Python:

Guido van Rossum is the inventor of Python Programming Language, he worked as an implementer on a team building a language called ABC at "Centrum Wiskunde Informatica" (CWI). In the year 1989, he started to prepare a Simple Scripting Language to resolve the problems of ABC programming language, as part of this, he prepared a simple virtual machine, a simple parser, and a simple runtime with various parts of ABC programming Language.

Guido van Rossum has given name to this programming Language as "Python" not by thinking Snakes name, He has given name "Python" to the programming language as he is a big fan of "Monty Python's Flying Circus" TV Show.

Guido van Rossum has introduced the first version of Python in February 1991 with the Version 0.9.0 with the features like Exception Handling, Functions, and the core data types of list, dict, str and so on.

Python version 1.0 was released in January 1994, It includes the major new features like functional programming tools like lambda, map, filter and reduce,....

In October 2000, Python 2.0 was introduced, It has included the features like list comprehensions, a full garbage collector and supporting unicode.....

In December 2008, Python 3.0 version was introduced , it is also called as “Python 3000” or “Py3K”, it includes the following enhancements.

1. Print is now a function
2. Views and iterators instead of lists
3. The rules for ordering comparisons have been simplified.
4. There is only one integer type left, i.e. int. long is int as well.
5. The division of two integers returns a float instead of an integer.
6. Text Vs. Data Instead Of Unicode Vs. 8-bit
   and some other Syntax changes.

Note: Python 3.0 version is not backward Compatible with Python 2.0 version.

The latest Version of the Python is "Python 3.7.2" released in 23rd Dec 2018, it includes the following enhancements.

1. Legacy C Locale Coercion
2. Forced UTF-8 Runtime Mode
3. Built-in breakpoint()
4. New C API for Thread-Local Storage
5. Customization of Access to Module Attributes
6. New Time Functions With Nanosecond Resolution
7. Show DeprecationWarning in __main__
8. Core Support for typing module and Generic Types
9. Hash-based .pyc Files

   and some other syntax changes.

Course Content:

Core Python
===========

1. Introduction
2. Language Fundamentals
3. Operators
4. Flow Control
5. String Data Type
6. List Data Structure
7. Tuple Data Structure
8. Set Data Structure
9. Dictionary Data Structure
10. Functions
11. Modules

12. Packages
13. Pattern Programs

Advanced Python
===============
14. OOPs
15. Exception Handling
16. File Handling
17. Multi Threading
18. Python Database Connectivity[PDBC]
19. Regular Expressions & Web Scraping
20. Decorator Functions
21. Generator Functions
22. Assertions
23. Python Logging
24. Python-GUI[TKinter]
25. Python-Distributed Applications[RPyC].

1. Introduction
----
1. Python History

2. Differences between Python and Others[C, C++ and JAVA]:
----
1. Python is very simple Programming language when compared with C, C++ and JAVA:

----
1. Simple Syntaxes in Python
2. Readability is good in Python
3. Less no of Instructions in Python
4. Easy to write programs and easy to execute applications.
5. Not required to specify data types explicitly in python.

6. Less Error prone programming language.

2. C and C++ are static Programming Languages but JAVA and Python are Dynamic Programming Languages:

C and C++ are allocating memory for the data at compilation time, so that, C and C++ are Static programming languages.

JAVA is able to allow memory allocation for the data at runtime , so that, JAVA is dynamic programming Language.

In case of Python, no sperate process for compilation and execution, in Python every thing is goiung on at runtime only including memory allocation, so that, Python is bydefault Dynamic Programming Language.

3. Pre-Processor is required in C and C++, but, Pre-Processor is not required in Java and Python:

In C and C++, Preprocessing activities are required like evaluating #include<> statements, #define statements,... so Pre-processor is required in C and C++.

In JAVA and Python , Pre-Processing activities are not existed , so that, Pre-Processor is not required in Java and Python.

4. C and C++ are Platform Dependent programming Languages , but, JAVA and Python are platform Independent Programming Language:

C and C++ are platform dependent programming Languages , because, C and C++ allows their applications to perform compilation and execution on the same Operating System.

Java is a platform Independent Programming language, because, JAVA allows its applications to perform compilation is on one Operating System and execution is on another Operating System.

Note: Java is a platform Independent Programming Language because of JVMs only, but, JVM is platform dependent.

Python is a Platform Independent Programming Language, because, Python allows its applications to run under all the Operating Systems like Windows, Linux, Unix, Solaries,..... with out having modifications.

Note: Python is a platform Independent Programming Language because of PVMs only, but, PVM is platform dependent tool, not only PVM, the complete Python Software is platform Dependent.

5. C is Procedure Oriented Progreamming Language, C++ and Java are Object Oriented Programming languages , but, Python is both Procedure oriented and Object Oriented Programming language:

C is a procedure oriented programming language, because, C programming language is using function/Procedure orented features like Functions, Lambdas,......

C++ and Java are Object Oriented Programming languages, because, C++ and Java are following Object Oriented features like Class, Object, Cncapsulation, Abstraction, Inheritance, Polymorphism,.......

Python is both Procedure Oriented and Object Oriented Programming Language, because, Python is following Procedure oriented features like Functions, Lambdas, Filters, map, reduce,..... and Object Oriented Features like class, object, encapsulation, abstraction, Inheritance, Polymorphism,.....

6. C, C++ and Java are statically Typed programming Languages, but, Python is dynamically Typed Programming Language:

If we represent Data as per the types in any Programming Languiage then that Programming Language is called as Typed Programming language.

C, C++ and Java are Statically Typed Programming Languages, because, in C, C++ and Java applications before representing data first we must confirm which type of data we are representing.
EX:

1. int i = 10; ---> Valid
2. int i;
   i = 10; ---> Valid
3. i = 10; --> Invalid

Python is dynamically Typed Programming Language, because, in python applications , data types will be confirmed internally after providing data, not before providing data.
EX:
i = 10;
print(type(i)) ---> int

EX:
f = 22.22
type(f) ----> float

7. C++ is using Destructors to destroy objects, Java is using Garbage Collector to destroy objects, but, Python is using both Destructors methods and Garbage Collector to destroy objects:

In general, In Object Oriented Programming Languages , it is convention to represent data in the form of Objects as per Object Oriented features. In Object Oriented Programming Languages, to create objects we will use

"Constructors" and to destroy objects we will use "Destructors".

In C++ , there is no Garbage Collector kind of component to destroy objects , so that, in C++ applications, developers must destroy objects explicitly, to destroy objects explicitly developers must use "Destructors" feature in C++.

In Java, developers are not responsible to destroy objects, because, JAVA has an implicit component in the form of "Garbage Collector" to destroy objects, so that, Destructors are not required in java.

In Python, Both Destructor method and and Garbage Collector are existed to destroy objects. In general, we will use destructors methods for explicit object destruction and Python will use Garbage Collector for implicit Objects destruction.

Note: In Python applications, we will use destructor methods to include a set of instructions inorder to execute while destroying objects.

8.C++ and Python are allowing Multiple Inheritance, but, JAVA is not allowing Multiple Inheritance:

Inheritance: it is a relation between classes, it will provide variables and functions from one class[parent Class/Base Class/Super class] to another class[Chaild class / Derived Class/ Sub class] inorder to improve Code Reusability.

The main advantage of Inheritance is "Code Reusability".

Initially, there are two types of Inheritances.

1. Single Inheritance
2. Multiple Inheritance

By combining single and multiple inheritances three more inheritances are defined.

3. Multi Level Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance.

1. Single Inheritance:

It is a relation between classes, it will bring variables and methods from only one super class to one or more no of sub classes.

2. Multiple Inheritance

It is a relation between classes, it will bring variables and methods from more than one super class to one or more no of sub classes.

In Java, multiple inheritance is not possible , because, in Java applications if we declare more than one variable with the same name and with different values, more than one method with the same name and with the different implementations in more than one super class and if we access that variable and method in the respective sub class then which super class vasriable and which super class method will be executed is a confusion state, but, JAVA is a simple programming language, it is not allowing multiple inheritance in its programming.

In Python, Multiple Inheritance is allowed, because, Python is following a protocol called as MRO[Method Resolution Order] , it has provided solution for the confusion of Multiple Iheritance , as per MRO, if we access any super class member in the respective sub class then PVM will search for the respective member in all the super classes as per the order in which we specified in sub class declaration.
EX:

```
---
class A():
  def m1():
     ---X-impl--

class B():
  def m1():
     --Y-impl---

class C(A,B):
   ---
c = C()
c.m1() # X-impl

class D(B,A):
   -----
d = D()
d.m1() # Y-impl
```

9. Pointers are exited in C ande C++ , but, Pointers are not existed in Java and Python:

Pointer is a variable, it able to store address locations of the data structers, where data structers may be a variable, an array, a struct, another pointer variable.

In general, pointers are recognized and initilized at the time of compilation.

Q)Why pointers are not existed in Java and Python?

Ans:
----
1. Pointer variables required static memory allocation, but, JAVA and Python are following dynamic memory allocation.

2. Pointer variables are supported by Static Programming Languages, but, Java and Python are dynamic Programming Languages.

3. Pointers are very much suitable in Platform Dependent Programming languages, but, Java and Python are platform Independent Programming Languages.

4. Pointer variables are able to provide less security for the data, but, JAVA and Python are very good Secure programming languages, they must provide very good security for the applications data.

5. Pointers is a bit confusion oriented feature, but, JAVA and Python are simple programming lnaguages , they

must not provide confusion to the developers.

Q)What are the differences between pointer variables, Reference variables and Id variables?

Ans:
----

1. Pointer variables are existed in C and C++.
   Reference variables are existed in Java.
   Id variables[Reference Variables] are existed in Python.

2. Pointer variables are able to refer a block of memory by storing its address   location.

   Reference variables are able to store a block of memory[Object] by storing   Object reference value, where Object reference value is an hexa decimal of Hashcode, where hashcode is an unique identity for each and every object   provided Heap manager in the form of an integer value.

   In Python, Id variables are able to refer a block of memory [Object] by storing   Object id value, where Object Id value is an unique identity provided by Memory   Manager in the form of an integer value.

3. Pointer variables required static memory allocations.
   Reference variables and Id variables required dynamic memory allocation.

4. Pointer variables are recognized and initialized at compilation time.
   Reference variables and Id variables are recognized and initialized at runtime.

10. C a C++ are following call by value and Call by Reference, but, Java and Python are following Call By value only:

In any Programming Language, if we pass primitive data like byte, short, int, long,float, double.... as parameters to the methods or functions then the parameter passing mechanism is "Call by Value".

In any Programming Language, if we pass address locations as parameters to the methods or functions then the parameter passing mechanism is "call By Reference".

In C and C++, if we pass pointer variables as parameters to the methods or functions then the parameter passing mechanism is "Call By Reference", because, Pointer variables are able to store address locations.

In Java and Python, if we pass Object reference value or Object Id value as parameters to the methods or functions then the parameter passing mechanism is "Call by Value " only, not "Call By Reference", because, in JAVA, Object reference value is not address location and in Python Object Id value is not address location.

11. C and C++ are compilative Programming Languages, Python is an interpretive programming language, but, JAVA is both Compilative and Interpretive programming language :

C and C++ are Compilative Programming languages, because, in C and C++ applications majority of the operations like Memory allocations are going on at compilation time only.

Python is an interpretive programming language, becasue, in Python applications every operation is going on at runtime only, that is by using an interpreter.

Java is both Compilative and interpretive programming language, because, in Java applications to check developers mistakes and to convert program from High level representations to low level representations we need compilation and to execute java programs we need an interpretor inside JVM.

12. Operator overloading is possible in C++ and Python, but, Operator Overloading is not possibe in JAVA:

The process of defining an operator with more than one operation or functionality is called as Operator Overloading.

Operator Overloading is an implementation for Polymorphism.

The main advantage of Polymorphism is "Flexibility" to develop applications.
EX:
```
a = 10
b = 20
c = a + b // + is used for Arithmetic Addition
print(c)// OP: 30

str1 = "abc"
```

```
str2 = "def"
str3 = str1 + str2 // + is used for String concatination
print(str3)// OP: abcdef
```

In C++, operator overloading is possible by defining functions explicitly.

In JAVA, some of the predefined operators are declared as overloaded operators with fixed functionalities, but, JAVA has not provided any option to perform operator overloading explicitly at developers level.

In Python, we are able to get Operator overloading implicitly and explictly, we are able to provide operator overloading by defining some magic functions.

13. Java is more portable than Python:

Java is more portable programming language, because, JAVA is able to provide very good environment to prepare the applications like Standalone applications, Web Applications, Diustributed applications, Mobile Based Applications,..... and JAVA applications will take less execution time and performance of the Java applications is more.

Python is less portable programming langiuage, because, it is not suitable in mobile application development and its applications will take more execution time and performance of Python applications is very less.

14. C , C++ and Java are following proper structer to prepare applications, but , Python is not following proper structer to prepare applications:

C is a Procedure Oriented Programming language, it has follow procedure orientation style[Structer] to prepare application.
1. Macros
2. #define
3. #include
4. Global Variables
5. Structs
6. User defined Functions
7. Main Function

C++ and Java are Object Oriented Programming Langiuages, they will follow Object Orientation style or Structer to prepare applications.
1. Comment Section
2. Package Section
3. Import Section
4. Classes/interfaces Section

5. Main Class Section.

Python is both Procedure Oriented and Object Oriented programming language, it will not follow any structer to prepare applications, it includes both Procedure oriented elements and Object Oriented elements in any direction.

----

3. Python Features:

To describe the nature of Python Programming Language we have to go for Python Feature.

Python has provided the following features to describe its nature.

1. Simple and Easy To Learn
2. Freeware and Open Source
3. High level Programming Language
4. Platform Independent
5. Portable
6. Dynamically Typed
7. Both Procedure Oriented and Object Oriented
8. Interpreted
9. Multi Threadded
10. Extendable / Extensive
11. Embedded

12.Extensive Library
13.Multi Purpose Programming Language
14.Multi Tech Support

1. Simple and Easy To Learn:

Python is very simple and easy programming language to learn and to work, because,
1. Python Syntaxes and Statements are very simple, they are available like general   english statements.
2. Python has limited no of keywords/Reserved words[30+] and they have limited   functionalities.
3. Less no of instructions are sufficient to prepare Python applications.
4. In Python applications, development time will be reduced, development cost will   be reduced and productivity will be increased.

2. Freeware and Open Source:

Python is a Freeware, because, we are able to download and utilize python software with out taking any licence.

Python is Open Source Software, because, Python source code is coming along with Python software when we download, if source code available along with python software then we are able to modify the existed python software and we are able to utilize that python software as per our requirements.

3. High level Programming Language:

Python is a high level programming language,because,
1. Python Syntaxes and Statements are very simple to understand and to use in   applications[Developers Friendly Programming Language].

2. Python is providing an abstraction layer for all the low level activities like   Memory Management, Security Management, H/W management,..... to the developers.

4. Platform Independent:

Python is a platform independent programming language, because, Python applications are executed in almost all the Operating Systems like windows, Linux, Unix,... with out having changes in python applications.

Note: Python is platform independent programming language because of PVMs only, but, PVM is platform dependent.

5. Portable:

Python is a portable programming language, because,
1. Python is able to allow its applications to execute under all the Operating   Systems and under all the H/W Systems.
2. Python programming language is very much suitable programming language for some   other products like Selenum, Hadoop,....

6. Dynamically Typed:

Python is a dynamically Typed Programming Language,because, in Python applications, we will represent data with out specifying any particular data type, but, in python applications, Python programming language will decide which type of data we represented after providing data.

7. Both Procedure Oriented and Object Oriented:

Python is both procedure oriented and Object Oriented Programming languages, because, python is following procedure oriented features like functions, lambdas, maps, reduces, filters,.... and Python is following Object Oriented Features like class, object, encapsulation, abstraction, inheritance, polymorphism,....

8. Interpreted:

In Python, there is no seperate process for compilation, directly we will execute python applications . IN general, Python programs are executed by PVM[Python Virtual Machine], in PVM , an intepretor is responsible to execute python applications.

Note: In Python, PVM will take responsibility to check syntax errors by using "Syntax Error Checkker".

9. Multi Threadded:

Python is Multiple Threadded programming Language, because, Python is able to provide very good environment to create and execute more than one thread at a time.

10.Extendable / Extensive:

Python is Extendable / Extensive programming language, because, it able to include other programming languages functionalities in python applications as per the requirement.

11.Embedded:

Python is an embedded programming language, because, Python is able to allow to embeed its functionalities in other programming language applications.

12.Extensive Library:

Python is having very good predefined library to prepare diffderent types of applications like PDBC applications, GUI Applications, Network applications,....

To prepare PDBC[Python Database Connectivity] applications Python jas provided the complete library in the form of PDBC module.

To prepare GUI applications, Python has provided predefined library in the form of seperate module called as "tkinter",.....

13.Multi Purpose Programming Language:

Python is Multi Purpose programming language, becauyse, Python is able to provide very good envirtonment to prepare the following types of applications.
1. Standalone Applications.
2. Desktop / GUI applications

3. Database Related applications
4. Web applications
5. Distributed applications
6. Nueral Network based Applications
7. AI related applications
8. Data Science Related applications

14.Multi Tech Support:

Python is giving support to the multiple technologies like C, Java, .Net,......
EX:
---
1. Python is providing "Jython" as a flavour to run under Java environment.
2. Python is providing "Cython" as a flavour to run under C environment.
3. Python is providing "Iron Python" as a flavour to run under .Net environment
4. Python is providing "Anaconda Python" as a flavour to run under Data Science   environmemnt.

Steps to prepare Python Applications:

To prepare Python applications we have to use the following steps.
1. Download and Install Python Software
2. Write and Execute Python Applications

1. Download and Install Python Software:
----
1. Open "www.python.org" website in Browser.
2. Click on "Downloads".
3. Select "Windows x86-64 executable installer" link
4. Copy The downloaded Software and maintain that Software in our Softwares dump

5. Double Click on "python-3.7.4-amd64.exe" setup file.
6. Select "Install Laucher For all USers".[Bydefault Selected]
7. Select "Add Python 3.7 to PATH".
8. Click on "Customize Installation" Link.
9. Click on "Next" button.
10.Change Python installation location from

"C:\Users\Nagoor\AppData\Local\Programs\Python\Python37" to "C:\Python\python37".
11. Click on "Install" button.
12. Click on "Yes" button.
13. Click on "Close" button.

2. Write and Execute Python Applications:

To Write and Execute Python applications we will use the following ways.

a) By Using Python Provided IDLE[Integrated Development and Learning Environment]
b) By Using Python Shell in Command Prompt
c) By Using Editplus or Some Other Editors.
d) By Using PyCharm IDE

a) By Using Python Provided IDLE:
----
a)Search for IDLE(Python3.7 64 bit) in System Search.
b)Select IDLE(Python3.7 64 bit)
c)Write Python program on IDLE and Execute Python Program

```
>>> print("Welcome to Python IDLE")      --> Click on Enter Button
    Welcome to Python IDLE
```

b) By Using Python Shell in Command Prompt:
----
a)Open Command Prompt.
b)Use "python" command in Command Prompt.
c)Write Python program and Execute Python Program.

```
>>> print("Welcome To Python Shell in CMD")        -->
Click on Enter button
    Welcome To Python Shell in CMD
```

c) By Using Editplus or Some Other Editors:

To prepare and execute application by using Editplus, we have to use the following steps.

a)Download and Install Editplus:

1)Open "https://www.editplus.com/download.html" url in Browser.
2)Select " Download EditPlus Version 4 (64-bit)" Link.
3)Copy the downloaded "epp430_64bit" setup file from downloads and keep it in   our Softwares dump.
4)Double Click on "epp430_64bit.exe" setup file.
5)Click on "Accept" button.
6)Click on "Yes" button.
7)Click on "Start Copy" Button.
8)Click on "OK" button.

b)Open Python File and Write Python Program in Editplus:

1)Double Click on "Editplus" on Desktop.
2)Click on b"Yes" button.
3)Click on "OK" button.
4)Provide Subscription code and Click on OK button.

5)Select "File"
6)Select "New"
7)Select "Others"
8)Select "Python"
9)Click on "OK" button.
10)Write Python Program

```
print("Welcome To Python-Editplus")
```

c)Save Python File :
a)Select "File"
b)Select "Save"
c)Provide FileName.py
d)Click on "Save" button.

d)Execute Python File
a)Open Command Prompt.
b)Goto the location Where Python files are saved.
c)Use either "python" command or "py" command on Command Prompt.

```
 D:\python10>python welcome.py
  Welcome To Python Editplus

D:\python10>py welcome.py
  Welcome To Python Editplus
```

d) By Using PyCharm IDE

To prepare and execute applications by Using PyCharm IDE we have to use the following steps.

1. Download and Install PyCharm IDE
2. Open PyCharm IDE and Create Python Project
3. Create Python File and Write Python Program
4. Run Python Program

1. Download and Install PyCharm IDE:
----

1. Open "https://www.jetbrains.com/pycharm/download/" in Browser.
2. Click on "Download" under "Community".
3. Copy the downloaded PyCharm Software from Downloads to Our Softwares Dump   location.

4. Double Click on "pycharm-community-2019.2.exe" Setup File.
5. Click on "Yes" button.
6. Click on "Next" button.
7. Change PyCharm installation location from
   "C:\Program Files\JetBrains\PyCharm Community Edition 2019.2" to
   "C:\PyCharm\PyCharm Community Edition 2019.2"
8. Click on "Next" button.
9. Click on "Next" button.

10.Click on "Install" button.
11.Click on "Finish" button.

2. Open PyCharm IDE and Create Python Project:
----

1. Seaarch and Select "Jetbrains PyCharm Community Edition" in our System Search.
2. Select "Light" Radio Button.
3. Select "Skip Remaining and Set Defaults" button.
4. Click on "Create New Project" Link.
5. Specify the Project Location[D:\Python10\PyCharm]
6. Click on "OK"
7. Click on "Create" button.

1. Select "File"
2. Select "New Project"
3. Provide Project Name [app01]
4. Click "Create" button.
5. Select "This Window".
6. Right Click on "app01" project.
7. Select "New"
8. Select "Python Package".
9. Provide Package Name [com.durgasoft]
10.Click on "OK" button.

3. Create Python File and Write Python Program:
----

1. Right Click on "durgasoft" package.
2. Select "New"
3. Select "Python File".
4. Provide file name [welcome.py]
5. Write Program in welcome.py file.

```
print("Welcome To PyCharm IDE")
```

4. Run Python Program

When we run first time:
  a)Right CLick on Editor
  b)Select "Run Welcome".
From Second Time onwards:
  Click on "Run" button.

