data","a") as fp:
 fp.writelines(str(x)+"\n")
 print("Data written to the file")
#Program for Reading the Data from KBD and write it to the file
#FileWriteEx5.py
print("Enter Ur Data OR Message from KBD and Press @ to stop:")
with open("hyd.info","at") as fp:
 while(True):
 kbddata=input()
 if(kbddata!="@"):
 fp.write(kbddata+"\n")
 else:
 print("Data written to the file")
 break
```

**DAY-76 13/06/2024 THURSDAY**

**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 Emp details and save them as Record in emp file

```
#EmpPickEx1.py
import pickle
def saverecord():
 with open("emp.pic","ab") as fp:
 #get Emp values from KBD
 print("-----")
 eno=int(input("Enter Employee Number:"))
 ename=input("Enter Employee Name:")
 sal=float(input("Enter Employee Salary:"))
 dsg=input("Enter Employee Designation:")
 print("-----")
 #create an empty list object
 lst=list()
 lst.append(eno)
 lst.append(ename)
 lst.append(sal)
 lst.append(dsg)
 #Save lst data into the file
 pickle.dump(lst,fp)
 print("Employee Record Saved in a File Sucessfully")
 print("-----")
```

#Main Program

```
saverecord()
```

---

#program for Accepting Emp details and save them as Record in emp file

```
#EmpPickEx2.py
import pickle
def saverecord():
 with open("emp.pic","ab") as fp:
 while(True):
 try:
 #get Emp values from KBD
 print("-----")
 eno=int(input("Enter Employee Number:"))
 ename=input("Enter Employee Name:")
 sal=float(input("Enter Employee Salary:"))
 dsg=input("Enter Employee Designation:")
 print("-----")
 #create an empty list object
 lst=list()
 lst.append(eno)
 lst.append(ename)
 lst.append(sal)
 lst.append(dsg)
 #Save lst data into the file
 pickle.dump(lst,fp)
 print("Employee Record Saved in a File Sucessfully")
 print("-----")
 ch=input("Do u want Insert Another Record(yes/no):")
 if(ch.lower()=='no'):
 print("Thx for using Program")
 break
 except ValueError:
 print("Don't enter alnums,strs and symbols for empno,sal")
```

```
#Main Program
saverecord()

#program for Reading the Records from File where employee records present
#EmpUnPickEx1.py
import pickle
with open("emp.pic","rb") as fp:
 print("-----")
 while(True):
 try:
 record = pickle.load(fp)
 for val in record:
 print("\t{}".format(val),end="\t")
 print()
 except EOFError:
 print("-----")
 break
```

```
#program for Reading the Records from File where employee records present
#EmpUnPickEx2.py
import pickle
with open("emp.pic","rb") as fp:
 print("-----")
 while(True):
 try:
 record = pickle.load(fp)
 for val in record[0:2]:
 print("\t{}".format(val),end="\t")
 print()
 except EOFError:
 print("-----")
 break
```

---

#Program for Counting Number of Lines , words and letters from given file

```
try:
 filename=input("Enter Any File Name:")
 with open(filename,"rt") as fp:
 lines=fp.readlines()
 nl,nw,nc=0,0,0
 for line in lines:
 nl=nl+1
 nw=nw+len(line.split())
 nc=nc+len(line)
 else:
 print("-----")
 print("Number of Lines=",nl)
 print("Number of Words=",nw)
 print("Number of Chars=",nc)
 print("-----")
 except FileNotFoundError:
 print("File does not Exist")
```

---

#Program for Demonstrating Random Access File

```
#RandomAccessFile.py
with open("hyd.info","r") as fp:
 print("-----")
 print("Initial Index of fp=",fp.tell())
 filedatal=fp.read(5)
 print("File Data=",filedata)
 print("Now Index of fp=",fp.tell())
 print("-----")
 filedatal = fp.read(5)
 print("File Data=", filedatal)
 print("Now Index of fp=", fp.tell())
 print("-----")
 filedatal = fp.read(10)
 print("File Data=", filedatal)
 print("Now Index of fp=", fp.tell())
```

```

print("-----")
filedata = fp.read()
print("File Data=", filedata)
print("Now Index of fp=", fp.tell())
print("=====---")
filedata = fp.read()
print("File Data=", filedata)
print("Now Index of fp=", fp.tell())
print("-----")
#reset fp at 10th Index
fp.seek(10)
filedata = fp.read(6)
print("File Data=", filedata)
print("Now Index of fp=", fp.tell())
print("-----")

```

**DAY-77 14/06/2024 FRIDAY**

---

### 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.writerows(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)

**TID NAME EXP SUB**

**10 Sreeja 10 PYTHON**

**20 MahaLaxmi 12 C**

**30 Farhana 15 Java**

**40 SriLaxmi 12 C#.NET**

**50 offlinegirl 10 C++**

---

**ENO NAME SAL DESG**

**100 RS 2.3 SE**

**200 TR 3.4 PM**

**300 MC 5.6 HR**

**400 KV 0 Trainer**

**500 RT 1.2 TR**

**600 KL 1.9 HR**

**700 DR 3.4 SE**

---

**CID NAME STATE**

**1000 Suresh TS**

**2000 Hassan AP**

**3000 KVR AP-TS**

**4000 Rossum NL**

**5000 Hemanath TS**

**6000 Naveen TS**

**7000 Nagesh CH**

**8000 Nirakar MUM**

---

#Program for Reading CSV File Data in the form of Dict Format.

```
#CSVReadDictEx2.py
import csv
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\teacher.csv","r") as fp:
 dictro=csv.DictReader(fp)
 for record in dictro:
 for k,v in record.items():
 print("\t{}--->{}".format(k,v))
 print("-----")
```

---

Program for Reading CSV File Data in the form of Tabular Format.

```
#CSVReadEx1.py
import csv
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\emp.csv","r") as fp:
 csvro=csv.reader(fp) # here csvro is an object <class, csv.reader>
 print("-" * 50)
 for record in csvro:
 for val in record:
 print("\t{}".format(val),end=" ")
 print()
 print("-" * 50)
```

---

#Program for Reading CSV File Data in the form of Dict Format.

```
#CSVReadDictEx1.py
import csv
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\emp.csv","r") as fp:
 dictro=csv.DictReader(fp)
 for record in dictro:
 for k,v in record.items():
 print("\t{}--->{}".format(k,v))
 print("-----")
```

---

Program for Reading CSV File Data in the form of Tabular Format.

```
#CSVReadEx1.py
import csv
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\teacher.csv","r") as fp:
 csvro=csv.reader(fp) # here csvro is an object <class, csv.reader>
 print("-" * 50)
 for record in csvro:
 for val in record:
 print("\t{}".format(val),end=" ")
 print()
 print("-" * 50)
```

---

#Program for Creating CSV File with Dict data by using csv.DictWriter()

```
#CSVWriteDictEx1.py
import csv # Step-1
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\teacher.csv","a") as fp: # Step-2
 #Choose the Header Names--Step-3
 colnames=["TID","NAME","EXP","SUB"]
 #Choose the records--Step-4
 records=[{"TID":10,"NAME":"Sreeja","EXP":10,"SUB":"PYTHON"},
 {"TID": 20, "NAME": "MahaLaxmi", "EXP": 12, "SUB": "C"},
 {"TID":30,"NAME":"Farhana","EXP":15,"SUB":"Java"},
 {"TID":40,"NAME":"SriLaxmi","EXP":12,"SUB":"C#.NET"},
```

```

{"TID":50,"NAME":"offlinegirl","EXP":10,"SUB":"C++"}]
#Create an object of DictWrite Object by using DictWriter() of csv module
dictwro=csv.DictWriter(fp,fieldnames=colnames) # Step-5--here dictwro is an object of <class, csv.DictWriter>
#Write header names
dictwro.writeheader() # Step-6
#write records to the file
dictwro.writerows(records) #Step-7
print("CSV File Created--verify")

#Program for Creating CSV File by using csv.writer()
#CSVWriteEx1.py
import csv #Step-1
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\citizen.csv","a") as fp: # Step-2
 #Choose Header Names OR Col Names--step-3
 hnames=["CID","NAME","STATE"]
 #Choose Set of Records in the form list in list--step-4
 records=[[1000,"Suresh","TS"],
 [2000,"Hassan","AP"],
 [3000,"KVR","AP-TS"],
 [4000,"Rossum","NL"],
 [5000,"Hemanath","TS"],
 [6000,"Naveen","TS"],
 [7000,"Nagesh","CH"]]
 #create an object of csv.Writer class by using writer() of csv module
 csvwro=csv.writer(fp) # here csvwro is an object of <class, csv.Writer>--Step-5
 csvwro.writerow(hnames) # Step-6--writing the colnames to the CSV File
 csvwro.writerows(records) # Step-7--writing the records to the CSV File
 print("CSV File Created --verify")

#Program for adding Record to Existing CSV File by using csv.writer()
#CSVWriteEx2.py
import csv #Step-1
with open("E:\\KVR-PYTHON-4PM\\FILES\\NOTES\\citizen.csv","a") as fp: # Step-2
 record=[8000,"Nirakar","MUM"] # Step-3
 csvwro=csv.writer(fp) # Step-4
 csvwro.writerow(record) # Step-5
 print("New Record adding to existing CSV File--verify")

```

---

**DAY-78 15/06/2024 SATURDAY**

---

### Working JSON Files (Java Script Object Notation)

---

=>JSON stands for Java Script Object Notation.  
=>JSON File Format is a Language Independent Concept and It can be used in all the languages bcoz JSON File Format is one of the Light Weight File Format in Data Exchanging Between Client and Server Side Application in the Internet world (Web Application Development).  
=>Since JSON File Format Exchanging Data between Client and Server Side Application in the form (Key,value) and It is called Dictionary and In Python It is related dict Data Type.  
=>To take Any Information in the form json file, It Must saved on Some File Name with extension .json(FileName.json) where It contains (Key,Value)  
=>To Impelment JSON File Format in Python Programming, we must use a Pre-defined Module called "json".  
=>In Python Programing , JSON File format is Shown Below.

```
varname='{"Key1":"Val1","Key2":"Val2",...,"Key-n":"Val-n" }'
```

---

=====

Functions in json module

=====

=====

=====

Parse JSON (Convert from JSON Str Data to Python Dict)

=====

=>json.loads() Function 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 Data

```
=>json.load() Function can read the data from JSON file which contains a JSON Data and placed in dict data.
Syntax:
```

```
dictobj=json.load(file_Pointer)
```

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 Dict Data to JSON file

```
=>json.dump() Function 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")
```

```
=====X=====
#Program for Demonstrating Saving Dict Data into JSON File
#DictDataToJsonFile.py
import json
d={"FN":"Guido","LN":"Rossum","ma1":"rossum@psf.com","state":"NL"}
print(d,type(d))
print("-----")
#Saving Dict Data into JSON File
with open("jstudent.json","a") as fp:
 json.dump(d,fp)
 print("Dict Data Saved as record in Json File--verify")
```

```
#Program for Demonstrating Converting Dict Data into JSON File String Format Data
#DictDataToJsonStrfmt.py
import json
d={"FN":"Guido","LN":"Rossum","ma1":"rossum@psf.com","state":"NL"}
print(d,type(d))
print("-----")
#Convert Dict type data into json str data
```

```

jsonstrdata=str(d)
print(jsonstrdata,type(jsonstrdata))
#DynamicCSVFile.py
import csv
csvfilename=input("Enter CSV File Name with an extension .csv :")
noh=int(input("Enter How Many Header Name u want {} File: ".format(csvfilename)))
if(noh<=0):
 print("{} Invalid Header Names--Not Possible to Create CSV File")
else:
 colnames=[]
 for i in range(1,noh+1):
 col=input("Enter {} Col Name for {}: ".format(i,csvfilename))
 colnames.append(col)
 else:
 nor=int(input("Enter How Many Records u want to Enter for {} File:".format(csvfilename)))
 if(nor<=0):
 print("{} Invalid Number of records for {}".format(nor,csvfilename))
 else:
 records = [] # outer List to Take records as inner list
 for i in range(1,nor+1):
 print("-----")
 print("Enter {} Record:".format(i))
 record=list() # inner list for single record
 for j in range(len(colnames)):
 val1=input("Enter Value for {}:".format(colnames[j]))
 record.append(val1)
 else:
 records.append(record) # adding single record to outer list
 else:
 with open(csvfilename,"a") as fp:
 wo=csv.writer(fp)
 wo.writerow(colnames)
 wo.writerows(records)
 print("{} Created Successfully--very".format(csvfilename))

```

---

#Program for Demonstrating Loading JSON File data into Dict Data

#JsonFileDialogToDict.py

import json

try:

```

fp=open("jstudent.json","r")
d=json.load(fp)
print("-----")
for key,val in d.items():
 print("\t{}-->{}".format(key,val))
print("-----")

```

except FileNotFoundError:

print("Json File Does not Exist")

finally:

fp.close()

---

#Program for Demonstrating Converting JSON File String Format Data into Dict Data

#JsonStrFormatToDict.py

import json

jsonstr='{"FN":"Guido","LN":"Rossum","ma1":"rossum@psf.com","state":"NL"}'

print(jsonstr, type(jsonstr))

print("-----")

#Convert json str format into dict format--to do this we must use loads()

d=json.loads(jsonstr)

for key,val in d.items():

print("\t{}-->{}".format(key,val))

print("-----")

{"FN": "Guido", "LN": "Rossum", "ma1": "rossum@psf.com", "state": "NL"}

**DAY-79 17/06/2024 MONDAY**

---

=====

**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 FileExistsError

=>mkdir() can create only one folder at a time and if we try to create folderS hierarchy then we get FileNotFoundError.

=>in mkdir(), if we specify any folder name with escape sequence ( \n \u \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
#rmdir.py
import os
try:
 os.rmdir("D:\KVR")
 print("Folder removed Successfully-verify")
except FileNotFoundError:
 print("folder name does not exist")
except OSError:
 print("rmdir() can remove those folder which are empty--check ur path")
```

#### 4) Removing Folders Hierarchy. (removedirs())

=>For Removing Folders Hierarchy, we use removedirs()  
=>Syntax: os.removedirs("Folders Hierarchy")  
=>Here Folders Hierarchy represent Root Folder\sub folder\sub-sub folder so on...  
=>if the folder name not exist then we get FileNotFoundError  
=> if we specify any folder name with escape sequence ( \n \u \digits,\t..etc) then  
we get OSError.

##### Examples

```
#Program for Removing Folders Hierarchy
#removedirsex.py
import os
try:
 os.removedirs("D:\\India\\Hyd\\ampt\\python\\python")
 print("Folders Hierarchy Removed Successfully-verify")
except FileNotFoundError:
 print("The specified folders hierarchy does not exist")
except OSError:
 print("remove those folder which are empty-Check ur path of folder names")
```

#### 5) Removing File Name from Folder.

=>To remove the file name from folder, we use remove()  
=>Syntax: os.remove("Absolute Path of File Name")  
=>If the file name does not exist then we get FileNotFoundError

##### Examples

```
#Program for removing the file name from folder
#RemoveFileEx.py
import os
try:
 os.remove("E:\\KVR-PYTHON-7AM\\MODULES\\SE3.py")
 print("File Name removed Sucessfully")
except FileNotFoundError:
 print("File does not exist")
```

#### 6) Renaming a Folder/File Name.

=>To rename a folder, we rename()  
=>Syntax: os.rename("Old Folder Name","New Folde Name")  
=>Syntax: os.rename("Old Folder Name","New Folde Name")  
=>If the Old Folder Name does not exist then we get FileNotFoundError.

##### Examples

```
#Program for renaming a folder anme
#RenameFolderEx.py
```

```
import os
try:
 os.rename("D:\\KVR","D:\\PYTHON")
 print("Folder Name renamed")
except FileNotFoundError:
 print("File does not exist")
```

---

## 7) List the file names in folder.

---

=>To list the file names in folder, we use listdir()  
=>Syntax: os.listdir("Absolute Path of Folder Name")  
=>If the Folder Name does not exist then we get FileNotFoundError.

---

Examples:

---

```
#Program for Listing files ijn folder
#ListFileFolderEx.py
import os
try:
 FolderName=input("Enter Folder name to list files:")
 fileslist=os.listdir(FolderName)
 print("-"*50)
 print("List of Files:")
 print("-"*50)
 for filename in fileslist:
 print("\t{}".format(filename))
 print("-"*50)
except FileNotFoundError:
 print("Folder does not exist")
```

---

#Python Program for Creating Folder

```
#CreateFolderEx1.py
import os
try:
 os.mkdir("C:\\\\HYD")
 print("Folder Created ")
except FileExistsError:
 print('Folder already created')
except FileNotFoundError:
 print("Folder Does not Exist")
```

---

#Program for Creating Folders Hierarchy(

```
#CreateFoldersHierarchy.py
import os
try:
 os.makedirs("C:\\INDIA\\TS\\HYD\\NIT\\PYTHON")
 print("Folder Hierarchy Created")
except FileExistsError:
 print("Folder Hierarchy Alerady Exist")
```

---

#Program for Listing the Files in a Folder

```
#ListFilesEx1.py
import os
try:
 FilesList=os.listdir(".") # here dot (.) Refers Current Working Folder--->C:\\Users\\KVR\\PycharmProjects\\4PMFILES
 print("-----")
 print("List of Files")
 print("Number of Files=",len(FilesList))
 print("-----")
 for filename in FilesList:
 print(filename)
 print("-----")
except FileNotFoundError:
 print("Folder Does not Exist")
```

---

#Program for Listing the Files in a Folder

```
#ListFilesEx2.py
import os
```

```
try:
 foldername=input("Enter a Folder Name for Listing Files:")
 FilesList=os.listdir(foldername)
 print("-----")
 print("List of Files")
 print("Number of Files=",len(FilesList))
 print("-----")
 for filename in FilesList:
 print(filename)
 print("-----")
except FileNotFoundError:
 print("Folder Does not Exist")
#Program for Listing the Files in a Folder
#ListFilesEx3.py
import os
try:
 foldername=input("Enter a Folder Name for Listing Python Files:")
 FilesList=os.listdir(foldername)
 print("-----")
 print("List of Files")
 print("Number of Files=",len(FilesList))
 nop=0
 print("-----")
 for filename in FilesList:
 if(filename.endswith(".py")):
 print(filename)
 nop=nop+1
 else:
 print("Number of Python Files=",nop)
 print("-----")
 except FileNotFoundError:
 print("Folder Does not Exist")
#Program for Listing the Files in a Folder
#ListFilesEx4.py
import os
try:
 foldername=input("Enter a Folder Name for Listing Python Files:")
 FilesList=os.listdir(foldername)
 print("-----")
 print("List of Files")
 print("Number of Files=",len(FilesList))
 nop=0
 print("-----")
 for filename in FilesList:
 if(filename[-3:]==".py"):
 print(filename)
 nop=nop+1
 else:
 print("Number of Python Files=",nop)
 print("-----")
 except FileNotFoundError:
 print("Folder Does not Exist")
#Program for Listing the Files in a Folder
#ListFilesEx4.py
import os
try:
 foldername=input("Enter a Folder Name for Listing Python Files:")
 FilesList=os.listdir(foldername)
 print("-----")
 print("List of Files")
 print("Number of Files=",len(FilesList))
 nop=0
 print("-----")
 for filename in FilesList:
 if(filename.startswith("CSV")):
 print(filename)
```

```
nop=nop+1
else:
 print("Number of Python Files=",nop)
 print("-----")
except FileNotFoundError:
 print("Folder Does not Exist")

#Program for Removing the File Name
#RemoveFileName.py
import os
try:
 os.remove("kv.py")
 print("File Name Removed")
except FileNotFoundError:
 print("File Name does not Exist")

#Program for Removing a Folder
#RomoveFolder.py
import os
try:
 os.rmdir("C:\\HYD")
 print("Folder Removed")
except FileNotFoundError:
 print("FOlder does not Exist")
except OSError:
 print("Folder is not Empty--make sure that Folder must be empty")

#Program for Renaming a File
#RenameFileName.py
import os
try:
 os.rename("fw4.py","xyz.py")
 print("File Name Renamed")
except FileNotFoundError:
 print("Source File Does not Exist")

#Program for Renaming the Folder Name
#RenameFolder.py
import os
try:
 os.rename("C:\\BANG","C:\\BANG")
 print("Folder Renamed")
except FileNotFoundError:
 print("Source Folder Does not Exist")

#Program for Removing a Folder Hierachy
#RomoveFolderHierarchy.py
import os
try:
 os.removedirs("C:\\INDIA\\TS\\HYD\\NIT\\PYTHON")
 print("Folder Hierarchy Removed")
except FileNotFoundError:
 print("Folder Hierarchy does not Exist ")
except OSError:
 print("Folder is not Empty--make sure that Folder must be empty")
```

**DAY-80 18/06/2024 TUESDAY**

---

=====

**Python Data Base Communication (PDBC)**

=====

**Index**

---

- =>Limitations of File
- =>Advantages of Data Base Softwares
- =>Importance of Python Database Communication (PDBC)
- =>Module for Python Database Communication (PDBC)
- =>Third Party Module Set-up for Python Database Communication (PDBC)
- 
- =>Steps for Developing Python Database Communication (PDBC) Applications
- =>Types of Queries

- a) DDL Queries
- b) DML Queries
- c) DRL Queries

=>Executing DDL Queries from Python Program

- a) create
- b) alter
- c) drop

=>Programming Examples

=>Executing DML Queries from Python Program

- a) insert
- b) delete
- c) update

=>Programming Examples

=>Executing DRL Queries from Python Program

- a) select

=>Programming Examples

---

=>Meta Data Programming

=>Programming Examples

---

BOOK NAME

---

Oracle Developers Guide--By Ivan Bayras

BPB Publications.

---

### **Python DataBase Communication ( PDBC )**

---

=>Even we achieved the Data Persistence by using Files, Files has the following Limitations.

1. Files of any language does not contain security bcoz Files are unable to provide security in the form of User Name and Password.
2. Files are unable to store large amount of data
3. File are differing from One OS to another OS (Files are OS dependent)
4. Querying and Processing the data from Files is Very Complex bcoz file data is organized w.r.t Indices and identifying the indices is very complex.
5. Files does not contain Column Names (Except CSV Files) and complex to Process data

=>To Overcome the limitation of files and to achieve the Data Persistence, we must use the concept of any RDBMS DataBase Softwares ( Oracle, MYSQL, Mongo DB, DB2, SQL Server, Postgey SQL, SQLITE3.....etc).