3. Language Fundamentals:

To prepare Python applications, we need some constructs bydefault from Python Programming language called as "Language Fundamentals".

To prepare Python applications, Python has provided the following fundamentals.
1. Tokens
2. Data Types
3. Type Casting
4. Python Statements

1. Tokens:

Lexeme: Logical Individual Unit in programming is called as Lexeme.
Token: The Collection of lexemes come under a particular group called as Token.
EX:
---
a = b + c * d

Lexemes: a, =, b, +, c, *, d ---> 7 Lexemes

Tokens:
Identifiers: a, b, c, d
Operators: =, +, *

Tokens: 2 types of tokens

To prepare Python applications, Python has provided the following tokens.
1. Identifiers
2. Literals
3. Keywords/Reserved Words
4. Operators

1. Identifiers:

Providing names to the programming elements like variables, functions, classes, ...
called as "Identifiers".

To provide identifiers in Python applications we have to follow a set of rules and regulations.

1. All python identifiers must not be started with a number, they may start with an alphabet or _ symbol, but, the subsequent symbols may be a number, an alphabet or _ symbol.
EX:
---
eno = 111 ---> Valid
9eno = 9999 ---> Invalid
emp9Name = "Durga" ---> Valid
_ename = "Durga" ---> Valid
emp_Addr = "Hyd" --> Valid
$esal = 5000.0 ---> Invalid

2. All Python identifiers are not allowing all operators and all special symbols except _ symbol.
EX:
---
empName = "Durga" ---> valid
emp.Name = "Durga" ---> Invalid

emp+Addr = "Hyd" ---> Invalid
emp-Sal = 5000.0 ---> Invalid
emp_Addr = "Hyd" ---> Valid
emp@Hyd = "Durga" ---> Invalid

3. All Python Identifiers are not allowing spaces in the middle.
EX:
---
empName = "Durga" ---> Valid
emp Name = "Durga" ---> Invalid
tempEmpAddr = "Hyd" ---> Valid
temp Emp Addr = "Hyd" ---> INvalid

4. In Python applications, we are unable to use keywords and reserved words as identifiers.
EX:
if = 10 --> INvalid
for = 20 ---> Invalid

5. In Python applications, we are able to use all fundamental data types names and sequence data types names as identifiers.
EX:
int = 10 --> Valid
float = 20 ---> Valid
bool = 50 ---> Valid

```
list = 100 --> Valid
set = 200 --> Valid
tuple = 70 --> Valid
list = [10, 20, 30, 40, 50] --> Valid
```

6. All Python identifiers are case sensitive.
EX:

```
ename = "Durga"
ENAME = "Ravi"
print(ename)
print(ENAME)
```
OP:

```
Durga
Ravi
```

Along with the above rules and regulations, Python has provided the following suggestions to provide identifiers.

1.In Python applications, it is suggestible to provide identifiers with a particular meaning.
EX:
```
xxx = "abc123" ----> Not Suggestible
accNo = "abc123"  ----> Suggestible
```

2. In Python applications, there is no length restriction

for the identifiers , we can prepare identifiers with any length but it is suggestible to manage length of the identifiers around 10 symbols.
EX:
temporaryemployeeaddress = "Hyd" ---> Not Suggestible
tempEmpAddr = "Hyd" ---> Suggestible

3. If we have multiple words in an identifiers then it is suggestible to seperate multiple words with the special notations like '_' symbols.
EX:
tempEmpAddr = "Hyd" ---> Not Suggestible
temp_Emp_Addr = "Hyd" --> Suggestible

Q)What is the difference between variable and identifier?

Ans:

Variable is a memory location where data is stored, where name of the variable is called as an Identifier.

Note: In Programmnig Languages, when we access variable then we are able to get the data which was stored at the respective memory location, we are unable to get memory location directly.

2. Literals:

Literal is a constant assigned to the variable
EX:
```
a = 10
a ---> variable/Identifier
= ---> Operator
10 --> Constant[Literal]
```

There are two types of literals in Python.
1. Numeric Literals
2. Non Numeric Literals

1. Numeric Literals:
----
1. int literals : 10, 20, 30,....
2. float Literals : 123.234, 456.345,...

Note: Upto Python2.x version 'long' data type is existed and its literals are existed, but, from Python3.x version long data type is not existed and its literals are not existed.

EX:
```
a = 10
print(a)
print(type(a))
```

OP:
10
<class 'int'>

EX:
a = 234.345
print(a)
print(type(a))
OP:
234.345
<class 'float'>

2. Non Numeric Literals:
----
1. bool: True, False
2. str : if we provide any thing in ' '[Single quatotions], " "[Double quatotions],
        ''' '''[Triple Quatotions] then that literal is str literal.

Note: To represent string data in a single line then we are able to use '' or "" or ''' ''' , but, to represent data in multiple lines then we are able to use ''' ''' only.

EX:

#b = true --> Error

```
b = True
print(b)
print(type(b))

c = False
print(c)
print(type(c))
```
OP:
```
True
<class 'bool'>
False
<class 'bool'>
```

EX:

```
s1 = 'Durga Software Solutions'
print(s1)
print(type(s1))
```
OP:

```
Durga Software Solutions
<class 'str'>
```

EX:

```
s2 = "Durga Software Solutions"
print(s2)
print(type(s2))
OP:
Durga Software Solutions
<class 'str'>
```

EX:

```
s3 = '''Durga Software Solutions'''
print(s3)
print(type(s3))
OP:
Durga Software Solutions
<class 'str'>
```

EX:

```
str = '''Durga
Software
Solutions'''
print(str)
print(type(str))
OP:
Durga
Software
Solutions
<class 'str'>
```

In String literals, it is not possible to provide single quatotions insise single quatotions directly, but, it is possible with \, that is , \'.

EX:

```
---
str = 'Durga 'Software' Solutions'
Status: Syntax Error.
```

EX:

```
---
str = 'Durga \'Software\' Solutions'
print(str)
OP:
Durga 'Software' Solutions
```

In String literals, we are able to provide double quotations inside single quotation directly.

EX:

```
---
str = 'Durga "Software" Solutions'
print(str)
OP:
Durga "Software" Solutions
```

In String literals, it is not possible to provide triple quotations inside single quotation with or with /.

EX:

```
---
str = 'Durga '''Software''' Solutions'
Status: Syntax Error
```

EX:

```
---
str = 'Durga \'''Software\''' Solutions'
OP:
Durga 'Software' Solutions
```

In String literals, in side double quotations, we are able to provide single quotations and triple quotations directly ,but, we are unable to provide double quotations directly, but, we are able to provide doubles quotations with \.

EX:

```
---
str1 = "Durga 'Software' Solutions"
print(str1)
#str2 = "Durga "Software" Solutions" --> Error
str2 = "Durga \"Software\" Solutions"
print(str2)
str3 = "Durga '''Software''' Solutions"
print(str3)
```

OP:

```
---
```

```
Durga 'Software' Solutions
Durga "Software" Solutions
Durga '''Software''' Solutions
```

In String literals, inside triple quotations, we are able to provide single quotations and double quotations directly, we are unable to provide triple quotations directly , but, it is possible to provide triple quotations with \.

EX:

```
str1 = '''Durga 'Software' Solutions'''
print(str1)
str2 = '''Durga "Software" Solutions'''
print(str2)
#str3 = '''Durga '''Software''' Solutions''' --> Error
str3 = '''Durga \'''Software\''' Solutions'''
print(str3)
```
OP:

```
Durga 'Software' Solutions
Durga "Software" Solutions
Durga '''Software''' Solutions
```

Q)Find the right options to display the following string literal
```
   'Durga' "Software" '''Solutions'''
1)str = '\'Durga\' "Software" \'''Solutions\''''
```

```
  print(str)
2)str = "'Durga' \"Software\" '''Solutions'''"
  print(str)
3)str = ""'Durga' "Software" \"'Solutions\"""
  print(str)
4)None
```

Ans: 2, 3

Q) eno = 111
   ename = 'Durga'
   esal = 50000.0
Find data types order of the variables ename,eno and esal respectovily?
1. str, float, int
2. str, int, float
3. float, str, int
4. None

Ans: 2

Number Systems:

In all the programming languiages, to represent numbers we have to use some standards, they are Number Systems.

There are four number systems in Programming Languages
1. Binary Number System[BASE-2]
2. Octal Number System[BASE-8]
3. Decimal Number System[BASE-10]
4. Hexa Decimal Number System[BASE-16]

All the four number systems are allowed in python , but, the default number system is "Decimal Number System".

1. Binary Number System[BASE-2]

Alphbet: 0, 1
Prefix : 0b or 0B
EX:
---
a = 0b1010 ---> Valid
b = 0B1100 ---> Valid
c = 0b1020 ---> Invalid
d = 01100 ----> Invalid
e = 10 -------> It is not binary number, it is decimal number.

2. Octal Number System:

Alphabets: 0,1,2,3,4,5,6 and 7.
Prefix   : 0o or 0O [zero and O]

EX:
---
a = 10   ------> It is not octal number, it is decimal number
b = 01234 -----> Invalid
c = 0o4567-----> Valid
d = 0O34567----> Valid
e = 0O5678-----> Invalid
f = 0023456----> Invalid

3. Decimal Number System:

Alphabet : 0,1,2,3,4,5,6,7,8,9
Prefix: No prefix
EX:
a = 10 ---------> valid
b = 123456 -----> Valid
c = 56789 ------> Valid
d = 789abc -----> Invalid
e = 012345 -----> Invalid

4. Hexa Decimal Number System:

Alphabet : 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f
Prefix : 0x or 0X
EX:
a = 10 ----------> It is decimal number, it is not hexa

decimal number
b = 0x1234567 ----> Valid
c = 0X6789abc ----> valid
d = 0x123def -----> Valid
e = 0Xadefg ------> Invalid

Q)Match the following

a. Binary Number System	1. 1234
b. Octal number System	2. 0o23456
c. Decimal number System	3. 0X123abc
d. hexa Decimal Number System	4. 0B1010

a)a-1, b-2, c-3, d-4
b)a-4, b-2, c-1, d-3
c)a-1, c-2, d-3,c-4
d)a-3, b-1, c-4, d-2

Ans: b

Note: In Python applications, if we provide any number in any number system then PVM will recognize the provided number and its number system on the basis of prefix value , PVM will convert that provided number from the specified number system to decimal number System, PVM will process that converted number as like decimal number and PVM will display that numbner as like Decimal number.

In Python applications, we are able to perform arithmetic operations over the numbers which are existed in different number systems

EX:

```
---
a = 0b1010 // 10
b = 0B1100 // 12
c = a + b
print("a :", a)
print("b :", b)
print("c :", c)
```

OP:

```
a : 10
b : 12
c : 22
```

EX:

```
---
a = 0b1010
b = 0O10
c = a + b
d = a - b
e = a * b
f = a / b
print("a :", a)
print("b :", b)
print("c :", c)
print("d :", d)
print("e :", e)
print("f :", f)
```

OP

```
a: 10
b: 8
c: 18
d: 2
e: 80
f: 1.25
```

In Python applications, we are able to convert numbers from one number system to another system by using the following predefined functions.

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

1. bin()

This predefined function can be used to convert all numbers from all number systems into Binary Number System
EX:

```
---
a = 10
b = 0o10
c = 0x10
dec_To_Binary = bin(a)
oct_To_Binary = bin(b)
hex_To_Binary = bin(c)
print(dec_To_Binary)
print(oct_To_Binary)
print(hex_To_Binary)
```
OP:
```
---
0b1010
0b1000
0b10000
```

2. oct()

This predefined function can be used to convert all numbers from all the number systems to Octal number system.
EX:
```
---
a = 0b1010
b = 10
c = 0x10
binary_To_Oct = oct(a)
```

```
dec_To_Oct = oct(b)
hex_To_oct = oct(c)
print(binary_To_Oct)
print(dec_To_Oct)
print(hex_To_oct
```
OP:

```
0o12
0o12
0o20
```

3. hex()

This predefined function can be used to convert all numbers from all number systems to Hexa Decimal Number System.
EX:

```
a = 0b1010
b = 10
c = 0o10
binary_To_Hex = hex(a)
dec_To_Hex = hex(b)
oct_To_Hex = hex(c)
print(binary_To_Hex)
print(dec_To_Hex)
print(oct_To_Hex)
```

OP:

```
0xa
0xa
0x8
```

Note: In Python , the predefined functions like bin(), oct() and hex() will return a value of type 'str'.
EX:

```
print(type(bin(10)))
print(type(oct(10)))
print(type(hex(10)))
```
OP:

```
<class 'str'>
<class 'str'>
<class 'str'>
```

Note: In Python, type() predefined function can be used to return type of the variable or data.

3. Keywords/Reserved Words

If any predefined word has both word recognization and internal functionality then that predefined word is called as Keyword or Reserved Word.

To prepare python applications, Python has provided the following set of Keywords.

'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'

All the above keywords are provided by Python software in the form of keyword.py file at "C:\Python\Python37\Lib".

To get all Python Keywords on console we have to use the following code.

```
import keyword
print(keyword.kwlist)
```

EX:

```
C:\Python\Python37\Lib\keyword.py
-----------------------------------
kwlist=['False', 'None', 'True', 'and',.....]
```

Note: In Python, all keywords are provided in lower case letters except True, None and False. In True, False and

None only first symbol is Upper case letter.

Q)Find the valid keyword in Python?
a)true
b)none
c)False
d)None of the above
Ans: C

4. Operators:

Operator is a symbol, it will perform a particular operation over the provided operands.

To prepare Python applications, Python has provided the following list of Operators.

1. Arithmetic Operators

+, -, *, **[Power operator], /, //[Floor Division], %[Modulo]

2. Assignment Operators:

=, +=, -=, *=, /=, %=

3. Comparision Operators:

==, !=, <, >, <=, >=

4. Logical Boolean operators:

&, |, ^, not

Note: In Python, we can rerpesent & and | operators in word form also like 'and' and 'or'.

5. Logical Bitwise Operators:

&[and], |[or], ^, <<, >>, ~

Note: ~ is called as Bitwise 1's compliment Operator.

6. Ternary Operator:

Expr1 if Expr2 else Expr3

7. Identity Operators:
----
is, is not

8. Membership Operators:
----
in, not in

Example on Arithmetic Operators:

```
a = 10
b = 5
print("a :", a)
print("b :", b)
print("a+b :", a+b)
print("a-b :", a-b)
print("a*b :", a*b)
print("a/b :", a/b)
```

OP:

```
a : 10
b : 5
a+b : 15
a-b : 5
a*b : 50
a/b : 2.0
```

Q)What is the difference between * and ** operators?

Ans:

In Python applications, * operator will be used to perform Arithmetic Multiplication operation.

In Python applications, ** operator is representing power operator.
EX:

```
a = 10
b = 3
print("a :", a)
print("b :", b)
print("a*b :",a*b)
print("a**b :",a**b)
```

OP:

```
a : 10
b : 3
a*b : 30
a**b : 1000
```

Q)What is the difference between / and // operators?

Ans:

In Python applications, / operator will perform Arithmetic division operation and it will return always float value as result whether the operands are int or float.

In Python applications, // operator will perform Arithmetic Division operator and it will return the results as per the following conditions.

1. If both the operands are int type then the result of // operator is int only.
2. If either of the operands or both the operands are float then the result of //   operator is float.

EX:

```
---
a = 11
b = 2
c = a / b
print(c)
print(type(c))
```

OP:

```
---
5.5
<class 'float'>
```

EX:

```
a = 5.5
b = 2.2
c = a / b
print(c)
print(type(c))
```

OP:

```
----
```

```
2.5
<class 'float'>

EX:
---
a = 10
b = 2
c = a // b
print(c)
print(type(c))
OP:
---
5
<class 'int'>

EX:
----
a = 5.5
b = 2
c = a // b
print(c)
print(type(c))
OP:
----
2.0
<class 'float'>
```

```
EX:
----
a = 5
b = 2.2
c = a // b
print(c)
print(type(c))
OP:
----
2.0
<class 'float'>

EX:
---
a = 5.5
b = 2.2
c = a // b
print(c)
print(type(c))
OP:
----
2.0
<class 'float'>
```

Example on Assignment Operators:

```
a = 10
```

```
print(a)
a += 2
print(a)
a -= 5
print(a)
a *= 2
print(a)
a /= 2
print(a)
a %= 2
print(a)
```

OP:

```
10
12
7
14
7.0
1.0
```

Example on Comparision Operator:

```
a = 10
b = 20
print(a == b)
print(a != b)
print(a < b)
print(a <= b)
print(a > b)
print(a >= b)
```

OP:

```
False
True
True
True
False
False
```

Example on Logical Boolean Operators:

If we want to evaluate boolean operators then we have to use the following table.

A	B	A&B	A|b	A^B	not A
T	T	T	T	F	F
T	F	F	T	T	F
F	T	F	T	T	T
F	F	F	F	F	T

```
a = True
```

```
b = False
print(a&a)
print(a&b)
print(b&a)
print(b&b)
print(a and b)
print()
print(a|a)
print(a|b)
print(b|a)
print(b|b)
print(a or b)
print()
print(a^a)
print(a^b)
print(b^a)
print(b^b)
print()
print(not a)
print(not b)
```

OP:

```
----
True
False
False
False
```

```
False

True
True
True
False
True

False
True
True
False

False
True
```

Example on Logical Bitwise Operators:

To evaluate Logical bitwise operators we have to use the following table.

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
a = 10
b = 2
print(a & b)
print(a | b)
print(a ^ b)
print(~a)
print(a << b)
print(a >> b)
```

OP:

```
---
2
10
8
-11
40
2
```

Evaluation:

```
a = 10 ----> 1010
b = 2  ----> 0010
-------------------
   a&b ----> 0010 -----> 2
   a|b ----> 1010 -----> 10
   a^b ----> 1000 -----> 8
   ~a -----> ~10 ------> -10-1 = -11
   a<<b ---> 10 << 2 --> 00001010
                          00101000 ---> 40
   a>>b ---> 10 >> 2 --> 00001010
                       00000010 -------> 2
```

Example on Ternary Operator:

Syntax: Expr1 if Expr2 else Expr3

EX:

```
a = 10
b = 20

min = a if a < b else b
max = a if a > b else b

print("a :", a)
print("b :", b)
print("min :", min)
print("max :", max)
```

OP:

```
a : 10
b : 20
min : 10
max : 20
```

Example on Identity Operators:

The main intention of Identity operators is to check whether the two variables values are same or not same.

In Python, there are two Identity operators.
1. is
2. is not

Where 'is' operator will check whether the provided two variables values are same or not, if the variables values are same then 'is' operator will return True value , if the two variables values are not same then 'is' operator will return false value.

Where 'is not' operator will check whether the provided two variables values are not same or not, if the provided two variables values are not same then 'is not' operator will return True value, if the provided two variables values are same then 'is not' operator will return False value.

EX:
```
a = 10
b = 10
c = 20
print(a is b)
print(a is c)
print(a is not b)
print(a is not c)
```

```
OP:
---
True
False
False
True
```

Example on Membership Operators:

The main intention of membership operators is to check whether the specified element is the members of the specified sequence type or not.

There are two types member ship operators in Python.
1. in
2. not in

Where 'in' operator will check whether the specified element is member of the specified sequence type or not, if the specified element is member of the specified sequence then 'in' operator will return True value , if the specified element is not the member of the specified sequence then 'in' operator will return False value.

Where 'not in' operator will check whether the specified element is not the member of the specified sequence or not, if the specified element is not the member in the specified sequence then 'not in' operator will return True value, if the specified element is existed in the specified sequence then 'not in' operator will return False value.

EX:

```
l = [1,2,3,4,5]
print(1 in l)
print(10 in l)
print(5 not in l)
print(10 not in l)
```

OP:

```
True
False
False
True
```

EX:

Consider the following Python code.

```
a = 10
b = 'abc'
c = 22.22
```

d = True
e = +23456E456

Provide the data types of the variables a,b,c,d respectivily.
Ans: int, str, float, bool, float

EX:
Consider the following Code:
a = 10
b = 2
print(a/b)
print(a//b)
print(a%b)

What will be the result.
Ans: 5.0, 5, 0

EX:
Consider the following Code
a = 13
b = 3
print(a/b)
print(a//b)
print(a%b)
What will be the Result.
Ans: 4.3, 4, 1

EX:
Consider the following Code

```
a = 12.5
b = 4
print(a/b)
print(a//b)
print(a%b)
```

What is the result.
Ans:3.1, 3.0, 0.5

EX:

```
---
a = 5
b = 2
print(a*b)
print(a**b)
```

What is the result.
Ans: 10, 25

EX:

```
----
a = 5
print(a+2*a+1)
```

Ans:16

EX:

```
---
a = 13
print(a/3//2)
Ans: 2.0
EX:
---
a = 6
print(a*2**2)
Ans:24

a*2**2
6*2**2
6*4
24
```

Operators Priorities order Decreasing Mode:

id
()
Exponents, **
*, /,//, %
+, -
and

EX:

```
a = 5+6*4/2
print(a)
Ans: 17.0
```

EX:

```
a = 2 * (3 + 4 ** 2) + 3 * (4 ** 2 - 2)
print(a)
```

Ans: 80

```
2 * (3 + 4 ** 2) + 3 * (4 ** 2 - 2)
2 * (3 + 16) + 3 * (4 ** 2 - 2)
2 * (3 + 16) + 3 * (16 - 2)
2 * 19 + 3 * (16 - 2)
2 * 19 + 3 * 14
38 + 3 * 14
38 + 42
80
```

EX:

```
---
a = 8 // 6 % 5 + 2 ** 3 - 2
print(a)
```

Ans: 7

```
8 // 6 % 5 + 2 ** 3 - 2

1 % 5 + 2 ** 3 - 2

1 % 5 + 8 - 2
```

1 + 8 - 2
9 - 2
7

EX:
---
a = 3
b = 10
a += 3**2
a -= b // 2 // 3
print(a)
Ans:11

a = 3
b = 10
a += 3**2 --> a = a + 3 ** 2 --> a = 3 + 9 --> a = 12
a -= b // 2 // 3 --> a = a - b // 2 // 3 --> a = 12 - 10 // 2 // 3

```
                                   a = 12 - 5 // 3
                                   a = 12 - 1
                                   a = 11
```

EX:
Which of the following expression will generate max value
1. 8%3*4

2. 8-3*4
3. 8//3*4
4. 8/3*4

Ans: 4

```
print(8%3*4)--> 8
print(8-3*4) --> -4
print(8//3*4)---> 8
print(8/3*4) ----> 10.666666
```

/ and // are having same priority.

2. Data Types:

In any programming language if the data is provided as per the classifications or types then that programming language is called as Typed Programming language.

There are two types of programming Languages as per the Data types.
1. Statically Typed Programming language.
2. Dynamically Typed Programming Language.

1. Statically Typed Programming language:

In any Proigramming Language, if we are providing data

types before representing data then that programming language is called as Statically Typed Programming Language.
EX:

```
1. int a = 10; ---> valid
2. int a;
   a = 20; -------> Valid
3. a = 30; -------> Invalid
```

Note: C, C++ and Java are statically Typed Programming Languages.

2. Dynamically Typed Programming Language:

In any programming language, if we represent data with out providing data types explicitly and if the data types are calculated after providing data then that programming languages are called as Dynamically Typed Programming Languages.
EX: Python.

EX: a = 10 ---> valid
EX: int a = 20; --> Invalid.

Note: In Python , every data is represented in the form of an object and these objects are come under the respective data types.

In Python, it is possible to get data type of the provided data by using a predefined function like type(--)
EX:
```
a = 10
print(a)
print(type(a))
```
OP:
```
<class 'int'>
```

To prepare Python applications, Python has provided the following data types.

1. Fundamental Data Types
   a)Numeric Data Types
      int
      long
      float
      complex
      Note: Upto Python2.x version long data is existed, but, from Python3.x verison long data type is not existed.
   b)Non Numeric Data Types
      None
      bool
      str

2. Sequence Data Types / Advanced Data Types
   1. bytes
   2. bytearray
   3. range
   4. list
   5. tuple
   6. set
   7. frozenset
   8. dict

int :

--> It able to store only integral data.
--> Its objects are immutable objects, these objects are not allowing modifications on their content, if we are trying to perform modifications over the data then data is allowed for modification but the resultent modified data will be stored by creating new object.
EX:
```
a = 10
print(a)
print(type(a))
print(id(a))
```
OP:
```
10
<class 'int'>
140726104257056
```

EX:
```
a = 10
print(a)
print(id(a))
a = a + 10
print(a)
print(id(a))
```

OP:
```
10
140726104257056
20
140726104257376
```

In Pythn applications, we are able to represent numbers in binary, octal, decimal and hexa decimal as int type data.
EX:

```
a = 0b1010
b = 0o123
c = 10
d = 0x123abc
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```
OP:
```
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
```

Q)How many no of objects are created for int type in the following Python code?
```
a = 10
b = 10
c = 10
print(a)
print(b)
print(c)
```

Ans: 1

```
print(id(a))
print(id(b))
print(id(c))
```
OP:
```
140726104257056
140726104257056
140726104257056
```

EX:

```
a = 10
a = a + 10
a = a + 10
print(a)
OP: 30
```

Ans: 3 objects are created.

Explanation:

```
a = 10
print(id(a))
a = a + 10
print(id(a))
a = a + 10
print(id(a))
print(a)
```

OP:

```
140725848994336
140725848994656
140725848994976
30
```

2. float:

--> It includes integral part and fractional part.
--> Its objects are immutable objects.
EX:

```
f = 12.234
print(f)
print(type(f))
print(id(f))
```
OP:

```
12.234
<class 'float'>
1910119359920
```
EX:

```
f = 12.22
print(id(f))
f = f + 10
print(id(f))
f = f + 10
print(id(f))
print(f)
```
OP:
```
3222635177392
3222635177328
3222635177360
32.22
```

3. Complex Type:

--> Its data includes real value and imaginary value.
--> To represent data in Compex Type we have to use the following syntax.
     a+bj
Where a is real part
Where b is imaginary part

EX:

```
a = 10+2j
print(a)
print(type(a))
print(id(a))
```

OP
```
(10+2j)
<class 'complex'>
1863307100496
```

In Complex Numbers, we are able to get real and imaginary data seperatly by using the predefined variables "real" and "imag".
EX:

```
a = 10+2j
print(a)
print(a.real)
print(a.imag)
```

OP:

```
(10+2j)
10.0
2.0
```

In Complex Numbers, we are able to provide int, float values as real part and amaginary part, but, bydefault, they will take float values.

EX:

```
a = 22.22+2.5j
print(a)
print(type(a))
```

OP:

```
(22.22+2.5j)
<class 'complex'>
```

In Complex Numbers, we are able to provide real value by using the number systems like binary number system, octal number system , decimal number system and hexa decimal number system, but, imaginary part is able to allow the value in decimal number system only.

Note: In Complex numbers if we provide real and imaginary values in any number system then PVM will convert these numbers from the specified number system to decimal number system of float type and PVM will process that numbers as like float values.
EX:

```
a = 0o10 + 2j
print(a)
print(type(a))
print(a.real)
print(a.imag)
```

In Python applications, we are able to perform all the arithmetic operations like addition, subtraction , Multiplication,.. over the complex numbers.
EX:

```
a = 10 + 10j
b = 5 + 5j
c = a + b
d = a - b
e = a * b
print(a)
print(b)
print(c)
print(d)
print(e)
OP:
---
(10+10j)
(5+5j)
(15+15j)
(5+5j)
100j
```

Expl:

```
a = 10 + 10j
b = 5 + 5j

a+b --> (10+5) + (10+5)j --> 15 + 15j
a-b --> (10-5) + (10-5)j --> 5 + 5j

a*b --> 10*5 + 10*5j + 10j*5 + 10j*5j
                              2
    --> 50 + 50j + 50j + 50j
    --> 50 + 100j - 50
    --> 100j
```

In Python, Complex data type objects are immutable objects.

EX:

```
---
a = 10 + 10j
print(a)
print(id(a))
a = a + (5 + 5j)
print(a)
print(id(a))
OP:
---
(10+10j)
2510474314160
(15+15j)
2510474314320
```

None Type:

In Python applications, it is not possible to declare any variable with out explicit initialization, we must provide intialization for the variables explicitly.

```
a --> Invalid
a = 10 --> Valid
```

In python applicartions, if we want to declare a variable with no data then we have to use 'None' as value to the variables, where 'None' is representing no data.
EX:

```
a = None
print(a)
print(type(a))
print(id(a))
```

OP:

```
None
<class 'NoneType'>
140726009281760
```

bool:

This data type is able to provide two values like True and False .
EX:

```
b = True
print(b)
print(type(b))
print(id(b))
```
OP:

```
True
<class 'bool'>
140726009235792
```

In Python, bool type objects are immutable objects.
EX:

```
b = False
print(b)
print(id(b))
b = True
print(b)
print(id(b))
```
OP:

```
False
140726009235824
True
140726009235792
```

In python , boolean values True and False are represented internally in the form of 1 and 0 respectivly and it is possible to perform Arthmetic Operations like Addition, Subtraction, Multiplication,... over boolean values.
EX:

```
a = True
print(int(a))
b = False
```

```
print(int(b))
c = True + False
print(c)
d = True - False
print(d)
f = True * False
print(f)
OP:
---
1
0
1
1
0
```

str:

It is the collection of alphabets, digits, special symbols,........
IN Python applications, we are able to represent str values by using ' or " or
''' .
EX:

```
a = 'Durga Software Solutions'
b = "Durga Software Solutions"
c = '''Durga
```

```
Software
Solutions'''
print(a)
print(b)
print(c)
```
OP:

```
Durga Software Solutions
Durga Software Solutions
Durga
Software
Solutions
```

str type objects are immutable objects.
EX:

```
a = "Durga "
b = a + "Software "
c = b + "Solutions"
print(a)
print(b)
print(c)
print(id(a))
print(id(b))
print(id(c))
```

OP:

```
---
Durga
Durga Software
Durga Software Solutions
1693420140400
1693421470960
1693420046096
```

In Python applications, we are able to read elements from String individually on the basis of index values. In str values, we are able to read elements in both forward direction and backward direction. In forward direction we must use +ve index values and it must be started from 0 and in backward direction we must use -ve index values and it must be started with -1.

To retrive elements from str type on the basis of index values we have to use the following syntax.

```
        refVar[index_Value]
```

Note: If we provide index_Value in outside range of the String index values then PVM will provide an error like "IndexError: string index out of range".

EX:

```
---
str = "Durga"
print(str)
print(str[2])
print(str[4])
print(str[-2])
print(str[-4])
#print(str[5]) --> IndexError
print(str[-5])
#print(str[-6]) --> IndexError
OP:
---
Durga
r
a
g
u
D
```

Note: To get length of the string we have to use len(--) predefined function.

EX:

```
str = "Durga Software Solutions"
print(str)
print(len(str))
OP:
Durga Software Solutions
24
```

In Python applications, if we want to read elements from String type we will use for loop.
EX:

```
str = "Durga Software Solutions"
for x in range(0, len(str)):
    print(x,"--->",str[x])
print()
for x in range(0, len(str)):
    index = -(x+1)
    print(index,"--->",str[index])
```

OP:

```
0 --> D
1 --> u
-------
-------
24 --> s

-1 --> s
-2 --> n
-----
-----
-25 --> D
```

EX:

```
str = "Durga Software Solutions"
for x in str:
    print(x)
```

OP:
```
D
u
--
--
s
```

bytes Type:

-->It is a sequence data type, it able to represent sequence of numbers.
-->It able to allow the numbers from 0 to 255.
-->To represent numbers in bytes data we have to use the following steps.

      1. Represent numbers in List by using [].
      2. Convert numbers from List to bytes data type by using bytes() function.

EX:

```
list = [1,2,3,4,5]
print(list)
print(type(list))
```

```
b = bytes(list)
print(b)
print(type(b))
```

```
OP:
---
[1, 2, 3, 4, 5]
<class 'list'>
b'\x01\x02\x03\x04\x05'
<class 'bytes'>
```

In Python applications, bytes data type is able to allow the numbers from 0 to 255, if we provide numbers in out side range of 0 to 255 then PVM will provide an error like "ValueError: bytes must be in the range (0, 256)"

EX:

```
l = [253, 254, 255, 256]
b = bytes(l)
print(b)
```

```
Status: ValueError: bytes must be in range(0, 256)
```

bytes objects are immutable objects, they are not allowing modifications on their content, if we are trying to perform modifications then PVM will raise an error like "TypeError: 'bytes' object does not support item

assignment"
EX:

```
l = [1,2,3,4,5]
b = bytes(l)
print(b)
b[2] = 10
print(b)
```

Status: "TypeError: 'bytes' object does not support item assignment"

In bytes type we are able to retrive numbers individually by providing index value, if the index value is in out side range of the bytes index values then PVM will rise an error like "indexError: index out of range".

In bytes type we are able to read elements in both forward direction and backward direction, to read elements in forward direction we have to provide +ve index values and it will be started with 0. If we want to read elements in backward direction then we have to provide -ve index values and it must be started with -1.
EX:

```
l = [1,2,3,4,5,6,7,8,9,10]
b = bytes(l)
```

```
print(b)
print(b[2])
print(b[-2])
print(b[6])
print(b[-5])
#print(b[10])-->IndexError: index out of range
```

OP:

```
b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n'
3
9
7
6
```

bytearray Type:

--> It is a sequence data type, it able to represent sequence of numbers.
--> To represent numbers in bytearray type we have to use the following two steps.
    1. Represent Numbers in List data type by using [] .
    2. Convert numbers from List type to bytearray type by using bytearray() fun.

EX:

```
list = [1,2,3,4,5]
```

```
print(list)
print(type(list))
ba = bytearray(list)
print(ba)
print(type(ba))
```

OP:

```
[1, 2, 3, 4, 5]
<class 'list'>
bytearray(b'\x01\x02\x03\x04\x05')
<class 'bytearray'>
```

--> bytearray type is able to allow the numbers from 0 to 255 only, if we provide numbers in out side range of 0 to 255 then PVM will provide any error like "ValueError: byte must be in range(0, 256)".
EX:

```
list = [253, 254, 255, 256]
ba = bytearray(list)
print(ba)
```

Status: ValueError: byte must be in range(0, 256)

--> bytearray type objects are mutable objects, they are able to allow modifications on their content.

EX:

```
list = [1,2,3,4,5]
ba = bytearray(list)
for x in ba:
    print(x,end=" ")
print()
ba[2] = 100
for x in ba:
    print(x,end=" ")
```

OP:

```
1 2 3 4 5
1 2 100 4 5
```

--> In Python applications, we are able to read individual elements from bytearray on the basis of index values. If we provide any index value which is in out side rage of the bytearray index values then OVM will raise an error like "IndexError: bytearray index out of range"

In bytearray type, we are able to read elements in both forward direction and backward direction, to read elements in forward direction we have to use +ve index value and it must be started with 0 . To read elements in backward direction we have to use -ve index value and it

must be started with -1.
EX:

```
list = [1,2,3,4,5,6,7]
ba = bytearray(list)
print(ba[3])
print(ba[-5])
print(ba[5])
print(ba[-4])
#print(ba[7]) --> IndexError: bytearray index out of
range
print(ba[len(ba)-2])
```

OP:

```
4
3
6
4
6
```

In Python applications, to read elements from bytearray type we are able to use for loop and for-Each loop. To read elements in both forward direction and backward direction then we must use for loop only.If we use for-Each loop then we are able to read elements in forward direction only.

EX:

```
list = [1,2,3,4,5,6,7, 8, 9, 10]
ba = bytearray(list)
for x in range(0, len(ba)):
    print(ba[x],end=" ")
print()
for x in range(0, len(ba)):
    print(ba[-(x+1)],end=" ")
```

OP:
```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
```

EX:

```
list = [1,2,3,4,5,6,7, 8, 9, 10]
ba = bytearray(list)
for x in ba:
    print(x,end=" ")
```

OP:

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

Q)What are the differences between bytes data type and bytearray data type?

Ans:
----
1. To rpresent elements in bytes data type we have to use a predefined function   like bytes().
   To represent elements in bytearray type we have to use a predefined function   like bytearray().

2. bytes type objects are immutable objects.
   bytearray type objects are mutable objects.

Note: In Python, both bytes data type and bytearray data type are able to allow only numbers as elements and they are existed in fixed size in nature. If we are trying to access elements over its size then PVM will provide an error.
EX:

```
list = [1,2,3,4,5]
ba = bytearray(list)
for x in ba:
    print(x,end=" ")
#ba[5] = 6 --> IndexError: bytearray index out of range
#print(ba[5]) --> IndexError: bytearray index out of range
```

range type:

--> It is a sequence data type, it able to generate sequence of numbers.
--> To get numbers from range type we must use for loop.
--> To generate sequence of numbers from range type we have to use the following   syntaxes.
1. range(end)
--> It will generate sequence of numbers starts from 0 and upto the specified end-1 by using 1 step length.
EX:

```
r = range(10)
for x in r:
    print(x,end=" ")
```

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

2. range(Start, End)
--> It will generate sequence of numbers starts from the specified Start value and upto the Specified End-1 and by using 1 step length.
EX:

```
r = range(0,10)
```

```
for x in r:
    print(x,end=" ")
```
OP:

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

EX:

```
r = range(11,20)
for x in r:
    print(x,end=" ")
```
OP:

```
11 12 13 14 15 16 17 18 19
```

3. range(Start, End, Step)
--> It will generate sequence of numbers from the specified start value and upto the specified end-1 value by using the specified step length.
EX:

```
r = range(0,20,2)
for x in r:
    print(x,end=" ")
```
Op:
```
0 2 4 6 8 10 12 14 16 18
```

List:

--> It is a sequence data type, it able to represent sequence of elements.
--> In List type, to represent elements we have to use [].
EX:

```
list = [1,2,3,4,5]
print(list)
print(type(list))
print(id(list))
```
OP:
```
[1, 2, 3, 4, 5]
<class 'list'>
3204551692872
```

--> List type is not in fixed size nature, it is existed with dynamically growable nature, that is, it able to increase or decrease its size depending on the no of elements which are existed in List.
EX:

```
list = [1,2,3,4]
print("List Elements :",list)
print("Length : ",len(list))
print()
```

```
list.append(5)
list.append(6)
print("List Elements :",list)
print("Length : ",len(list))
print()
list.append(7)
list.append(8)
print("List Elements :",list)
print("Length : ",len(list))
print()
list.remove(8)
list.remove(7)
print("List Elements :",list)
print("Length : ",len(list))
```

OP:

```
---
List Elements : [1, 2, 3, 4]
Length :  4

List Elements : [1, 2, 3, 4, 5, 6]
Length :  6

List Elements : [1, 2, 3, 4, 5, 6, 7, 8]
Length :  8

List Elements : [1, 2, 3, 4, 5, 6]
Length :  6
```

In Python List type is able to allow heterogeneous elements.
EX:

```
list = [1,2,3,4]
print(list)
list.append(22.22)
list.append(33.33)
print(list)
list.append(True)
list.append(False)
print(list)
list.append("abc")
list.append("xyz")
print(list)
```
OP:

```
[1, 2, 3, 4]
[1, 2, 3, 4, 22.22, 33.33]
[1, 2, 3, 4, 22.22, 33.33, True, False]
[1, 2, 3, 4, 22.22, 33.33, True, False, 'abc', 'xyz']
```

--> In List data type, we are able to read elements individually by using index values. In List type, it is possible to read elements in both forward direction and Backward direction, to read elements in forward direction we have to provide +ve index valuees and it must be started with 0 , to read elements in backward direction we have to provide -ve index values and it must be started with -1.

EX:

```
list = [1,2,3,4,5,6,7]
print(list)
print(list[2])
print(list[-4])
print(list[5])
print(list[-6])
print()
for x in range(0,len(list)):
    print(list[x], end=" ")
print()
for x in range(0,len(list)):
    print(list[-(x+1)], end=" ")
print()
for x in list:
    print(x, end=" ")
```

OP:

```
[1, 2, 3, 4, 5, 6, 7]

3
4
```

```
6
2

1 2 3 4 5 6 7
7 6 5 4 3 2 1
1 2 3 4 5 6 7
```

--> List is able to allow duplicate elements.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE"]
print(list)
list.append("BBB")
list.append("DDD")
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'BBB', 'DDD']
```

--> List is following insertion order, it is not following Sorting order.
EX:

```
list = ["AAA", "FFF", "BBB", "EEE", "CCC", "DDD"]
print(list)
```
OP:

```
----
['AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD']
```

--> List is able to allow None elements in any number.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", None, None, None]
print(list)
```

OP:

```
['AAA', 'BBB', 'CCC', 'DDD', None, None, None]
```

--> List objects are mutable objects, they are able to allow modifications on their content.
EX:

```
list = ["AAA", "BBB"]
print(list)
print(id(list))
list.append("CCC")
list.append("DDD")
print(list)
print(id(list))
list.append("EEE")
list.append("FFF")
print(list)
print(id(list))
OP:
---
['AAA', 'BBB']
2879331586632
['AAA', 'BBB', 'CCC', 'DDD']
2879331586632
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
2879331586632
```

Tuple Type:

--> It is a sequence type, it able to store sequence of elements.
--> It is index based, it able to allow to read elements on the basis of index   values, It allows to read elements in both forward direction and backward direction, to read elements in forward direction we have to use +ve idex values   and it must be started with 0, to read elements in Backward direction we have   to provide -ve index values and it must be started with -1.
--> It allows duplicate elements.
--> It allows heterogeneous elements.
--> It is following insertion order.
--> It does not follow Sorting order.
--> Its objects are immutable objects.
--> It allows any no of None elements.

EX:

```
tuple = (1,2,3,4, 2, 5, None, "AAA", "BBB", None)
print(tuple)
print(type(tuple))
print(tuple[2])
print(tuple[-2])
#tuple[1] = "XXX"--> TypeError: 'tuple' object does not support item assignment
```
OP:

```
(1, 2, 3, 4, 2, 5, None, 'AAA', 'BBB', None)
<class 'tuple'>
3
BBB
```

Q)What are the differences between List and Tuple?

Ans:
----
1. In List , all the elements are represented with [].
   In Tuple , all the elements are rpresented with ().

2. List is having append() and remove() functions to perform manipulations over the   elements.
   Tuple is not having append() and remove() functions to perform manipulations   over the elements.

3. List Objects are Mutable
   Tuple Objects are Immutable.

4. List is dynamically Gorwable in nature.
   Tuple is fixed size in nature.

set Type:

--> It is a sequence data type, it able to store sequence of elements.
--> To represent elements in set type we have to use {}.
EX:

```
set = {"AAA", "BBB", "CCC", "DDD", "EEE", "FFF"}
print(set)
print(type(set))
```
OP:

```
{'BBB', 'CCC', 'DDD', 'FFF', 'EEE', 'AAA'}
<class 'set'>
```

--> Set is not following both insertion order and Sorting order.
EX:

```
set = {"AAA", "BBB", "CCC", "DDD"}
```

```
print(set)
OP:
---
{'BBB', 'AAA', 'CCC', 'DDD'}
```

--> Set is not index based, if we are trying to read elements on the basis of index values then PVM will provide an error like
"TypeError: 'set' object is not subscriptable"

EX:

```
set = {"AAA", "BBB", "CCC", "DDD"}
print(set)
#print(set[1]) --> Error
Status: TypeError: 'set' object is not subscriptable
```

--> To read elements from set we have to use for-Each loop, not normal for loop.

EX:

```
set = {"AAA", "BBB", "CCC", "DDD"}
print(set)
for x in set:
    print(x)
OP:
---
{'AAA', 'CCC', 'BBB', 'DDD'}
```

```
AAA
CCC
BBB
DDD
```

--> Set is not allowing duplicate elements.
EX:

```
set = {"AAA", "BBB", "CCC", "DDD", "BBB", "CCC"}
print(set)
```

OP:

```
{'AAA', 'CCC', 'BBB', 'DDD'}
```

--> Set is able to allow heterogeneous elements.
EX:

```
set = {"AAA", "BBB", 10, 20, 22.22, 33.33, True, False}
print(set)
```
OP:

```
{False, 33.33, True, 'AAA', 10, 20, 22.22, 'BBB'}
```

EX:

```
set = {"AAA", "BBB"}
```

```
print(set)
set.add(10)
set.add(20)
print(set)
set.add(22.22)
set.add(33.33)
print(set)
set.add(True)
set.add(False)
print(set)
```

Op:

```
{'BBB', 'AAA'}
{'BBB', 10, 'AAA', 20}
{'BBB', 33.33, 10, 'AAA', 20, 22.22}
{False, 'BBB', 33.33, True, 10, 'AAA', 20, 22.22}
```

--> Set type objects are mutable objects, they are able to allow modifications on their content.
EX:

```
set = {10, 20}
print(set)
set.add(30)
set.add(40)
print(set)
```

```
set.add(50)
set.add(60)
print(set)
```

OP:

```
{10, 20}
{40, 10, 20, 30}
{40, 10, 50, 20, 60, 30}
```

--> Set is able to allow None element[Only one, not more than one].
EX:

```
set = {10, 20, None, None}
print(set)
```

OP:
```
{10, 20, None}
```

Q)What are the differences between List and Set?