1. All RDBMS DataBase Softwares Provides Security bcoz RDBMS DataBase Softwares considers User names and Password.
2. All RDBMS DataBase Softwares stores large amount of data Compared to Files
3. All RDBMS DataBase Softwares Arch Remains Same on all types of OSes ( OS Independent)
4. Querying and Processing the data from All RDBMS DataBase Softwares is Very Simple bcoz data of All RDBMS DataBase Softwares organized records in the form of Tables with Column Names.
5. The Data Present in any RDBMS DataBase Softwares organized in the form 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 OR oracledb Module

=>Examples : pip install cx\_Oracle (upto Python 3.10)  
                  pip install oracledb ( from python 3.11 and higher)

=>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 (at any Windows command prompt)

=>Examples : pip install mysql-connector-python (at any Windows command prompt)

=>In order Develop Python DataBase Communication ( PDBC ) Applications, we must have following Pre-Requisites

1. Python Software must be Installed
2. Database Software must be Installed
3. Database Related Module Name must be Installed

---

### Steps for Developing Python Data Base Communication (PDBC) Applications

---

=>To Develop Python Data Base Communication (PDBC) Applications, we need the following Pre-Requisites.

1. Install Python Software
2. Install Data Base Software ( Example: Oracle, MySQL, SQL Server, MongoDB, DB2, Postgrey SQL..etc)
3. Install Corresponding Data Base Module for Python Programmers
  - ( upto Python3.10 Version, Install cx\_Oracle module
  - from Python 3.11 and Higher, Install oracledb module
  - For MySQL, Install mysql-connector and mysql-conneter-python..etc )

=>After Installing the above Software, we develop the Application, which Establish the Communication Python Software and Database Software. To doing this we need to learn the following the Steps.

\*\*\*\*\*  
Step-1: import Appropriate Data Base Module and Other Modules if required.

Step-2: Every Python Program Must Obtain the Connection

Step-3: Every Python Program Must Create an object of Cursor

Step-4: Every Python Must Design the Query ,Place the Query in Cursor object and send for Execute in Database.

Step-5: Every Python Program Must Process the Result of the Query

Step-6: Every Python Program is Recommned to Close the Connection from Database Software.

---

DAY-81 19/06/2024 WEDNESSDAY

---

---

### Communication between Python Program and Oracle

---

=>To Develop Python and oracle Communication (PDBC) Applications, we need the following Pre-Requisite

1. Install Python Software
2. Install Oracle Data Base Software
3. Install Corresponding Data Base Module for Python Programmers
  - ( upto Python3.10 Version, Install cx\_Oracle module
  - from Python 3.11 and Higher, Install oracledb module)

=>After Installing the above Software, we develop the Application, which Establish the Communication Python Software and Oracle Database Software. To doing this we need to learn the following the Steps.

\*\*\*\*\*  
Step-1: import cx\_Oraxcle OR oracledb depends on Python Version and Other Modules if required.

Step-2: Every Python Program Must Obtain the Connection from Oracle Database

Step-3: Every Python Program Must Create an object of Cursor

Step-4: Every Python Must Design the Query ,Place the Query in Cursor object and send for Execute in Database.

Step-5: Every Python Program Must Process the Result of the Query

Step-6: Every Python Program is Recommned to Close the Connection from Database Software.

---

### Explanation

---

Step-1: import cx\_Oraxcle OR oracledb depends on Python Version and Other Modules if required.

-----=>When Python Programmer wants to communicat with Oracle Database then Python Programmer Must import either

cx\_Oracle Module (upto Python 3.10) OR oracledb module (from Python 3.11 and Higher).

Examples: import cx\_Oracle as crc  
(OR)  
import oracledb as orc

---

-----Step-2: Every Python Program Must Obtain the Connection from Oracle Database

---

=>After importing either cx\_Oracle or oracledb module, Python program Must the get the Connection from Oracle Database.

=>To get the Connection from Oracle Database, we must use connect(), which is present in cx\_Oracle OR oracledb module.

---

Syntax: varname=oracledb.connect("username/password@DNS/ServiceID")  
OR  
varname=oracledb.connect("username/password@IPAddress/ServiceID")

=>Here varname---->Represents Connection Object of type <class, oracledb.Connection>

=>here oracledb---->Represents Name of Pre-Defined Third Party Module

=>Here connect()--->It is One of the Pre-DefinedFunction used for Obtaining the connection from Oracle Database

=>here username--->Represents user name of Oracle Database

=>here password--->Represents Password of Oracle Database

=>here DNS----->DNS stands for Domain Naming System/Service.

DNS Represents Name of the Physical machine where Database

Software Resides

The Default DNS of Every Computer is "localhost".

=>here IPAddress--->IPAddress stands for Internet Protocol Address.

IPAddress Represents Physical Address of the Machine where

Database Software Resides.

The Default IPAddress of Every Computer is 127.0.0.1 (Loop Back Address).

=>here ServiceID---->Represents Alias name or Alternative Name to Original Name.

To Find ServiceID of Oracle Database, we use the following Query at

SQL Environment

SQL> select \* from global\_name;  
OUTPUT

-----  
GLOBAL\_NAME

-----  
orcl <---Service ID

=>Here "username/password@DNS/ServiceID" is called Connection URL of Oracle Database.

=>If we write OR Specify any part of "username/password@DNS/ServiceID" as wrong then we get exception called "oracledb.DatabaseError "

---

Step-3: Every Python Program Must Create an object of Cursor

---

=>The purpose of Creating an object of Cursor is that "To carry the query from Python Program to Oracle Database and brings the result from Oracle Database and handover to Python Program".

=>To Create an object of Cursor, we use a pre-defined function called cursor() which is present in connection object.

=>Syntax: varname=connobj.cursor()

=>Here varname is an object of <class, oracledb.Cursor>

---

Step-4: Every Python Must Design the Query ,Place the Query in Cursor object and send for Execute in Database.

---

=>A Query is a Request / Question to the Data Base Software for Performing any Database Operation from Python Program.

=>To Execute the Query from Python Program in Oracle Database, we use a pre-defined function execute(), which is present in cursor object.

=>Syntax: curobj.execute("Query")

=>Here Query can be either 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 dropping the table from Database Software:

=>Syntax: SQL> drop table tablename

=>Examples: SQL > drop table employee

## Types of Queries in Database Softwares

=>In RDBMS Softwares, SQL Queries are classified into 3 Types. They are

1. DDL (Data Definition Language) Queries--create, alter, drop
2. DML(Data Manipulation Language) Queries--insert,update,delete
3. DRL (Data Retrieval Language) Queries ----select

#Program for Demonstrating Connection from Oracle DB

#OracleConnTest1.py

import oracledb as orc # Step-1

try:

    con=orc.connect("system/tiger@localhost/orcl") # Step-2

    #Here con is an object of <class, oracledb.Connection>

    print("Python Program Obtains Connection From Oracle DB")

    print("Type of con=",type(con))

except orc.DatabaseError as db:

```
print("Problem in Oracle DB: ",db)
#Program for Demonstrating Connection from Oracle DB
#OracleConnTest2.py
import oracledb as orc # Step-1
try:
 con=orc.connect("system/tiger1@127.0.0.1/orcl") # Step-2
 #Here con is an object of <class, oracledb.Connection>
 print("Python Program Obtains Connection From Oracle DB")
 print("Type of con=",type(con))
except orc.DatabaseError as db:
 print("Problem in Oracle DB: ",db)
```

---

```
#Program for Demonstrating to create an object of Cursor
#OracleCursor.py
import oracledb as orc # step-1
con=orc.connect("system/tiger@localhost/orcl")
print("Python Program Obtains Connection From Oracle DB")
print("Type of con=",type(con))
print("-----")
cur=con.cursor() # Step-3
print("Python Program created cursor object")
print("type of cur=",type(cur)) # here cur is an object of <class 'oracledb.Cursor'>
```

---

```
#Program for Removing the Table
#OracleDropTable.py
import oracledb as orc # Step-1
def removetable():
 try:
 con=orc.connect("system/tiger@localhost/orcl") # Step-2
 cur=con.cursor() # Step-3
 #Step-4
 cq="drop table student"
 cur.execute(cq)
 print("Table dropped successfully")
 except orc.DatabaseError as db:
 print("Problem in Oracle DB: ", db)
```

```
#main Program
removetable()
#Program for Creating table employee with Suitable Col Names
#OracleTableCreateEx1.py
import oracledb as orc # Step-1
try:
 con=orc.connect("system/tiger@localhost/orcl") # Step-2
 cur=con.cursor() # Step-3
 #Step-4
 cq="create table employee(eno number(2) primary key,name varchar2(10) not null,marks number(5,2) not null)"
 cur.execute(cq)
 print("Table Create successfully")
except orc.DatabaseError as db:
 print("Problem in Oracle DB: ", db)
```

---

```
#Program for Creating table employee with Suitable Col Names
#OracleTableCreateEx2.py
import oracledb as orc # Step-1
try:
 con=orc.connect("system/tiger@localhost/orcl") # Step-2
 cur=con.cursor() # Step-3
 #Step-4
 cq=input("Enter a Query to create a table in Oracle:")
 cur.execute(cq)
 print("Table Created successfully")
except orc.DatabaseError as db:
 print("Problem in Oracle DB: ", db)
```

---

```
import oracledb as orc
print(orc._version_)
```

## **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

---

```
#OracleAlterWithAdd.py
import oracledb as orc
def altertable():
 try:
 con=orc.connect("system/tiger@localhost/orcl")
 cur=con.cursor()
 aq="alter table employee add(compname varchar2(10) not null)"
 cur.execute(aq)
 print("Table Altered--verify")
 except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)
```

#Main Program

altertable() # function call

---

```
#OracleAlterWithModify.py
import oracledb as orc
def altertable():
 try:
 con=orc.connect("system/tiger@localhost/orcl")
 cur=con.cursor()
 aq="alter table employee modify(eno number(3),name varchar2(15))"
 cur.execute(aq)
 print("Table Altered--verify")
```

```
except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)
```

---

```
#Main Program
altertable() # function call
```

```
#OracleRecordDeleteEx1.py
```

```
import oracledb as orc
```

```
def deleterecord():
```

```
try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 cur.execute("delete from employee where eno=275")
 con.commit()
 print("{} Record deleted--verify".format(cur.rowcount))
except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
```

```
#Main Program
```

```
deleterecord()
```

---

```
#OracleRecordDeleteEx2.py
```

```
import oracledb as orc
```

```
def deleterecord():
```

```
while(True):
 try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 #Get employee Number from KBD
 empno=int(input("Enter Employee Number:"))
 cur.execute("delete from employee where eno=%d" %empno)
 con.commit()
 if(cur.rowcount>0):
 print("{} Record Deleted--verify".format(cur.rowcount))
 else:
 print("{} Record Does Not Exist".format(empno))
 print("-----")
 ch=input("Do u want to delete another record(yes/no):")
 if(ch.lower()=="no"):
 print("Thx for Using Program")
 break
 except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
 except ValueError:
 print("Don't enter non-int value for empno")
```

```
#Main Program
```

```
deleterecord()
```

---

```
#OracleRecordInsertEx1.py
```

```
import oracledb as orc
```

```
def insertrecord():
```

```
try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 iq="insert into employee values(200,'Rossum',1.6,'PSF')"
 cur.execute(iq)
 con.commit()
 print("Record Inserted--verify")
except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
```

```
#Main Program
```

```
insertrecord()
```

---

```
#OracleRecordInsertEx2.py
```

```
import oracledb as orc
```

```
def insertrecord():
```

```
try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
```

```
cur=con.cursor()
print("-----")
#Get the employee values from KBD
empno=int(input("Enter Employee Number:"))
empname=input("Enter Employee Name:")
empsal=float(input("Enter Employee Salary:"))
cname=input("Enter Employee Comp Name:")
print("-----")
iq="insert into employee values(%d,'%s',%f,'%s')"
cur.execute(iq %(empno,empname,empsal,cname))
#OR cur.execute("insert into employee values(%d,'%s',%f,'%s')" %(empno,empname,empsal,cname))
con.commit()
print("{} Record Inserted--verify".format(cur.rowcount))
print("-----")
except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
```

#Main Program

insertrecord()

---

```
#OracleRecordInsertEx2.py
import oracledb as orc
def insertrecord():
 while(True):
 try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 print("-----")
 #Get the employee values from KBD
 empno=int(input("Enter Employee Number:"))
 empname=input("Enter Employee Name:")
 empsal=float(input("Enter Employee Salary:"))
 cname=input("Enter Employee Comp Name:")
 print("-----")
 iq="insert into employee values(%d,'%s',%f,'%s')"
 cur.execute(iq %(empno,empname,empsal,cname))
 #OR cur.execute("insert into employee values(%d,'%s',%f,'%s')" %(empno,empname,empsal,cname))
 con.commit()
 print("{} Record Inserted--verify".format(cur.rowcount))
 print("-----")
 ch=input("Do u want to Insert another record(yes/no):")
 if(ch.lower() == "no"):
 print("Thx for Using this Program")
 break
 except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
 except ValueError:
 print("Don't enter empno and salary as Non-Int/float value")
```

#Main Program

insertrecord()

---

```
#OracleUpdateRecordEx1.py
import oracledb as orc
def updaterecord():
 try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 cur.execute("update employee set sal=1.2 where eno=225")
 con.commit()
 print("{} Record Updated--verify".format(cur.rowcount))
 except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
```

#Main Program

updaterecord()

---

```
#OracleUpdateRecordEx2.py
import oracledb as orc
```

```

def updaterecord():
 while(True):
 try:
 con=orc.connect("system/tiger@127.0.0.1/orcl")
 cur=con.cursor()
 #Get employee Number from KBD
 empno=int(input("Enter Employee Number for updating emp sal:"))
 newsal=float(input("Enter New Salary for Employee:"))
 cur.execute("update employee set sal=%f where eno=%d" %(newsal,empno))
 con.commit()
 if(cur.rowcount>0):
 print("{} Record Updated--verify".format(cur.rowcount))
 else:
 print("{} Record Does Not Exist".format(empno))
 print("-----")
 ch=input("Do u want to Update another record(yes/no):")
 if(ch.lower()!="no"):
 print("Thx for Using Program")
 break
 except orc.DatabaseError as db:
 print("Problem in Oracle in DB:",db)
 except ValueError:
 print("Don't enter non-int/float value for empno,salary")

```

#Main Program  
updaterecord()

**DAY-83 21/06/2024 FRIDAY**

---

### **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: a) if specified number of records==0 then this function obtains all records(in the case of cx\_Oracle Module

python 3.8,3.9 and 3.10 only )

case-1: b) if specified number of records==0 then this function select no records (in the case of oracledb--python 3.11,3.12 versions Module)

case-2: if specified number of records>0 and specified number of records<=Total Number of Records then this function gives specified number of records in the case of both cx\_Oracle and oracledb.

case-3: if specified number of records>Total Number of Records then this function obtains all records in the case of

both cx\_Oracle and oracledb

case-4: if specified number of records<0 then this function never gives any records in the case of both cx\_Oracle and

oracledb

case-5: if we don't specify specified number of records then this function obtains all records in the case of both cx\_Oracle and oracledb

---

### 3) fetchall()

---

=>fetchall() is used for obtaining all the records from cursor object in the form of tuples of list.

---

#Program for Demonstrating How to read Records from table--fetchone()

#OracleSelectRecordsEx1.py

import oracledb as orc

def selectrecords():

try:

    con=orc.connect("system/tiger@localhost/orcl")

    cur=con.cursor()

    #Read the Records from Employee Table

    sq="select \* from employee"

    cur.execute(sq)

    #get the records

    print("-" \* 50)

    while (True):

        record = cur.fetchone()

        if(record!=None):

            for val in record:

                print("\t{}".format(val),end="\t")

                print()

        else:

            print("-" \* 50)

            break

    except orc.DatabaseError as db:

        print("Problem in Oracle DB:",db)

#Main Program

selectrecords()

---

#Program for Demonstrating How to read Records from table--fetchmany()

#OracleSelectRecordsEx2.py

import oracledb as orc

def selectrecords():

try:

    con=orc.connect("system/tiger@localhost/orcl")

    cur=con.cursor()

    #Read the Records from Employee Table

    sq="select \* from employee"

    cur.execute(sq)

    #get the records

    print('-----')

    records=cur.fetchmany(3)

    for record in records:

        for val in record:

            print("\t{}".format(val),end="\t")

            print()

        print('-----')

    except orc.DatabaseError as db:

        print("Problem in Oracle DB:",db)

#Main Program

selectrecords()

---

#Program for Demonstrating How to read Records from table--fetchall()

#OracleSelectRecordsEx2.py

import oracledb as orc

def selectrecords():

try:

    con=orc.connect("system/tiger@localhost/orcl")

    cur=con.cursor()

    #Read the Records from Employee Table

    sq="select \* from employee"

```
cur.execute(sq)
#get the records
print('-----')
records=cur.fetchall()
for record in records:
 for val in record:
 print("\t{}".format(val),end="\t")
 print()
print('-----')
except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)
```

#Main Program  
selectrecords()

---

```
#program Obtaining Column names of Table
#OracleTableColNamesEx1.py
import oracledb as orc
def selectcolnames():
 try:
 con=orc.connect("system/tiger@localhost/orcl")
 cur=con.cursor()
 #Read the Records from Employee Table
 sq="select * from employee"
 cur.execute(sq)
 #get the col names
 print('-----')
 metadata=cur.description
 for colnames in metadata:
 print(colnames[0],end="\t")
 print()
 print('-----')
 except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)
```

#Main Program  
selectcolnames()

---

```
#OracleTableRecordsColNamesEx1.py
import oracledb as orc
def selectrecordscolnames():
 try:
 con=orc.connect("system/tiger@localhost/orcl")
 cur=con.cursor()
 #Read the Records from Employee Table
 sq="select * from %s "%input("Enter Table Name:")
 cur.execute(sq)
 #get the col names
 print('-----')
 metadata=cur.description
 for colnames in metadata:
 print(colnames[0],end="\t\t")
 print()
 print('-----')
 #get records
 records=cur.fetchall()
 for record in records:
 for val in record:
 print("\t{}".format(val),end="\t\t")
 print()
 print('-----')
 except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)
```

#Main Program  
selectrecordscolnames()

---

```
#OracleTableRecordsOrderEx1.py
import oracledb as orc
```

```

def selectrecordscolnames():
 try:
 con=orc.connect("system/tiger@localhost/orcl")
 cur=con.cursor()
 #Read the Records from Employee Table
 sq="select * from employee order by sal "
 cur.execute(sq)
 #get the col names
 print('-----')
 metadata=cur.description
 for colnames in metadata:
 print(colnames[0],end="\t\t")
 print()
 print('-----')
 #get records
 records=cur.fetchall()
 for record in records:
 for val in record:
 print("\t{}".format(val),end="\t\t")
 print()
 print('-----')
 except orc.DatabaseError as db:
 print("Problem in Oracle DB:",db)

```

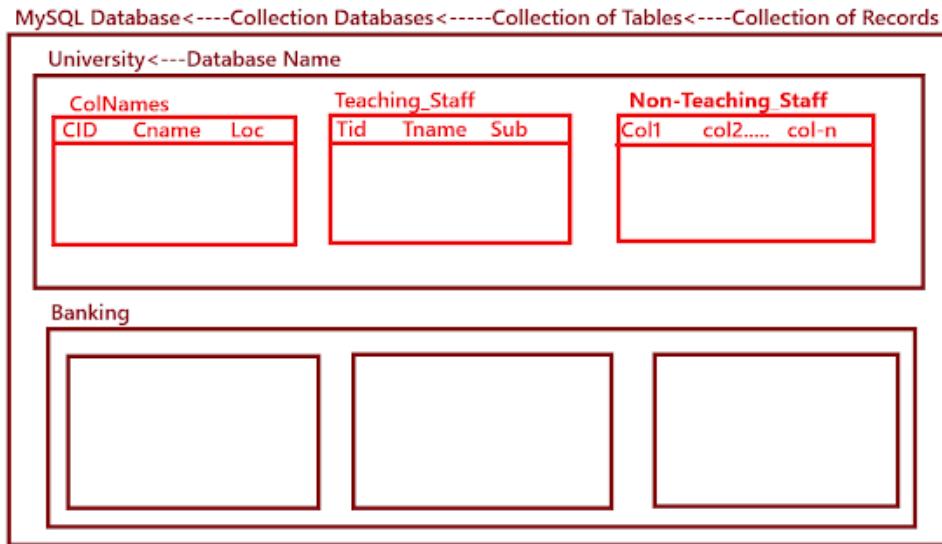
#Main Program

selectrecordscolnames()

---

**DAY-84 22/06/2024 SATURDAY**

---



### Communication between Python Program and MySQL Database Software

---

Steps

---

1. Import mysql.connector Module and Other Modules If Required
  2. Every Python Program Must get CONNECTION from MySQL Database Software
  3. Every Python Program Must create an object of CURSOR
  4. Every Python Program Must Design the Query, Place the Query in the Cursor Object and EXECUTE
  5. Every Python Program must PROCESS the Result coming from Database Software through Cursor Object.
  6. Every Python Program need to Close the Connection (Optional)
-

## Explanation

**Step-1. Import mysql.connector Module and Other Modules If Required**

=>When Python Programmer wants to communicate with MySQL Database then Python Programmer Must import mysql.connector module

**Step-2:** Every Python Program Must get CONNECTION from MySQL Database Software

=>After mysql.connector module, Python program Must the Connection from MySQL Database.  
=>To get the Connection from MySQL Database, we mus use connect(), which is present in mysql.connector module.

=>Here varname---->Represents Connection Object

=>here mysql.connector---->Represents Name of Pre-Defined Third Party Module

=>Here connect()--->It is One of the Pre-DefinedFunction uses for Obtaining the connection from MySQL Database

=>here username--->Represents user name of MySQL Database

=>here password--->Represents Password of MySQL Database

=>here Host----->Represents DNS stands for Domain Naming System.

Software Resides DNS Represents Name of the Physical machine where Database

The Default DNS of Every Computer is "localhost".

=>here IPAddress--->IPAddress stands for Internet Protocol Address.

IPAddress Represents Physical Address of the Machine where Database Software Resides.

The Default IPAddress of Every Computer is 127.0.0.1 (Loop Back Address).

=>Here "host="DNS/IPAddress", user="user Name", passwd="password" is calle Connection URL of MySQL Database.

=>If we write OR Specify any part of Connection URL wrong then we get exception called "mysql.connector.DatabaseError "

## Examples:

```
#Program for Obtaining Connection from MySQL Database
#MySQLConnectionTestEx1.py
import mysql.connector
try:
 con=mysql.connector.connect(host="localhost",
 user="root",
 passwd="root")
 print("Python Program Got Connection from MySQL")
except mysql.connector.DatabaseError as db:
 print("Problem in MySQL ",db)
```

**Step-3.** Every Python Program Must create an object of CURSOR

=>After Obtaining the Connection from Oracle Database by the Python Program, later we must create an object of Cursor.

=>Here Cursor is an object which is used Taking the Query from Python Program, Handover to Oracle Database(Any database),

and gets the Result from database software and handover to Python Program.

=>To create an object of Cursor, we must use cursor() which is present in Connection Class object.