Ans:
----
1. List is able to represent elements in the form of [].
   Set is able to represent all the elements in the form of {}.

2. List is index based.
   Set is not index based.

3. List is able to allow duplicate elements.
   Set is not allowing duplicate elements.

4. List is following insertion order.
   Set is not following insertion order.

5. List is able to allow None elements in any number.
   Set is able to allow only one None element.

6. List has append() function to add elements.
   Set has add() function to add elements.

7. List allows both for loop and for-Each to read elements.
   Set allows only for-Each loop to read elements.

8. Immutable form of List is Tuple.
   Immutable form of Set is FrozenSet.

FrozenSet Type

--> It is a Sequence data type, it able to represent sequence of elements.

--> To represent elements in frozenset type we have to use the following steps.
    1. Prepare Elements in Set type by using {}.
    2. Convert all the elements from Set type to frozenset type by using
       frozenset() predefined function.
--> It is not index based.
--> It is not following insertion order and sorting order.
--> It is not allowing duplicate elements.
--> It allows only one None element.
--> It allows heterogeneous elements.
--> It allows for-Each loop to read elements.
--> Its Objects are immutable objects.

EX:

```
set = {10, 20, 30, 40, 50, 20, 30, "AAA", "BBB", 22.22, 33.33, None, None}
fs = frozenset(set)
print(fs)
print(type(fs))
for x in fs:
    print(x,end=" ")
```

Q)What are the differences between Set and frozenset?

Ans:

----
1. To represent elements in Set we will use {}.
   To repersent elements in frozenset we must represent elelments in set by using   {} then we must convert that elements into frozenset by using frozenset()   function.

2. set objects are mutable objects.
   frozenset objects are immutable objects.

3. set is in dynamically growable in nature.
   frozenset is in fixed size in nature.

4. add() and remove() functions are existed in Set type.
   add() and remove() functions are not existed in frozenset type.

dict Type:

--> It is a sequence data type, it able to store sequence of elements in the form   of Key-Value pairs.
--> To represent elements in dict data type we will use the following pattern.
    {Key1:Val1, Key2:Val2,....Key-n:Val-n}
EX:

```
stud_Dict = {111:'AAA', 222: 'BBB', 333:'CCC',
444:'DDD'}
```

```
print(stud_Dict)
print(type(stud_Dict))
print(id(stud_Dict))
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD'}
<class 'dict'>
2171314711912
```

--> In dict type, duplicate elements are not possibl at keys side, but, duplicate elements are possible at values side.
EX:

```
dict = {111:'AAA', 222: 'BBB', 333:'CCC', 444:'DDD',
222:'XXX', 555:'CCC'}
print(dict)
```
OP:

```
{111: 'AAA', 222: 'XXX', 333: 'CCC', 444: 'DDD', 555:
'CCC'}
```

--> It is not index based, but, we can get values on the basis of keys.
EX:

```
dict = {1:111, 2:222, 3:333, 4:444}
```

```
print(dict)
print(dict[1])
print(dict[2])
OP:
---
{1: 111, 2: 222, 3: 333, 4: 444}
111
222
```

--> It does not follow sorting order, but, it follows insertion order w.r.t the keys.

EX:

```
---
dict = {'a':'AAA', 'f':'FFF', 'b': 'BBB', 'e':'EEE', 'c':'CCC',
'd':'DDD'}
print(dict)
OP:
---
{'a': 'AAA', 'f': 'FFF', 'b': 'BBB', 'e': 'EEE', 'c': 'CCC', 'd':
'DDD'}
```

--> Dict type is able to allow heterogeneous elements at both keys side and values side.

EX:

```
----
dict = {1:111, 22.22:23456.3456, False:True, 'a':'AAA',
100:'XXX'}
```

```
print(dict)
OP:
----
{1: 111, 22.22: 23456.3456, False: True, 'a': 'AAA', 100:
'XXX'}
```

--> Dict data type is able to allow only one None element at keys side, but, any no of None elements at values side.

EX:

```
----
dict = {None: 'AAA', None:'BBB', 1:None, 2:None}
print(dict)
OP:
---
{None: 'BBB', 1: None, 2: None}
```

3. Type Casting:

The process of converting data from one data type to another data type is called as Type Casting.

To perform type casting Python has provided a set of predefined functions.
1. int()
2. float()
3. complex()

4. bool()
5. str()

1. int():

It can be used to convert the numbers from all the data types to int type except complex type.
EX:

```
a = 22.22
str = '10'
bool = True
complex = 10+2j

float_To_Int = int(a)
print(float_To_Int)

str_To_Int = int(str)
print(str_To_Int)

bool_To_Int = int(bool)
print(bool_To_Int)

#complex_To_Int = int(complex) --> TypeError: can't
convert complex to int
#print(complex_To_Int)
```

```
OP:
---
22
10
1
```

2. float():

--> It can be used to convert data from all the data types to float data type except from Complex.

Note: if we are trying to convert complex number to float number by using float() function then PVM will generate an error like "TypeError: can't convert complex to float"

EX:

```
a = 10
b = True
str = '22.22'
c = 10+2j

int_To_Float = float(a)
print(int_To_Float)

bool_To_Float = float(b)
```

```
print(bool_To_Float)

str_To_Float = float(str)
print(str_To_Float)

#complex_To_Float = float(c) --> TypeError: can't
convert complex to float
#print(complex_To_Float)
```

OP:

```
10.0
1.0
22.22
```

3. complex()

It can be used to convert data from all the data types to Complex data type.

Note: While converting data from int to complex, float to complex , bool to complex and str to complex then the provided values will be placed at real part in the generated complex number, in the generated complex number 0 will be provided as imaginary part.
EX:

```
a = 10
```

```
b = 22.22
c = True
str = '20'

int_To_Complex = complex(a)
print(int_To_Complex)

float_To_Complex = complex(b)
print(float_To_Complex)

bool_To_Complex = complex(c)
print(bool_To_Complex)

str_To_Complex = complex(str)
print(str_To_Complex)
```

OP:

```
(10+0j)
(22.22+0j)
(1+0j)
(20+0j)
```

If privide complex number in the form of String while converting data from String type to complex number then the provided complex number will be converted as it is from String type to complex type.

EX:

```
str1 = "10"
str1_To_Complex = complex(str1)
print(str1_To_Complex)# 10+0j

str2 = "20+5j"
str2_To_Complex = complex(str2)
print(str2_To_Complex)# 20+5j   or  (20+5j)+0j
```

OP:

```
(10+0j)
(20+5j)
```

4. bool():

It can be used to convert data from all the data types to bool data type.
EX:

```
a = 100
b = 22.22
c = 10 + 5j
d = "abc"

int_To_Bool = bool(a)
```

```
print(int_To_Bool)

float_To_Bool = bool(b)
print(float_To_Bool)

complex_To_Bool = bool(c)
print(complex_To_Bool)

str_To_Bool = bool(d)
print(str_To_Bool)
```

OP:

```
True
True
True
True
```

case-1: While converting data from int to bool , if the provided int value is non zero [+ve values and -ve values] then bool() function will return True value, if the provided int value is 0 then bool function will return 'False' value.

EX:

```
a = -100
b = 0
```

```
c = 100
print(bool(a))
print(bool(b))
print(bool(c))
OP:
---
True
False
True
```

Note: Same above situation is existed in the case of float to bool conversion.

EX:

```
a = -22.22
b = 0.0
c = 33.33
print(bool(a))
print(bool(b))
print(bool(c))
OP:
---
True
False
True
```

Case-2: In Python applications, it is possible to convert

data from None type to bool type, always, it will provide 'False' as an output.
EX:

```
a = None
print(bool(a))
```
OP:

```
False
```

Case-3: While converting data from str type to bool type , if we provide any data as String then bool() function will return 'True' value, but, if we provide empty string[""] then bool() function will return 'False' value.
EX:

```
a = "0"
b = "100"
c = "abc"
d = " "#space
e = ""#No space
f = "False"
print(bool(a))
print(bool(b))
print(bool(c))
print(bool(d))
print(bool(e))
print(bool(f))
```
OP:

```
True
True
True
True
False
True
```

5. str()

The main intention of this function is to convert data from all the data types to str type.
EX:

```
a = 10
b = 22.22
c = 10+5j
d = None
e = True

int_To_Str = str(a)
print(int_To_Str)
float_To_Str = str(b)
print(float_To_Str)
complex_To_Str = str(c)
```

```
print(complex_To_Str)
none_To_Str = str(d)
print(none_To_Str)
bool_To_Str = str(e)
print(bool_To_Str)
```

OP:

```
10
22.22
(10+5j)
None
True
```

4. Python Statements

Statement is the collection of expressions.

To prepare Python applications, Python has provided the following type of statements.

1. General Purpose Statements
2. Input And Output Statements
3. Conditional Statements
4. Iterative Statements
5. Transfer Statements

1. General Purpose Statements:

These Python statements are very much common and very much frequent in Python applications.
EX:
Declaring variables
Declaring functions
Declaring Classes
Creating Objects for the classes
Accessing variables and functions
....
----

2. Input And Output Statements:

Input:

If we use any statement to provide input data to the Python applications then that statements are called as "Input Statements".

In Python applications, we are able to provide three types of input data to the python programs.

1. Static Input
2. Dynamic Input
3. Command Line Input / Command Line Arguments

1. Static Input:

If we provide input data to the python applications at the time of writing programs then that input data is called as Static input.
EX:

```
a = 10
b = 5
print("ADD :", (a+b))
print("SUB :", (a-b))
print("MUL :", (a*b))
```
OP:

```
15
5
50
```

2. Dynamic Input:

If we provide input data to the python applications at Runtime then that input data is called as Dynamic Input.

To take dynamic input in python applications, we have to use the following predefined functions.

1. row_input()
2. input()

Q)What is the differernce between row_input() and input() functions?

Ans:

In Python2.x version, row_input() can be used to read all the types of dynamic input and it will return the dynamic input in the form of String. In Python2.x version input() function can be used to read any type of dynamic input and it will return that dynamic input in the provided format like int, float, bool, ...

In python3.x version, row_input() function is not existed, only input() function is existed and it is same as row_input() function of Python2.x version, that is, input() function is able to read all the types of dynamic input but it able to return in the form of String type.
EX:

```
str = input("Enter Data : ")
print(str)
print(type(str))
```
OP:

```
Enter Data : Durga Software Solution
Durga Software Solution
<class 'str'>
```

In Python , input() function is able to read data in the form of String type even we enter the data of types int, float, bool,..... If we want to get dynamic input data in the form of the data types like int, float, bool,... then we have to use the following steps.

1. Read Dynamic input in the form of String by using input() function.
2. Convert data from String type to other data types like int, float, bool,.... by using the predefined functions like int(), float(), bool(),...

EX:

```
a = input("First value  : ")
b = input("Second Value : ")
fval = int(a)
sval = int(b)
print("a   :",a)
print("b   :",b)
print("fval:", fval)
print("sval:", sval)
print("ADD :", (fval + sval))
print("SUB :", (fval - sval))
print("MUL :", (fval * sval))
```

OP:

```
---
First value  : 10
Second Value : 5
a   : 10
b   : 5
fval: 10
sval: 5
ADD : 15
SUB : 5
MUL : 50
```

EX:

```
---
eno = int(input("Employee Number  : "))
ename = input("Employee Name    : ")
esal = float(input("Employee Salary  : "))
eaddr = input("Employee Address : ")
empType = bool(input("Employee is Temp : "))
print()
print("Employee Details")
print("--------------------")
print("Employee Number   :", eno)
print("Employee Name     :", ename)
print("Employee Salary   :", esal)
print("Employee Address  :", eaddr)
print("Temporary Employee:", empType)
```
OP:

```
Employee Number  : 111
Employee Name    : AAA
Employee Salary  : 50000.0
Employee Address : Hyd
Employee is Temp :  [To give False value then dont
provide any data]

Employee Details
---------------------
Employee Number   : 111
Employee Name     : AAA
Employee Salary   : 50000.0
Employee Address  : Hyd
Temporary Employee: False
```

Q)Is it possible to take more than one dynamic input by using single input() function?

Ans:

No, it is not possible to take more than one dynamic

input from console by using single input() function. If we provide multiple values at a time on console then single input() function will read all the multiple values as single dynamic input of type str.

EX:

```
str = input("Enter Multiple Values : ")
print("str :",str)
print(type(str))
```

OP:

```
Enter Multiple Values : 10 20 30
str : 10 20 30
<class 'str'>
```

Note: If we want to get multiple values from our single dynamic input then we have to perform split operation over the dynamic input and get multiple values in the form of List and assign multiple values to multiple variables.

EX:

```
str = input("Enter Multiple Values : ")
list = str.split()
a,b,c = list   # a,b,c = ['10','20','30']
print(int(a))
print(int(b))
print(int(c))
```

OP:

```
Enter Multiple Values : 10 20 30
10
20
30
```

EX:

```
a,b,c = [int(x) for x in input("Enter Multiple Values :
").split()]
print(a)
print(b)
print(c)
```

OP:

```
Enter Multiple Values : 10 20 30
10
20
30
```

EX:

```
a,b = [int(x) for x in input("Enter two Values : ").split()]
print("ADD :", (a+b))
print("SUB :", (a-b))
print("MUL :", (a*b))
```
OP:

```
Enter two Values : 10 5
ADD : 15
SUB : 5
MUL : 50
```

If we want to evaluate the expressions which we provided as dynamic input then we have to use eval() function.
Note: input() function will read the provided expression as single dynamic input of type Str and it is not performing any evaluations over the provided expression.
EX:

```
expr = eval(input("Enter Expression : "))
print(expr)
```
OP:

```
Enter Expression : 10+20*3
70
```

Q)Is it possible to read list / tuple / set of values as dynamic input directly?

Ans:

No, input() is able to read all the types of data as String. If we want to get List, Tuple, Set, dict of elements as dynamic input then we have to use the following steps.
1. Read list / tuple/ set/ dict type elements as dynamic input in the form of   String by using input() function.
2. Convert String dynamic input into List, tuple, set and dict by using eval()   function.
EX:

```
list = eval(input("Enter List Elements with [] : "))
tuple = eval(input("Enter Tuple Elements with or with out () : "))
set = eval(input("Enter Set Elements with {} : "))
dict = eval(input("Enter dict Elements with {k:v} : "))

print(list, "----->", type(list))
print(tuple, "----->", type(tuple))
print(set, "----->", type(set))
print(dict, "----->", type(dict))
```

OP:
```
Enter List Elements with [] : [10,20,30,40,50]
Enter Tuple Elements with or with out () :
(10,20,30,40,50)
```

```
Enter Set Elements with {} : {10,20,30,40,50}
Enter dict Elements with {k:v} :
{10:100,20:200,30:300,40:400,50:500}
[10, 20, 30, 40, 50] -----> <class 'list'>
(10, 20, 30, 40, 50) -----> <class 'tuple'>
{40, 10, 50, 20, 30} -----> <class 'set'>
{10: 100, 20: 200, 30: 300, 40: 400, 50: 500} ----->
<class 'dict'>
```

3. Command Line Input / Command Line Arguments:

If we provide input data to the python program by specifying values along with "python" or "py" command on console or command prompt then that input data is called as Command Line input or Command Line Arguments.

```
EX: D:\python10>python Add.py 10 20
```

If we use the above command on command prompt then PVM will perform the following actions.
1. PVM will read all the command line input including python file name from console
2. PVM will store all the command line input in the form of String in a list   refered by argv in sys module.
Note: To get elements from 'argv' list type , we have to import argv in the present python file.

```
      from sys import argv
```

EX:

Test.py

```
from sys import argv
print(argv)
for x in argv:
            print(x)
```

OP:
```
D:\python10>python Test.py 10 20 30
['Test.py', '10', '20', '30']
Test.py
10
20
30
```

EX:Cal.py

```
from sys import argv
a = int(argv[1])
b = int(argv[2])
print("ADD :",(a+b))
print("SUB :",(a-b))
print("MUL :",(a*b))

D:\python10>py Cal.py 10 5
ADD : 15
SUB : 5
MUL : 50
```

EX: Test.py

```
------------
from sys import argv
print("No Of Command Line Arguments :",len(argv)-1)
sum = 0
print("Command line Arguments List  :",end=" ")
for x in range(1,len(argv)):
            print(argv[x],end=" ")
            sum = sum + int(argv[x])
print()
print("SUM of Command Line Arguments:",sum)
```

OP:

```
D:\python10>py Test.py 10 20 30 40
No of Command Line Arguments: 4
Command Line Input List : 10 20 30 40
SUM of Command Line Arg : 100
```

In Command line Inputs case, PVM will read all the command line inputs from console on the basis of space

seperator. If we want to provide a String data as comand line input including multiple multiple words then we have to use " "[Double quotations], it is not possible to use Single quotations and triple quotations.
EX:Test.py

```
from sys import argv
print(argv)
print(len(argv)-1)
```

OP:

```
D:\python10>py Test.py Durga Software Solutions
['Test.py', 'Durga', 'Software', 'Solutions']
3

D:\python10>py Test.py 'Durga Software Solutions'
['Test.py', "'Durga", 'Software', "Solutions'"]
3

D:\python10>py Test.py "Durga Software Solutions"
['Test.py', 'Durga Software Solutions']
1

D:\python10>py Test.py '''Durga Software Solutions'''
['Test.py', "'''Durga", 'Software', "Solutions'''"]
3
```

Output Statements:

If we send data from Python applications to output devices like Console, Printer, Network,..... then that Data is called as Output Data and this operation is called as Output Operation.

To perform Output operation if we use any statement then that statement is called as Output Statement.

To perform out put operation in python applications we will use a predefined function in the form of print(--), Where print() function was defined in such a way to display the provided data or the provided variables values and to keep cursor in the next line.

EX:

```
std_Roll_No = int(input("Student Roll Number   : "))
std_Name = input("Student Name          : ")
std_Aggr = float(input("Student Aggregate     : "))
std_Qual = eval(input("Student Qualifications : "))
std_Marks = eval(input("Student Marks        : "))
std_Courses = eval(input("Student Courses  : "))
std_Year_Of_Passes = eval(input("Student Year Of
Passes : "))
```

```
print()
print("Student Details")
print("--------------------")
print("Student Roll Number    :", std_Roll_No)
print("Student       Name      :", std_Name)
print("Student Aggregate %    :", std_Aggr)
print("Student Qualifications:", std_Qual)
print("Student Marks          :", std_Marks)
print("Student Courses        :", std_Courses)
print("Student Year of Passes:", std_Year_Of_Passes)
```

OP:

```
Student Roll Number   : 111
Student Name          : Durga
Student Aggregate     : 85.5
Student Qualifications : ["BTech","MTech","PHD"]
Student Marks         : (66,67,89,87,69,77)
Student Courses  : {"JAVA", "Python", "Oracle"}
Student Year Of Passes :
{"BTECH":2000,"MTECH":2002,"PHD":2007}

Student Details
--------------------
Student Roll Number   : 111
Student  Name         : Durga
Student Aggregate %   : 85.5
Student Qualifications: ['BTech', 'MTech', 'PHD']
Student Marks         : (66, 67, 89, 87, 69, 77)
Student Courses        : {'JAVA', 'Oracle', 'Python'}
Student Year of Passes: {'BTECH': 2000, 'MTECH': 2002,
'PHD': 2007}
```

In print() function, we are able to perform concatination operation by using , and + .

To perform concatination in print() function if we use + operator then we must provide both the data of String type, it is not possible to perform concatination between String and other data types.

EX:

```
age = 25
print("Age :"+age)
```

Status: TypeError: can only concatenate str (not "int") to str

EX:

```
name = "Durga"
print("Name :"+name)
```

OP:

```
Name :Durga
```

EX:

```
print("Durga"+"Software"+"Solutions")
```

```
OP:DurgaSoftwareSolutions
```

To perform concatination operation in print() function if we use , then we are able to get the concatination result with space seperator and it is possible to perform concatination between String type and any other type including string, int, float, bool,....

EX:

```
name = "Durga"
age = 25
salary = 25000.0
print("Name   :",name)
print("Age    :",age)
print("Salary :",salary)
```
OP:

```
Name   : Durga
Age    : 25
Salary : 25000.0
```

EX:

```
print("Durga","Software","Solutions")
```

OP:

Durga Software Solutions

In Print() function, if we want to concatinate a particular value to the string at the end and to keep cursor at the same line we have to use "end" keyword.
EX:

```
seconds = 45
minuts = 50
hours = 10
day = "Mon Day"
date = 15
month = 8
year = 2019
print("Today Date And Time ",end=":")
print(day,end="  ")
print(date,end=":")
print(month,end=":")
print(year,end="  ")
print(hours,end=":")
print(minuts,end=":")
print(seconds)
```

OP:

```
Today Date And Time :Mon Day  15:8:2019  10:50:45
```

In print() function, if we want to display multiple values with a particular seperator then we have to use "sep" keyword.

EX:

```
---
a = 10
b = 20
c = 30
d = 40
print("a,b,c,d values :",a,b,c,d,sep=" ")
```

OP:

```
---
a,b,c,d values : 10 20 30 40
```

EX:

```
---
seconds, minuts, hours, day, date, month, year = 45,
50, 10, "Monday", 15, 8, 2019
print("Today Date And Time ",end=":")
print(day,end="  ")
print(date,month,year,sep="/",end="  ")
print(hours,minuts,seconds,sep=":")
```

OP:

```
---
Today Date And Time :Monday  15/8/2019  10:50:45
```

In print() function , if we want to perform concatination over a particular string upto the specified number of times then we have to use '*' notation.

Syntax:

----

1. print("str"*n) --> It will perform concatination upto n times
2. print(n*"str") --> It will perform concatination upto n times
3. print(n1*"str"*n2)--> It will perform concatination upto n1*n2 times

EX:

```
print("Durgasoft "*4)
print(4*"Durgasoft\t")
print(4*"Durgasoft\n"*4)
```

OP:

```
Durgasoft Durgasoft Durgasoft Durgasoft
Durgasoft	Durgasoft	Durgasoft	Durgasoft

Durgasoft
-----
--16 times-
```

In python applications, we are able to format the outputs by using '%' operator through print() function.

%i ---> int
%d ---> int
%s ---> str, list, tuple,....
%f ---> float

EX:

```
a = 10
b = "abc"
c = 22.22
print("a , b and c values are %d %s %f"%(a,b,c))
```

OP:

```
a , b and c values are 10 abc 22.220000
```

In general, we will use formatted output while preparing template messages as outputs.
Note: Template messages is a message with variables, where variables will have values at runtime.
EX:

```
name = input("Name       : ")
```

```
age = int(input("Age        : "))
company = input("Company    : ")
salary = float(input("Salary     : "))
skills =  eval(input("Skill Set :"))
print("I am %s, my age is %d years and i am working
with %s  \n and My Salary is %f and My Skill set is
%s"%(name,age,company,salary,skills))
```

OP:

```
Name       : Anil
Age        : 32
Company    : TCS
Salary     : 55000.0
Skill Set :["Java", "Python", "Oracle", "UI Techs"]
I am Anil, my age is 32 years and i am working with TCS

and My Salary is 55000.000000 and My Skill set is
['Java', 'Python', 'Oracle', 'UI Techs']
```

In print() function we are able to format the data by using place holders also.
In python applications, w are able to provide place holders in print() function in the following two ways.
1. Index Based Place holders.
2. Name Based Place holders.

1. Index Based Place holders:

Syntax: {index}
To provide value to the place holder we have to use format() function with value.
EX:

```
a = 10
b = 20
c = 30
print("a value is {0} \nb value is {1} \nc value is
{2}".format(a,b,c))
```

OP:

```
a value is 10
b value is 20
c value is 30
```

2. Name Based Place Holders.

Syntax: {name}
To provide value to the Name based place holder we have to use format() function with value assignment to the respective place holder name.
EX:

```
a = 10
b = 20
c = 30
print("a value is {x} \nb value is {y} \nc value is
{z}".format(x=a,y=b,z=c))
OP:
---
a value is 10
b value is 20
c value is 30
```

In Python applications, in print() function, we are able to provide both index based place holders and name based place holders , but, first we have to provide index based place holders next we have to provide name based place holder. If we provide name based place holders first and index based place holders next then PVM will raise an error like "Syntax Error: positional argument follows keyword argument".

EX:

```
a = 10
b = 20
c = 30
d = 40
print("a value is {0} \nb value is {1} \nc value is {x}\nd
```

value is {y}".format(a,b,x=c,y=d))
OP:

```
a value is 10
b value is 20
c value is 30
d value is 40
```

EX:

```
a = 10
b = 20
c = 30
d = 40
print("a value is {x} \nb value is {y} \nc value is {2}\nd
value is {3}".format(x=a,y=b,c,d))
```
Status: SyntaxError: positional argument follows keyword argument

3. Conditional Statements:

These statements are able to allow to execute a block of instructions on the basis of a particular condition.
EX: if
Syntax-1:

```
if condition:
  --instructions--

Syntax-2:
---------
if condition:
  ---instructions---
else:
  ---insrtuctions----

Syntax-3:
---------
if condition:
  ---instructions----
elif condition:
  ---instructions----
elif condition:
  ---instructions----
---
---
else:
  ---instructions----

EX:
---
name = input("Enter Name : ")
if name == "Durga":
    print("Your Name is %s"%name)
```

OP:

```
Enter Name : Durga
Your Name is Durga
```

EX:

```
name = input("Enter Name : ")
if name == "Durga":
    print("Your Name is Durga")
else:
    print("Your Name is not Durga, Your Name is
%s"%name)
```

OP:

```
Enter Name : Anil
Your Name is not Durga, Your Name is Anil
```

EX:

```
no = int(input("Enter a number from 1 to 7 : "))
if no == 1:
    print("Monday")
elif no == 2:
    print("Tuesday")
```

```
elif no == 3:
    print("Wensday")
elif no == 4:
    print("Thursday")
elif no == 5:
    print("Friday")
elif no == 6:
    print("Saturday")
elif no == 7:
    print("Sunday")
else:
    print("Sorry, No week day for your number, please
enter number from 1 to 7")
```

OP:

```
Enter a number from 1 to 7 : 5
Friday
```

OP:

```
Enter a number from 1 to 7 : 10
Sorry, No week day for your number, please enter
number from 1 to 7
```

EX:

```
no = int(input("Enter a Number : "))
if no % 2 == 0 :
    print("%d is Even Number"%no)
else:
    print("%d is Odd NUmber"%no)
```

OP:

```
Enter a Number : 10
10 is Even Number
```

OP:

```
Enter a Number 5
5 is Odd Number
```

EX:

```
no1 = int(input("Enter First Number  : "))
no2 = int(input("Enter Second Number : "))
if no1 < no2:
    print("%d is Biggest Number"%no2)
elif no1 > no2:
    print("%d is Biggest Number"%no1)
else:
    print("Both are Equal")
```

OP:

```
Enter First Number  : 10
Enter Second Number : 20
20 is Biggest Number
```

OP:

```
Enter First Number  : 10
Enter Second Number : 10
Both are Equal
```

EX:

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

if a > b:
    if a > c:
        print("%d is Biggest Number"%a)
    else:
        print("%d is Biggest Number"%c)
elif b > c:
    if b > a:
        print("%d is Biggest Number"%b)
    else:
        print("%d is Biggest Number"%a)
elif c > a:
    if c > b:
        print("%d is Biggest Number" %c)
    else:
        print("%d is Biggest Number" %b)
else:
    print("All are Equal")
```

```
OP:
---
Enter First Number  : 10
Enter Second Number : 20
Enter Third Number  : 30
30 is Biggest Number

OP:
---
Enter First Number  : 10
Enter Second Number : 10
Enter Third Number  : 20
20 is Biggest Number

OP:
---
Enter First Number  : 10
Enter Second Number : 10
Enter Third Number  : 10
All are Equal
```

Note: In C, C++ and Java switch programming consatruct is existed , but, in Python switch programming construct is not existed, we are able to manage switch programming construct by using if-elif-else syntax.

4. Iterative Statements:

These statements are able to allow to execute a set of instructions repeatedly on the basis of a particular conditional expression.
EX: for, while
Note: In Python do-while iterative statement is not existed.

for :

```
for var in range(start, end, step)/sequence_Data_Type:
   ----instructions------
```
EX:

```
for x in range(10):
    print(x)
```

EX:

```
for x in range(0,10):
    print(x)
```
EX:

```
for x in range(0,10,1):
    print(x)
```

OP:

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

EX:

```
for x in range(0,10,1):
    print(9-x)
```

```
OP:
---
9
8
7
6
5
4
3
2
1
0
```

EX:

```
no = int(input("Enter Number  : "))
for x in range(0,no,2):
    print(x)
```

```
OP:
---
0
2
4
6
8
10
12
```

```
14
16
18
```

EX:

```
no = int(input("Enter Number  : "))
for x in range(1,no,2):
    print(x)
```
OP:

```
1
3
5
7
9
11
13
15
17
19
```

EX:[Prime Number]

```
no = int(input("Enter Number  : "))

for x in range(2,no+1):
    b = True
    for y in range(1,x):
        if not((y==1) or (y==x)) :
            if x%y == 0:
                b = False

    if b == True:
        print("%d is Prime Number"%x)
```

OP:

```
---
Enter Number  : 20
2 is Prime Number
3 is Prime Number
5 is Prime Number
7 is Prime Number
11 is Prime Number
13 is Prime Number
17 is Prime Number
19 is Prime Number
```

Note: Providing a loop inside a loop is called as nested loop.

EX:

```
---
for x in range(0, 10, 1):
    for y in range(0, 10, 1):
        print(x,"  ", y )
OP:
----
0  0
0  1
----
----
0  9
----
----
----
9  0
----
9  9
```

If we want to read elements from the data types like str, list, tuple, set, ... then we have to use for-Each loop.

EX:

```
---
str = "Durga Software Solutions"
for x in str:
    print(x,end="  ")

OP:
---
D  u  r  g  a     S  o  f  t  w  a  r  e     S  o  l  u  t  i  o  n
s
```

EX:

```
list = [10,20,30,40,50,60,70,80,90, 100]
for x in list:
    print(x,end="  ")
```

OP:

```
10  20  30  40  50  60  70  80  90  100
```

EX:

```
dict = {"A":"AAA", "B":"BBB", "C":"CCC", "D":"DDD",
"E":"EEE", "F":"FFF"}
for x in dict:
    print(x,"--->",dict[x])
```
OP:

```
A ---> AAA
B ---> BBB
C ---> CCC
D ---> DDD
E ---> EEE
F ---> FFF
```

while:

Syntax:

```
while condition:
  ----instructions-----
```

EX:

```
a = 0;
while a < 10:
    print(a)
    a = a + 1
```

OP:

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

EX:

```
---
str = "Durga Software Solutions"

a = 0;
while a < len(str):
    print(str[a])
    a = a+ 1
OP:
---
D
u
r
g
a

S
o
f
t
w
a
r
e

S
o
l
```

u
t
i
o
n
s

EX:

```
list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
result = 0
a = 0
print("List of Elements :", end = " ")
while a < len(list):
    print(list[a],end="  ")
    result = result + list[a]
    a = a+ 1
print()
print("SUM of List Elements :", result)
print("AVG of all List Elements :",(result/len(list)))
```

OP:

```
List of Elements : 10  20  30  40  50  60  70  80  90  100

SUM of List Elements : 550
AVG of all List Elements : 55.0
```

EX:

```
dict = {"A":"AAA", "B":"BBB", "C":"CCC", "D":"DDD",
"E":"EEE", "F":"FFF"}
list = list(dict.keys())

a = 0;
while a < len(list):
    print(list[a],"---->",dict[list[a]])
    a = a + 1
```
OP:

```
A ----> AAA
B ----> BBB
C ----> CCC
D ----> DDD
E ----> EEE
F ----> FFF
```

5. Transfer Statements:

These statements are able to bypass flow of execution from one instruction to another instruction.

EX: break, continue

break:

In general, break statement will be used in loops, it will bypass flow of execution to out side of the current loop.
EX:

```
print("Before Loop")
for x in range(0,10):
    if x == 5:
        break
    print(x)
print("After Loop")
```
OP:

```
Before Loop
0
1
2
3
4
After Loop
```

In python applications, if we provide 'break' statement inside the nested loop then break statement will give effect to the respective nested loop only, not to outer loop.
EX:

```
for x in range(0,10):
    for y in range(0,10):
        if y == 5:
            break;
        print(x,"---->",y)
```

OP:

```
0 0
0 1
---
0 4
---
---
9 0
9 1
---
9 4
```

continue:

It will skip the remaining instructions execution in current iteration and it will continue with next iteration.

EX:

```
for x in range(0,10):
    if x == 5:
        continue
    print(x)
```
OP:

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

In Python applications, if we provide 'continue' statement inside nested loop then 'continue' statement will give effect to nested loop only, it will not give effect to outer loop.
EX:

```
for x in range(0,10):
    for y in range(0, 10):
        if y == 5:
            continue
        print(x,"---->",y)
```
OP:

```
---
0 ----> 0
0 ----> 1
0 ----> 2
0 ----> 3
0 ----> 4
0 ----> 6
0 ----> 7
0 ----> 8
0 ----> 9
-----
-----
9 ----> 0
9 ----> 1
9 ----> 2
9 ----> 3
9 ----> 4
9 ----> 6
9 ----> 7
9 ----> 8
9 ----> 9
```

pass:

It can be used to pass flow of execution to out side of the current block.
EX:

```
a = 20
if a == 10:
    print("a value is %d"%a)
else:
   pass
print("After if-else")
```

OP:

```
After if-else
```

del:

It can be used to delete a particular variable from Python program scope. If we delete any variables by using 'del' keyword then we are unable to use that variable, in the case of we use that variable then PVM will raise an error like "NameError: name 'a' is not defined".
EX:

```
a = 10
print("a value is %d"%a)
del a
print("a value is %d"%a)
```

OP
```
a value is 10
"NameError: name 'a' is not defined".
```

Patterns:

Q1)Write a Python program to display the following patterns?
----
1)
```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```
EX:

```
for x in range(0,10):
    for y in range(0,10):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(y,end=" ")
    print()
```

3)

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

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

```
for x in range(0,10):
    for y in range(0,10):
        print(9-y,end=" ")
    print()
```

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

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(x,end=" ")
    print()
```

5)

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

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(9-x,end=" ")
    print()
```

6)

```
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
```

```
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
A B C D E F G H I J
```

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(chr(65+y),end=" ")
    print()
```

7)
```
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
J I H G F E D C B A
```
EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(chr(74-y),end=" ")
    print()
```

8)

```
A A A A A A A A A A
B B B B B B B B B B
C C C C C C C C C C
D D D D D D D D D D
E E E E E E E E E E
F F F F F F F F F F
G G G G G G G G G G
H H H H H H H H H H
I I I I I I I I I I
J J J J J J J J J J
```

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(chr(65+x),end=" ")
    print()
```

9)

```
J J J J J J J J J J
I I I I I I I I I I
H H H H H H H H H H
G G G G G G G G G G
F F F F F F F F F F
```

```
E E E E E E E E E E
D D D D D D D D D D
C C C C C C C C C C
B B B B B B B B B B
A A A A A A A A A A
```

EX:

```
for x in range(0,10):
    for y in range(0,10):
        print(chr(74-x),end=" ")
    print()
```

Q2)Write python programs to display the following patterns?

----

1)

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

EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(y,end=" ")
    print()
```

3)

```
9
```

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

EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(9-y,end=" ")
    print()
```

4)

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

EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(x,end=" ")
    print()
```

5)
```
9
8 8
7 7 7
6 6 6 6
5 5 5 5 5
4 4 4 4 4 4
3 3 3 3 3 3 3
2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
```
EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(9-x,end=" ")
    print()
```
6)
```
A
A B
```

```
A B C
A B C D
A B C D E
A B C D E F
A B C D E F G
A B C D E F G H
A B C D E F G H I
A B C D E F G H I J
```
EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(chr(65+y),end=" ")
    print()
```

7)

```
J
J I
J I H
J I H G
J I H G F
J I H G F E
J I H G F E D
J I H G F E D C
J I H G F E D C B
J I H G F E D C B A
```
EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(chr(74-y),end=" ")
    print()
```

8)

```
A
B B
C C C
D D D D
E E E E E
F F F F F F
G G G G G G G
H H H H H H H H
I I I I I I I I I
J J J J J J J J J J
```

EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(chr(65+x),end=" ")
    print()
```

9)

```
J
I I
```

```
H H H
G G G G
F F F F F
E E E E E E
D D D D D D D
C C C C C C C C
B B B B B B B B B
A A A A A A A A A A
```
EX:

```
for x in range(0,10):
    for y in range(0,x+1):
        print(chr(74-x),end=" ")
    print()
```

Q3)Write Python applications to display the following patterns?
----
1)
```
         *
        * *
       * * *
      * * * *
     * * * * *
    * * * * * *
   * * * * * * *
```

```
    * * * * * * * *
  * * * * * * * * *
* * * * * * * * * *
```
EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print("*",end=" ")
    print()
```

2)
```
                  0
                0 1
              0 1 2
            0 1 2 3
          0 1 2 3 4
        0 1 2 3 4 5
      0 1 2 3 4 5 6
    0 1 2 3 4 5 6 7
  0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```
EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(z,end=" ")
    print()
```

3)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(9-z,end=" ")
    print()
```

4)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(x,end=" ")
    print()
```

5)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(9-x,end=" ")
    print()
```

6)

```
                  A
                 A B
                A B C
               A B C D
              A B C D E
             A B C D E F
            A B C D E F G
           A B C D E F G H
          A B C D E F G H I
         A B C D E F G H I J
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(chr(65+z),end=" ")
    print()
```

7)

```
                  J
                 J I
                J I H
               J I H G
              J I H G F
             J I H G F E
            J I H G F E D
           J I H G F E D C
          J I H G F E D C B
         J I H G F E D C B A
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(chr(74-z),end=" ")
    print()
```

8)

```
                  A
```

```
        B B
       C C C
      D D D D
     E E E E E
    F F F F F F
   G G G G G G G
  H H H H H H H H
 I I I I I I I I I
J J J J J J J J J J
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(chr(65+x),end=" ")
    print()
```

9)

```
         J
        I I
       H H H
      G G G G
     F F F F F
    E E E E E E
   D D D D D D D
  C C C C C C C C
```

```
 B B B B B B B B B
A A A A A A A A A A
```
EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end=" ")
    for z in range(0, x+1):
        print(chr(74-x),end=" ")
    print()
```

Q4)Write Python programs to display the following patterns?
----

1)
```
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```
EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(y,end=" ")
    print()
```

3)

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

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

```
for x in range(0,10):
    for y in range(0,10-x):
        print(9-y,end=" ")
    print()
```

4)

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

```
for x in range(0,10):
    for y in range(0,10-x):
        print(x,end=" ")
    print()
```

5)

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

EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(9-x,end=" ")
    print()
```

6)

```
A B C D E F G H I J
A B C D E F G H I
A B C D E F G H
```

```
A B C D E F G
A B C D E F
A B C D E
A B C D
A B C
A B
A
```
EX:

```
for x in range(0,10):
    for y in range(0,10-x):
       print(chr(65+y),end=" ")
    print()
```

7)
```
J I H G F E D C B A
J I H G F E D C B
J I H G F E D C
J I H G F E D
J I H G F E
J I H G F
J I H G
J I H
J I
J
```
EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(chr(74-y),end=" ")
    print()
```

8)

```
A A A A A A A A A A
B B B B B B B B B
C C C C C C C C
D D D D D D D
E E E E E E
F F F F F
G G G G
H H H
I I
J
```

EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(chr(65+x),end=" ")
    print()
```

9)

```
J J J J J J J J J J
I I I I I I I I I
H H H H H H H H
```

```
G G G G G G G
F F F F F F
E E E E E
D D D D
C C C
B B
A
```

EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(chr(74-x),end=" ")
    print()
```

Q)Write Python programs to display the following patterns?
----

1)

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

```
        * *
         *
```

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(z,end=" ")
    print()
```

3)

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

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(9-z,end=" ")
    print()
```

4)

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

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(x,end=" ")
    print()
```

5)

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

```
       2 2 2
        1 1
         0
```
EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(9-x,end=" ")
    print()
```

6)

```
A B C D E F G H I J
 A B C D E F G H I
  A B C D E F G H
   A B C D E F G
    A B C D E F
     A B C D E
      A B C D
       A B C
        A B
         A
```
EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(chr(65+z),end=" ")
    print()
```

7)

```
J I H G F E D C B A
  J I H G F E D C B
    J I H G F E D C
      J I H G F E D
        J I H G F E
          J I H G F
            J I H G
              J I H
                J I
                  J
```

EX:

```
---
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(chr(74-z),end=" ")
    print()
```

8)

```
A A A A A A A A A A
  B B B B B B B B B
    C C C C C C C C
      D D D D D D D
        E E E E E E
          F F F F F
            G G G G
              H H H
                I I
                  J
```

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(chr(65+x),end=" ")
    print()
```

9)

```
J J J J J J J J J J
  I I I I I I I I I
    H H H H H H H H
      G G G G G G G
        F F F F F F
          E E E E E
            D D D D
```

```
              C C C
                B B
                  A
```

EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end=" ")
    for z in range(0,10-x):
        print(chr(74-x),end=" ")
    print()
```

Q6)Write Python programs to display the following patterns?