Syntax:

varname=connobj.cursor()

here varname is called Cursor Object whose type is <class, oracledb.Cursor>

---

**Step-4.** Every Python Program Must Design the Query, Place the Query in the Cursor Object and EXECUTE

---

=>A Query is one of the Request OR Question to database software from Program Language(Python Program).

=>To execute the Query which was placed inCursor object, we use execute() which is present in cursor object.

Syntax:- cursorobj.execute("Query")

=>Here the Query can be Either DDL or DML or DRL

#Program for Demonstrating How to get the Connection from MySQL Database

#MySQLConnTestEx1.py

import mysql.connector

try:

```
con=mysql.connector.connect(host="localhost",
 user="root",
 passwd="root")
```

print("Python Program got connection from MySQL")

except mysql.connector.DatabaseError as db:

print("Problem in MySQL: ",db)

---

#Program for Demonstrating How to get the Connection from MySQL Database

#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")

except mysql.connector.DatabaseError as db:

print("Problem in MySQL: ",db)

---

#Program for Demonstrating How to create Database on the name of 4pmbatch

#MySQLDataBaseCreate.py

import mysql.connector

def createdatabase():

try:

```
con = mysql.connector.connect(host="127.0.0.1",
 user="root",
 passwd="root")
```

cur=con.cursor()

#create data base in mysql

dc="create database kvr"

cur.execute(dc)

print("Database Created Successfully in MySQL--verify")

except mysql.connector.DatabaseError as db:

print("Problem in MySQL: ", db)

```
#Main program
createdatabase()
```

```

#MySQLTest.py
import mysql.connector
print(mysql.connector.__version__)

```

```
=====
Library Management System
=====
```

- ```
1. Add a New Book  
2. Delete a Book  
3. Update Book Deatils  
4. View Book Details  
5. View All Book Deatils  
6. Exit
```

```
=====  
Enter Ur Choice:  
=====
```

```
File System
```

```
-----  
LibraryMenu.py  
    menu()  
LibraryAdd.py  
    addbook()  
LibraryDelete.py  
    deletebook()  
LibraryUpdate.py  
    updatebook()  
LibraryView.py  
    viewallbooks()  
LibraryMainProject.py-----main Program
```

```
-----  
Table Name
```

```
-----  
Library
```

```
-----  
BNO          BNAME        PRICE  PUBLICATION
```

```
-----  
#LibraryAdd.py<----File Name and Module Name  
import oracledb as orc  
def addbook():  
    while(True):  
        try:  
            con=orc.connect("system/tiger@127.0.0.1/orcl")  
            cur=con.cursor()  
            print("-----")  
            #Get the Book values from KBD  
            bno=int(input("Enter Book Number:"))  
            bookname=input("Enter Book Name:")  
            bookprice=float(input("Enter Book Price:"))  
            pub=input("Enter Book Publication:")  
            print("-----")  
            iq="insert into library values(%d,'%s','%f','%s')"  
            cur.execute(iq %(bno,bookname,bookprice,pub))
```

```

#OR cur.execute("insert into employee values(%d,'%s','%f','%s')"
%(empno,empname,empsal,cname))
con.commit()
print("{} Record Inserted--verify".format(cur.rowcount))
print("-----")
ch=input("Do u want to Insert another record(yes/no):")
if(ch.lower() == "no"):
    print("Thx for Using this Program")
    break
if (ch.lower() != "yes"):
    print("Plz Learning Typing")
    break
except orc.DatabaseError as db:
    print("Problem in Oracle in DB:",db)
except ValueError:
    print("Don't enter empno and salary as Non-Int/float value")
-----
```

```

#LibraryDelete.py<----File Name and Module name
import oracledb as orc
def deletebook():
while (True):
    try:
        con = orc.connect("system/tiger@127.0.0.1/orcl")
        cur = con.cursor()
        # Get book Number from KBD
        bno = int(input("Enter Book Number:"))
        cur.execute("delete from library where bno=%d" % bno)
        con.commit()
        if (cur.rowcount > 0):
            print("{} Record Deleted--verify".format(cur.rowcount))
        else:
            print("{} Record Does Not Exist".format(bno))
        print("-----")
        ch = input("Do u want to delete another record(yes/no):")
        if (ch.lower() == "no"):
            print("Thx for Using Program")
            break
        if (ch.lower() != "yes"):
            print("Plz Learning Typing")
            break
    except orc.DatabaseError as db:
        print("Problem in Oracle in DB:", db)
    except ValueError:
        print("Don't enter non-int value for Book Number")
-----
```

```

#LibraryMainProject.py
from LibraryMenu import menu
from LibraryAdd import addbook
from LibraryView import viewallbooks,viewbook
from LibraryDelete import deletebook
from LibraryUpdate import updatebook
while(True):
    try:
        menu()
        ch=int(input("Enter UR Choice:"))
        match(ch):
            case 1:
                addbook()
            case 2:
```

```
    deletebook()
case 3:
    updatebook()
case 4:
    viewbook()
case 5:viewallbooks()
case 6:
    print("Thx for using Program")
    break
case _:
    print("Ur Selection of Operation is wrong--try again")
except ValueError:
    print("Don't alnums,strs and symbols for Choice--try again")
```

```
#LibraryUpdate.py<----File Name and Module Name
import oracledb as orc
def updatebook():
    while (True):
        try:
            con = orc.connect("system/tiger@127.0.0.1/orcl")
            cur = con.cursor()
            # Get employee Number from KBD
            bno = int(input("Enter Book Number for updating Book Name, Price and Publication:"))
            bname = input("Enter Book Name:")
            price = float(input("Enter Book Price:"))
            pub=input("Enter Book Publication:")
            cur.execute("update library set bname='%s',price=%f,pub='%s' where bno=%d"
            %(bname,price,pub,bno))
            con.commit()
            if (cur.rowcount > 0):
                print("{} Record Updated--verify".format(cur.rowcount))
            else:
                print("{} Record Does Not Exist".format(bno))
            print("-----")
            ch = input("Do u want to Update another BOOK(yes/no):")
            if (ch.lower() == "no"):
                print("Thx for Using Program")
                break
            if(ch.lower()!="yes"):
                print("Plz Learning Typing")
                break
        except orc.DatabaseError as db:
            print("Problem in Oracle in DB:", db)
        except ValueError:
            print("Don't enter non-int/float value for bno,price")
```

```

#LibaryView.py----File Name and Module Name
import oracledb as orc
def viewallbooks():
    try:
        con=orc.connect("system/tiger@localhost/orcl")
        cur=con.cursor()
        #Read the Records from Employee Table
        sq="select * from library"
        cur.execute(sq)
        #get the col names
        print("-----")
        metadata=cur.description
        for colnames in metadata:
            print(colnames[0],end="\t")
        print()
        print('-----')
        records = cur.fetchall()
        for record in records:
            for val in record:
                print("{}{}".format(val), end="\t")
            print()
        print('-----')
    except orc.DatabaseError as db:
        print("Problem in Oracle DB:",db)

def viewbook():
    try:
        con=orc.connect("system/tiger@localhost/orcl")
        cur=con.cursor()
        #Read the Book Number from KBD
        bno=int(input("Enter Book Number:"))
        sq="select * from Library where bno=%d"
        cur.execute(sq %bno)
        record=cur.fetchone()
        #get the records
        print("-" * 50)
        if(record!=None):
            print("Book Number:{}".format(record[0]))
            print("Book Name:{}".format(record[1]))
            print("Book Price:{}".format(record[2]))
            print("Book Publication:{}".format(record[3]))
        else:
            print("{} Book Number Does not Exist".format(bno))
        print("-" * 50)

    except orc.DatabaseError as db:
        print("Problem in Oracle DB:",db)

```

DAY-85 24/06/2024 MONDAY

```

#Program for Demonstrating How to get the Connection from MySQL Database
#MySQLConnTestEx1.py
import mysql.connector
try:
    con=mysql.connector.connect(host="localhost",
                                user="root",
                                passwd="root")
    print("Python Program got connection from MySQL")

```



```

cur=con.cursor()
#Get employee Number from KBD
empno=int(input("Enter Employee Number:"))
cur.execute("delete from employee where eno=%d" %empno)
con.commit()
if(cur.rowcount>0):
    print("{} Record Deleted--verify".format(cur.rowcount))
else:
    print("{} Record Does Not Exist".format(empno))
print("-----")
ch=input("Do u want to delete another record(yes/no):")
if(ch.lower()=='no'):
    print("Thx for Using Program")
    break
except mysql.connector.DatabaseError as db:
    print("Problem in MySQL in DB:",db)
except ValueError:
    print("Don't enter non-int value for empno")

```

#Main Program
deleterecord()

```

#OracleRecordInsertEx2.py
import mysql.connector
def insertrecord():
    while(True):
        try:
            con = mysql.connector.connect(host="127.0.0.1",
                                           user="root",
                                           passwd="root",
                                           database="4pmbatch")
            cur=con.cursor()
            print("-----")
            #Get the employee values from KBD
            empno=int(input("Enter Employee Number:"))
            empname=input("Enter Employee Name:")
            empsal=float(input("Enter Employee Salary:"))
            cname=input("Enter Employee Comp Name:")
            print("-----")
            iq="insert into employee values(%d,'%s',%f,'%s')"
            cur.execute(iq %(empno,empname,empsal,cname))
            #OR cur.execute("insert into employee values(%d,'%s',%f,'%s')"
            %(empno,empname,empsal,cname))
            con.commit()
            print("{} Record Inserted--verify".format(cur.rowcount))
            print("-----")
            ch=input("Do u want to Insert another record(yes/no):")
            if(ch.lower()=='no'):
                print("Thx for Using this Program")
                break
        except mysql.connector.DatabaseError as db:
            print("Problem in MySQL in DB:",db)
        except ValueError:
            print("Don't enter empno and salary as Non-Int/float value")

```

#Main Program
insertrecord()

#MySQLRecordUpdateEx.py

```

import mysql.connector,sys
def updaterecord():
    while(True):
        try:
            con = mysql.connector.connect(host="127.0.0.1",
                                          user="root",
                                          passwd="root",
                                          database="4pmbatch")
            cur=con.cursor()
            #Get employee Number from KBD
            empno=int(input("Enter Employee Number for updating emp sal:"))
            newsal=float(input("Enter New Salary of Employee:"))
            cname = input("Enter New Comp Name of Employee:")
            cur.execute("update employee set sal=%f,compname='%s' where eno=%d"
            %(newsal,cname,empno))
            con.commit()
            if(cur.rowcount>0):
                print("{} Record Updated--verify".format(cur.rowcount))
            else:
                print("{} Record Does Not Exist".format(empno))
            print("-----")
        while(True):
            ch=input("Do u want to Update another record(yes/no):")
            if(ch.lower()=='no'):
                print("Thx for Using Program")
                sys.exit()
            if(ch.lower()=='yes'):
                break
        except mysql.connector.DatabaseError as db:
            print("Problem in Oracle in DB:",db)
        except ValueError:
            print("Don't enter non-int/float value for empno,salary")

```

#Main Program
updaterecord()

```

#Program for Demonstrating How to create table in 4pmbatch database of MySQL
#Note: When we are creating the Table, u must mention the Database Name during Connection
establishment
#MySQLTableCreate.py
import mysql.connector
def createtable():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="4pmbatch")
        cur=con.cursor()
        #Query for creating Table
        tc="create table student(sno int primary key,name varchar(10) ,marks float,colpname varchar(10))"
        cur.execute(tc)
        print("Table Created Sucessfully--verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL: ", db)

```

#Main Program
createtable()

```

#MySQLTableRecordsColNamesEx.py
import mysql.connector
def selectrecordscolnames():

```

```

try:
    con = mysql.connector.connect(host="127.0.0.1",
                                  user="root",
                                  passwd="root",
                                  database="4pmbatch")
    cur=con.cursor()
    #Read the Records from Employee Table
    sq="select * from %s" %input("Enter Table Name:")
    cur.execute(sq)
    #get the col names
    print("-----")
    metadata=cur.description
    for colnames in metadata:
        print(colnames[0],end="\t\t")
    print()
    print('-----')
    #get records
    records=cur.fetchall()
    for record in records:
        for val in record:
            print("{}".format(val),end="\t\t")
        print()
        print('-----')
except mysql.connector.DatabaseError as db:
    print("Problem in MySQL DB:",db)

```

#Main Program
selectrecordscolnames()

```

#MySQLTest.py
import mysql.connector
print(mysql.connector.__version__)

```

=>Accept Sno,name,Marks in 3 Subjects such as C Marks(100), C++ Marks(100), Python Marks(100)

=>Calculate totmarks (totmarks=cm+ccpm+pythonm)

=>Calculate percentage of marks (percent=(totmarks/300)*100)

=>Decide the Grade

- i) If any student secured less than 40 in any one of the subject then give grade as "FAIL"
- ii) if totmarks ranges between 250 and 300 then Give grade as "DISTINCTION"
- iii) if totmarks ranges between 200 and 249 then Give grade as "FIRST"
- iv) if totmarks ranges between 150 and 199 then Give grade as "SECOND"
- v) if totmarks ranges between 120 and 149 then Give grade as "THIRD"

=>Save sno,sname,cm,ccpm,pythonm,totmarks,percent,grade as a record in RESULT Table

RESULT<----Table Name

SNO	NAME	CM	CPPM	PYTHONM	TOTMARKS	PERCENT
GRADE						

DAY-86 25/06/2024 TUESDAY

Object Oriented Principles or Features or Concepts (11 days)

Index:

=>What are the advantages of OOPs

=>List of Object Oriented Principles

1. Classes

2. Objects
 3. Data Encapsulation
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing(already discussed)
-

1. Classes

- =>Importance and purpose of Classes concept
 - =>Syntax for Defining Class
 - =>Types of Data Members
 - a) Instance Data Members
 - b) Class Level Data Members
 - =>Types of Methods
 - a) Instance Methods
 - b) Class Level Methods
 - c) Static Methods
 - =>What is "self" and "cls"
 - =>Programming Examples
-

2. Object:

- =>Importance and purpose of Object Concept
 - =>Syntax for creating Object
 - =>Programming Examples
 - =>PRAMMING Examples related to
 - Pickling and Un-Pickling
 - Data base communication with Classes and objects.
-

- =>Constructors in OOPs
 - =>Importance and purpose of Constructors
 - =>Types of Constructors
 - a) Default Constructors
 - b) Parameterized Constructors
 - =>Rules for Using Constructors
 - =>Programming Examples
-

- =>Destructrors in OOPs with Garbage Collector
 - =>Importance and purpose of Detstructrors
 - =>Syntax for defining Detstructrors
 - =>Internal flow of Detstructrors
 - =>Relationship between Detstructrors and Garbage Collector
 - =>gc module
 - =>Progammming Examples
-

- (3) Data Encapsulation and (4) Data Abstraction
 - =>Importance and purpose of Data Encapsulation
 - =>Importaance and purpose of Data Abstraction
 - =>Implementation of Data Ecapsulation and Data Abstraction
 - =>Programming Examples
-

5. Inheritance

- =>Importaance and purpose of Inheritance
- =>Types of Inheritances
 - a) single
 - b) multi level
 - c) hierarchical

- d) multiple
- e) Hybrid

=>Syntax for using Inheritance in Python

=>Programming Examples

Method Overriding in OOPs

- =>Importance and purpose of Method Overriding
- =>Memory management in Method Overriding
- =>Programming Examples

Constructor Overriding in OOPs

- =>Importance and purpose of Constructor Overriding
- =>Memory management in Constructor Overriding
- =>Programming Examples

6. Polymorphism

- =>Importance and purpose of Polymorphism
- =>Difference between Polymorphism and Inheritance
- =>Method Overriding with Polymorphism
- =>Constructor Overriding with Polymorphism
- =>`super()` and class name approaches in Polymorphism
- =>Programming Examples

===== Introduction to Object Oriented Principles or Features or Concepts =====

=>In Real Time, to develop any project, we need to choose a Language.

=>The Language, which we select for developing the Project can satisfy Two Types of Principles. They are

1. Procedure OR Functional Oriented Principles
2. Object Oriented Principles

=>In Otherwords, In Industry, we have Two Types of Programming Languages. They are

1. Procedure OR Functional Oriented Programming Languages.
2. Object Oriented Programming Languages.

1. Procedure OR Functional Oriented Programming Languages:

=>If Programming Language satisfies Procedure OR Functional Oriented Principles then that Languages are called Procedure OR Functional Oriented Programming Language.

=>Examples: C , Pascal,COBOL,PYTHON,upto Oracle7.3...etc

2. Object Oriented Programming Languages

=>If Programming Language satisfies Object Oriented Principles then that Languages are called Object Oriented Programming Language.

=>Examples: Lisp,Small Talk,Ruby,C++,Java,C#.net, PYTHON,from Oracle 8 onwards...etc

=>Even though PYTHON Belongs to Both Functional and Object Oriented Programming Language, Internally Every Thing treated as Object.

Every Thing in Python is an object--Advantages
(OR)

qAdvantages of OOPs

- *****
1. It Stores Unlimited amount Data.
 2. The large of Volume of Data Transferred between Multiple Remote Machines all at once and gets Effective Communication.
 3. The confidential Data Transferred between Multiple Remote Machines in the form of Cipher Text. So that Security Enhanced (improved)
 4. The Data always available around the objects and gives Less Memory

5. We can Build High end Re-Usable Applications by using Inheritance

Object Oriented Principles or Features or Concepts in Python

=>To say a Programming Language is Object Oriented then It has to satisfy the following Principles.

1. Classes
2. Objects
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism
7. Message Passing (already discussed)

DAY-87 26/06/2024 WEDNESDAY

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 + To Perform
Customized Operations"

=>To Develop Programmer-Defined Data Type by using Classes Concept, we use a key word called "class".

=>In Python Programming, Class Names are treated as Programmer-Defined Data Types.

=>In OOPS, All Programs Must start with Class Names (i.e without ClassesnConcept, we can't develop any program
or Application)

Definie Class

=>A Class is a Collection of Data Members and Methods (Functions in OOPs are called Methods).

=>When we define a Class, There is no memory space is created for Data Members and Methods But
whose Memory

Space is created when we create an Object w.r.t Class Name.

=>What are all the Data Members and Methods Available in the Class, which are appearing as it is as a Part
of Object with memory space. So that Programmers can store the Data in Data Members of Object and
Performs Operations on the Data Members of Object by using Methods.

Syntax for Defining a Class in Python

=>To Define a Class in Python, we use the following Syntax:

```
class <clsname>:  
    Class Level Data Members  
    def InstanceMethodName(self,list of formal params if any):  
        Specify Instance Data Members  
        Perform Specific Operations on Objects  
  
        @classmethod  
        def ClassLevelMethodName(cls,list of formal params if any):  
            Specify Class Level Data Memebrs  
            Performs Class Level Operations  
  
            @staticmethod  
            def StaticMethodName(list of formal params if any):
```

Performs Universal / Utility Operations

def __init__(self, list of formal params if any):

Block of Statements--Initializes the Object

def __del__(self):

Block of Statements--Destroys the Object Memory space

=====
NOTE:
=====

In Side of Class --what things Present

1. Data Members

- a) Instance Data Members
- b) Class Level Data Members

2. Methods

- a) Instance Methods
- b) Class Level Methods
- c) Static Methods

3. Constructors

- a) Default OR Parameter-Less Constructor
- b) Parameterized Constructor

4. Destructor

DAY-88 27/06/2024 THURSDAY

Types of Data Members in Class of Python

=>In a Class of Python, we can define Two Types of Data Members. They are

- 1. Instance Data Members
- 2. Class Level Data Members

1. Instance Data Members

=>Instance Data Members are those which are used for Storing Specific Values.

=>Instance Data Members are those whose Memory Space Created Every Time when we create an object and hence

Instance Data Members are called Object Level Data Members.

=>Programmatically, Instance Data Members can be defined in 3 Places.

- i) Through the Object of a Class
- ii) Inside of Instance Method Definition
- iii) Inside of Constructor Definition

=>Programmatically, Instance Data Members Must be accessed w.r.t Object Name or self

Syntax-1: ObjectName.Instance Data Member Name

Syntax-2: self.Instance Data Member Name (To be used Only Inside of Instance Method But not in Other places)

2. Class Level Data Members

=>Class Level Data Members are those which are used for Storing Common Values for all Objects of Same Class.

=>Class Level Data Members are those whose Memory Space Created Only Once Irrespective of Number of Objects

are created w.r.t a Class

=>Programmatically Class Level Data Members can be defined in 2 Places.