----

1)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(z,end=" ")
    print()
```

3)

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(9-z,end=" ")
    print()
```

4)

```
         0
        1 1
       2 2 2
      3 3 3 3
```

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(x,end=" ")
    print()
```

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(9-x,end=" ")
    print()
```

6)
```
         A
        A B
       A B C
      A B C D
     A B C D E
    A B C D E F
   A B C D E F G
  A B C D E F G H
 A B C D E F G H I
A B C D E F G H I J
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(chr(65+z),end=" ")
    print()
```

7)

```
         J
        J I
       J I H
      J I H G
     J I H G F
    J I H G F E
   J I H G F E D
  J I H G F E D C
 J I H G F E D C B
J I H G F E D C B A
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(chr(74-z),end=" ")
    print()
```

8)

```
         A
        B B
```

```
       C C C
      D D D D
     E E E E E
    F F F F F F
   G G G G G G G
  H H H H H H H H
 I I I I I I I I I
J J J J J J J J J J
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(chr(65+x),end=" ")
    print()
```

9)

```
         J
        I I
       H H H
      G G G G
     F F F F F
    E E E E E E
   D D D D D D D
  C C C C C C C C
```

```
 B B B B B B B B B
A A A A A A A A A A
```

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print(chr(74-x),end=" ")
    print()
```

Q7)Write Python programs to display the following patterns?
----
1)

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

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print("*",end=" ")
    print()
```

2)

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

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(z,end=" ")
    print()
```

3)

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

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(9-z,end=" ")
    print()
```

4)

```
          0
        1 1 1
      2 2 2 2 2
    3 3 3 3 3 3 3
  4 4 4 4 4 4 4 4 4
```

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(x,end=" ")
    print()
```

5)

```
          5
        4 4 4
      3 3 3 3 3
    2 2 2 2 2 2 2
```

```
 1 1 1 1 1 1 1 1 1
```
EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(5-x,end=" ")
    print()
```

6)
```
        A
      A B C
    A B C D E
  A B C D E F G
A B C D E F G H I
```

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(chr(65+z),end=" ")
    print()
```

7)

```
         J
       J I H
     J I H G F
   J I H G F E D
 J I H G F E D C B
```
EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(chr(74-z),end=" ")
    print()
```

8)

```
         A
       B B B
     C C C C C
   D D D D D D D
 E E E E E E E E E
```
EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(chr(65+x),end=" ")
    print()
```

9)

```
         E
       D D D
     C C C C C
   B B B B B B B
 A A A A A A A A A
```

EX:

```
for x in range(0,5):
    for y in range(0,5-x):
        print(" ",end=" ")
    for z in range(0,2*x+1):
        print(chr(69-x),end=" ")
    print()
```

Q8)Write Python Programs to display the following patterns:

----

1)

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

EX:

```
for x in range(0,10):
    for y in range(0,10-x):
        print(" ",end=" ")
    for z in range(0,2*x):
        print("*",end=" ")
    print()
```

Q9)Write python programs to display the following patterns?
----
1)

```
* * * * * * * * * * 
 * * * * * * * * * 
  * * * * * * * * 
   * * * * * * * 
    * * * * * * 
     * * * * * 
      * * * * 
       * * * 
        * * 
         * 
```
EX:

```
for x in range(0,10):
    for y in range(0,x):
        print(" ",end="")
    for z in range(0,10-x):
        print("*",end=" ")
    print()
```

Q10)Write Python programs to display the following patterns?

----

1)

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

```
       * * *
        * *
         *
```
EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print("*",end=" ")
    print()
for x in range(0, 9):
    for y in range(0,x+1):
        print(" ",end="")
    for z in range(0,9-x):
        print("*", end=" ")
    print()
```

Q11)Write Python programs to display the following patterns?

```
**********
**********
**********
**********
**********
   ****
   ****
   ****
   ****
   ****
```
EX:

```
for x in range(0,5):
    for y in range(0,10):
        print("*",end="")
    print()
for x in range(0,5):
    for y in range(0,3):
        print(" ",end="")
    for z in range(0,4):
        print("*",end="")
    print()
```

Q12)Write Python Programs to display the following patterns?
----
1)
```
         *
        * *
       * * *
      * * * *
     * * * * *
    * * * * * *
```

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

EX:

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print("*",end=" ")
    print()
for x in range(0,7):
    for y in range(0,3):
        print(" ",end=" ")
    for z in range(0,4):
        print("*",end=" ")
    print()
```

13)

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

EX:

```
for x in range(0,10):
    for y in range(0, x+1):
        print("*",end=" ")
    print()
for x in range(0,9):
    for y in range(0,9-x):
        print("*",end=" ")
    print()
```

14)

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

```
for x in range(0,10):
    for y in range(0,9-x):
        print(" ",end="")
    for z in range(0,x+1):
        print("*",end="")
    print()
for x in range(0,9):
    for y in range(0,x+1):
        print(" ",end="")
    for z in range(0,9-x):
        print("*",end="")
    print()
```

'str' Data Type:

String is the collection of Symbols, where the symbols may be the combination of alphabets, digits, ...

In Python applications, we are able to represent String data by using ' ' or " " or ''' '''.
EX:

```
str1 = 'Durga Software Solutions'
print(str1)
str2 = "Durga Software Solutions"
```

```
print(str2)
str3 = '''Durga Software SOlutions'''
print(str3)
str4 = '''Durga
Software
Solutions'''
print(str4)
```

OP:

```
Durga Software Solutions
Durga Software Solutions
Durga Software SOlutions
Durga
Software
Solutions
```

In Python applications, we are able to provide ' '[Single quotation] with / in single quaotation, we can provide double quotations dirdctly in single quotations and we are unable to provide triple quotations in single quotations. If we provide triple quotations with / then PVM will take /' as string and the remaining single quotations are treated as empty string and PVM will display only single quotation in the string.
EX:

```
str1 = 'Durga \'Software\' Solutions'
print(str1)
str2 = 'Durga "Software" Solutions'
print(str2)
str3 = 'Durga \'''Software\''' Solutions'
print(str3)
```
OP:

```
Durga 'Software' Solutions
Durga "Software" Solutions
Durga 'Software' Solutions
```

In double quatotations, it is possible to provide single quotations and triple quotations directly , but, it is not possible to provide double quotations directly , but, we can provide double quotations in double quotations with \ .

EX:

```
str1 = "Durga 'Software' Solutions"
print(str1)
str2 = "Durga \"Software\" Solutions"
print(str2)
str3 = "Durga '''Software''' Solutions"
print(str3);
```
OP:

```
Durga 'Software' Solutions
Durga "Software" Solutions
Durga '''Software''' Solutions
```

In triple quotations, we are able to provide single quotations and double quotations directly, we are unable to provide triple quotations directly but we are able to provide triple quotations with \ .

EX:

```
str1 = '''Durga 'Software' Solutions'''
print(str1)
str2 = '''Durga "Software" Solutions'''
print(str2)
str3 = '''Durga \'''Software\''' Solutions'''
print(str3)
```

EX:

```
Durga 'Software' Solutions
Durga "Software" Solutions
Durga '''Software''' Solutions
```

EX:

```
str1 = '\'Durga\' "Software" \'''Solutions\''''
print(str1)
```

```
str2 = '''Durga' \"Software\" '''Solutions'''
print(str2)
str3 = """Durga' "Software" \'''Solutions\''' """
print(str3)
OP:
---
'Durga' "Software" 'Solutions'
'Durga' "Software" '''Solutions'''
'Durga' "Software" '''Solutions'''
```

In Python applications, we are able to read individual characters from String by using normal for loop , for-Each loop and by using while loop.

EX:

```
str = "Durga Software Solutions"
for x in range(0,len(str)):
    print(str[x],end="  ")
print()
for x in str:
    print(x,end="  ")
print()
a = 0
while a < len(str):
    print(str[a],end="  ")
    a = a + 1
```

OP:

```
D u r g a   S o f t w a r e   S o l u t i o n
s
D u r g a   S o f t w a r e   S o l u t i o n
s
D u r g a   S o f t w a r e   S o l u t i o n
s
```

In string data types, we are able to read string elements in both forward direction and Backward direction. If we want to read elements in forward direction then we have to provide +ve index values to str data type and it must be started with 0. If we want to read elements in Backward direction then we must provide -ve index values to str data type and it must be started from -1 .
EX:

```
str = "Durga Software Solutions"
print("Forward Direction : ", end="")
for x in range(0, len(str)):
    print(str[x], end=" ")
print()
print("Backward Direction : ", end="")
for x in range(0,len(str)):
    print(str[-(x+1)], end=" ")
```

OP:

```
Forward Direction : D u r g a   S o f t w a r e   S o l u t i
o n s
Backward Direction : s n o i t u l o S   e r a w t f o S   a
g r u D
```

In String type data, we are able to retrieve elements individually by using slice operator.
Syntax:
----
str[Val1:Val2:Val3]
Val1 ---> Start Index, it may be +ve or -ve value
Val2 ---> End Index, it may be +ve or -ve value
Val3 ---> Step length, it may be +ve or -ve value

If we provide +ve value to step length then PVM will read elements in forward direction from the specified start index and upto the specified End Index-1.

If we provide -ve value to step length then PVM will read elements in Backward direction from the specified start index and upto the specified End Index-1.

If the step Length value is +ve value then
The Default value for Start Index is 0

The default value for End Index is len(str)

If we provide -ve value to step length then
The default value for Start Index is -1
The default value for end Index is -(len(str)+1)

The default value for step length is 1.

EX:

```
str = "Durga Software Solutions"
print(str[0:5:1])
print(str[6:14:1])
print(str[15:24:1])
print(str[0:24:2])
print(str[:24:1])
print(str[0::1])
print(str[0:24:])
print(str[::])
```
OP:

```
Durga
Software
Solutions
DraSfwr ouin
Durga Software Solutions
Durga Software Solutions
Durga Software Solutions
Durga Software Solutions
```

EX:

```
str = "Durga Software Solutions"
print(str[-1:-25:-1])
print(str[-1:-10:-1])
print(str[-11:-19:-1])
print(str[-20:-25:-1])
print(str[:-25:-1])
print(str[-1::-1])
print(str[::-1])
print(str[::])
```

OP:

```
snoituloS erawtfoS agruD
snoituloS
erawtfoS
agruD
snoituloS erawtfoS agruD
snoituloS erawtfoS agruD
snoituloS erawtfoS agruD
Durga Software Solutions
```

EX:

```
---
str = "Durga Software Solutions"
print(str[4:-25:-1])
print(str[-18:14:1])
print(str[13:5:-1])
print(str[-1:14:-1])

OP:
---
agruD
Software
erawtfoS
snoituloS
```

In the case of slice operator even if we provide start index and end index values in out side range of the string data index values PVM will not provide any error like IndexError.

EX:

```
---
str = "Durga Software Solutions"
print(str[0:1000:1])
print(str[-100:1000:1])
print(str[-1:-1000:-1])

OP:
----
```

```
Durga Software Solutions
Durga Software Solutions
snoituloS erawtfoS agruD
```

Note: If we are trying to retrieve an element from String data on the basis of a particular index value which is not in the range of String data index values then PVM will provide an error like "IndexError: string index out of range".
EX:

```
str = "Durga Software Solutions"
print(str[30]) # IndexError: string index out of range
print(str[-30])# IndexError: string index out of range
```

Note: If we provide 0[zero] value to step length in slice operator then PVM will provide an error like "ValueError: slice step cannot be zero".
EX:

```
str = "Durga Software Solutions";
print(str[10:20:0])
Status: ValueError: slice step cannot be zero
```

In String type, we are able to perform concatination operation by using + and * operators. + operator is able to perform concatination over two string values , if we provide any other data type along with string value then PVM will raise an error. * operator is also called as Repeate operator , it able to perform concatination operation on the same string repeatedly upto the specified no of times. * operator required one operand is string and another operand is int value.

EX:

```
print("Durga "+"Software "+"Solutions")
str1 = "Durga "
str2 = "Software "
str3 = "Solutions"
str4 = str1+str2+str3
print(str4)
#str5 = "Durga"+5 ---> Error
```

OP:

```
Durga Software Solutions
Durga Software Solutions
```

EX:

```
print("Durga "*5)
print(5*"Durga ")
print(3*"Durga "*3)
str = "Durga "*3
print(str)
```

```
OP:
---
Durga Durga Durga Durga Durga
Durga Durga Durga Durga Durga
Durga Durga Durga Durga Durga Durga Durga Durga
Durga
Durga Durga Durga
```

Note: In Python applications, ',' operator is acting as concatination operator inside print() function only, it is not possible to use ',' operator as concatination operator in out side of the print() function, but we can use + and * concatination operators in print() function and in out side of the print() function.

In String data type, if we want to compare two string values we have to use the following comparision operators.

```
   == ,  <, >
```

Where '==' operator will check whether two string values are same or not.

Where '<' operator will check whether the provided first operand is existed before the provided second operand or not in dictionary order.

Where '>' operator will check whether the provided first operand is existed after the provided second operand or not in dictionary order.

EX:

```
str1 = "abc"
str2 = "def"
str3 = "abc"
print(str1 == str2)
print(str2 == str3)
print(str3 == str1)
print(str1 < str2)
print(str2 < str3)
print(str3 < str1)
print(str1 > str2)
print(str2 > str3)
print(str3 > str1)
```

OP:

```
False
False
True
True
False
```

```
False
False
True
False
```

In string data type, we are able to check whether a string is existed or not existed in the original string by using "Membership Operators".

In Python, there are two membership operators
1. in
2. not in

EX:

```
str = "Durga Software Solutions"
print("Durga" in str)
print("Software" in str)
print("Solutions" in str)
print("Tech" in str)
print("Tech" not in str)
print("Durga" not in str)
```

OP:

```
True
True
```

```
True
False
True
False
```

In String type, if we want to split a string into no of sub strings in the form of a list then we have to use split() predefined function. split() function will take
space as default delimiter and it is possible to provide our own delimiter as parameter.

EX:

```
str = "Durga Software Solutions"
list = str.split()
print(list)
for x in list:
    print(x)
print()
str = "durga@durgasoft.com"
list = str.split("@")
print(list)
for x in list:
    print(x)
```

OP:

```
['Durga', 'Software', 'Solutions']
Durga
Software
Solutions

['durga', 'durgasoft.com']
durga
durgasoft.com
```

In string type, if we want to get the location of a particular substring in the original string then we have to use the following two predefined functions.

1. find()
2. index()

Q)What is the difference between find() and index() function?

Ans:

To get the location of a particular substring if we use find() function then find() function will check whether the provided sub string is existed or not in the original string, if the provided sub string is existed then find() function will return the index of the provided substring, if the provided substring is not existed in the original

string then find() function will return -1 value.

To get the location of a particular substring if we use index() function then index() function will check whether the provided substring is existed or not in the original string, if it is existed then index() function will return the index of the provided substring, if the provided substring is not existed then index() function will return an error like "ValueError: substring not found".

EX:

```
str = "Durga Software Solutions"
print(str.find("Software"))
print(str.index("Software"))
print(str.find("Hyderabad"))
print(str.index("Hyderabad")) --> ValueError: substring
not found
```
OP:

```
6
6
-1
ValueError: substring not found
```

In the case of find() and index() functions, if we want to get the location of a particular substring in the specified range then we have to provide start index value and end index value as parameters to find() and index() functions.

EX:

```
str = "Durga Software Solutions"
print(str.find("So",5, 14))
print(str.find("So", 0, 5))
print(str.index("So", 5, 14))
print(str.index("So", 0, 5))
```

OP:

```
6
-1
6
ValueError: substring not found
```

11/09/19

If we want ti find the location of a particular substring in backward direction in the goiven string then we have to use the following two predefined functions.

rfind()

rindex()

Where rfind(--) function will check whether the provided sub string is existed or not, if it is existed then rfind() function will get index value in backward direction and rfind() function will return that index value. If the specified substring is not existed then rfind() function will return -1 value.

Where rindex(--) function will check whether the provided sub string is existed or not, if it is existed then rindex() function will get index value in backward direction and rindex() function will return that index value. If the specified substring is not existed then rindex() function will raise an error like "ValueError: substring not found".
EX:

```
str = "Durga Software Solutions"
print(str)
print(str.rfind("So"))
print(str.rfind("Hyderabad"))
print(str.rindex("So"))
print(str.rindex("Hyderabad"))--> "ValueError: substring not found".
```
OP:

```
15
-1
15
"ValueError: substring not found".
```

In the case of rfind() and rindex() functions, if we want to get the location of a particular substring in the specified range in Backward direction then we have to provide start index value and end index value as parameters to rfind() and rindex() functions.
EX:

```
str = "Durga Software Solutions"
print(str.rfind("So",14,24))
print(str.rfind("So",5, 14))
print(str.rfind("So", 0, 5))

print(str.rindex("So",14,24))
print(str.rindex("So",5, 14))
print(str.rindex("So", 0, 5))
```

OP:
```
15
6
-1
15
6
```

ValueError: substring not found

In Python applications, if we want to get length of the string we have to use predefined function like len().
EX:

```
str = "Durga Software Solutions"
print(len(str))
```
OP:

```
24
```

In string type values, if we want to remove before and afetr spaces for a string then we have to use the following predefined functions.
1. strip()
2. lstrip()
3. rstrip()

Where strip() function will remove before spaces and after spaces fvor a particular string.
Where lstrip() will remove left spaces for a string.
Where rstrip() will remove right spaces for a particular string.
EX:

```
str = "      Durga Software Solutions       "
```

```
print(str,"Hyderabad")
print(str.strip(),"Hyderabad")
print(str.lstrip(), "Hyderabad")
print(str.rstrip(), "Hyderabad")
OP:
---
     Durga Software Solutions      Hyderabad
Durga Software Solutions Hyderabad
Durga Software Solutions      Hyderabad
     Durga Software Solutions Hyderabad
```

In String type values, to manipulate lower case letters and Upper case letters, String type has provided the following set of predefined functions.

1. lower()
2. islower()
3. upper()
4. isupper()
5. swapcase()
6. title()
7. isTitle()
8. capitalize()

Where lower() function will convert all the letters in string into lower case letters.

Where islower() function will check whether the data in a string is lower case or not.

Where upper() function will convert all the letters into upper case.

Where isupper() function will check whether the data in string is in upper case letters or not.

Where swapcase() function will convert all lower case letters into upper case letters and all upper case letters into lower case letters.

Where title() function will make all the first letter of each and every word as upper case letter.

Where istitle() function will check whether the data is title() or not.

Where capitalize() function will make the first letter in the first word as upper case letter.

EX:

```
str = "Durga Software Solutions"
print(str)
print()
str1 = str.lower()
```

```
print(str1)
print(str1.islower())
print()
str2 = str.upper()
print(str2)
print(str2.isupper())
print()
str3 = str.swapcase()
print(str3)
print()
str4 = str.title()
print(str4)
print(str4.istitle())
print()
str5 = str.capitalize()
print(str5)
```

OP:

```
Durga Software Solutions

durga software solutions
True

DURGA SOFTWARE SOLUTIONS
True
```

```
dURGA sOFTWARE sOLUTIONS

Durga Software Solutions
True

Durga software solutions
```

In Python applications, if we want join all the elements of a list or set or tuple with a particular delimiter or seperator then we have to use join() function.
Syntax: "Delimiter".join(list/set/tuple)
EX:

```
list = ["11", "09", "2019"]
str1 = "-".join(list)
print(str1)
str2 = "/".join(list)
print(str2)
```

OP:

```
11-09-2019
11/09/2019
```

In String type values, to check whether the data containes alphanumiric or not , numeric values or not, alphabets or not ,.... Python has provided the following predefined functions.

isalpha()
isalnum()
isdigit()

EX:

```
str1 = "Durga"
print(str1)
print(str1.isalpha())
print()
str2 = "Durga123"
print(str2)
print(str2.isalnum())
print()
str3 = "1234";
print(str3)
print(str3.isdigit())
```

OP:

```
Durga
True
```

```
Durga123
True

1234
True
```

To get how many times a particular sub string is repeated in a given string we have to use a predefinede function like count().

EX:

```
str = "Durga Software Solutions"
print(str.count("So"))
```

OP:

```
2
```

List Type:

--> If we want to represent a group of elements as single entity there we will use List.
--> To represent elements in list we have to use [].
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
```

```
print(list)
```

EX

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
```

--> List is index based, we are able to read elements from List on the basis of index values, We are able to read elements in bot forward direction and backward direction, to provide elements in forward direction we have to provide +ve index values, to provide elements in backward direction we will use -ve index values.

EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(list[0])
print(list[1])
print(list[2])
print(list[3])
print(list[4])
print(list[-1])
print(list[-2])
print(list[-3])
```

OP

```
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
AAA
BBB
CCC
DDD
EEE
FFF
EEE
DDD
```

--> List is following insertion order, it is not following sorting order.
EX:

```
---
list = ["AAA", "FFF", "BBB", "EEE", "CCC", "DDD"]
print(list)
```

OP:

```
---
['AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD']
```

--> It able to allow duplicate elements, we can recognize the duplicate elements on the basis of index values.
EX:

```
----
list = ["AAA", "BBB", "CCC", "BBB", "AAA"]
```

print(list)
OP:

['AAA', 'BBB', 'CCC', 'BBB', 'AAA']

--> List is able to allow heterogeneos elements.
EX:
---
list = ["AAA", True, 22.22, 10]
print(list)
OP:

['AAA', True, 22.22, 10]

--> List is able to allow more than one None element.
EX:
---
list = ["AAA", "BBB", "CCC", "DDD", None, None, None]
print(list)
OP:

['AAA', 'BBB', 'CCC', 'DDD', None, None, None]

--> To retrive elements from List we are able to use both for loop and while loop.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print()
for x in range(0,len(list)):
    print(list[x])
print()
for x in list:
    print(x)
print()
x = 0
while x < len(list):
    print(list[x])
    x = x + 1
```

OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']

AAA
BBB
CCC
DDD
EEE
FFF

AAA
BBB
```

```
CCC
DDD
EEE
FFF

AAA
BBB
CCC
DDD
EEE
FFF
```

--> We are able to read elememts from List type by using slice operator.
Syntax:
list[start_Index: end_Index: Step_Length]
start_Index: it will take start index value in both +ve and -ve values.
end_Index : It will take end index value in both +ve and -ve values.
step_Length: It will take step length to move pointer to the elements in order to retrieve.
If step_Length value is +ve then it will retrieve elements in forward direction, if step_Length value is -ve value then it will retrive elements in backward direction.

Default values in Forward direction:
start_Index : 0
end_Index : len(list)-1

Default values in Backward direction:
start_Index : -1
end_Index : -len(list)

Default value for step_Length is 1.

EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(list[0:5:1])
print(list[:5:1])
print(list[::1])
print(list[::])
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
```

EX:

```
---
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(list[::-1])
print(list[:-5:-1])
print(list[-2:-5:-1])
```

OP:

```
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['FFF', 'EEE', 'DDD', 'CCC', 'BBB', 'AAA']
['FFF', 'EEE', 'DDD', 'CCC']
['EEE', 'DDD', 'CCC']
```

EX:

```
----
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(list[2:-2:1])
print(list[-2:1:-1])
print(list[0:-3:-1])
print(list[len(list)-1:-1:1])
```

OP:

```
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['CCC', 'DDD']
['EEE', 'DDD', 'CCC']
```

```
[]
[]
```

len()
--> To get no of elements which are existed in list.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(len(list))
```

OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
6
```

count()
--> It able to return an integer value, it will represent the no of times a particular element is repeated.

append()
--> It can be used to add the specified element to the list.
EX:

```
list = []
print(list)
```

```
list.append("AAA")
list.append("BBB")
list.append("CCC")
list.append("DDD")
print(list)
```

OP:

```
[]
['AAA', 'BBB', 'CCC', 'DDD']
```

EX:

```
list = eval(input("Enter List Elements : "))
print(list)
print(type(list))
```

OP:

```
Enter List Elements :
["AAA","BBB","CCC","DDD","EEE","FFF"]
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
<class 'list'>
```

EX:

```
list = []
```

```
for x in range(0,6):
    list.append(input("Enter %d element : "%x))
print(list)
```

Note: List is Mutable , it able to allow changes on its elements.

EX:

```
list = ["AAA", "BBB", "CCC"]
print(list)
list.append("DDD")
print(list)
list.append("EEE")
print(list)
list.append("FFF")
print(list)
```

OP:

```
['AAA', 'BBB', 'CCC']
['AAA', 'BBB', 'CCC', 'DDD']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
```

18/09/19

----

insert()

--> It can be used to insert the specified element at the specified index, at the specified index, if element is existed then PVM will keep the new element at the specified index and PVM will adjust the existed element to the next index, if no element is existed at the specified index then the specified element is added to the List as last element.

EX:

```
list = ["AAA", "BBB", "CCC", "DDD"]
print(list)
list.insert(2, "XXX")
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD']
['AAA', 'BBB', 'XXX', 'CCC', 'DDD']
```

EX:

```
list = ["AAA", "BBB", "CCC", "DDD"]
print(list)
list.insert(10, "XXX")
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD']
['AAA', 'BBB', 'CCC', 'DDD', 'XXX']
```

Q)What are the differences between append() function and insert() function in List?

Ans:
-----
1. append() function is able to add the specified element as lat element.

   insert() function is able to add the specified element at the specified index, if no element is existed then the specified element will be added to the list as last element.

2. append() function will perform addition operation with out index values.
   insert() function will perform addition operation on the basis of index values.

index()
--> It able to return an index value where the first occurence of the specified element.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "BBB", "EEE",
"BBB"]
print(list)
print(list.index("BBB"))
OP:
---
['AAA', 'BBB', 'CCC', 'DDD', 'BBB', 'EEE', 'BBB']
1
```

remove()
--> It can be used to remove the specified element from the List, if the specified element is not existed then remove() function will raise an error like "ValueError: list.remove(x): x not in list".

EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
list.remove("BBB")
print(list)
//list.remove("ZZZ") --> ValueError: list.remove(x): x not
in list
OP:
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['AAA', 'CCC', 'DDD', 'EEE', 'FFF']
```

```
ValueError: list.remove(x): x not in list
```

Note: In List, remove() function is able to remove only one element, if we want to remove all duplicate elements of the specified element from List then we have to provide own logic explicitly.

EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF", "BBB",
"BBB"]
print(list)
val = list.count("BBB")
while val != 0:
    list.remove("BBB")
    val = list.count("BBB")
print(list)
```

OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF', 'BBB', 'BBB']
['AAA', 'CCC', 'DDD', 'EEE', 'FFF']
```

clear()

-->It can be used to remove all the elements from the list.

EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
```

```
print(list)
list.clear()
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
[]
```

pop()
--> It able to remove and return lat element of the list.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
print(list.pop())
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
FFF
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
```

Note: If we access pop() function on a empty List then PVM will raise an error like "IndexError: pop from empty list".
EX:

```
list = []
print(list)
print(list.pop())
print(list)
OP:
---
IndexError: pop from empty list
```

Q)What are the differences between remove() and pop() function?

Ans:
----
1. remove() function is able to remove the specified element from List.
   pop() function will remove the last element from List.

2. remove() function will return None type element.
   pop() function is able to return the removed element.

3. If we access remove() function on an empty list then PVM will provide an error   like "ValueError: list.remove(x): x not in list".

   If we access pop() function on an empty list then PVM will provide an error like
   "IndexError: pop from empty list".

reverse()
--> It will arrange all the elemenmts of List inreverse order.
EX:

```
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
print(list)
list.reverse()
print(list)
```
OP:

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['FFF', 'EEE', 'DDD', 'CCC', 'BBB', 'AAA']
```

sort()
--> It able to arrange all the elements of a list in sorting order, bydefault, in scending order. If we want to arrange all the elements in reverse of sorting order that is in descending order then we have to use "reverse" attribute in sort() function with "True" value.
Note: sort() function and sort(reverse=False) function are having same functionality.
EX:

```
list1 = ["AAA", "FFF", "BBB", "EEE", "CCC", "DDD"]
print(list1)
```

```
list1.sort()
print(list1)
list2 = ["AAA", "FFF", "BBB", "EEE", "CCC", "DDD"]
print(list2)
list2.sort(reverse=True)
print(list2)
list3 = ["AAA", "FFF", "BBB", "EEE", "CCC", "DDD"]
print(list3)
list3.sort(reverse=False)
print(list3)
```

OP:

```
['AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD']
['FFF', 'EEE', 'DDD', 'CCC', 'BBB', 'AAA']
['AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
```

copy()
--> It can be used to copy the elements from one List to another List.
EX:

```
list1 = ["AAA", "BBB", "CCC", "DDD", "EEE"]
print(list1)
print(id(list1))
list2 = list1.copy()
print(list2)
print(id(list2))
OP:
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
2363647611464
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
2363649341320
```

Note1:

```
list1 = ["AAA", "BBB", "CCC", "DDD", "EEE"]
print(list1)
print(id(list1))
list2 = list1
print(list2)
print(id(list2))
OP:
---
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
2624374854216
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
2624374854216
```

In the above code, Single List object is refered by list1 and list2 reference variables.

Note2:

```
list1 = ["AAA", "BBB", "CCC", "DDD", "EEE"]
print(list1)
print(id(list1))
list2 = list1[::]
print(list2)
print(id(list2))
```
OP

```
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
1689520198216
['AAA', 'BBB', 'CCC', 'DDD', 'EEE']
1689551353736
```

In the above example Duplicate List is cfreated by slice operator.

--> In List Type elements, we are able to compare the contents of two list by using ==, !=, <, <=, >, >= .

EX:

```
list1 = ["AAA", "BBB", "CCC"]
list2 = ["DDD", "EEE", "FFF"]
list3 = ["AAA", "BBB", "CCC", "DDD"]
```

```
print(list1 == list2)
print(list1 != list2)
print(list1 < list2)
print(list1 > list2)
print(list1 <= list2)
print(list1 >= list2)
print(list1 == list3)
print(list1 < list3)
print(list1 <= list3)
print(list1 > list3)
print(list1 >= list3)
```

OP:

```
---
False
True
True
False
True
False
False
True
True
False
False
```

20/09/19

--> To check whether an element is member of a particular list or not we have to use Membership Operators.
EX: in, not in

EX:

```
list = [10,20,30,40,50]
print(10 in list)
print(100 in list)
print(10 not in list)
print(100 not in list)
```
OP:

```
True
False
False
True
```

--> IN Python applications, if we want to concatinate two lists then we have to use '+' operator.

EX:

```
list1 = [10,20,30,40,50]
list2 = [60,70,80,90,100]
```

```
list3 = list1 + list2
print(list3)
print("list1 Ref : ", id(list1))
print("list2 Ref : ", id(list2))
print("list3 Ref : ", id(list3))
OP:
---
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
list1 Ref :  2670100107848
list2 Ref :  2670100108360
list3 Ref :  2670130411336
```

--> In Python applications, if we want to concatinate a particular list elements in repeatedly then we have to use '*'[repeate] operator.

EX:

```
---
list1 = [10,20,30,40,50]
print("List1 :",list1)
list2 = list1*3
print("List2 :",list2)
list3 = 3*list1
print("List3 :",list3)
list4 = 2*list1*2
print(list4)
print("List1 Ref :",id(list1))
print("List2 Ref :",id(list2))
print("List3 Ref :",id(list3))
print("List4 Ref :",id(list4))
```
OP:

```
List1 : [10, 20, 30, 40, 50]
List2 : [10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20,
30, 40, 50]
List3 : [10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20,
30, 40, 50]
[10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40,
50, 10, 20, 30, 40, 50]
List1 Ref : 1747908186696
List2 Ref : 1747910047496
List3 Ref : 1747910046088
List4 Ref : 1747911363848
```

---> In Python applications, we are able to declare one List in another List, where the internal List called as "Nested List".
EX:

```
list1 = [1,2,3,4,5]
list2 = [6,7,8,9,10]
list3 = [11,12,13,14,15]
list4 = [list1, list2, list3]
list5 = [[1,2,3,4,5], [6,7,8,9,10], [11,12,13,14,15]]
print(list1)
print(list2)
print(list3)
print(list4)
print(list5)
```

OP:

```
[1, 2, 3, 4, 5]
[6, 7, 8, 9, 10]
[11, 12, 13, 14, 15]
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

Tuple:

--> In Python applications, if we want to provide a group of elements as single   with out allowing modifications then we ahve to use Tuple.
--> It is same as List, but, it is not mutable, it is immutable.
--> It able to follow insertion order.
--> It is not following Sorting order.
--> It allows duplicate elements.
--> It allows heterogenous elements.
--> It allows None elements in more than one.

EX:

```
tuple = ("AAA", "FFF", "BBB", "EEE", "CCC", "DDD",
"BBB", 10, 20, 30, None, None )
print(tuple)
```

OP:

```
('AAA', 'FFF', 'BBB', 'EEE', 'CCC', 'DDD', 'BBB', 10, 20, 30,
None, None)
```

--> To represent Tuple elements either we have to use () or not to use any symbols
Note1: In Python applications, we are able to represent tuple elements either by using () or with out using () .
Note2: In Python applications, if we want to provide only one element in Tuple with out using () then we have to use , after element.
EX:

```
tuple1 = (10, 20, 30,40, 50)
tuple2 = 10,20,30,40,50
tuple3 = ()
tuple4 = (10)
tuple5 = 10,
print(type(tuple5))
print(tuple1)
print(tuple2)
print(tuple3)
print(tuple4)
print(tuple5)
```
OP:

```
<class 'tuple'>
(10, 20, 30, 40, 50)
(10, 20, 30, 40, 50)
()
10
(10,)
```

In Python applications, we are able to read elements from Tuple in the following two ways.
1. By Using index values.
2. By Using Slice operator.

1. By Using index values.
EX:

```
tuple = ["AAA", "BBB", "CCC", "DDD"]
print(tuple[0])
print(tuple[1])
print(tuple[2])
print(tuple[3])
print(tuple[-2])
```
OP:

```
AAA
BBB
CCC
DDD
CCC
```

EX:

```
tuple = ["AAA", "BBB", "CCC", "DDD"]
for x in range(0, len(tuple)):
    print(tuple[x])
print()
for x in tuple:
    print(x)
print()

val = 0
while(val < len(tuple)):
    print(tuple[val])
```

OP:

```
AAA
BBB
CCC
DDD
```

2. By Using Slice operator.

Syntax: RefVar[Start_Index: End_Index: Step_Length]

If Step_Length value is + ve value then PVM will read elements in Forward direction.
If Step_Length value is -ve value then PVM will read elements in backward direction.

In Forward Direction,
Default value for Start_index is 0
Default Value for End_Index is len(Ref_Var)-1

In Backward Direction,
Default value for Start_index is -1
Default Value for End_Index is -len(Ref_Var)

EX:

```
tuple = (10,20,30,40,50,60,70,80,90)
print(tuple)
print(tuple[2:6:1])
print(tuple[4:8:1])
print(tuple[6:2:1])
print(tuple[:8:1])
print(tuple[2::1])
print(tuple[::1])
print(tuple[::])
OP:
---
(10, 20, 30, 40, 50, 60, 70, 80, 90)
(30, 40, 50, 60)
(50, 60, 70, 80)
()
(10, 20, 30, 40, 50, 60, 70, 80)
(30, 40, 50, 60, 70, 80, 90)
(10, 20, 30, 40, 50, 60, 70, 80, 90)
(10, 20, 30, 40, 50, 60, 70, 80, 90)
```

EX:

```
tuple = (10,20,30,40,50,60,70,80,90)
print(tuple)
print(tuple[-2:-6:-1])
print(tuple[-4:-8:-1])
print(tuple[-8:-2:-1])
print(tuple[:-9:-1])
print(tuple[-3::-1])
print(tuple[::-1])
print(tuple[-1:-1000:-1])
```

OP:

```
(10, 20, 30, 40, 50, 60, 70, 80, 90)
```

```
(80, 70, 60, 50)
(60, 50, 40, 30)
()
(90, 80, 70, 60, 50, 40, 30, 20)
(70, 60, 50, 40, 30, 20, 10)
(90, 80, 70, 60, 50, 40, 30, 20, 10)
(90, 80, 70, 60, 50, 40, 30, 20, 10)
```

EX:

```
tuple = (10,20,30,40,50,60,70,80,90)
print(tuple)
print(tuple[0:len(tuple):2])
print(tuple[-1:1:-1])
print(tuple[-9:9:1])
print(tuple[2:-1:])
print(tuple[-1000:2000:])
print(tuple[-5:8:-1])
```

OP:

```
(10, 20, 30, 40, 50, 60, 70, 80, 90)
(10, 30, 50, 70, 90)
(90, 80, 70, 60, 50, 40, 30)
(10, 20, 30, 40, 50, 60, 70, 80, 90)
(30, 40, 50, 60, 70, 80)
(10, 20, 30, 40, 50, 60, 70, 80, 90)
()
```

--> In Python applications, it is possible to convert elements from List , Set,dict ... to Tuple type by using tuple() function.
Note: While converting elements from dict type to tuple type, tuple is able to get only keys from dict data type.
EX:

```
list = [10,20,30,40,50,60,70,80,90]
print(list)
print(type(list))
tuple = tuple(list)
print(tuple)
print(type(tuple))
```
OP:

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
<class 'list'>
(10, 20, 30, 40, 50, 60, 70, 80, 90)
<class 'tuple'>
```

EX:

```
set = {10,20,30,40,50,60,70,80,90}
print(set)
print(type(set))
tuple = tuple(set)
```

```
print(tuple)
print(type(tuple))
OP:
---
{70, 40, 10, 80, 50, 20, 90, 60, 30}
<class 'set'>
(70, 40, 10, 80, 50, 20, 90, 60, 30)
<class 'tuple'>
EX:
---
dict = {10:100, 20:200, 30:300, 40:400, 50:500}
print(dict)
print(type(dict))
tuple = tuple(dict)
print(tuple)
print(type(tuple))
OP:
---
{10: 100, 20: 200, 30: 300, 40: 400, 50: 500}
<class 'dict'>
(10, 20, 30, 40, 50)
<class 'tuple'>
```

--> Tuple is an immutable object, it will not allow modifications on its content, if we are trying to perform modifications over its content tghen PVM will raise an error like "TypeError: 'tuple' object does not support

item assignment".
EX:

```
tuple = (10,20,30,40,50)
print(tuple)
tuple[2] = 100
print(tuple)
```
OP:

```
"TypeError: 'tuple' object does not support item
assignment"
```

--> IN Python applications, we are able to perform concatination operation over the tuple elements by using + operator and *[Repeate] operator.
EX:

```
tuple1 = (10,20,30,40,50)
tuple2 = (60,70,80,90,100)
tuple3 = tuple1 + tuple2
print(tuple3)
tuple4 = 2*tuple1
print(tuple4)
tuple5 = tuple1*3
print(tuple5)
tuple6 = 2*tuple1*2
print(tuple6)
```

OP:

```
(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
(10, 20, 30, 40, 50, 10, 20, 30, 40, 50)
(10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40,
50)
(10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40,
50, 10, 20, 30, 40, 50)
```

Tuple Functions:
----
1. len()
--> It can be used to get size of the Tuple.
EX:

```
tuple = (10,20,30,40,50)
print(len(tuple))
```
OP:

```
5
```

2. count()
--> It can be used to return an integer value representing how many times a particular element is repeated in tuple.
EX:

```
tuple = (10,20,30,40,50,20,40,60,70,20,80)
print(tuple.count(20))
OP:
---
3
```

3. index()
--> It can be used to return an index value of the specified element in tuple.
--> If the specified element is existed in more than one time then index() function   will return index value where the specified elements first occurence is existed.
--> If the specified element is not existed then PVM will raise an error like   "ValueError: tuple.index(x): x not in tuple".
EX:

```
tuple = (10,20,30,40,50,20,40,60,70,20,80)
print(tuple)
print(tuple.index(20))
OP:
---
(10, 20, 30, 40, 50, 20, 40, 60, 70, 20, 80)
1
```

EX:

```
tuple = (10,20,30,40,50,20,40,60,70,20,80)
print(tuple)
print(tuple.index(100))
```
OP:

```
ValueError: tuple.index(x): x not in tuple
```

Note: If we want to search for the specified element in a particular range then we have to pass start index and end index as parameters to index() function.

EX:

```
tuple = (10,20,30,40,50,20,40,60,70,20,80)
print(tuple)
print(tuple.index(20,0,3))
print(tuple.index(20,4,7))
print(tuple.index(20,8,10))
print(tuple.index(20,6,8)) --> ValueError: tuple.index(x):
x not in tuple
```
OP:

```
(10, 20, 30, 40, 50, 20, 40, 60, 70, 20, 80)
1
5
9
ValueError: tuple.index(x): x not in tuple
```

sorted()
--> This function will take tuple as parameter and it will return List of elements in a particular sorting order. If we want to get all the sorting elements in reverse order then we have to use "reverse" attribute with "True" value.
EX:

```
tuple = "AAA", "FFF", "BBB", "EEE", "DDD", "CCC"
print(tuple)
print(sorted(tuple))
print(sorted(tuple, reverse=True))
print(sorted(tuple, reverse=False))
```
OP:

```
('AAA', 'FFF', 'BBB', 'EEE', 'DDD', 'CCC')
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
['FFF', 'EEE', 'DDD', 'CCC', 'BBB', 'AAA']
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
```

min() and max()
--> min() function can be used to return minimum value from Tuple.
--> max() function can be used to return max value frim Tuple.
EX:

```
tuple = "AAA", "FFF", "BBB", "EEE", "DDD", "CCC"
print(tuple)
print(min(tuple))
print(max(tuple))
```

OP:

```
('AAA', 'FFF', 'BBB', 'EEE', 'DDD', 'CCC')
AAA
FFF
```

cmp()
--> It can be used to compare two tuple elements as per dictionary order.
Syntax: cmp(tuple1, tuple2)
If tuple1 < tuple2 then cmp() will retun -ve value.
If tuple1 > tuple2 then cmp() will return +ve value.
If tuple1 == tuple2 then cmp() will return 0 value.

Note: cmp() function is existed upto Python2.x version, it is not existed in Python3.x version.

--> In python applications, we are able to keep more than one value or variable values to the tuple type directly , it is called as Tuple Packing. We are able to assign values from Tupe type to individual variables, it is called as Tuple Unpacking.

EX:

```
a = 10
b = 20
c = 30
d = 40

tuple = a,b,c,d  ------> Tuple Packing
print(tuple)

i,j,k,l = tuple -------> Tuple Unpacking
print(i,j,k,l)
```

Q)What are the differences between List and Tuple?

Ans:
----
1. List elements are represented in [].
   Tuple elements are represented in ().

2. List is mutable.
   Tuple Immutable.

3. In the requirements where we want to change the elements continously there we   will use List.
   In the requirement where we want to provide fixed no of elements with out having   modifications through out

the application there we will use Tuple.

Set :

--> It is a sequence data type, it able to store more than one element.
--> It is not index based.
--> It does not allow index based operations like slice oprator,....
--> It does not follow insertion order.
--> It does not allow duplicate elements.
--> It allows heterogeneous elements.
--> It allows only one None element.
--> To represent elements in set we will use {} .
EX:

```
set = {10,20,30,40,50, "AAA", True, 22.22, None, None}
print(set)
print(type(set))
print(id(set))
```
OP:

```
{True, 40, 10, None, 50, 20, 22.22, 'AAA', 30}
<class 'set'>
1731558505160
```

--> In Python apoplications, we are able to read elements by using for-each loop only, it is not possible to use normal for loop and while loop.
EX:

```
set = {10,20,30,40,50}
for x in set:
    print(x)
```
OP:

```
40
10
50
20
30
```

--> IN Python applications, we are able to convert elements from the sequence types like List, Tuple, Range,... to Set by using a predefined function like set()
EX:

```
list = [10,20,30,40,50]
tuple = (10,20,30,40,50)
range = range(0,5)
dict = {10:100, 20:200, 30:300, 40:400, 50:500}

s1 = set(list)
```

```
s2 = set(tuple)
s3 = set(range)
s4 = set(dict)

print("List To Set   :",s1)
print("Tuple To Set  :",s2)
print("Range To Set  :",s3)
print("Dict To Set   :",s4)
```

OP:

```
List To Set   : {40, 10, 50, 20, 30}
Tuple To Set  : {40, 10, 50, 20, 30}
Range To Set  : {0, 1, 2, 3, 4}
Dict To Set   : {50, 20, 40, 10, 30}
```

--> IN Python applications, to represent an empty set if we use {} then PVM will treate the provided curly braces are dict type bydefault, where if we want to provide an empty set then we have to use set() function.

EX:

```
set1 = {}
print(type(set1))

set2 = set()
print(set2)
print(type(set2))
```

--> In Python applications, to add elemens to the Set we will use add() function.
EX:

```
set = {"AAA", "BBB"}
print(set)
set.add("CCC")
set.add("DDD")
set.add("EEE")
set.add("FFF")
print(set)
```
OP:

```
{'AAA', 'BBB'}
{'EEE', 'DDD', 'CCC', 'BBB', 'AAA', 'FFF'}
```

EX:

```
set = set()
count = int(input("No of Inputs  : "))
for x in range(0, count):
    set.add(input("Enter %d value  : "%(x+1)))
print(set)
```
OP:

```
No of Inputs  : 5
```

```
Enter 1 value  : AAA
Enter 2 value  : BBB
Enter 3 value  : CCC
Enter 4 value  : DDD
Enter 5 value  : EEE
{'CCC', 'AAA', 'DDD', 'EEE', 'BBB'}
```

--> If we want to add all the elements of List, Tuple, Range to set at a time then we have to use update() function.

EX:

```
set = set()
print(set)
list = [10,20,30,40,50]
tuple = ("AAA", "BBB", "CCC", "DDD", "EEE")
range = range(0,5)

set.update(list, tuple, range)
print(set)
```

OP:

```
set()
{0, 1, 2, 3, 4, 'DDD', 40, 'CCC', 10, 50, 20, 'BBB', 'EEE',
'AAA', 30}
```

Q)What are the differences betweeb add() function and

update() function?

Ans:

1.add() function can be used to add single element at a time.
update() function can be used to add elements of sequence data types at a time.

2. add() function will take single value as parameter.
   update() function will take more than one value as parameter.

--> In Python applications, we are able to copy all elements from one set to another set.
EX:

```
set1 = {10,20,30,40,50}
print(set1)
print(type(set1))
print(id(set1))

set2 = set1.copy()
print(set2)
print(type(set2))
print(id(set2))
```

OP:

```
{40, 10, 50, 20, 30}
<class 'set'>
1899202214600
{50, 20, 40, 10, 30}
<class 'set'>
1899203252744
```

--> In Python applications, to remove a particular element from Set then we are able to use the following three functions.
  a)remove()
  b)pop()
  c)discard()

remove()
--> It can be used to remove a particular element from Set and it will return None   value.
--> If the specified element is not existed then remove() function will raise an   error like "KeyError: 100".
EX:

```
set = {10, 20, 30, 40, 50}
print(set)
print(set.remove(20))
print(set)
```

OP:

```
{40, 10, 50, 20, 30}
None
{40, 10, 50, 30}
```
EX:

```
set = {10, 20, 30, 40, 50}
print(set)
print(set.remove(200))
print(set)
```
Status: KeyError: 200

pop()
--> It will remove and return first element in its order.
--> If we access pop() function on an empty set then PVM will raise an error like "KeyError: 'pop from an empty set'".
EX:

```
set = {10,20,30,40,50}
print(set)
print(set.pop())
print(set)
```
OP:

```
{40, 10, 50, 20, 30}
```

```
40
{10, 50, 20, 30}
```

EX:

```
set = set()
print(set)
print(set.pop())
print(set)
```
OP:

```
KeyError: 'pop from an empty set'.
```

discard()
--> It can be used to remove a particular element from set.
--> If the specified element is not existed in set then dissard() function will not   raise any error.
EX:

```
set = {10,20,30,40,50}
print(set)
print(set.discard(20))
print(set)
```
OP:

```
{40, 10, 50, 20, 30}
None
{40, 10, 50, 30}
```

Q)What are the differences between remove(), pop() and discard() functions?