- i) Inside of Class Definition

ii) Inside of Class Level Method Definition

=>Programmatically, Class Level Data Members can be accessed with the following ways

Syntax-1: Class Name.Class Level Data Member Name

Syntax-2: cls.Class Level Data Member Name(To be used Only Inside of Class Level Method But not in Other

places)

Syntax-3: ObjectName.Class Level Data Member Name

Syntax-4: self.Class Level Data Member Name(To be used Only Inside of Instance Level Method But not in Other

places)

#Program for Demonstrating Class Level Data Members

#ClassLevelDataMemberEx1.py

class Student:

 crs="PYTHON" # here crs is called Class Level Data Member

#Main Program

s1=Student()

s2=Student()

print("Content of s1=",s1.__dict__) # {}

print("Content of s2=",s2.__dict__)# {}

Here crs is not a part of s1 and s2

#Here we are accessing Class Level Data member w.r.t Class Name

print("Content of s1-Class Level data Member=",Student.crs)

print("Content of s2-Class Level data Member=",Student.crs)

print("-----OR-----")

#Here we are accessing Class Level Data member w.r.t Object Name

print("Content of s1-Class Level data Member=",s1.crs)

print("Content of s2-Class Level data Member=",s1.crs)

#Program for Demonstrating Instance Data Members and Class Level Data Members

#ClassLevelDataMemberEx2.py

class Student:

 crs="PYTHON"

 city="HYD" # Here crs and city are called Class Level Data Members

#Main Program

s1=Student() # Object Creation--here s1 is an object

s2=Student() # Object Creation--here s2 is an object

print("-"*50)

print("Id of s1=",id(s1))

print("Id of s2=",id(s2))

print("-"*50)

#Add the Student Details to s1--Instance Data Members

s1.sno=10

s1.sname="Rossum"

s1.marks=45.67 # here sno,sname and marks are called Instance Data Members

#Add the Student Details to s2--Instance Data Members

s2.sno=20

s2.sname="Travis"

s2.marks=56.78

print("First Student Details")

print("\tStudent Number:",s1.sno)

print("\tStudent Name:",s1.sname)

print("\tStudent Marks:",s1.marks)

print("\tStudent Course:",Student.crs)

print("\tStudent City:",Student.city)

print("-"*50)

print("Second Student Details")

```
print("\tStudent Number:",s2.sno)
print("\tStudent Name:",s2.sname)
print("\tStudent Marks:",s2.marks)
print("\tStudent Course:",Student.crs)
print("\tStudent City:",Student.city)
print("-"*50)
```

```
#Program for Demostrating Instance Data Members and Class Level Data Members
#ClassLevelDataMemberEx2.py
class Student:
    crs="PYTHON"
    city="HYD" # Here crs and city are called Class Level Data Members

#Main Program
s1=Student() # Object Creation--here s1 is an object
s2=Student() # Object Creation--here s2 is an object
print("-"*50)
print("Id of s1=",id(s1))
print("Id of s2=",id(s2))
print("-"*50)
#Add the Student Details to s1--Instance Data Members
s1.sno=10
s1.sname="Rossum"
s1.marks=45.67 # here sno,sname and marks are called Instance Data Members
#Add the Student Details to s2--Instance Data Members
s2.sno=20
s2.sname="Travis"
s2.marks=56.78
print("First Student Details")
print("\tStudent Number:",s1.sno)
print("\tStudent Name:",s1.sname)
print("\tStudent Marks:",s1.marks)
print("\tStudent Course:",s1.crs)
print("\tStudent City:",s1.city)
print("-"*50)
print("Second Student Details")
print("\tStudent Number:",s2.sno)
print("\tStudent Name:",s2.sname)
print("\tStudent Marks:",s2.marks)
print("\tStudent Course:",s2.crs)
print("\tStudent City:",s2.city)
print("-"*50)
```

```
#DynamicInstanceClassLevelDataMembers1.py
class Student:
    crs="PYTHON"
    city="HYD" # Here crs and city are called Class Level Data Members

#Main Program
s1=Student() # Object Creation--here s1 is an object
s2=Student() # Object Creation--here s2 is an object
print("-"*50)
print("Id of s1=",id(s1))
print("Id of s2=",id(s2))
print("-"*50)
#Add the Student Details to s1--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:")) # here sno,sname and marks are called Instance Data Members
print("-"*50)
```

```
#Add the Student Details to s2--Instance Data Members
s2.sno=int(input("Enter Second Student Number:"))
s2.sname=input("Enter Second Student Name:")
s2.marks=float(input("Enter Second Student Marks:")) # here sno,sname and marks are called Instance Data Members
print("First Student Details")
print("\tStudent Number:",s1.sno)
print("\tStudent Name:",s1.sname)
print("\tStudent Marks:",s1.marks)
print("\tStudent Course:",s1.crs)
print("\tStudent City:",s1.city)
print("-"*50)
print("Second Student Details")
print("\tStudent Number:",s2.sno)
print("\tStudent Name:",s2.sname)
print("\tStudent Marks:",s2.marks)
print("\tStudent Course:",s2.crs)
print("\tStudent City:",s2.city)
print("-"*50)
```

```
#Program for Demostrating Instance Data Members
#InstanceDataMemberEx1.py
class Student:pass

#Main Program
s1=Student() # Object Creation--here s1 is an object
s2=Student() # Object Creation---here s2 is an object
print("-"*50)
print("Id of s1=",id(s1))
print("Id of s2=",id(s2))
print("-"*50)
#Add the Student Details to s1--Instance Data Members
s1.sno=10
s1.sname="Rossum"
s1.marks=45.67 # here sno,sname and marks are called Instance Data Members
#Add the Student Details to s2--Instance Data Members
s2.sno=20
s2.sname="Travis"
s2.marks=56.78
print("First Student Details")
print("\tStudent Number:",s1.sno)
print("\tStudent Name:",s1.sname)
print("\tStudent Marks:",s1.marks)
print("-"*50)
print("Second Student Details")
print("\tStudent Number:",s2.sno)
print("\tStudent Name:",s2.sname)
print("\tStudent Marks:",s2.marks)
print("-"*50)
```

```
#Program for Demostrating Instance Data Members
#InstanceDataMemberEx2.py
class Student:pass
```

```
#Main Program
s1=Student() # Object Creation--here s1 is an object
s2=Student() # Object Creation---here s2 is an object
print("-"*50)
print("Id of s1=",id(s1))
print("Id of s2=",id(s2))
```

```
print("-"*50)
print("Initial Content of s1={} and No. of Values={}".format(s1.__dict__,len(s1.__dict__)))
print("Initial Content of s2={} and No. of Values={}".format(s2.__dict__,len(s2.__dict__)))
print("-"*50)
#Add the Student Details to s1--Instance Data Members
s1.sno=10
s1.sname="Rossum"
s1.marks=45.67 # here sno,sname and marks are called Instance Data Members
#Add the Student Details to s2--Instance Data Members
s2.stno=20
s2.stname="Travis"
s2.marks=56.78
s2.cname="OUCET"
print("-"*50)
print("Now Content of s1={} \nNo. of Values={}".format(s1.__dict__,len(s1.__dict__)))
print("Now Content of s2={} \nNo. of Values={}".format(s2.__dict__,len(s2.__dict__)))
print("-"*50)
```

```
#Program for Demonstrating Instance Data Members
#InstanceDataMemberEx2.py
class Student:pass

#Main Program
s1=Student() # Object Creation--here s1 is an object
s2=Student() # Object Creation--here s2 is an object
print("-"*50)
print("Id of s1=",id(s1))
print("Id of s2=",id(s2))
print("-"*50)
print("Initial Content of s1={} and No. of Values={}".format(s1.__dict__,len(s1.__dict__)))
print("Initial Content of s2={} and No. of Values={}".format(s2.__dict__,len(s2.__dict__)))
print("-"*50)
#Add the Student Details to s1--Instance Data Members
s1.sno=10
s1.sname="Rossum"
s1.marks=45.67 # here sno,sname and marks are called Instance Data Members
s1.crs="PYTHON"
#Add the Student Details to s2--Instance Data Members
s2.stno=20
s2.stname="Travis"
s2.marks=56.78
s2.cname="OUCET"
s2.crs="PYTHON"
print("First Student Details")
for idmn,idv in s1.__dict__.items():
    print("\t{}--->{}".format(idmn,idv))
print("-"*50)
print("Second Student Details")
for idmn,idv in s2.__dict__.items():
    print("\t{}--->{}".format(idmn,idv))
print("-"*50)
#NOTE
# Here s1.crs="PYTHON" and s2.crs="PYTHON" is not recommended to write
# bcoz It is not recommended to write common multiple times and leads More memory space
```

DAY-89 28/06/2024 FRIDAY

Types of Methods in Class of Python

=>In a class of Python programming, we can define 3 Types of Methods. They are

-
1. Instance Methods
 2. Class Level Methods
 3. Static Methods
-

1. Instance Methods

=>Instance Methods are those which are used for performing Specific Operations on Objects and Hence Instance Methods are also called Object Level Methods.

=>The Syntax for Defining Instance Methods is

```
def instancemethodname(self,list of formal params if any):
```

Specify Instance Data Members
Perform Specific Operations

=>Instance Methods Must accessed w.r.t Object Name OR self.

objectname.InstanceMethodName()

(OR)

self.InstanceMethodName() (To be used Inside of Instance

Method Only)

=>What is "self"

=>"self" is one of the implicit object and it contains Address of Current Object

=>"self" always to be used as First Formal Parameter in Instance Method.

=>Since "self" is a Formal parameter, so that it can access inside of Corresponding Instance Method Definition only but not possible to access in other part of the Program.

2. Class Level Methods

=>Class Level Methods are used for Performing Common Operation on all objects of Same Class.

=>The Syntax for Defining Class Level Method is

```
@classmethod  
def classlevelmethodname(cls , List of formal Params if any):
```

Performs Common Operations for the Objects of same Class.
Specify Class Level Data Members

=>All Class Level Methods must be accessed w.r.t Class Name OR Object Name OR cls OR self

Syntax: clsname.ClassLevelMethodName()
OR
Syntax: cls.ClassLevelMethodName()
OR
Syntax: objectname.ClassLevelMethodName()
OR
Syntax: self.ClassLevelMethodName()

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 other part of Program.

3. Static Methods

=>Static Methods are used for performing Universal Operations or Utility Operations

=>Static Methods definition must be preceded with a predefined decorator called

@staticmethod and it never takes "cls" or "self" but always takes object of other classes.

=>The Syntax for Static Method is

```
@staticmethod  
def staticmethodname(list of Formal Params):
```

Utility Operation / Universal Operations

=>Static Methods can be accessed w.r.t Class Name OR object name OR cls OR self

```
ClassName.static method name()  
(OR)  
ObjectName.static method name()  
(OR)  
cls.static method name()  
(OR)  
self.static method name()
```

#Program for Demonstrating Instance Methods

#InstanceMethodEx1.py

```
class Student:
```

```
    def readstudvalues(self):  
        self.sno=int(input("Enter Student Number:"))  
        self.name = input("Enter Student Name:")  
        self.marks=float(input("Enter Student Marks:"))  
    def dispstudvalues(self):  
        print("-"*50)  
        print("\tStudent Number=",self.sno)  
        print("\tStudent Name:",self.name)  
        print("\tStudent Marks:",self.marks)  
        print("-" * 50)
```

#Main Program

```
s1=Student()
```

```
s2=Student()
```

#read the data dynamically for s1 from KeyBoard through Instance Method

```
s1.readstudvalues()
```

```
print("-----")
```

```
s2.readstudvalues()
```

```
print("-----")
```

```
print("First Student Details")
```

```
s1.dispstudenvalues()
```

```
print("Second Student Details")
```

```
s2.dispstudenvalues()
```

#Program for Demonstrating Instance Methods

#InstanceMethodEx2.py

```
class Student:
```

```
    def readstudvalues(self):  
        self.sno=int(input("Enter Student Number:"))  
        self.name = input("Enter Student Name:")  
        self.marks=float(input("Enter Student Marks:"))  
    #One Instance of Current Class calling another Instance Instance  
    #Of Same Class by using "self"  
    self.dispstudenvalues()
```

```
    def dispstudvalues(self):
```

```
        print("-"*50)  
        print("\tStudent Number=",self.sno)  
        print("\tStudent Name:",self.name)  
        print("\tStudent Marks:",self.marks)  
        print("-" * 50)
```

#Main Program

```
s1=Student()
```

```
s2=Student()
```

```
s1.readstudvalues()
s2.readstudvalues()
-----
#Program for adding of Two Numbers
#InstanceMethodEx3.py
class Sumop:
    def getvalues(self):
        so.a=float(input("Enter First Value:"))
        so.b = float(input("Enter Second Value:"))
    def addvalues(self):
        self.c=self.a+self.b
    def dispvalues(kvr):
        print("First value={}".format(kvr.a))
        print("Second Value:{}".format(kvr.b))
        print("Sum={}".format(kvr.c))
```

```
#Main Program
so=Sumop() # Object Creation
so.getvalues()
so.addvalues()
so.dispvalues()
```

```
#Program for adding of Two Numbers
#InstanceMethodEx3.py
class Sumop:
    def getvalues(self):
        so.a=float(input("Enter First Value:"))
        so.b = float(input("Enter Second Value:"))
    def addvalues(self):
        self.c=self.a+self.b
    def dispvalues(kvr):
        kvr.getvalues()
        kvr.addvalues()
        print("First value={}".format(kvr.a))
        print("Second Value:{}".format(kvr.b))
        print("Sum={}".format(kvr.c))
```

```
#Main Program
so=Sumop() # Object Creation
so.dispvalues()
```

```
#This Program Demonstrates Class Level Methods
#ClassLevelMethodEx1.py
class Employee:
    @classmethod
    def getcompdet(cls): # Class Level Method
        cls.compname="WIPRO"
        Employee.city="HYD"
    @classmethod
    def dispcompdet(cls):
        print("Comp Name:{}".format(cls.compname))
        print("Comp City={}".format(cls.city))
        print("-----OR-----")
        print("Comp Name:{}".format(Employee.compname))
        print("Comp City={}".format(Employee.city))
#Main Program
Employee.getcompdet()
```

```
Employee.dispcompdet()
```

```
#This Program Demonstrates Class Level Methods
#ClassLevelMethodEx2.py
class Employee:
    @classmethod
    def getcompdet(cls): # Class Level Method
        cls.compname="WIPRO"
        Employee.city="HYD"
    @classmethod
    def dispcompdet(cls):
        cls.getcompdet() # Calling Class level Method w.r.t cls
        print("Comp Name:{}".format(cls.compname))
        print("Comp City={}".format(cls.city))
        print("-----OR-----")
        print("Comp Name:{}".format(Employee.compname))
        print("Comp City={}".format(Employee.city))
#Main Program
```

```
Employee.dispcompdet()
```

```
#This Program Demonstrates Class Level Methods
#ClassLevelMethodEx3.py
class Employee:
    @classmethod
    def getcompdet(cls): # Class Level Method
        cls.compname="WIPRO"
        Employee.city="HYD"
    @classmethod
    def dispcompdet(cls):
        cls.getcompdet() # Calling Class level Method w.r.t cls
        print("Comp Name:{}".format(cls.compname))
        print("Comp City={}".format(cls.city))
        print("-----OR-----")
        print("Comp Name:{}".format(Employee.compname))
        print("Comp City={}".format(Employee.city))
#Main Program
eo=Employee() # Object Creation
eo.dispcompdet() # Calling Class level method w.r.t object name
```

```
#This Program Demonstrates Class Level Methods
#ClassLevelMethodEx3.py
class Employee:
    @classmethod
    def getcompdet(cls): # Class Level Method
        cls.compname="WIPRO"
        Employee.city="HYD"
    @classmethod
    def dispcompdet(cls):
        print("Comp Name:{}".format(cls.compname))
        print("Comp City={}".format(cls.city))
        print("-----OR-----")
        print("Comp Name:{}".format(Employee.compname))
        print("Comp City={}".format(Employee.city))
    def printcompdet(self): # Instance Method
```

```
self.getcompdet() # Calling Class level Method w.r.t self  
self.dispcompdet() # Calling Class level Method w.r.t self
```

```
#Main Program  
eo=Employee() # Object Creation  
eo.printcompdet()
```

```
#ClassLevelInstanceMethodEx1.py  
class Student:  
    @classmethod  
    def getunivdet(cls): # Class Level Method  
        cls.uname = "JNTU"  
        Student.city = "HYD"  
    def readstudvalues(self):  
        self.sno=int(input("Enter Student Number:"))  
        self.name = input("Enter Student Name:")  
        self.marks=float(input("Enter Student Marks:"))  
    def dispstudvalues(self):  
        self.readstudvalues() # Calling Instance Method w.r.t self  
        self.getunivdet() # Calling Class Level Method w.r.t self  
        print("-" * 50)  
        print("\tStudent Number=",self.sno)  
        print("\tStudent Name:",self.name)  
        print("\tStudent Marks:",self.marks)  
        print("\tStudent univ Name:",Student.uname)  
        print("\tstudent City:",Student.city)  
        print("-" * 50)  
#Main Program  
s=Student()  
s.dispstudenvalues()
```

DAY-90 29/06/2024 SATURDAY

```
#Program for Demonstrating Static Methods  
#StaticMethodEx1.py  
class Student:  
    def getstuddet(self):  
        self.sno=int(input("Enter Student Number:"))  
        self.sname=input("Enter Student Name:")  
        self.marks=float(input("Enter Student Marks:"))  
class Employee:  
    def getempdet(self):  
        self.eno=int(input("\nEnter Employee Number:"))  
        self.ename=input("Enter Employee Name:")  
class Teacher:  
    def getteacherdet(self):  
        self.tno=int(input("\nEnter Teacher Number:"))  
        self.tname=input("Enter Teacher Name:")  
        self.expr=float(input("Enter Teacher Exp:"))  
        self.subject = input("Enter Teacher Subject:")  
class Hyd:  
    @staticmethod  
    def dispobjectdata(objdata,objinfo):  
        print("-" * 40)  
        print("{} Information".format(objinfo))  
        print("-" * 40)  
        for key,value in objdata.__dict__.items():  
            print("\t{}-->{}".format(key,value))
```

```

        print("-" * 40)
#Main Program
s=Student()
e=Employee()
t=Teacher()
#
s.getstuddet()
e.getempdet()
t.getteacherdet()
#Today My Requirement is to display any object content by single method
# This type single method is called Static Method
#We are calling Static method w.r.t Class Name
Hyd.dispobjectdata(s,"Student")
Hyd.dispobjectdata(e,"Employee")
Hyd.dispobjectdata(t,"Teacher")
-----
#Program for Demonstrating Static Methods
#StaticMethodEx2.py
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.expr=float(input("Enter Teacher Exp:"))
        self.subject = input("Enter Teacher Subject:")
class Hyd:
    @staticmethod
    def dispobjectdata(objdata,objinfo):
        print("-"*40)
        print("{} Information".format(objinfo))
        print("-" * 40)
        for key,value in objdata.__dict__.items():
            print("\t{}-->{}".format(key,value))
        print("-" * 40)
#Main Program
s=Student()
e=Employee()
t=Teacher()
#
s.getstuddet()
e.getempdet()
t.getteacherdet()
#Today My Requirement is to display any object content by single method
# This type single method is called Static Method
#We are calling Static method w.r.t object Name
h=Hyd()
h.dispobjectdata(s,"Student")
h.dispobjectdata(e,"Employee")
h.dispobjectdata(t,"Teacher")
-----
#Program for Demonstrating Static Methods
#StaticMethodEx3.py
class Student:

```

```

def getstuddet(self):
    self.sno=int(input("Enter Student Number:"))
    self.sname=input("Enter Student Name:")
    self.marks=float(input("Enter Student Marks:"))
class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.expr=float(input("Enter Teacher Exp:"))
        self.subject = input("Enter Teacher Subject:")
class Hyd:
    @staticmethod
    def dispobjectdata(objdata,objinfo):
        print("-" * 40)
        print("{} Information".format(objinfo))
        print("-" * 40)
        for key,value in objdata.__dict__.items():
            print("\t{}-->{}".format(key,value))
        print("-" * 40)
#Main Program
s=Student()
e=Employee()
t=Teacher()
#
s.getstuddet()
e.getempdet()
t.getteacherdet()
#Today My Requirement is to display any object content by single method
# This type single method is called Static Method
#We are calling Static method w.r.t Name-Less Object
Hyd().dispobjectdata(s,"Student")
Hyd().dispobjectdata(e,"Employee")
Hyd().dispobjectdata(t,"Teacher")
-----
```

```

#Program for Demonstrating Static Methods
#StaticMethodEx4.py
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.expr=float(input("Enter Teacher Exp:"))
        self.subject = input("Enter Teacher Subject:")
class Hyd:
    @classmethod
    def getobjdata(cls,objdata,objinfo):
        #Calling Static Method from Class Level method w.r.t cls
        cls.dispobjectdata(objdata,objinfo)
    @staticmethod
```

```

def dispobjectdata(objdata,objinfo):
    print("-"*40)
    print("{} Information".format(objinfo))
    print("-" * 40)
    for key,value in objdata.__dict__.items():
        print("\t{}-->{}".format(key,value))
    print("-" * 40)
#Main Program
s=Student()
e=Employee()
t=Teacher()
#
s.getstuddet()
e.getempdet()
t.getteacherdet()
#Today My Requirement is to display any object content by single method
# This type single method is called Static Method
#We are calling Class Level method w.r.t Name-Less Object
Hyd.getobjdata(s,"Student")
Hyd.getobjdata(e,"Employee")
Hyd.getobjdata(t,"Teacher")
-----
#Program for Demonstrating Static Methods
#StaticMethodEx4.py
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.tname=input("Enter Teacher Name:")
        self.expr=float(input("Enter Teacher Exp:"))
        self.subject = input("Enter Teacher Subject:")
class Hyd:
    def getobjdata(self,objdata,objinfo):
        #Calling Static Method from Instance method w.r.t self
        self.dispobjectdata(objdata,objinfo)
    @staticmethod
    def dispobjectdata(objdata,objinfo):
        print("-"*40)
        print("{} Information".format(objinfo))
        print("-" * 40)
        for key,value in objdata.__dict__.items():
            print("\t{}-->{}".format(key,value))
        print("-" * 40)
#Main Program
s=Student()
e=Employee()
t=Teacher()
#
s.getstuddet()
e.getempdet()
t.getteacherdet()
#Today My Requirement is to display any object content by single method
# This type single method is called Static Method

```

```

#We are calling Instance method w.r.t Name-Less Object
Hyd().getobjdata(s,"Student")
Hyd().getobjdata(e,"Employee")
Hyd().getobjdata(t,"Teacher")
-----
#program for Generating Mul table for a Given Number
#MulTableoops.py<---File Name and Acts as Module Name
class MulTable:
    def getvalue(self):
        while(True):
            try:
                self.n=int(input("Enter a Number for generating mul table:"))
                if(self.n>0):
                    break
                else:
                    print("\t{} Is Invalid Input-enter +ve val".format(self.n))
            except ValueError:
                print("\tDon't enter alnums,strs and special symbols")
    def gettable(self):
        if(self.n<=0):
            print("{} is Invalid input".format(self.n))
        else:
            print("-"*50)
            print("Mul Table for {}".format(self.n))
            print("-" * 50)
            for i in range(1,11):
                print("\t{} x {} = {}".format(self.n,i,self.n*i))
            print("-" * 50)
-----
#MulTableDemoOops.py<---Main Program
from MulTableoops import MulTable
mt=MulTable()
mt.getvalue()
mt.gettable()
-----
#Employee.py<---File Name and Module Name
class Emp:
    def getempdata(self,eno,ename,sal):
        self.eno=eno
        self.ename=ename
        self.sal=sal
    def dispempdata(self):
        print("\t{}\t{}\t{}\t{}".format(self.eno,self.ename,self.sal))
-----
#EmployeePicOoops.py<---File Name and Module Name
from Employee import Emp
import pickle
class EmployeePic:
    def saveempdata(self):
        with open("emp.data","ab") as fp:
            while(True):
                #get employee Data
                print("-"*50)
                eno=int(input("Enter Employee Number:"))
                name=input("Enter Employee Name:")
                sal=float(input("Enter Employee Salary:"))
                print("-" * 50)
                #Create our own Object
                eo=Emp()
                eo.getempdata(eno,name,sal)
                #Save eo data to the file

```

```

pickle.dump(eo,fp)
print("Employee Data Saved Sucessfully in File")
print("-" * 50)
ch=input("Do u want to Insert Another record(yes/no):")
if(ch.lower() == "no"):
    print("Thx for using This Program")
    break

#Main Program
EmployeePic().saveempdata() # Calling Instance Method w.r.t Name-Less Object

```

#EmployeeUnPicOops.py<---File Name and Module Name

```

import pickle
class EmployeeUnPic:
    def reademprecords(self):
        try:
            print("-----")
            with open("emp.data","rb") as fp:
                while(True):
                    try:
                        record = pickle.load(fp)
                        record.dispempdata()
                    except EOFError:
                        print("-----")
                        break
        except FileNotFoundError:
            print("File does not Exist")

```

#Main Program

```

EmployeeUnPic().reademprecords()

```

#Program for Reading employee Details from KBD and Insert then as records

```

#EmployeeWithOracleOops.py
import oracledb as orc
class Employee:
    def getempdata(self):
        self.eno = int(input("Enter Employee Number:"))
        self.name = input("Enter Employee Name:")
        self.sal = float(input("Enter Employee Salary:"))
    def saveempdataoracle(self):
        while(True):
            try:
                print("-----")
                self.getempdata()
                #Write PDBC Code
                con=orc.connect("system/tiger@127.0.0.1/orcl")
                cur=con.cursor()
                #prepare Insert Query
                iq="insert into emp values(%d,'%s',%f)"
                cur.execute(iq %(self.eno,self.name,self.sal))
                con.commit()
                print("Employee Records Saved Sucessfully:")
                print("-----")
                ch = input("Do u want to Insert Another record(yes/no):")
                if (ch.lower() == "no"):
                    print("Thx for using This Program")
                    break
            except orc.DatabaseError as db:
                print("Problem in Oracle db:",db)

```

Types of Methods in Class of Python

=>In a class of Python programming, we can define 3 Types of Methods. They are

1. Instance Methods
2. Class Level Methods
3. Static Methods

1. Instance Methods

=>Instance Methods are those which are used for performing Specific Operations on Objects and Hence Instance Methods are also called Object Level Methods.

=>The Syntax for Defining Instance Methods is

```
def instancemethodname(self,list of formal params if any):
```

Specify Instance Data Members
Perform Specific Operations

=>Instance Methods Must accessed w.r.t Object Name OR self.

objectname.InstanceMethodName()

(OR)

self.InstanceMethodName() (To be used Inside of Instance

Method Only)

=>What is "self"

=>"self" is one of the implicit object and it contains Address of Current Object

=>"self" always to be used as First Formal Parameter in Instance Method.

=>Since "self" is a Formal parameter, so that it can access inside of Corresponding Instance Method Definition only but not possible to access in other part of the Program.

2. Class Level Methods

=>Class Level Methods are used for Performing Common Operation on all objects of Same Class.

=>The Syntax for Defining Class Level Method is

```
@classmethod  
def classlevelmethodname(cls , List of formal Params if any):  
-----  
Performs Common Operations for the Objects of same Class.  
Specify Class Level Data Members  
-----
```

=>All Class Level Methods must be accessed w.r.t Class Name OR Object Name OR cls OR self

Syntax: clsname.ClassLevelMethodName()
OR
Syntax: cls.ClassLevelMethodName()
OR
Syntax: objectname.ClassLevelMethodName()
OR
Syntax: self.ClassLevelMethodName()

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 other part of Program.

3. Static Methods

- =>Static Methods are used for performing Universal Operations or Utility Operations
- =>Static Methods definition must be preceded with a predefined decorator called @staticmethod and it never takes "cls" or "self" but always takes object of other classes
- =>The Syntax for Static Method is

```
@staticmethod  
def staticmethodname(list of Formal Params):
```

Utility Operation / Universal Operations

=>Static Methods can be accessed w.r.t Class Name OR object name OR cls OR self

ClassName.static method name()
(OR)
ObjectName.static method name()
(OR)
cls.static method name()
(OR)
self.static method name()

DAY-91 01/07/2024 MONDAY

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 one of the special method which is automatically / Implicitly called by PVM During Object Creation and whose purpose is to initlize the object without leaving the object empty.

Syntax for defining Constructor:

```
def __init__(self, list of formal params if any):
```

Block of Statements-- Performs Initialization

Rules or Properties of Constructors

1. The Name of the constructor is always def __init__(self,.....)
 2. Constructors will be called by PVM automatically / implicitly during object creation
 3. Constructors will not return any value except None.
 4. In Python, Constructors can participate in Inheritance Process.
 5. In Python, Constructors can be Overridden .

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 Initialization Process

Example:

```
#program for demonstrating Default Constructor
#DefaultConstEx1.py
class Test:
    def __init__(self):
        print("i am from default constructor:")
        self.a=10
        self.b=20
        print("\ta={}\tb={}".format(self.a,self.b))

#main program
t1=Test()# Object creation calls default constructor
t2=Test()# Object creation calls default constructor
t3=Test()# Object creation calls default constructor
```

2. Parameterized Constructor:

=>A Parameterized Constructor is one, which always takes Formal Parameters after self.
=>The purpose of Parameterized Constructor is that " To Initialize Multiple objects of same class with Different Values".

=>Syntax: def __init__(self,list of formal params):

Block of statements----Performs Initialization Process

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
```

```

#Program for Demonstrating Class and Objects with Constructors for Initializing the Object
#ConstEx1.py
class Student:
    def __init__(self): # Default OR Parameter-Less Constructor Definition
        print("-----")
        print("I am from Default OR Parameter-Less Constructor")
        self.sno=10
        self.sname="Rossum"
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("-----")

#Main Program
s1=Student() # During the Object Creation--PVM Calls Default Constructor Implicitly / Automatically
s2=Student() # During the Object Creation--PVM Calls Default Constructor Implicitly / Automatically

#Program for Demonstrating Class and Objects with Constructors for Initializing the Object
#ConstEx2.py
class Student:
    def __init__(self,sno,sname): # Parameterized Constructor Definition
        print("-----")
        print("I am from Parameterized Constructor")
        self.sno=sno
        self.sname=sname
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("-----")

#Main Program
s1=Student(100,"Maha Lakshmi") # Object Creation--PVM Calls Parameterized Constructor
s2=Student(200,"Swathi") # Object Creation--PVM Calls Parameterized Constructor
-----
#Program for Demonstrating Default Constructor
#ConstEx3.py
class Test:
    def __init__(self): # Default Constructor
        print("-----")
        print("I am from default constructor")
        self.a=1
        self.b=2
        print("Val of a=",self.a)
        print("Val of b=",self.b)
        print("-----")

#Main Program
t1=Test() # Object Creation--PVM Calls Default Constructor
t2=Test() # Object Creation--PVM Calls Default Constructor
-----
#Program for Demonstrating Parameterized Constructor
#ConstEx4.py
class Test:
    def __init__(self,a,b): # Parameterized Constructor
        print("-----")
        print("I am from Parameterized constructor")

```

```

        self.a=a
        self.b=b
        print("Val of a=",self.a)
        print("Val of b=",self.b)
        print("-----")

#Main Program
t1=Test(10,20) # Object Creation--PVM Calls Parameterized Constructor
t2=Test(100,200) # Object Creation--PVM Calls Parameterized Constructor
t3=Test(1000,2000) # Object Creation--PVM Calls Parameterized Constructor
=====

#Program for Demonstrating Both Default and Parameterized Constructor

#ConstEx5.py
class Test:
    def __init__(self,a=1,b=2): # Parameterized Constructor
        print("-----")
        print("I am from Default / Parameterized constructor")
        self.a=a
        self.b=b
        print("Val of a=",self.a)
        print("Val of b=",self.b)
        print("-----")

#Main Program
t1=Test() # Object Creation--PVM Calls Default Constructor
t2=Test(10,20) # Object Creation--PVM Calls Parameterized Constructor
t3=Test(b=1000) # Object Creation--PVM Calls Parameterized Constructor
t3=Test(b=3000,a=2000) # Object Creation--PVM Calls Parameterized Constructor
=====

#Program for Calculating Factorial of a Number by using Classes and Object with Constructor
#FactEx.py
class Factorial:
    def __init__(self,n):
        self.n=n
    def calfact(self):
        f=1
        if(self.n<0):
            print("{} is -Ve Number-No Factorial".format(self.n))
        else:
            for i in range(1,self.n+1):
                f=f*i
            else:
                print("Factorial({})={}".format(self.n,f))

#Main Program
while(True):
    try:
        fo=Factorial(int(input("Enter a Number for Cal Factorial:"))) # Object Creation--calls
Paraneterized Constrctor
        fo.calfact()
    except ValueError:
        print("\tDon't Enter alnums,strs and symbols for Numbers for cal Factorial")
    else:
        break
=====

#Program for Demonstrating Class and Objects without using Constructors
#Non-ConstEx.py

```

```

class Student:
    def setstudvals(self):
        self.sno=10
        self.sname="Rossum"
#Main Program
s=Student() # Object Creation
print("Content of s=",s.__dict__) # { }--empty
#How do u place the Instance Data Members inside of object s--by using Instance Method
#s.setstudvals() # For Setting the values in object s--we are calling a Method Explicitly
print("Content of s=",s.__dict__) # { 'sno':10,'sname':'RS'}--Non-Empty
-----!

```

#Program for Demonstrating Class and Objects without using Constructors

#Non-ConstEx.py

```

class Student:
    def setstudvals(self):
        self.sno=10
        self.sname="Rossum"

```

#Main Program

s=Student() # Object Creation

print("Content of s=",s.__dict__) # { }--empty

#How do u place the Instance Data Members inside of object s--by using Instance Method

#s.setstudvals() # For Setting the values in object s--we are calling a Method Explicitly

print("Content of s=",s.__dict__) # { 'sno':10,'sname':'RS'}--Non-Empty

DAY-92 02/07/2024 TUESDAY

Destructors in Python

and
Garbage Collector

=>We know that Garbage Collector is one of the in-built program in python, which is running behind of every python program and whose role is to collect un-used memory space and it improves the performance of python based applications.

=>Every Garbage Collector Program is internally calling its Own Destructor Functions.

=>The destructor function name in python is `def __del__(self)`.

=>By default ,The destructor always called by Garbage Collector when the program execution completed for de-allocating the memory space of objects which are used in that program. Where as constructor called By PVM implicitly during object is creation 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(called automatic Garbage Collection).

b) Make the object reference as `None` for calling Forcefull Garbage Collection(called Forcefull Garbage Collection)

Syntax : `objname=None`

c) delete the object by using `del` operator for calling Forcefull Garbage Collection(Called 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 by automatic enabling of GC

#Program for Demonstrating With Destructor

#DestEx1.py

import time

class Employee:

```
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}\n".format(self.eno))
        print("Employee Name:{}\n".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
```

#Main Program

print("Program Execution Started")

eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor

eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor

print("Program Execution Ended")

time.sleep(10)

#Here Garbage Collector calls its Destructor at end of the Program execution automatically and This type of GC is called Automatic Gabage collection.

#Program for Demonstrating With Destructor

#DestEx2.py

import time,sys

class Employee:

```
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("ID Current Object={}\n".format(id(self)))
        print("Employee Number:{}\n".format(self.eno))
        print("Employee Name:{}\n".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        global memspace
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
        memspace=memspace-sys.getsizeof(self)
        print("\tCurrent Object Removed:{}\n".format(id(self)))
        print("\tNow Available Memory Space={}\n".format(memspace))
```

#Main Program

```
print("Program Execution Started")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor
eo3=Employee(300,"DR") # Object Creation--PVM Calls Parameterized Constructor
#Calculate the memory space of eo1,eo2 and eo3
memspace=sys.getsizeof(eo1)+sys.getsizeof(eo2)+sys.getsizeof(eo3)
print("Total Memory Space of This Program=",memspace)
print("Program Execution Ended")
time.sleep(10)
#Here Garbage Collector calls its Destructor at end of the Program execution automatically and This type of
GC is called Automatic Gabage collection.
=====
```

```
#Program for Demonstrating With Destructor
#DestEx3.py
import time
class Employee:
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
```

```
#Main Program
print("Program Execution Started")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
print("No Longer Interested to maintain to the object eo1")
time.sleep(5)
eo1=None # GC Calls Its Destructor Forcefully to remove the memory space of eo1
eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor
print("No Longer Interested to maintain to the object eo2")
time.sleep(5)
eo2=None # GC Calls Its Destructor Forcefully to remove the memory space of eo2
eo3=Employee(300,"DR") # Object Creation--PVM Calls Parameterized Constructor
print("No Longer Interested to maintain to the object eo3")
time.sleep(5)
eo3=None
print("Program Execution Ended")
=====
```

```
#Program for Demonstrating With Destructor
#DestEx4.py
import time
class Employee:
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
```

```
#Main Program
print("Program Execution Started")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
```

```

print("No Longer Interested to maintain to the object eo1")
time.sleep(5)
del eo1 # GC Calls Its Destructor Forcefully to remove the memory space of eo1
eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor
print("No Longer Interested to maintain to the object eo2")
time.sleep(5)
del eo2 # GC Calls Its Destructor Forcefully to remove the memory space of eo2
eo3=Employee(300,"DR") # Object Creation--PVM Calls Parameterized Constructor
print("No Longer Interested to maintain to the object eo2")
time.sleep(5)
del eo3
print("Program Execution Ended")
=====

```

```

#Program for Demonstrating With Destructor
#DestEx5.py
import time
class Employee:
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
#Main Program

```

```

print("Program Execution Started")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
eo2=eo1 # Deep Copy
eo3=eo2 # Deep Copy
print(eo1.__dict__)
print(eo2.__dict__)
print(eo3.__dict__)
print("Program Execution Ended")
time.sleep(10)
#Here Garbage Collector calls its Destructor at end of the Program execution automatically and This type of
GC is called Automatic Gabage collection.
=====
```

```

#Program for Demonstrating With Destructor
#DestEx6.py
import time
class Employee:
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
#Main Program

```

```

print("Program Execution Started")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
eo2=eo1 # Deep Copy
eo3=eo2 # Deep Copy
print("No Longer Interested to maintain to the object eo1")
```

```

time.sleep(5)
del eo1 # GC will not Call Its Destructor Forcefully to remove the memory space of eo1, bcoz still eo2 and
eo3 pointing to memory space
print("No Longer Interested to maintain to the object eo2")
time.sleep(5)
del eo2 # GC will not Call Its Destructor Forcefully to remove the memory space of eo2, bcoz still eo3
pointing to memory space
print("No Longer Interested to maintain to the object eo3")
time.sleep(5)
del eo3 # GC Calls Its Destructor Forcefully to remove the memory space of eo3
print("Program Execution Ended")
=====
```

```

#GCEX1.py
import gc
print("Program Execution Started")
print("Initially, Is GC Running=",gc.isenabled())
a=10
b=20
print("a=",a)
print("b=",b)
gc.disable()
print("Now Is GC Running after disable()=",gc.isenabled())
c=a+b
print("c=",c)
gc.enable()
print("Program Execution Ended")
print("Now Is GC Running after enable()=",gc.isenabled())
=====
```

```

#Program for Demonstrating With Destructor
#GCEX2.py
import time,sys,gc
class Employee:
    def __init__(self,eno,ename): # Constructor
        print("-----")
        self.eno=eno
        self.ename=ename
        print("ID Current Object=",id(self))
        print("Employee Number:{}.".format(self.eno))
        print("Employee Name:{}.".format(self.ename))
        print("-----")
    def __del__(self): # Destructor Definition
        global memspace
        print("GC Calls __del__() for Removing the Memory Space of Current Object")
        memspace=memspace-sys.getsizeof(self)
        print("\tCurrent Object Removed:",id(self))
        print("\tNow Available Memory Space=",memspace)
```

```

#Main Program
print("-----")
print("Program Execution Started")
print("Initially, Is GC Running=",gc.isenabled())
print("-----")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor
eo3=Employee(300,"DR") # Object Creation--PVM Calls Parameterized Constructor
#Calculate the memory space of eo1,eo2 and eo3
memspace=sys.getsizeof(eo1)+sys.getsizeof(eo2)+sys.getsizeof(eo3)
print("Total Memory Space of This Program=",memspace)
gc.disable()
```

```
print("Now, Is GC Running=",gc.isenabled())
time.sleep(10)
print("Program Execution Ended")
time.sleep(10)
#Here Garbage Collector calls its Destructor at end of the Program execution automatically and This type of
GC is called Automatic Gabage collection.
```

```
#Program for Demonstrating Without using Destructor
#Non-DestEx1.py
class Employee:
    def __init__(self,eno,ename):
        print("-----")
        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")
eo1=Employee(100,"RS") # Object Creation--PVM Calls Parameterized Constructor
eo2=Employee(200,"TR") # Object Creation--PVM Calls Parameterized Constructor
print("Program Execution Ended")
```

DAY-93 03/07/2024 WEDNESSDAY

```
#Program for Demonstrating the need of Data Encapsulation--Data Member Lavel through Constructor
#Account1.py<----File Name and Module Name
class Account:
    def __init__(self):
        self.__acno=10
        self.cname="Rossum"
        self.__bal=5.6
        self.__pin=4567
        self.bname="SBI"

#Program for Demonstrating the need of Data Encapsulation--Data Member Lavel through Instance Method
#Account2.py<----File Name and Module Name
class Account:
    def getAccData(self): # Instance Method
        self.__acno=10
        self.cname="Rossum"
        self.__bal=5.6
        self.__pin=4567
        self.bname="SBI"

#Program for Demonstrating the need of Data Encapsulation--Data Member Lavel through Instance Method
#Account3.py<----File Name and Module Name
class Account:
    def __getAccData(self): # Instance Method--encapsulated
        self.acno=10
        self.cname="Rossum"
        self.bal=5.6
        self.pin=4567
        self.bname="SBI"

#Program for Demonstrating the need of Data Encapsulation
```

```
#Account3.py<----File Name and Module Name
class Account:
    def _____init____(self): # Constructor---concept can't encapsulated
        self.acno=10
        self.cname="Rossum"
        self.bal=5.6
        self.pin=4567
        self.bname="SBI"
=====
```

```
#Program for Demonstrating the need of Data Encapsulation
#Account5.py<----File Name and Module Name
class __Account: # Here Class __Account is Encapsulated
    def getAccData(self): # Instance Method--encapsulated
        self.acno=10
        self.cname="Rossum"
        self.bal=5.6
        self.pin=4567
        self.bname="SBI"
=====
```

```
#AccountDetails.py
class Account:
    def __getAccData(self): # Instance Method--encapsulated
        self.acno=10
        self.cname="Rossum"
        self.bal=5.6
        self.pin=4567
        self.bname="SBI"
    def showdetails(self):
        self.__getAccData()
        print(self.__dict__)
=====
```

```
#Main Program
ac=Account()
#ac.__getAccData()---can't access from Main Program
ac.showdetails()
=====
```

```
#AccountInfo.py
class Account:
    def __init__(self): # Constructor
        self.__acno=10
        self.cname="Rossum"
        self.__bal=5.6
        self.__pin=4567
        self.bname="SBI"
    def showdetails(self):
        print("-----")
        print("Account Number:{}".format(self.__acno))
        print("Account Name:{}".format(self.cname))
        print("Account Bal:{}".format(self.__bal))
        print("Account Pin:{}".format(self.__pin))
        print("Account Branch:{}".format(self.bname))
        print("-----")
#Main Program
ac=Account()
#print("Account Number:{}".format(self.__acno))---AttributeError
ac.showdetails()
```

```
#This Program demonstrates Data Abstraction
#Others1.py
from Account1 import Account
ac=Account()
print("-----")
#print("Account Number={}".format(ac.acno))
print("Account Holder Name:{}".format(ac.cname))
#print("Account Holder Balance:{}".format(ac.bal))
#print("Account PIN:{}".format(ac.pin))
print("Account Holder Branch Name:{}".format(ac.bname))
print("-----")
=====

#This Program demonstrates Data Abstraction
#Others2.py
from Account2 import Account
ac=Account()
ac.getAccData()
print("-----")
#print("Account Number={}".format(ac.acno))
print("Account Holder Name:{}".format(ac.cname))
#print("Account Holder Balance:{}".format(ac.bal))
#print("Account PIN:{}".format(ac.pin))
print("Account Holder Branch Name:{}".format(ac.bname))
print("-----")
=====

#This Program demonstrates Data Abstraction
#Others3.py
from Account3 import Account
ac=Account()
#ac.getAccData()--can't access, bcoz getAccData() is encapsulated
=====

#This Program demonstrates Data Abstraction
#Others4.py
from Account4 import Account
ac=Account()
ac.___init___() # Acts as Method
print("-----")
print("Account Number={}".format(ac.acno))
print("Account Holder Name:{}".format(ac.cname))
print("Account Holder Balance:{}".format(ac.bal))
print("Account PIN:{}".format(ac.pin))
print("Account Holder Branch Name:{}".format(ac.bname))
print("-----")
=====

#This Program demonstrates Data Abstraction
#Others5.py
#This Program will not execute bcoz Account Class made as Encapsulated
from Account5 import Account
ac=Account()
ac.getAccData()
print("-----")
print("Account Number={}".format(ac.acno))
print("Account Holder Name:{}".format(ac.cname))
print("Account Holder Balance:{}".format(ac.bal))
print("Account PIN:{}".format(ac.pin))
print("Account Holder Branch Name:{}".format(ac.bname))
```

```
print("-----")
```

===== **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 Instance Data Members Level
- b) At Instance Methods Level

=>To implement Data Encapsulation in python programming, The Data Members , Methods must be preceded with double under score (__).

Syntax1:- (Data member Level through method)

```
class <ClassName>:  
    def methodname(self):  
        self.__Data MemberName1=Value1  
        self.__Data MemberName2=Value2  
        .....  
        self.__Data MemberName-n=Value-n
```

(OR)

Syntax1:- (Data member Level through Constructor)

```
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 __Instancemethodname(self):  
        self.Data MemberName1=Value1  
        self.Data MemberName2=Value2  
        .....  
        self.Data MemberName-n=Value-n
```

NOTE: It is not recommended to Encapsulate Class Data Members , Class Level Methods and Static Method bcoz they are meant for Common and Universal Purpose. So that they must be Public in Access.

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 Members can be encapsulated.

Note: We can also Encapsulate Class Name But In real Time, Hiding the class name is of no use bcoz we get

ImportError.

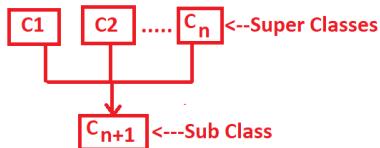
```
class __<clsname>:  
-----  
-----  
-----
```

DAY-94 04/07/2024 THURSDAY

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:

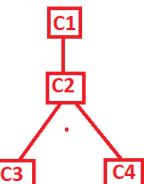
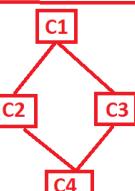


Diagram2:



Inheritance

=>Inheritance is one of distinct features of OOPs

=>The purpose of Inheritance is that " To build Re-usable Applications with Effective Memory Management 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 OR 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.

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 <classname-1>:  
-----  
-----  
class <classname-2>:  
-----  
-----  
-----  
-----  
class <classname-n>:  
-----  
-----  
  
class <classname-n+1>(classname-1,classname-2,...classname-n):  
-----  
-----  
-----
```

EXPLANATION:

=>Here <classname-n+1> Represents Name of the Derived OR Sub Class

=>Here classname-1,classname-2,...classname-n Represents Name of the Base OR Super Class

=>Here All the Data Members, Methods and Constructors are Inherited from classname-1,classname-2,...classname-n into

classname-n+1 Virtually OR Logically.

=>Here the Derived Class classname-n+1 can access the all the features of Base Classes.

=>When we develop any Inheritance Based Application, It is always Recommended to Create an object of Bottom Most

derived Class bcoz It Inherits all the Features of Base Class, intermediate Base Class(es) .

=>For Every Class in Python, there Exist an implicit Pre-Defined Super Class calss "object" and It provides Garbage Collection facility .

=====X=====

Types of Inheritances OR Re-usbale Tech in Python

=>Types of Inheritance is one of the Model / Diagram / Pattern which makes us to understand How the Features are Inherited from Base Class to Dervied 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

```
#InhProg1.py  
class C1:  
    def disp1(self):  
        print("C1--disp1()--Instance Method")  
class C2(C1):  
    def disp2(self):  
        print("C2--disp2()--Instance Method")  
class C3(C2):  
    def disp3(self):  
        print("C3--disp3()--Instance Method")  
#Main Program  
o3=C3()  
o3.disp1()  
o3.disp2()  
o3.disp3()
```

```

/-----
class C1:
    def disp1(self):
        print("C1--disp1()--Instance Method")
class C2:
    def disp2(self):
        print("C2--disp2()--Instance Method")
class C3:
    def disp3(self):
        print("C3--disp3()--Instance Method")

#Main Program
o1=C1()
o2=C2()
o3=C3()
o1.disp1()
o2.disp2()
o3.disp3()
/-----
=====
```

DAY-95 05/07/2024 FRIDAY

```
=====
```

```

#College.py<---File Name and Module Name
from Univ import Univ
class College(Univ):
    def getCollDet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispCollDet(self):
        print("-"*50)
        print("\tCollege Name:{}" .format(self.cname))
        print("\tCollege Location:{}" .format(self.cloc))
=====
```

```

#InhProg2.py
class C1:
    def getA(self):
        self.a=float(input("Enter Value for a:"))
class C2:
    def getB(self):
        self.b=float(input("Enter Value for b:"))
class C3(C1,C2):
    def operation(self):
        self.c=self.a+self.b
    def dispvalues(self):
        print("Sum({},{})={}" .format(self.a,self.b,self.c))

=====
```

```

#Main program
o3=C3()
o3.getA()
o3.getB()
o3.operation()
o3.dispvalues()
```

```
=====

#InhProg3.py
class C1:
    def getA(self):
        self.a=float(input("Enter Value for a:"))
class C2:
    def getB(self):
        self.b=float(input("Enter Value for b:"))
class C3(C1,C2):
    def operation(self):
        self.c=self.a+self.b
    def dispvalues(self):
        self.getA()
        self.getB()
        self.operation()
        print("Sum({},{})={}".format(self.a,self.b,self.c))

#Main program
o3=C3()
o3.dispvalues()
=====
```

```
#InhProg4.py
class Univ:
    def getUnivDet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispUnivDet(self):
        print("-"*50)
        print("\tUniversity Name:{}".format(self.uname))
        print("\tUniversity Location:{}".format(self.uloc))
class College(Univ):
    def getCollDet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispCollDet(self):
        print("-"*50)
        print("\tCollege Name:{}".format(self.cname))
        print("\tCollege Location:{}".format(self.cloc))
class Student(College):
    def getStudDet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.branch=input("Enter Student Branch:")
    def dispStudDet(self):
        print("-"*50)
        print("\tStudent Number:{}".format(self.sno))
        print("\tStudent Name:{}".format(self.sname))
        print("\tStudent Branch:{}".format(self.branch))
        print("-" * 50)
#Main Program
so=Student()
so.getStudDet()
so.getCollDet()
so.getUnivDet()
```

```
so.dispUnivDet()
so.dispCollDet()
so.dispStudDet()
-----
#InhProg4.py
class Univ:
    def getUnivDet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispUnivDet(self):
        print("-"*50)
        print("\tUniversity Name:{}".format(self.uname))
        print("\tUniversity Location:{}".format(self.uloc))
class College(Univ):
    def getCollDet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispCollDet(self):
        print("-"*50)
        print("\tCollege Name:{}".format(self.cname))
        print("\tCollege Location:{}".format(self.cloc))
class Student(College):
    def getStudDet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.branch=input("Enter Student Branch:")
        self.getCollDet()
        self.getUnivDet()
    def dispStudDet(self):
        self.dispUnivDet()
        self.dispCollDet()
        print("-"*50)
        print("\tStudent Number:{}".format(self.sno))
        print("\tStudent Name:{} ".format(self.sname))
        print("\tStudent Branch:{}".format(self.branch))
        print("-" * 50)
```

```
#Main Program
so=Student()
so.getStudDet()
so.dispStudDet()
-----
```

```
#Student.py<----File Name and Module Name
import College
import mysql.connector
class Student(College.College):
    def getStudDet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.branch=input("Enter Student Branch:")
        self.getCollDet()
        self.getUnivDet()
    def dispStudDet(self):
        self.dispUnivDet()
```

```

self.dispCollDet()
print("-"*50)
print("\tStudent Number:{}".format(self.sno))
print("\tStudent Name:{}".format(self.sname))
print("\tStudent Branch:{}".format(self.branch))
print("-" * 50)
def saveToMySQL(self):
    try:
        con=mysql.connector.connect(host="127.0.0.1",
                                     user="root",
                                     password="root",
                                     database="univdata")
        cur=con.cursor()
        #Quwery for Insert
        iq="insert into univ values(%d,'%s','%s','%s','%s','%s','%s')"
        cur.execute(iq %(self.sno,self.sname,self.branch,self.cname,self.cloc,self.uname,self.uloc))
        con.commit()
        print("{} Record Saved in Database--verify".format(cur.rowcount))
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL DB:",db)

```

#Univ.py<---File Name and Module Name

```

class Univ:
    def getUnivDet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispUnivDet(self):
        print("-"*50)
        print("\tUniversity Name:{}".format(self.uname))
        print("\tUniversity Location:{}".format(self.uloc))

```

#UnivCollegeStudent.py

```

from Student import Student
so=Student()
so.getStudDet()
so.dispStudDet()
so.saveToMySQL()

```

Method Overriding in Python

=>Method Overriding=Method Heading is same + Method Body is Different
(OR)

=>The process of re-defining the original method of base class into various derived classes for performing different operations is called Method Overriding.

=>To use Method Overriding in python program we must apply Inheritance Principle.

=>Method Overriding used for implementing Polymorphism Principle.

(PLOYMORPHISM<---METHOD OVERRIDING<----INHERITANCE<---CLASS AND
OBJECTS)

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)".

=>In Python Programming, Polymorphism can be Implemented in Two ways. They are

1. Method Overriding
 2. Constructor Overriding
-
- X-----

```
class C1:  
    def disp1(self):  
        print("C1--disp1()--Instance Method")  
class C2:  
    def disp2(self):  
        print("C2--disp2()--Instance Method")  
class C3:  
    def disp3(self):  
        print("C3--disp3()--Instance Method")
```

```
#Main Program  
o1=C1()  
o2=C2()  
o3=C3()  
o1.disp1()  
o2.disp2()  
o3.disp3()
```

```
#Non-PolyWithInh  
#PloyEx1.py  
class Circle:  
    def drawcircle(self):  
        print("Drawing Circle:")  
class Rect(Circle):  
    def drawrect(self):  
        print("Drawing Rect:")
```

```
#main program  
ro=Rect()  
ro.drawcircle()  
ro.drawrect()
```

```
#PloyEx1.py  
class Circle:  
    def draw(self): #Original Method  
        print("Drawing circle")
```

```

class Rect(Circle):
    def draw(self): # Overridden Method
        print("Drawing Rect")
        super().draw()
class Square(Rect):
    def draw(self): # Overridden Method
        print("Drawing Square")
        super().draw()
#Main program
so=Square()
so.draw()

```

DAY-96 06/07/2024 saturday

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)
 super().methodname()
Syntax2:- super().__init__(list of values)
 super().__init__()

=>with super() we are able to call only immediate base class method / Constructor but unable to call Specified method / Constructor 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)
 ClassName.methodname(self)
Syntax2:- ClassName.__init__(self, list of values)
 ClassName.__init__(self)

X

```

#PloyEx6.py
class Circle:
    def __init__(self): #Original Constructor
        print("Drawing circle--constructor")
class Square(Circle):
    def __init__(self):#Overridden Constructor

```

```

print("Drawing Square--constructor")
super().__init__()

class Rect(Square):
    def __init__(self):#Overridden Constructor
        print("Drawing Rect--constructor")
        super().__init__()
#Main Program
ro=Rect() # Object Creation
=====

#PolyEx7.py
class Circle:
    def __init__(self): # Original Constructor
        self.r=float(input("Enter Radius:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{}".format(self.ac))
class Square(Circle):
    def __init__(self):#Overridden Constructor
        self.s=float(input("enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{}".format(self.sa))
        super().__init__()
class Rect(Square):
    def __init__(self):# Overridden Constructor
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ra=self.l*self.b
        print("Area of Rect={}".format(self.ra))
        super().__init__()

```

```

#Main Program
r=Rect() # Object Creation
=====

#PloyEx1.py
class Circle:
    def draw(self): #Original Method
        print("Drawing circle")
class Rect(Circle):
    def draw(self): # Overridden Method
        print("Drawing Rect")
        super().draw()
class Square(Rect):
    def draw(self): # Overridden Method
        print("Drawing Square")
        super().draw()
#Main program
so=Square()
so.draw()
=====

#PolyEx2.py
class Circle:
    def area(self): # Original Method
        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):#Overridden Method
        self.s=float(input("enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{}".format(self.sa))
        super().area()
class Rect(Square):
    def area(self):# Overridden Method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ra=self.l*self.b
        print("Area of Rect={}".format(self.ra))
        super().area()

#Main Program
r=Rect()
r.area()
=====
```

```

#PolyEx3.py
class Circle:
    def area(self): # Original Method
        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):#Overridden Method
        self.s=float(input("enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{}".format(self.sa))

class Rect(Square):
    def area(self):# Overridden Method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ra=self.l*self.b
        print("Area of Rect={}".format(self.ra))
        Circle.area(self) # Class Name arroach
        Square.area(self) # Class Name Approach
```

```

#Main Program
r=Rect()
r.area()

#PolyEx4.py
class Circle:
    def area(self): # Original Method
        self.r=float(input("Enter Radius:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{}".format(self.ac))
class Square:
    def area(self):# Original Method
        self.s=float(input("enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{}".format(self.sa))
```

```
class Rect(Circle,Square):
    def area(self):# Overridden Method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ra=self.l*self.b
        print("Area of Rect={}".format(self.ra))
        super().area()
        Square.area(self)
#Main Program
r=Rect()
r.area()
=====
```

```
#PolyEx8.py
class Circle:
    def __init__(self): # Original Constructor
        self.r=float(input("Enter Radius:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{}".format(self.ac))
class Square:
    def __init__(self):#Original Constructor
        self.s=float(input("enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{}".format(self.sa))
```

```
class Rect(Square,Circle):
    def __init__(self):# Overridden Constructor
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ra=self.l*self.b
        print("Area of Rect={}".format(self.ra))
        super().__init__()
        Circle.__init__(self)
```

```
#Main Program
r=Rect() # Object Creation
=====
```

```
#PolyEx9.py
class Circle:
    def __init__(self,r): # Original Constructor
        self.ac=3.14*r**2
        print("Area of Circle:{}".format(self.ac))
class Square:
    def __init__(self,s):#Original Constructor
        self.sa=s**2
        print("Area of Square:{}".format(self.sa))
class Rect(Square,Circle):
    def __init__(self,L,B):# Overridden Constructor
        self.ra=L*B
        print("Area of Rect={}".format(self.ra))
        print("-----")
        super().__init__(float(input("Enter Side:")))
        print("-----")
        Circle.__init__(self,float(input("Enter Radius:")))
```

```

#Main Program
L=float(input("Enter Length:"))
B=float(input("Enter Breadth:"))
r=Rect(L,B) # Object Creation
-----
#PolyEx10.py
class Circle:
    def area(self,r): # Original Method
        self.ac=3.14*r**2
        print("Area of Circle:{}".format(self.ac))
class Square:
    def area(self,s):#Original Method
        self.sa=s**2
        print("Area of Square:{}".format(self.sa))
class Rect(Square,Circle):
    def area(self,L,B):# Overridden Method
        self.ra=L*B
        print("Area of Rect={}".format(self.ra))
        print("-----")
        super().area(float(input("Enter Side:")))
        print("-----")
        Circle.area(self,float(input("Enter Radius:")))

```

```

#Main Program
L=float(input("Enter Length:"))
B=float(input("Enter Breadth:"))
r=Rect() # Object Creation
r.area(L,B)
-----

```

```

#PolyEx11.py
class Circle:
    def __init__(self):
        print("Drawing --Circle")
class Square(Circle):
    def __init__(self):
        print("Drawing --Square")
class Rect(Square):
    def area(self):
        print("Drawing --Rect")
#Main Program
ro=Rect()
ro.area()
#Ouput
# Drawing --Square
# Drawing --Rect
-----

```

```

#PolyEx11.py
class Circle:
    def __init__(self,c):
        print("Drawing --{}".format(c))
class Square(Circle):
    def __init__(self,s):
        print("Drawing --{}".format(s))
class Rect(Square):

```

```
def __init__(self,s=None):
    super().__init__("Square")
    Circle.__init__(self, "Circle")
def area(self):
    print("Drawing --Rect")
```

```
#Main Program
ro=Rect() # Object Creation
ro.area()
#Output
# Drawing --Square
# Drawing --Circle
# Drawing --Rect
```

DAY-97 08/07/2024 (MONDAY)

Multi Therading in Python-----3 Days

Index

=>Purpose of Multi Therading

=>Types of Applications

- a) Process Based Applications
- b) Thread Based Applications

=>Development of Thread Based Applications

=>Module Name for Development of Thread Based Applications--"threading"

=>Functions in threading Module

=>Programming Examples

=>Number of Approaches to Develop Thread Based Applications

- i) By using Functional Programming
- ii) By using Object Oriented Programming

=>Programming Examples

=>Creating Multiple Threads

=>Programming Examples

=>Synchnorization Technique in Python

(OR)

Dead Lock Occurrence and Avoiding

=>Implementation of Lock Class

=>Programming Examples

Introduction to Thread Based Applications

=>The purpose of multi threading is that "To provide Concurrent / Simultaneous execution / Parallel Execution".

=>Concurrent Execution is nothing but executing the operations all at once.

=>The advantage of Concurrent execution is that to get less execution time.

=>If a Python Program contains multiple threads then it is called Multi Threading program.

=>Def. of thread:

=>A Flow of Control is called thread.

=>The purpose of thread is that "To Perform certain operation whose logic developed in Functions / Methods concurrently."

=>By default Every Python Program contains Single Thread and whose name is "MainThread" and It provides

Sequential Execution.

=>Programmatically, In a Python Program we can create multiple sub / Child threads and whose purpose is that "To execute operations whose logic is written in Functions / Methods Concurrently".

=>Hence Programmatically a Python Program contains two types of Threads. They are

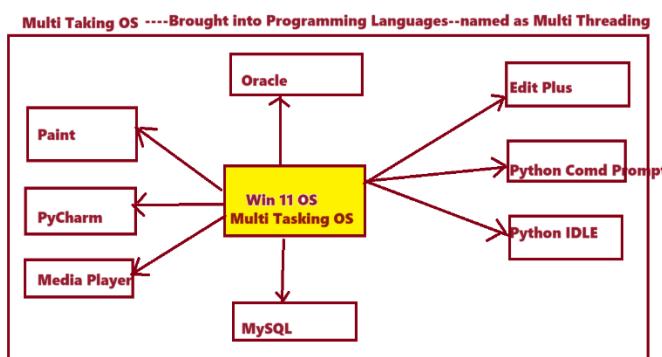
a) MainThread

b) Sub / Child Threads

=>MainThread is created / Initiated by PVM ,when program execution starts and the role of mainThread is to execute main program statements and Monitor the execution status of Sub threads(if sub threads present).

=>The Sub / Child Threads always executes operations whose logic is written in Functions / Methods Concurrently".

X



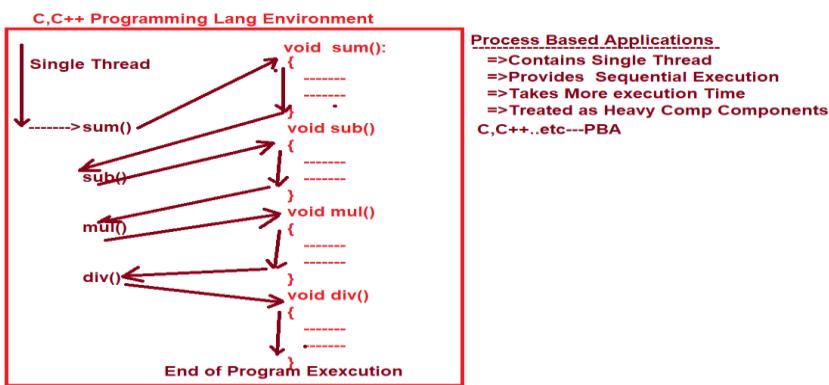
Types of Applications in Multi Threading

=>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

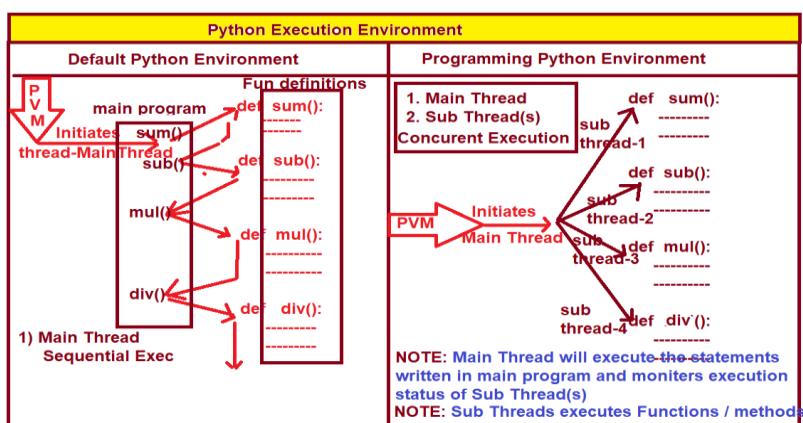
1. Process Based Applications

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 Programmatically we can create Multiple Threads.
- =>Thread 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

X

#Program for Domstrating Finding Default Thread

#DefaultThreadEx1.py

import threading

```

tname=threading.current_thread().name
print("Default Thread Name :{}".format(tname))
print("Number of threads by default=",threading.active_count())


---


#Program for Demonstrating MainTherad Executes all the functions One by One--
Sequential Execution
#DefaultThreadExecution.py
import threading
def welcome():
    print("Line:4 {}--Welcoming to Multi Threading
Concept".format(threading.current_thread().name))

def greet():
    print("Line No: 6 {}--Greeting--Good
Evening".format(threading.current_thread().name))

def hi():
    print("Line No:10 {}--Saying Hi".format(threading.current_thread().name))

#Main Program
print("Line-14--Program Execution Started:{}".format(threading.current_thread().name)) #
MainThread
welcome()
print("Line-16")
greet()
print("Line-18")
hi()
print("Line-20--Program Execution ended:{}".format(threading.current_thread().name))
=====

#Program computing Squares and cubes of list of Numbers by single default thread
#WithDeafultThreadOnly.py
import threading,time
def squares(lst):
    for val in lst:
        print("{}---square({})={}".format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def cubes(lst):
    for val in lst:
        print("{}---cube({})={}".format(threading.current_thread().name,val,val**3))
        time.sleep(1)

#Main Program
bt=time.time()
print("Program Execution Started")
lst=[2,14,5,19,-3,15,8,23,25]
squares(lst)
print("-----")
cubes(lst)
print("Program Execution Ended")
et=time.time()

```

```

print("Total Execution time with default thread only=", (et-bt))
#Program computing Squares and cubes of list of Numbers by Multiple Sub threads
#WithSubThreads.py
import threading, time
def squares(lst):
    for val in lst:
        print("{}---square({})={}".format(threading.current_thread().name, val, val**2))
        time.sleep(1)

def cubes(lst):
    for val in lst:
        print("{}---cube({})={}".format(threading.current_thread().name, val, val**3))
        time.sleep(1)

#Main Program
bt=time.time()
print("Program Execution Started:{}".format(threading.current_thread().name))
lst=[2,14,5,19,-3,15,8,23,25]
#Create First Sub Thread for Executing Squares
t1=threading.Thread(target=squares,args=(lst,) ) # Here t1 is Thread Object whose name-->Thread-1
#Create Second Sub Thread for Executing Cubes
t2=threading.Thread(target=cubes,args=(lst,) ) # Here t1 is Thread Object whose name-->Thread-2
#Dispatch the sub threads to the target functions by using start() of Thread class object
t1.start()
t2.start()
t1.join()
t2.join()
print("Program Execution Started:{}".format(threading.current_thread().name))
et=time.time()
print("Total Execution time with Sub Threads=", (et-bt))

```

DAY-98 09/07/2024 (TUESDAY)

Module Name for Development of Thread Based Applications

- =>The Module Name for Developing Thread Based Applications is "threading".
 - =>In Otherwords, "threading" is the Pre-Defined Module used for Developing Thread Based Applications.
 - =>"threading" module contains Variables, Functions and Class Names
-

Function Names in "threading" Module

- 1. current_thread()** : This Function is used for Obtaining Name of thread which is running by default
Syntax: varname=threading.current_thread()
print(varname.name)
OR
print(threading.current_thread().name)

- 2. active_count()** : This Function is used for Obtaining Number of threads which are actively Running
Syntax: varname=threading.active_count()
here varname gives number of threads which are actively Running
-

Class Name-1 in "threading" Module---Thread Class

- =>The Purpose of Thread class is that "To Create Sub Threads"
- =>The Purpose of Creating Sub Threads is that "To execute OR Perform Operations whose logic defined in Function / Method concurrently".
- =>Programmatically, Creating Sub Thread is nothing but creating an object of Thread Class

Instance Methods In Thread Class

1. Thread(target,args) : Syntax: varname=threading.Thread(target=Function/Method Name, args=())
here varname is an object of Thread class and It is Treated as Sub Thread
and whose default name is Thread-1,.....Thread-n
This Constructor is used for Creating Sub Thread and Specified which Function / Method to execute by specifying with target argument and It can also Takes args and purpose is that to take the values to target function and It is in the form of tuple.
2. start() :Syntax: threadobj.start()
This Function is used for Dispatching the sub thread to execute target function.
Without starting the sub thread, Sub thread can't execute target function.
3. setName(str) : Syntax: threadobj.setName("name to the sub thread")
This Function is used for setting the User-Friendly Name to the sub thread.
setName() is deprecated on the name of "name" attribute
Syntax: threadobj.name="name to the sub thread"
4. getName(): Syntax: varname=threadobj.getName()
This Function is used for getting the User-Friendly Name of the sub thread
getName() is deprecated on the name of "name" attribute
Syntax: print(threadobj.name)
5. join() : Syntax: threadobj.join()
This Method is used for Joining Sub Thread(s) as single Unit with MainThread.
6. is_alive() :Syntax: varname=threadobj.is_alive()
This Function is used for Checking whether the therad is running or not.
=>This Method Returns True provided thread is under Running
=>This Method Returns False provided thread is not in Running

Class Name-2 in "threading" Module---Lock Class

Methods In Lock Class

1. acquire()
2. release()

Number of Approaches to Develop Thread Based Applications

=>To Develop Thread Based Applications, we have Two Approaches. They are

1. Functional Programming
2. Object Oriented Programming

1. Functional Programming

Step-1: import threading module and Other Modules if Required
Step-2: Define a Function / Method where they contains Logic which is executed by Sub Thread
Step-3: Create Sub Thread by using Thread class
Step-4: Dispatch the sub thread by using start()

2. Object Oriented Programming

Step-1: import threading module and Other Modules if Required
Step-2: Define a Class and Choose appropriate Methods where they contains Logic which is executed by Sub Thread

Step-3: Create Sub Thread by using Thread class

Step-4: Dispatch the sub thread by using start()

#Program for Finding Number of threads actively running by default

#CountActiveThreads.py

import threading

noact=threading.active_count()

print("Number of Active Threads=",noact)

print("-----OR-----")

print("Number of Active Threads=",threading.active_count())

#This Program Obtains Current Thread in Python Environment

import threading

t[threading.current_thread()]

print("Default Thread Name=",t.name)

print("-----OR-----")

print("Default Thread=",threading.current_thread().name)

#Program for understanding whether the thread is running or not

#isAliveEx.py

import threading,time

def display(lst):

for val in lst:

print("{}--->{}".format(threading.current_thread().name,val))
time.sleep(1)

#main Program

print("Program Execution Started")

#Create sub thread

t1=threading.Thread(target=display,args=([10,25,15,35,45,20],))

print("Is sub therad running before start=",t1.is_alive())

#dispatch the sub thread

t1.start()

print("Number of active therads=",threading.active_count())

print("Is sub therad running after start=",t1.is_alive())

print("Program Execution Ended")

#This Program makes us to understand How Join the Sub Threads

#JoinMethodEx.py

import threading,time

def display(lst):

for val in lst:

print("{}--->{}".format(threading.current_thread().name,val))
time.sleep(1)

#main Program

print("Program Execution Started")

#Create sub thread

t1=threading.Thread(target=display,args=([10,25,15,35,45,20],))

print("Is sub therad running before start=",t1.is_alive())

#dispatch the sub thread

t1.start()

print("Number of active threads=",threading.active_count())

print("Is sub therad running after start=",t1.is_alive())

t1.join()

print("\nNumber of active threads after join=",threading.active_count())

print("Is sub threads running after join=",t1.is_alive())

print("Program Execution Ended")

#Program for Gemerating 1 to n Numbers where n is +ve by using threads

#NumberGenFunEx1.py

import threading,time

def generate(n):

```

if(n<=0):
    print("{}--->{} is Invalid Input".format(threading.current_thread().name,n))
else:
    print("-----")
    print("Numbers within {}".format(n))
    print("-----")
    for val in range(1,n+1):
        print("{}---Value:{}".format(threading.current_thread().name,val))
        time.sleep(0.25)
    else:
        print("-----")

#Main Program
n=int(input("Enter How Many Numbers u want to generate:"))
t1=threading.Thread(target=generate,args=(n,))
t1.start()
-----
#Program for Generating 1 to n Numbers where n is +ve by using threads
#NumberGenOopsEx1.py
import threading,time
class Numebrs:
    def generate(self,n):
        if(n<=0):
            print("{}--->{} is Invalid Input".format(threading.current_thread().name,n))
        else:
            print("-----")
            print("Numbers within {}".format(n))
            print("-----")
            for val in range(1,n+1):
                print("{}---Value:{}".format(threading.current_thread().name,val))
                time.sleep(0.25)
            else:
                print("-----")

#Main Program
n=int(input("Enter How Many Numbers u want to generate:"))
t1=threading.Thread(target=Numebrs().generate,args=(n,))
t1.start()
-----
#Program for Generating 1 to n Numbers where n is +ve by using threads
#NumberGenOopsEx2.py
import threading,time
class Numbers:
    def __init__(self,n):
        self.n=n
    def generate(self):
        if(self.n<=0):
            print("{}--->{} is Invalid Input".format(threading.current_thread().name,self.n))
        else:
            print("-----")
            print("Numbers within {}".format(self.n))
            print("-----")
            for val in range(1,n+1):
                print("{}---Value:{}".format(threading.current_thread().name,val))
                time.sleep(0.25)
            else:
                print("-----")

#Main Program
n=int(input("Enter How Many Numbers u want to generate:"))
no=Numbers(n) # Object Creation--PVM Calls Parameterized Constructor
t1=threading.Thread(target=no.generate)

```

```

t1.start()
=====

#Program for Generating 1 to n Numbers where n is +ve by using threads
#NumberGenOOPSEx3.py
import threading,time
class Numbers:
    def __init__(self,n):
        self.n=n
    def generate(self):
        if(self.n<=0):
            print("{}--->{} is Invalid Input".format(threading.current_thread().name,self.n))
        else:
            print("-----")
            print("Numbers within {}".format(self.n))
            print("-----")
            for val in range(1,self.n+1):
                print("{}---Value:{}".format(threading.current_thread().name,val))
                time.sleep(0.25)
            else:
                print("-----")

#Main Program
threading.Thread(target=Numbers(int(input("Enter How Many Numbers u want to
generate:")))).generate().start()
=====

#program for Understanding How set Name to the Thread and getting the name of the thread
#SetGetNamesEx.py
import threading
def welcome(name1):
    print("{}---->Hi {},Good Evening\n".format(threading.current_thread().name,name1))
#Main Program
print("Program Execution Started:{}".format(threading.current_thread().name))
#create sub Threads
t1=threading.Thread(target=welcome,args=("Travis",))
#Assign User-Friendly Name to the sub thread
t1.name="KVR" # t1.setName("KVR") here setName() is Deprecated on the name of "name" attribute
t1.start()
print("Name of Sub thread=",t1.name) # t1.getName() is Deprecated on the name of "name" attribute
print("Program Execution ended:{}".format(threading.current_thread().name))
=====

#Program for Understanding How to create sub threads
#StartMethodEx.py
import threading
def welcome(name1):
    print("{}---->Hi {},Good Evening\n".format(threading.current_thread().name,name1))
#Main Program
print("Program Execution Started:{}".format(threading.current_thread().name))
#create sub Threads
t1=threading.Thread(target=welcome,args=("Travis",))
t1.start()
print("Program Execution ended:{}".format(threading.current_thread().name))
=====

#This Program makes us to understand How to develop thread based application by using OOPs
#SubThreadWithOOPS.py
import threading,time
class Hyd:
    def display(self,lst):
        for val in lst:
            print("{}--->{}".format(threading.current_thread().name,val))
            time.sleep(1)

```

```

#Main Program
print("Program Execution Started")
#Create sub thread
h=Hyd()
t1=threading.Thread(target=h.display,args=[10,25,15,35,45,20])
t1.start()
t1.join()
print("Program Execution Ended")
-----
#This Program makes us to understand How to develop thread based application by using OOPs
#SubThreadWithOops1.py
import threading,time
class Hyd:
    def display(self,lst):
        for val in lst:
            print("{}--->{}".format(threading.current_thread().name,val))
            time.sleep(1)
#Main Program
print("Program Execution Started")
#Create sub thread
t1=threading.Thread(target=Hyd().display,args=[10,25,15,35,45,20])
t1.start()
t1.join()
print("Program Execution Ended")

```

DAY-99 10/07/2024 (WEDNESDAY)

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 generating odd number and even numbers by separate threads
#MultipleThreadsFunEx1.py
import threading,time
def odd(n):

```

```

if(n<=0):
    print("{}---{} Invalid Input".format(threading.current_thread().name,n))
else:
    for val in range(1,n+1,2):
        print("{}---Odd Value:{}".format(threading.current_thread().name,val))
        time.sleep(1)

def even(n):
    if(n<=0):
        print("{}---{} Invalid Input".format(threading.current_thread().name,n))
    else:
        for val in range(2,n+1,2):
            print("{}---Even Value:{}".format(threading.current_thread().name,val))
            time.sleep(1)

#Main Program
#Create a sub thread for generating Odd Numbers
t1=threading.Thread(target=odd,args=(int(input("Enter How Many Odd Numbers u want Generate:")),))
#Create a sub thread for generating even Numbers
t2=threading.Thread(target=even,args=(int(input("Enter How Many Even Numbers u want Generate:")),))
#Dispatch the sub threads
t1.start()
t2.start()

#NOTE: we created Multiple Therads and we Defined Multiple Functions
=====

#Program for geenerating odd number and even numbers by separate threads
#MultipleTheradsOopsEx1.py
import threading,time
class OddEven:
    def odd(self,n): # Instance Method1
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            for val in range(1,n+1,2):
                print("{}---Odd Value:{}".format(threading.current_thread().name,val))
                time.sleep(1)

    def even(self,n): # Instance Method2
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            for val in range(2,n+1,2):
                print("{}---Even Value:{}".format(threading.current_thread().name,val))
                time.sleep(1)

#Main Program
#Create a sub thread for generating Odd Numbers
t1=threading.Thread(target=OddEven().odd,args=(int(input("Enter How Many Odd Numbers u want Generate:")),))
#Create a sub thread for generating even Numbers
t2=threading.Thread(target=OddEven().even,args=(int(input("Enter How Many Even Numbers u want Generate:")),))
#Dispatch the sub threads
t1.start()
t2.start()

#NOTE: we created Multiple Therads and we Defined Multiple Functions
=====
#Program for geenerating odd number and even numbers by separate threads
#MultipleTheradsOopsEx2.py

```

```

import threading,time
class Odd:
    def odd(self,n): # Instance Method1
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            for val in range(1,n+1,2):
                print("{}---Odd Value:{}".format(threading.current_thread().name,val))
                time.sleep(1)

class Even:
    def even(self,n): # Instance Method2
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            for val in range(2,n+1,2):
                print("{}---Even Value:{}".format(threading.current_thread().name,val))
                time.sleep(1)

#Main Program
#Create a sub thread for generating Odd Numbers
t1=threading.Thread(target=Odd().odd,args=(int(input("Enter How Many Odd Numbers u want Generate:")),))
#Create a sub thread for generating even Numbers
t2=threading.Thread(target=Even().even,args=(int(input("Enter How Many Even Numbers u want Generate:")),))
#Dispatch the sub threads
t1.start()
t2.start()

```

#NOTE: we created Multiple Therads and we Defined Multiple Functions

```

#Program for Demonstarting Dead Lock Occurence
#Non-SynchEx1.py
import threading,time
def table(n):
    if(n<=0):
        print("{}---{} Invalid Input".format(threading.current_thread().name,n))
    else:
        print("-"*40)
        print("{}--Mul Table for:{}".format(threading.current_thread().name,n))
        print("-"*40)
        for i in range(1,11):
            print("\t{}---{} x {}={}.".format(threading.current_thread().name,n,i,n*i))
            time.sleep(1)

```

```

#Main Program
#Create Sub Threads
t1=threading.Thread(target=table,args=(7,))
t1.name="suresh"
t2=threading.Thread(target=table,args=(17,))
t2.name="Rakesh"
t3=threading.Thread(target=table,args=(6,))
t3.name="nagur"
t4=threading.Thread(target=table,args=(19,))
t4.name="kvr"
#Dispatch the sub threads
t1.start()
t2.start()
t3.start()
t4.start()

```

```

-----  

#Program for Demonstarting Dead Lock Occurence  

#Non-SynchOoopsEx1.py  

import threading,time  

class MulTable:  

    def table(self,n):  

        if(n<=0):  

            print("{}---{} Invalid Input".format(threading.current_thread().name,n))  

        else:  

            print("-"*40)  

            print("{}--Mul Table for:{}".format(threading.current_thread().name,n))  

            print("-"*40)  

            for i in range(1,11):  

                print("\t{}---{} x {}={}".format(threading.current_thread().name,n,i,n*i))  

                time.sleep(1)  

#Main Program  

#Create Sub Threads  

t1=threading.Thread(target=MulTable().table,args=(7,))  

t1.name="suresh"  

t2=threading.Thread(target=MulTable().table,args=(17,))  

t2.name="Rakesh"  

t3=threading.Thread(target=MulTable().table,args=(6,))  

t3.name="nagur"  

t4=threading.Thread(target=MulTable().table,args=(19,))  

t4.name="kvr"  

#Dispatch the sub threads  

t1.start()  

t2.start()  

t3.start()  

t4.start()  

-----  

#Program for Demonstarting Dead Lock Occurence  

#SynchFunEx1.py  

import threading,time  

def table(n):  

    L.acquire() # Obtain the Lock before execution--step-2  

    if(n<=0):  

        print("{}---{} Invalid Input".format(threading.current_thread().name,n))  

    else:  

        print("-"*40)  

        print("{}--Mul Table for:{}".format(threading.current_thread().name,n))  

        print("-"*40)  

        for i in range(1,11):  

            print("\t{}---{} x {}={}".format(threading.current_thread().name,n,i,n*i))  

            time.sleep(1)  

        print("-"*40)  

    L.release() # release the lock after complete execution--step-3  

#Main Program  

#Create an object of Lock of threading Module  

L=threading.Lock() # Step-1  

#Create Sub Threads  

t1=threading.Thread(target=table,args=(7,))  

t1.name="suresh"  

t2=threading.Thread(target=table,args=(-17,))  

t2.name="Rakesh"  

t3=threading.Thread(target=table,args=(6,))  

t3.name="nagur"  

t4=threading.Thread(target=table,args=(19,))  

t4.name="kvr"

```

```

#Dispatch the sub threads
t1.start()
t2.start()
t3.start()
t4.start()
-----
#Program for Demonstarting Dead Lock Occurence
#SynchOopsEx1.py
import threading,time
class MulTable:
    def table(self,n):
        L.acquire() # Obtain the Lock before execution--step-2
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            print("-"*40)
            print("{}--Mul Table for:{}".format(threading.current_thread().name,n))
            print("-"*40)
            for i in range(1,11):
                print("\t{}---{} x {}={}".format(threading.current_thread().name,n,i,n*i))
                time.sleep(0.25)
        L.release() # release the lock after complete execution--step-3

#Main Program
#Create an object of Lock of threading Module
L=threading.Lock() # Step-1
#Create Sub Threads
t1=threading.Thread(target=MulTable().table,args=(7,))
t1.name="suresh"
t2=threading.Thread(target=MulTable().table,args=(17,))
t2.name="Rakesh"
t3=threading.Thread(target=MulTable().table,args=(6,))
t3.name="nagur"
t4=threading.Thread(target=MulTable().table,args=(19,))
t4.name="kvr"
#Dispatch the sub threads
t1.start()
t2.start()
t3.start()
t4.start()
-----
#Program for Demonstarting Dead Lock Occurence
#SynchOopsEx2.py
import threading,time
class MulTable:
    @classmethod
    def getlock(cls):
        #Create an object of Lock of threading Module
        cls.L=threading.Lock() # Step-1--here L is Class Level data members
    def table(self,n):
        MulTable.L.acquire() # Obtain the Lock before execution--step-2
        if(n<=0):
            print("{}---{} Invalid Input".format(threading.current_thread().name,n))
        else:
            print("-"*40)
            print("{}--Mul Table for:{}".format(threading.current_thread().name,n))
            print("-"*40)
            for i in range(1,11):
                print("\t{}---{} x {}={}".format(threading.current_thread().name,n,i,n*i))
                time.sleep(0.000125)
        MulTable.L.release() # release the lock after complete execution--step-3

#Main Program

```

```

MulTable.getlock() # Calling Class Level Method for enabling Lock Object creation
#Create Sub Threads
t1=threading.Thread(target=MulTable().table,args=(7,))
t1.name="suresh"
t2=threading.Thread(target=MulTable().table,args=(17,))
t2.name="Rakesh"
t3=threading.Thread(target=MulTable().table,args=(6,))
t3.name="nagur"
t4=threading.Thread(target=MulTable().table,args=(19,))
t4.name="kvr"
t5=threading.Thread(target=MulTable().table,args=(26,))
t5.name="Nimit"
t6=threading.Thread(target=MulTable().table,args=(4,))
t6.name="Hemnath"
#Dispatch the sub threads
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
-----
#TrainReservationFunEx1.py
import threading,time
def reservation(nos):
    L.acquire()
    global totnos
    if(nos>totnos):
        print("\tHi {}, {} Seats are not available --try next
time".format(threading.current_thread().name,nos))
        time.sleep(2)
    else:
        totnos=totnos-nos
        print("\tHi {}, {} Seats are Reserved-Hpy
Journey".format(threading.current_thread().name,nos))
        time.sleep(2)
        print("\tNow Available Seats={}".format(totnos))
        time.sleep(2)
        if(totnos==0):
            print("\tTRAIN IS FULL")
    L.release()

#Main Program
L=threading.Lock()
totnos=14 # Total Number of seats
#Create Number of sub threads--Passengers
t1=threading.Thread(target=reservation, args=(2,))
t1.name="Asim"
t2=threading.Thread(target=reservation, args=(5,))
t2.name="Anup"
t3=threading.Thread(target=reservation, args=(8,))
t3.name="swathi"
t4=threading.Thread(target=reservation, args=(3,))
t4.name="neha"
t5=threading.Thread(target=reservation, args=(2,))
t5.name="Maha Lakshmi"
t6=threading.Thread(target=reservation, args=(2,))
t6.name="RS"
t7=threading.Thread(target=reservation, args=(1,))
t7.name="TR"
#Dispatch the sub threads

```

```

t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
t7.start()
-----
#TrainReservationOOPsEx1.py
import threading,time
class Train:
    @classmethod
    def getlock(cls):
        cls.L=threading.Lock()
        cls.totnos=14 # Total Number of seats
    def __init__(self,nos):
        self.nos=nos

    def reservation(self):
        Train.L.acquire()

        if(self.nos>Train.totnos):
            print("\tHi {}, {} Seats are not available --try next
time".format(threading.current_thread().name,self.nos))
            time.sleep(2)
        else:
            Train.totnos=Train.totnos-self.nos
            print("\tHi {}, {} Seats are Reserved-Hpy
Journey".format(threading.current_thread().name,self.nos))
            time.sleep(2)
            print("\tNow Available Seats={}".format(Train.totnos))
            time.sleep(2)
            if(Train.totnos==0):
                print("\tTRAIN IS FULL")
        Train.L.release()

#Main Program
Train.getlock() # Calling Class Level Method for enabling Lock Object creation
#create Number of sub threads--Passengers
t1=threading.Thread(target=Train(2).reservation)
t1.name="Asim"
t2=threading.Thread(target=Train(5).reservation)
t2.name="Anup"
t3=threading.Thread(target=Train(8).reservation)
t3.name="swathi"
t4=threading.Thread(target=Train(3).reservation)
t4.name="neha"
t5=threading.Thread(target=Train(2).reservation)
t5.name="Maha Lakshmi"
t6=threading.Thread(target=Train(2).reservation)
t6.name="RS"
t7=threading.Thread(target=Train(1).reservation)
t7.name="TR"
#Dispatch the sub threads
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
t7.start()

```

Development of Networking Applications

=>As a Part of Network Programming, we can 2 Types of Programs. They are

1. Server Side Program
 2. Client Side Program
-

Steps for Developing Server Side Programs.

Step-1 : import socket module and other modules if required

Step-2 : Create an Object of Socket (Called Server Socket)

Step-3 : Every Server Socket Must BINDS with Certain Machine and Port Number + Every Server Socket Must be

configured in such way that to how many Client Sockets the Server Socket communicates.

Step-4 : Every Server Socket Must ACCEPT Client Socket

Step-5 : Every Server Socket Must READ Client Socket Data and PROCESS

Step-6 : Every Server Socket Must give RESPONSE to Client Socket

Step-7 : Repeat Step-4,Step-5,Step-6 Until All Requests Completed from Client Socket

Steps for Developing Client Side Programs.

Step-1 : import socket module and other modules if required

Step-2 : Create an Object of Socket (Called Client Socket)

Step-3 : Every Client Socket must get CONNECTION from Server Side Program (Server Socket)

Step-4 : Every Client Socket Must SEND the Request to Server Socket.

Step-5 : Every Client Socket Must RECEIVE the Response from Server Socket

Step-6: Repeat Step-4 and Step-5 Until All Requests Completed from Client Socket

Introduction to Network Programming

=>The purpose of Network Programming is that "To Share the Data OR Information between Multiple Devices which are located either in Same Network OR in Different Network."

=>Definition of Network:

A Network is a Collection of Interconnected Autonomous Computers connected with Server.

=>To develop the Programs in Networking, we have to write 2 Types of Programs. They are

1. Server Side Program
 2. Client Side Program
-

1. Definition Server Side Program

=>A Server Side Program is one, which will receive the Request from Client Side Program, Process the Client Side Program Request and Gives Response Back to Client Side Program.

2. Client Side Program

A Client Side Program is one , which Makes Request to Server Side Program and Obtains Response from Server Side Program.

3. Definition of DNS(Domain Naming Service)

=>A DNS is one of the Physical Machine, where the Server Side Program Resides.

=>The Default DNS of Every Computer is "localhost".

4. Definition of IP Address(Internet Protocol Address)

=>An IP Address is one of the Numerical Address of Physical Machine, where the Server Side Program Resides.

=>The Default IP Address of Every Computer is "127.0.0.1" (Loop Back Address)

5. Definition of Port Number

=>A Port Number is one of the Logical Numerical ID where Server Side Program Runs

Module Name Required for Developing Networking Applications

=>The Module Name Required for Developing Networking Applications is "socket".

=>The module socket contains the Following Functions.

1. socket()

=>Syntax: varname=socket.socket()

=>This function is used for Creating an object of Socket.

=>If we create at Client Side Program then It is called Client Socket.

=>If we create at Server Side Program then It is called Server Socket.

=>An object of socket is treated as Bi-Directional Communication Entity.

2. bind()

Syntax: serversockobj.bind(("DNS/IP Address",portno))

=>This Function is used for Binding the Socket Object at Certain DNS (Machine) OR IPAddress and Port Number in the tuple.

Examples:

ss=socket.socket()

ss.bind(("localhost",9999))

Here ss is called server socket

3. listen()

=>Syntax: serversockobj.listen(No. of Clients)

=>This Function is used for Configuring the Server Socket (Server Side Program) in such way that To how Many

Clients Server Side Program can communicate.

Examples: ss.listen(2)

4. accept()

=>Syntax: var1,var2 = socketobj.accept()

=>This Function is used for accepting the Client Socket Request .

=>Here Var1 represents Client Side Program Object (i.e Client Socket object)

=>here Var2 represents Client Side Program IP Address.

Examples

cs,ca=ss.accept()

=>Here cs contains Client Side Program Requested Data

=>here ca contains Client Side Program IP Address.

5. recv() with decode()

Syntax: bytesobj=clientsocket.recv(1024 OR 2048 OR 4096)

strdata=bytesobj.decode()

OR

strdata=clientsocket.recv(1024 OR 2048 OR 4096).decode()

=>This Function is used for Receiving the Client Side Requested Data At server Side Program and we can also use at Client side Program for Receiving the Server Program Response.

6. send() with encode()

=>Syntax: bytesdata=str(Non-StrData).encode()
clientsocketobj.send(bytesdata)

OR

clientsocketobj.send(str(Non-StrData).encode())

=>This Function is used for Sending the Request of Client Side Program to Server Side Program and we can also use at Server Side Program for Sending Response to Client Side Program.

7. connect()

=>Syntax: Clientsockobj.connect(("DNS/IP Address",Portno))

=>This Function is used for Obtaining The Connection from server side program (Server Socket) by the Client Side
Program (Client Socket)

Examples

```
cs=socket.socket()  
cs.connect(("localhost",9999))
```

DAY=101 13/07/2024 (SATURDAY)

```
#ClientChat.py  
import socket  
while(True):  
    s=socket.socket()  
    s.connect(("127.0.0.1",8888))  
    cdata=input("Student-->")  
    if(cdata=="@"):  
        print("Bye Sir")  
        break  
    else:  
        s.send(cdata.encode())  
        sdata=s.recv(1024).decode()  
        print("Teacher-->{}".format(sdata))
```

```
#ServerChat.py  
import socket  
s=socket.socket()  
s.bind(("127.0.0.1",8888))  
s.listen(2)  
while(True):  
    cs,ca=s.accept()  
    cdata=cs.recv(1024).decode()  
    print("Student--->{}".format(cdata))  
    sdata=input("Teacher-->")  
    cs.send(sdata.encode())
```

#b) Write a Client Side Program in such a way that It must read a number from Key Board, send to Server Side Program and Get Its Square from Server Side Program.

```
#ClientSquare.py  
import socket  
s=socket.socket()  
s.connect(("localhost",9999))  
print("CSP Got Connection From SSP")  
n=input("Enter a Number for Finding Its Square:")  
s.send(n.encode())  
result=s.recv(1024).decode()  
print("square({})={}".format(n,result))
```

#a) Write a Server Side Program in such a way that It should get a number from Client Side Program, Square It and gives Square of that number as a Result to Client Side Program---Program--(A)

```
#ServerSquare.py
import socket
s=socket.socket()
s.bind(("localhost",9999))
s.listen(2)
print("SSP is ready to accept any CSP Request:")
while(True):
    try:
        cs,ca=s.accept()
        cdata=cs.recv(1024).decode()
        print("Client Data at Server side={}".format(cdata))
        #convert str cdata into float type
        cval=float(cdata)
        res=cval**2
        cs.send(str(res).encode())
    except ValueError:
        cs.send("Don't enter alnums,strs and symbols".encode())
```

```
#ClientEmpDataEx1.py
import socket
s=socket.socket()
s.connect(("localhost",7878))
print("CSP Got Connection from SSP")
eno=input("Enter Employee Number for Getting Other details:")
s.send(eno.encode())
empres=s.recv(1024).decode()
print("-"*40)
print("Employee Details")
print(empres)
```

```
#ClientEmpDataEx2.py
import socket
s=socket.socket()
s.connect(("localhost",7878))
print("CSP Got Connection from SSP")
eno=input("Enter Employee Number for Getting Other details:")
s.send(eno.encode())
empres=s.recv(1024).decode()
print("-"*40)
print("Employee Details")
print(empres)
```

```
#ServerEmpDataEx1.py
import socket
import oracledb as orc
s=socket.socket()
s.bind(("localhost",7878))
s.listen(2)
print("SSP is Ready to accept any CSP Request")
while(True):
    try:
        cs,ca=s.accept()
        empno=int(cs.recv(1024).decode())
        #we must Data Base Code
        con=orc.connect("system/tiger@localhost/orcl")
        cur=con.cursor()
        cur.execute("select * from employee where eno=%d" %empno)
```

```

#get the record
record=cur.fetchone()
if(record==None):
    cs.send(str("Emp Number :{} does not Exist".format(empno)).encode())
else:
    s11="-"*50
    s1="Employee Number:{}".format(record[0])
    s2="Employee Name:{}".format(record[1])
    s3="Employee Salary:{}".format(record[2])
    s4="Employee Comp Name:{}".format(record[3])
    s22="-"*50
    res=s11+"\n"+s1+"\n"+s2+"\n"+s3+"\n"+s4+"\n"+s22
    cs.send(str(res).encode())

except ValueError:
    cs.send("Don't Enter Alnums,strs and symbols for empno".encode())
except orc.DatabaseError:
    cs.send("Problem in Oracle DB".encode())
=====
#ServerEmpDataEx2.py
import socket
import mysql.connector
s=socket.socket()
s.bind(("localhost",7878))
s.listen(2)
print("SSP is Ready to accept any CSP Request")
while(True):
    try:
        cs,ca=s.accept()
        empno=int(cs.recv(1024).decode())
        #we must Data Base Code
        con=mysql.connector.connect(host="localhost",
                                     user="root",
                                     passwd="root",
                                     database="4pmbatch")
        cur=con.cursor()
        cur.execute("select * from employee where eno=%d" %empno)
        #get the record
        record=cur.fetchone()
        if(record==None):
            cs.send(str("Emp Number :{} does not Exist".format(empno)).encode())
        else:
            s11="-"*50
            s1="Employee Number:{}".format(record[0])
            s2="Employee Name:{}".format(record[1])
            s3="Employee Salary:{}".format(record[2])
            s4="Employee Comp Name:{}".format(record[3])
            s22="-"*50
            res=s11+"\n"+s1+"\n"+s2+"\n"+s3+"\n"+s4+"\n"+s22
            cs.send(str(res).encode())

    except ValueError:
        cs.send("Don't Enter Alnums,strs and symbols for empno".encode())
    except mysql.connector.DatabaseError:
        cs.send("Problem in MySQL DB".encode())

```

DAY=102 15/07/2024 (MONDAY)

=====

Regular Expressions in Python

=>The purpose of Regular Expressions in any language is that "To build Robust Applications by Performing Data Validations".

=>Regular Expressions is one of the Programming Language Independent Concept and It is Implemented by all Programming Languages for building Robust Applications by Performing Data Validations.

=>Regular Expressions Concept implemented in Python By using a Pre-Defined Module called "re".

=>In Otherwords "re" is the pre-defined modules used for Building Robust Application by performing Data Validations.

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 Development of Universal Protocols such as http, https, smtp,nntp,pop...etc

=>Regular Expressions are used in Pattern Matching

Definition of Regular Expressions

=>A Regular Expressions is one of the Search Pattern which is the Combination of Alphabets, Digits and Special Symbols and It is used to Find OR Match OR Search in Given Data and Obtains Desired Result.

Module Name Required for Developing Regular Expressions OR Pre-defined Functions in re module

=>The Module Name Required for Developing Regular Expressions is "re".

=>In Otherwords "re" is the pre-defined modules used for Building Robust Application by performing Data Validations.

=>The 're' module contains the following essential Functions.

1) finditer():

Syntax:- varname=re.finditer("search-pattern","Given data")

=>here varname is an object of type <class,'Callable_Itetaror'>

=>This function is used for searching the "search pattern" in given data iteratively and it returns table of entries which contains start index , end index and matched value based on the search pattern.

2).findall():

Syntax:- varname=re.findall("search-pattern","Given data")

=>here varname is an object of <class,'list'>

=>This function is used for searching the search pattern in entire given data and find all occurrences / matches and it returns all the matched values in the form an object <class,'list'> but not returning Start and End Index Values.

3) search():

Syntax:- varname=re.search("search-pattern","Given data")

=>here varname is an object of <class,'re.match'> or <class,'NoneType'>

=>This function is used for searching the search pattern in given data for first occurrence / match only but not for other occurrences / matches.

=>if the search pattern found in given data then it returns an object of match class which contains matched value and start and end index values and it indicates search is successful.

=>if the search pattern not found in given data then it returns None which is type <class, "NoneType"> and it indicates search is un-successful

4) group():

=>This function is used obtaining matched value by the findIter() and search()

=>This function present in match class of re module

Syntax:- varname=matchobj.group()

5) start():

=>This function is used obtaining starting index of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.start()

6) end():

=>This function is used obtaining end index+1 of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.end()

7) sub() Function

=> This function replaces the matches with the text of your choice:

```
import re  
txt = "The rain in Spain"  
x = re.sub("\s", "9", txt)  
print(x)----- The9rain9in9Spain
```

#RegExpr1.py

```
import re  
gd="Python is an oop lang.Python is also Fun prog lang"  
sp="Python"  
words=re.findall(sp,gd) # here words is of type <class, list>  
print("List of Occurences=",words)
```

#RegExpr2.py

```
import re  
gd="Python is an oop lang.Python is also Fun prog lang"  
sp="Python"  
res=re.search(sp,gd) # here res can be an object of <class, re.match> or <class, NoneType>  
if(res!=None):  
    print("Search is Successful")  
    print("Start Index=",res.start())  
    print("End Index=",res.end())  
    print("Value=",res.group())  
else:  
    print("Search is Un-Successful")
```

#RegExpr3.py

```
import re  
gd="Python is an oop lang.Python is also Fun prog lang"  
sp="Python"  
matres=re.finditer(sp,gd) # here matres is an object of <class, 'callable_iterator'>  
ctr=0  
print("-----")  
for res in matres: # here res is an object of <class,re.match class>  
    print("Start Index:{} End Index:{} Value:{}{}".format(res.start(),res.end(),res.group()))  
    ctr=ctr+1  
print("-----")  
print("{} Found {} Time(s)".format(sp,ctr))
```

#RegExpr4.py

```

import re
matres=re.finditer("Python","Python is an oop lang.Python is also Fun prog lang")
ctr=0
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
    ctr=ctr+1
print("-----")
print("{} Found {} Time(s)".format("Python",ctr))
=====

#Searching for either 'a' or 'b' or 'c' only
#RegExpr5.py
import re
matres=re.finditer("[abc]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====

#Searching for all except 'a' or 'b' or 'c'
#RegExpr6.py
import re
matres=re.finditer("[^abc]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
```

DAY=103 16/07/2024 (TUESDAY)

Programmer-Defined Character Classes

=>Programmer-Defined Character Classes are developed by Python programmers and They are used for Designing OR Preparing Search Patterns and They are used for Searching OR Finding OR Matching in Given Data and Obtains Desired Result.

=>The Syntax for Programmer-Defined Character Classes is

"[Programmer-Defined Character Classes]"

=>The Programmer-Defined Character Classes are given below

1. [abc]----->searches for either 'a' or 'b' or 'c' only
2. [^abc]----->Searches for all except 'a' or 'b' or 'c'
3. [a-z] ----->Searches for all lower case alphabets only
4. [^a-z]----->Searches for all except lower case alphabets.
5. [A-Z] ----->Searches for all Upper case alphabets only
6. [^A-Z] ----->Searches for all except Upper case alphabets
7. [0-9]----->Searches for Digits only
8. [^0-9]----->Searches for all except Digits
9. [A-Za-z]----->Searches for all alphabets (Upper and Lower Case) only
10. [^A-Za-z]---->Searches for all except alphabets (Upper and Lower Case)
11. [A-Za-z0-9]-->Searches all Alpha-numeric values (Upper and Lower Case+ Digits) only
12. [^A-Za-z0-9]-->Searches all Special Symbols except alpha-numrics (Upper and Lower Case+ Digits)
13. [A-Z0-9]----->searches for all Upper Case Alphabets and Digits only
14. [^A-Z0-9]----->searches for all except Upper Case Alphabets and Digits
15. [a-z0-9]----->searches for all Lower Case Alphabets and Digits only
16. [^a-z0-9]---->searches for all except Lower Case Alphabets and Digits

#Searching for all all lower case alphabets

#RegExpr7.py

import re

matres=re.finditer("[a-z]","aAbp6*Bka4#Yw@9qcRd5Kx3")

```

print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all except lower case alphabets
#RegExpr8.py
import re
matres=re.finditer("[^a-z]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all Upper case alphabets
#RegExpr9.py
import re
matres=re.finditer("[A-Z]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all except Upper case alphabets
#RegExpr10.py
import re
matres=re.finditer("[^A-Z]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all Digits
#RegExpr11.py
import re
matres=re.finditer("[0-9]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all except Digits
#RegExpr12.py
import re
matres=re.finditer("[^0-9]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all alphabets
#RegExpr13.py
import re
matres=re.finditer("[A-Za-z]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all except alphabets
#RegExpr14.py

```

```

import re
matres=re.finditer("[^A-Za-z]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all alpha-numerics
#RegExpr15.py
import re
matres=re.finditer("[A-Za-z0-9]","aAbp6*Bka4#Yw@9qcRd5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all special symbols
#RegExpr16.py
import re
matres=re.finditer("[^A-Za-z0-9]","aAbp6*Bka4#Yw@9qcR%d5Kx3")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
```

Pre-Defined Character Classes

=>Pre-Defined Character Classes are developed by Python Language Developers and they are available in Python Software and They are used by Python Language Programmers for Designing OR Preparing Search Patterns and They are used for Searching OR Finding OR Matching in Given Data and Obtains Desired Result.

=>The Syntax for Pre-Defined Character Classes is

"\Pre-Defined Character Class"

=>The Pre-Defined Character Classes are given below

-
1. \d----->Searches for Digits Only OR [0-9]
 2. \D----->Searches for all except Digits OR [^0-9]
 3. \w----->Searches for all word character OR Alpha-numeric value OR [A-Za-z0-9]
 4. \W----->Searches for all except word character OR Alpha-numeric value OR [^A-Za-z0-9]
 5. \s----->Searches for Space Characters only.
 6. \S----->Searches for all except Space Character .
-

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.

=>The Quantifiers in Regular Expressions are

-
- 1)"K"----->Searches for Exactly One K.
 - 2) K+----->Searches for One K or More K's
 - 3) "K*"---->Searches for Zero K or One K or More K's
 - 4) "K?"--->Searches for Zero K or One K
 - 5) "."---->Searching all occurrences of Letter of given Data.
-

Special Formula : Quantifiers combined with Programmer and Pre-Defined Character Classes

-
- 1) \d+--->Searching One OR More Digits OR [0-9]+
 - 2) \d{10} OR [0-9]{10}----->Searches 10 Digits Exactly

3) \d{m} OR [0-9]{m}----->searches M-Digit Number
4) \d{m,n}----->Searches for Minimum M-Digit Number and Maximum n-Digit Number Only.

-
- 1) \w+---Searching One OR MoreWord Character OR [A-Za-z0-9]+
 - 2) \w{5} OR [A-Za-z0-9]{5}-----Searches 5 Letter Word Exactly
 - 3) \w{m} OR [A-Za-z0-9]{m}----->searches M-Letter Word
 - 4) \w{m,n}OR [A-Za-z0-9]{m,n}---->Searches for Minimum M-Letter Word and Maximum n-Letter Word Only.
 - 5)[A-Za-z]{3}-----Searches for 3 Letter Word contains Purely Alphabets
 - 6) \dd.\dd----->Searches for Floting Point Value contains An Integer Part with 2 Digits and Decimal Part with 2 Digits.

=====X=====

#Searching for all Digits

#RegExpr17.py

import re

matres=re.finditer(r"\d","aAbp6*Bka4#Yw@9qcR%d5Kx3")

print("-----")

for res in matres: # here res is an object of <class,re.match class>

 print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))

print("-----")

=====

#Searching for all Digits

#RegExpr18.py

import re

matres=re.finditer(r"\D","aAbp6*Bka4#Yw@9qcR%d5Kx3")

print("-----")

for res in matres: # here res is an object of <class,re.match class>

 print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))

print("-----")

=====

#Searching for all word character OR Alpha-numeric value

#RegExpr19.py

import re

matres=re.finditer(r"\w","aAbp6*Bka4#Yw@9qcR%d5Kx3")

print("-----")

for res in matres: # here res is an object of <class,re.match class>

 print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))

print("-----")

=====

#Searching for all Special Symbols

#RegExpr20.py

import re

matres=re.finditer(r"\W","a Abp6*Bka4#Yw@9qcR%d5Kx3")

print("-----")

for res in matres: # here res is an object of <class,re.match class>

 print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))

print("-----")

=====

#Searching for all Space Characters

#RegExpr21.py

import re

matres=re.finditer(r"\s","a Abp6*Bka4#Yw@9qcR%d5Kx3")

print("-----")

for res in matres: # here res is an object of <class,re.match class>

 print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))

print("-----")

=====

#Searching for all except Space Character

#RegExpr22.py

import re

matres=re.finditer(r"\S","a Abp6*Bka4#Yw@9qcR%d5Kx3")

```

print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for Exactly One K
#RegExpr23.py
import re
matres=re.finditer("K","KVKKVKKKVKV")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for One or More K's----- K+
#RegExpr24.py
import re
matres=re.finditer("K+","KVKKVKKKVKV")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for zero K or One or More K's----- K*
#RegExpr25.py
import re
matres=re.finditer("K*","KVKKVKKKVKV")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for zero K or One ----- K?
#RegExpr26.py
import re
matres=re.finditer("K?","KVKKVKKKVKV")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Searching for all values ---- .
#RegExpr27.py
import re
matres=re.finditer(".","KVKKVKKKVKV")
print("-----")
for res in matres: # here res is an object of <class,re.match class>
    print("Start Index:{} End Index:{} Value:{}".format(res.start(),res.end(),res.group()))
print("-----")
=====
#Program for validating Mobile Number
#MobileNumberValidationEx1.py
import re
while(True):
    mno=input("Enter Ur Mobile Number:")
    if(len(mno)==10):
        res=re.search(r"\d{10}",mno)
        if(res!=None):
            print("\t{} is a Valid Mobile Number.".format(mno))
            break
    else:

```

```
        print("\t{} is a Valid Mobile Number bcz it contains non-int value-try  
again".format(mno))  
    else:  
        print("\t{} Invalid Mobile Number whose length is not 10 Digits-try again".format(mno))  
=====
```

DAY=104 17/07/2024 (TUESDAY)

ossum mail id is rossum_p@psf.com , Travis mail id is travis_numpy@numpy.org , Kinney mail id is kinney_12@pandas.org and Ricthe mail id is ritche12@bellabs.net.in and Kvr mail id is kvr1.python@gmail.com and Anup mail id is anup_python@wipro.com and Suresh mail id is surech@tcs.com and Nagur mail id is nagur1@hcl.com and Hemnath mail id is hemnath_hyd@ibm.com and Asim mail id is asim@ca.co.in and Neha mail id is neha_python@psf.net.in

Rossum got 88 marks , Travis got 77 marks , Kinney got 49 marks and Ricthe got 97 marks and Kvr got 11 marks and Anup got 44 marks and Suresh got 55 marks and Nagur got 64 marks and Hemnath got 88 marks

```
#Program for extracting mails of employees from  
# given file by using Regular Expressions  
#ExtractMailsEx1.py  
import re  
try:  
    with open("E:\\KVR-PYTHON-4PM\\REG EXPR\\NOTES\\empmails.data","r") as fp:  
        filedatalist=fp.read()  
        mailslist=re.findall(r"\S+@\S+", filedatalist)  
        print("Mails List")  
        for mail in mailslist:  
            print(mail)  
except FileNotFoundError:  
    print("File does not exist")  
=====
```

```
#Program for extracting Names and mails of employees from  
# given file by using Regular Expressions  
#ExtractNamesMailsEx1.py  
import re  
try:  
    with open("E:\\KVR-PYTHON-4PM\\REG EXPR\\NOTES\\empmails.data","r") as fp:  
        filedatalist=fp.read()  
        nameslist=re.findall("[A-Z][a-z]+",filedata)  
        mailslist=re.findall(r"\S+@\S+", filedata)  
        print("=*50)  
        print("Names\tMail-ID")  
        print("=* 50)  
        for names,mails in zip(nameslist,mailslist):  
            print("{}\t{}\n".format(names,mails))  
        print("=* 50)  
except FileNotFoundError:  
    print("File does not exist")  
=====
```

```
#Program for extracting marks of Students from given text by using Regular Expressions  
#MarksListEx1.py  
import re  
gd="Rossum got 88 marks , Travis got 77 marks , Kinney got 49 marks and Ricthe got 97 marks and Kvr got 11 marks"  
sp=r"\d{2}"  
markslist=re.findall(sp, gd)  
nameslist=re.findall("[A-Z][a-z]+", gd)  
print("-----")  
print("List of Marks")  
print("-----")
```

```

for marks in markslist:
    print(marks)
print("-----")
=====
#Program for validating Mobile Number
#MobileNumberValidationEx1.py
import re
while(True):
    mno=input("Enter Ur Mobile Number:")
    if(len(mno)==10):
        res=re.search(r"\d{10}",mno)
        if(res!=None):
            print("\t{} is a Valid Mobile Number.".format(mno))
            break
        else:
            print("\t{} is a Valid Mobile Number bcz it contains non-int value-try again".format(mno))
    else:
        print("\t{} Invalid Mobile Number whose length is not 10 Digits-try again".format(mno))
=====
#Program for extracting Names of Students from given text by using Regular Expressions
#NamesListEx1.py
import re
gd="Rossum is the father of python , Travis is the developer of numpy , Kinney is the developer of pandas and Ricthe is developer c and Kvr is faculty for python"
sp="[A-Z][a-z]+"
nameslist=re.findall(sp,gd)
print("-----")
print("List of Names")
print("-----")
for names in nameslist:
    print(names)
print("-----")
=====
#Program for extracting Names and marks of Students from given text by using Regular Expressions
#NamesMarksListEx1.py
import re
gd="Rossum got 88 marks , Travis got 77 marks , Kinney got 49 marks and Ricthe got 97 marks and Kvr got 11 marks"
sp=r"\d{2}"
markslist=re.findall(sp,gd)
nameslist=re.findall("[A-Z][a-z]+",gd)
print("-----")
print("Names\tMarks")
print("-----")
for names,marks in zip(nameslist,markslist):
    print("{}\t{}".format(names,marks))
print("-----")
=====
#Program for extracting Names and marks of Students from
# given file by using Regular Expressions
#NamesMarksWithFiles.py
import re
try:
    with open("E:\\KVR-PYTHON-4PM\\REG EXPR\\NOTES\\studinfo.data", "r") as fp:
        filedata=fp.read()
        nameslist=re.findall("[A-Z][a-z]+",filedata)
        markslist=re.findall(r"\d{2}",filedata)
        print("*50")
        print("Names\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")
=====
```