Ans:
----
1. remove() and discard() functions are used to remove a particular element.
   pop() function is able to remove first element in its order.

2. After removing element, remove() and discard() functions are able to return   None value.
   After removing element, pop() function is able to return the removed element.

3. If the specified eleemnt is not existed then remove() function will reutrn   KeyError.
   If we access pop() function over an empty set then pop() function will return   KeyWrror.
   If the specified element is not existed then discard() function will not raise   any error and it will return None value.

--> To remove all elements from Set we will use a predefined function like clear().
EX:

```
set = {10,20,30,40,50}
print(set)
set.clear()
print(set)
```
OP:

```
{40, 10, 50, 20, 30}
set()
```

--> In Python appliopcations, we are able to perform the mathematical operations like Union [|], Intersection [&], difference[-], Symmetric_Defference[^],...

set1.union(set2) : All the elements from set1 and set2.
set1.intersection(set2) : Only Common elements from Set1 and set2
set1.difference(set2) : elements existed in set1 and not existed in set2
set1.symmetric_Difference(set2): All Elements from set1 and set2 except the elements which are commoin in set1 and set2.
EX:

```
---
set1 = {10, 20, 30, 40, 50, 60, 70}
set2 = {5,10,15,20,25,30,35,40,45,50}
set3 = {5, 15, 25, 35, 45}
print(set1.union(set3))
print(set1 | set3)
print()
print(set1.intersection(set2))
print(set1 & set2)
print()
print(set2.difference(set1))
print(set2 - set3)
print()
print(set1.symmetric_difference(set2))
print(set1 ^ set2)
OP:
---
{35, 5, 70, 40, 10, 45, 15, 50, 20, 25, 60, 30}
{35, 5, 70, 40, 10, 45, 15, 50, 20, 25, 60, 30}

{40, 10, 50, 20, 30}
{40, 10, 50, 20, 30}

{35, 5, 45, 15, 25}
{40, 10, 50, 20, 30}

{35, 5, 70, 45, 15, 25, 60}
```

```
{35, 5, 70, 45, 15, 25, 60}
```

--> In Python applications, we are able to apply membership operators over set elements.
in, not in

EX:

```
set = {10,20,30,40,50}
print(set)
print(10 in set)
print(10 not in set)
print(100 in set)
print(100 not in set)
```
OP:

```
{40, 10, 50, 20, 30}
True
False
False
True
```

Q)What are the differences between List and Set Data Types?

Ans:
----

1. List is index based.
   Set is not index based.

2. List is able to allow duplicate elements.
   Set is not allowing duplicate elements.

3. List is following insertion order.
   Set is not following insertion Order.

4. List is able to allow any no of None elements.
   Set is able to allow only one None element.

5. List is able to allow elements in the form of [].
   Set is able to allow elements in the form of {}.

6. List immutable form is Tuple.
   Set immutable form is Frozenset.

Dict Type:

--> In Python , the sequence data types like List, Tuple, Set,... are able to store data in the form of individual elements, but, dict sequence type is able to allow data in the form of key-Value pairs.

Note: Java has similar data type in the form of Map.

--> To represent elements in dict type we have to use the following syntax.

```
    {key1:val1, key2:val2,.....key-n:val-n}
```

EX:

```
dict = {"A":"AAA", "B":"BBB", "C":"CCC", "D":"DDD",
"E":"EEE"}
print(dict)
print(type(dict))
print(id(dict))
```

OP:

```
{'A': 'AAA', 'B': 'BBB', 'C': 'CCC', 'D': 'DDD', 'E': 'EEE'}
<class 'dict'>
1994696568168
```

--> In dict type, all keys must be unique, they must not allow duplicate elements,   but, Values may be duplicated.If we duplicate a particular key then old value is   replaced with the provided new value.

EX:

```
dict = {"A":"AAA", "B":"BBB", "C":"CCC",
"D":"DDD","B":"XXX"}
print(dict)
```

OP:

```
{'A': 'AAA', 'B': 'XXX', 'C': 'CCC', 'D': 'DDD'}
```
EX:

```
dict = {"A":"AAA", "B":"BBB", "C":"CCC", "D":"DDD",
"E":"BBB", "F":"CCC"}
print(dict)
```
OP:

```
{'A': 'AAA', 'B': 'BBB', 'C': 'CCC', 'D': 'DDD', 'E': 'BBB', 'F':
'CCC'}
```

--> dict is following insertion order w.r.t the keys, dict is not following sorting   order.
EX:

```
dict = {"A":"AAA",  "F":"FFF", "B":"BBB", "E":"EEE",
"C":"CCC", "D":"DDD" }
print(dict)
```
OP:

```
{'A': 'AAA', 'F': 'FFF', 'B': 'BBB', 'E': 'EEE', 'C': 'CCC', 'D':
'DDD'}
```

--> dict type is allowing heterogeneous elements at both keys side and values.
EX:

```
dict = {"A":"AAA",  True:False, 10:100, 22.22:33.33 }
print(dict)
OP:
---
{'A': 'AAA', True: False, 10: 100, 22.22: 33.33}
```

--> Dict type is allowing only one None elements is allowed at keys side but any no of None elements are allowed at values.

EX:

```
---
dict = {"A":"AAA",  "B":"BBB", None:100, None:200,
10:None, 20:None}
print(dict)
OP:
---
{'A': 'AAA', 'B': 'BBB', None: 200, 10: None, 20: None}
```

-->In Python, we are able to represent empty dict type in the following two types.

1. By using {}.
2. By using dict()

EX:

```
----
dict1 = {}
print(dict1)
print(type(dict1))
```

```
dict2 = dict()
print(dict2)
print(type(dict2))
```

OP:

```
{}
<class 'dict'>
{}
<class 'dict'>
```

--> In Python applications, it is possible to convert elements from List type, Set type, tuple type to dict type, but, in List type ,Set type and tuple type we have to provide Key-Values by using ().

EX:

```
list = [(10,100),(20,200),(30,300),(40,400)]
print(list)
print(type(list))
dict = dict(list)
print(dict)
print(type(dict))
```

OP:

```
[(10, 100), (20, 200), (30, 300), (40, 400)]
```

```
<class 'list'>
{10: 100, 20: 200, 30: 300, 40: 400}
<class 'dict'>
```

EX:

```
set = {(10,100), (20,200), (30,300), (40,400)}
print(set)
print(type(set))
dict = dict(set)
print(dict)
print(type(dict))
```
OP:

```
{(30, 300), (40, 400), (10, 100), (20, 200)}
<class 'set'>
{30: 300, 40: 400, 10: 100, 20: 200}
<class 'dict'>
```

EX:

```
tuple = ((10,100),(20,200),(30,300),(40,400))
print(tuple)
print(type(tuple))
dict = dict(tuple)
print(dict)
print(type(dict))
```

OP:

```
((10, 100), (20, 200), (30, 300), (40, 400))
<class 'tuple'>
{10: 100, 20: 200, 30: 300, 40: 400}
<class 'dict'>
```

Q)Find the valid dict type declarations from the following examples?

```
dict1 = {10:100, 20:200, 30:300, 40:400}
dict2 = dict()
dict3 = dict([(10,100), (20,200), (30,300), (40,400)])
dict4 = dict({(10,100), (20,200), (30,300), (40,400)})
dict5 = dict(((10,100), (20,200), (30,300), (40,400)))
```
Ans: All

--> In Python applications, we are able to add key-value pairs to the dict type by using the following syntax.

```
        refVar[key] = Value
```
EX:

```
student_Dict = {}
while True:
    roll_No = int(input("Student Role No  :"))
    name = input("Student Name    :")
    student_Dict[roll_No] = name
    option = input("Onemore Student[yes/no]?  :")
    if option == "yes":
        continue
    else:
        break
print(student_Dict)
```
OP:

```
Student Role No  :111
Student Name    :AAA
Onemore Student[yes/no]?  :yes
Student Role No  :222
Student Name    :BBB
Onemore Student[yes/no]?  :yes
Student Role No  :333
Student Name    :CCC
Onemore Student[yes/no]?  :yes
Student Role No  :444
Student Name    :DDD
Onemore Student[yes/no]?  :no
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD'}
```

--> To retrieve elements from dict type we will use the following syntax.

```
     refVar[key]
```
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD"}
print(dict)
print("111 --->",dict[111])
print("222 --->",dict[222])
print("333 --->",dict[333])
print("444 --->",dict[444])
```

OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD'}
111 ---> AAA
222 ---> BBB
333 ---> CCC
444 ---> DDD
```

--> In dict type, we are able to read elements by using for-each loop.

EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
for x in dict:
    print(x,"---->",dict[x])
```

OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
```

```
'EEE', 666: 'FFF'}
111 ----> AAA
222 ----> BBB
333 ----> CCC
444 ----> DDD
555 ----> EEE
666 ----> FFF
```

--> In dict type, we are able to get value of a particular key by using get() function.

EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
for x in dict:
    print(x,"---->",dict.get(x))
```

OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
111 ----> AAA
222 ----> BBB
333 ----> CCC
444 ----> DDD
555 ----> EEE
666 ----> FFF
```

In case of get() function, if the specified key is not existed then get() function will return None bydefault, in this context, we are able to display an alternative message inplace of None, for this, we have to provide message as second parameter to get() function.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
print(111,"--->",dict.get(111))
print(222,"--->",dict.get(222))
print(333,"--->",dict.get(333))
print(444,"--->",dict.get(444))
print(555,"--->",dict.get(555))
print(666,"--->",dict.get(666))
print(777,"--->",dict.get(777))
print(888,"--->",dict.get(888,"Index 888 is not existed in
dict"))
```

--> IN Python applications, it is possible to create duplicate dict object by using copy() predefined function.
EX:

```
dict1 = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
```

```
print("Original Dict :",dict1)
print("Original Dict Ref :",id(dict1))
dict2 = dict1
print("Duplicate Dict by = :",dict2)
print("Duplicate Dict Ref :",id(dict2))
dict3 = dict1.copy()
print("Duplicate Dict by copy() :",dict3)
print("Duplicate Dict Ref :",id(dict3))
```
OP:

```
Original Dict : {111: 'AAA', 222: 'BBB', 333: 'CCC', 444:
'DDD', 555: 'EEE', 666: 'FFF'}
Original Dict Ref : 2921059462504
Duplicate Dict by = : {111: 'AAA', 222: 'BBB', 333: 'CCC',
444: 'DDD', 555: 'EEE', 666: 'FFF'}
Duplicate Dict Ref : 2921059462504
Duplicate Dict by copy() : {111: 'AAA', 222: 'BBB', 333:
'CCC', 444: 'DDD', 555: 'EEE', 666: 'FFF'}
Duplicate Dict Ref : 2921059467576
```

Note: Assigning one dict reference variable to another reference variable will not create duplicate object, it will copy single Dict object referrence value to another variable.

--> dict type is mutable.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD", 
555:"EEE", 666:"FFF"}
print(dict)
dict[333] = "XXX"
print(dict)
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555: 
'EEE', 666: 'FFF'}
{111: 'AAA', 222: 'BBB', 333: 'XXX', 444: 'DDD', 555: 
'EEE', 666: 'FFF'}
```

---> In dict type, if we want to remove key-value pair from dict type then we have to use the following three ways.
1. By using 'del' keyword.
2. By using pop().
3. By using popitem().

1. By using 'del' keyword.
---> It will return single key-valir from dict type.
Syntax: del dict_Ref_Var[key]
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD", 
555:"EEE", 666:"FFF"}
```

```
print(dict)
del dict[333]
print(dict)
print(dict[333])
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
{111: 'AAA', 222: 'BBB', 444: 'DDD', 555: 'EEE', 666:
'FFF'}
KeyError: 333
```

2. By using pop() function:
--> It can be used to remove a particular key-value pair on the basis of the provided key in pop() function and it will return the value of the removed
key-value pair.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
print(dict.pop(333))
print(dict)
print(dict[333])
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
CCC
{111: 'AAA', 222: 'BBB', 444: 'DDD', 555: 'EEE', 666:
'FFF'}
KeyError: 333
```

3.By using popitem():
--> It can be used to remove last key-value pair from dict, it will not take any parameter and it will return the removed key-value pair in the form of tuple.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
print(dict.popitem())
print(dict)
print(dict.popitem())
print(dict)
```

OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
(666, 'FFF')
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
```

```
'EEE'}
(555, 'EEE')
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD'}
```

Note: After deleting key-value pair from dict type then we are unable to access that key-value.

Q)What are the differences between pop() and popitem()?

Ans:
----
1. pop() function is able to remove a particular key-value pair on the basis of   the provided key.
   popitem() is able to remove last key-value pair from dict.

2. pop() is able to return value of the removed key-value pair.
   popitem() is able to return removed key-value pair in the form of tuple.

3. pop() required a particular key as parameter which we want to remove.
   popitem() will not take any parameter.

--> To remove all key-value pairs from dict we will use

clear() function.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
dict.clear()
print(dict)
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
{}
```

--> To get no of key-value pairs from dict we will use len() function.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(dict)
print(len(dict))
```
OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
6
```

--> To copy all key-value pairs from one dict type to another dict type we will use update() function.
EX:

```
dict2 = {444:"DDD", 555:"EEE", 666:"FFF"}
print(dict1)
print(dict2)
dict1.update(dict2)
print(dict1)
```

OP:

```
{111: 'AAA', 222: 'BBB', 333: 'CCC'}
{444: 'DDD', 555: 'EEE', 666: 'FFF'}
{111: 'AAA', 222: 'BBB', 333: 'CCC', 444: 'DDD', 555:
'EEE', 666: 'FFF'}
```

--> From dict type,
1. To get all keys as a list then we have to use keys() function.
2. To get all values as list then we have to use values() function.
3. To get all key-value pairs in the form of list we have to use items() function.
EX:

```
dict = {111:"AAA", 222:"BBB", 333:"CCC", 444:"DDD",
555:"EEE", 666:"FFF"}
print(list(dict.keys()))
print(list(dict.values()))
print(list(dict.items()))
OP:
---
[111, 222, 333, 444, 555, 666]
['AAA', 'BBB', 'CCC', 'DDD', 'EEE', 'FFF']
[(111, 'AAA'), (222, 'BBB'), (333, 'CCC'), (444, 'DDD'),
(555, 'EEE'), (666, 'FFF')]
```

Functions:

In general, in python applications, we need to execute a set of instructions repeatedly, if we provide the same set of instructions again and again as per the requirement then Code redundency will be increased, it is not suggestible in Python applications.

In the above context, to reduce code redudnency we have to improve Code Reusability, to provide code reusabilitu we have to use Functions.

Function is a set of instructions to repersent a particular action.

The main adv of Functions is Code Reusability, Code Maintanence.

There are two types of Functions in Python.
1. Predefined Functions
2. User Defined Functions

Predefined Functions:

These functions are defined by Python Programming Language and they are coming along with Python Software.
EX:
---
len()
sorted()
id()
type()
eval()

User Defined Functions

These functions are defined by the developers as per their applications requirement.
Syntax:
----
def FunctionName([Param_List]):

```
   ---instructions----
   [return Values]
```

Where "def" keyword can be used to define function.

Where "FunctionName" is an unique identity for the function inorder to access.

Where ParameterList is optional, it can be used to pass some input data to the function order to perform an action.

Where return statement can be used to return one or more values from the function.

EX:

```
def sayHello():
    print("Hello User!")

sayHello()
sayHello()
sayHello()
```

OP:

```
Hello User!
Hello User!
Hello User!
```

EX:

```
def sayHello(name):
    print("Hello",name,"!")

sayHello("Durga")
sayHello("Anil")
sayHello("Rama")
```

OP:

```
Hello Durga !
Hello Anil !
Hello Rama !
```

EX:

```
def wish(name, message):
    print("Hello",name,",",message)

wish("Durga", "Good Morning!")
wish("Anil", "Good Afternoon!")
wish("Rama", "Good Evening!")
```

OP:

```
Hello Durga , Good Morning!
Hello Anil , Good Afternoon!
Hello Rama , Good Evening!
```

EX:

```
def add(a,b):
    result = a + b
    return result

result1 = add(10,20)
print("ADD :",result1)
result2 = add(20,30)
print("ADD :",result2)
result3 = add(30,40)
print("ADD :",result3)
```

OP:

```
ADD : 30
ADD : 50
ADD : 70
```

EX:

```
def calculate(a,b):
    print("ADD :",(a+b))
    print("SUB :",(a-b))
    print("MUL :",(a*b))
    print("DIV :",(a/b))
    print("MOD :",(a%b))
calculate(10,5)
```

OP:

```
ADD : 15
SUB : 5
MUL : 50
DIV : 2.0
MOD : 0
```

In Python functions, we are able to return more than one value as per the requirement.

EX:

```
def calculate(a,b):
    add = a + b
    sub = a - b
    mul = a * b
    div = a / b
    mod = a % b
    return add, sub, mul, div, mod

add_Result, sub_Result, mul_Result, div_Result,
mod_Result = calculate(20,3)
print("ADD :", add_Result)
print("SUB :",sub_Result)
print("MUL :",mul_Result)
print("DIV :",div_Result)
print("MOD :", mod_Result)
```

OP:

```
ADD : 23
SUB : 17
MUL : 60
DIV : 6.666666666666667
MOD : 2
```

EX:

```
def getEmpDetails():
    eno = int(input("Employee Number : "))
    ename = input("Employee Name  : ")
    esal = float(input("Employee Salary : "))
    eaddr = input("Employee Address : ")
    return eno, ename, esal, eaddr
```

```
def displayEmpDetails(eno, ename, esal, eaddr):
    print("Employee Details")
    print("-------------------------")
    print("Employee Number   : ",eno)
    print("Employee Name     : ",ename)
    print("Employee Salary   : ",esal)
    print("Employee Address  : ",eaddr)

eno , ename, esal, eaddr = getEmpDetails()
print()
displayEmpDetails(eno, ename, esal, eaddr)
```

OP:

```
Employee Number : 111
Employee Name  : AAA
Employee Salary : 5000.0
Employee Address : Hyd

Employee Details
-------------------------
Employee Number   :  111
Employee Name     :  AAA
Employee Salary   :  5000.0
Employee Address  :  Hyd
```

In Python Functions, if we provide argument list at

Function definition then that argument is called as Formal Arguments.

In Python Function calls, if we provide argument values list then that argument values are called as Actual Arguments.
EX:

```
def add(a,b):
   print(a+b)

add(10,20)
```

Where a,b are Format Arguments and 10,20 are actual arguments.

In Python, thyere are four types of arguments are existed for the functions.
1. Positional Arguments.
2. Keyward Arguments
3. Default Arguments
4. Variable length Arguments.

1. Positional Arguments:

Positional arguments are normal arguments to the functions, whose values must be provided on the basis

of the argument positions in function calls.
EX:

```
def wish(name, message):
    print("Hello",name,message)

wish("Durga", "Good Morning!")
```
OP:

```
Hello Durga Good Morning!
```

In the above example, if we provide "Good Morning" as first parameter and "Durga" as second parameter then Function result will be changed.
EX:

```
def wish(name, message):
    print("Hello",name,message)

wish("Good Morning!", "Durga")
```
OP:

```
Hello Good Morning! Durga
```

2. Keyward Arguments:

In the case of Keyward arguments, we will provide

argument values directly by assigning values to the argument names in functions calls.
EX:

```
def wish(name, message):
    print("Hello",name,message)

wish(name="Durga", message="Good Morning!")
wish(message = "Good Morning!", name="Durga")
```
OP:

```
Hello Durga Good Morning!
Hello Durga Good Morning!
```
Note: In the case of Keyward arguments, we can provide arguments in any order.

Note: In Python Functions, we are able to provide both positional arguments and keyward arguments in single function,but, first we have to provide positional arguments next we have to provide keyward arguments.If we voilate this rule then PVM will raise an error like "SyntaxError: positional arguments follows keyward arguments.
EX:

```
def add(a,b,c):
    print(a+b+c)
```

```
add(10,b = 20, c = 30)
add(a = 10,20,30) --> SyntaxError: positional arguments follows keyward arguments
OP:
---
SyntaxError: positional argument follows keyword argument
```

3. Default Arguments:

In the case of default arguments, we will provide initial values to the arguments at Function definition, if user dont want to provide values to these arguments then the initial values will be assigned to the arguments, if user provides values to these arguments at function call then new values will override default values and new values will be existed to the arguments.

EX:

```
def wish(name="Durga", message="Good Morning!"):
    print("Hello",name,",",message)
wish()
wish("Anil")
wish("Nag", "Good Afternoon!")
OP:
---
Hello Durga , Good Morning!
Hello Anil , Good Morning!
Hello Nag , Good Afternoon!
```

EX:

```
def
createAccount(accNo,accHolderName,accType="Savings
", balance=10000):
    print("Account Details")
    print("----------------------")
    print("Account Number       :",accNo)
    print("Account Holder Name  :",accHolderName)
    print("Account Type         :",accType)
    print("Account Balance      :",balance)

createAccount("abc123", "Durga")
print()
createAccount("xyz123", "Nag", "Current",100000)
```

OP:

```
Account Details
----------------------
Account Number       : abc123
Account Holder Name  : Durga
Account Type         : Savings
Account Balance      : 10000
Account Details
----------------------
Account Number       : xyz123
Account Holder Name  : Nag
Account Type         : Current
Account Balance      : 100000
```

Note: In Python Function, it is possible to provide both Positional Arguments and default arguments in single function, but, first we have to provide positional argument then we have to provide default arguments. If we voilate this rule then PVM will raise an error like "SyntaxError: non-default argument follows default argument".

EX:

```
def add(a,b=20,c=30):
    print(a+b+c)
add(10)
```

OP:

```
60
```

EX:

```
def add(a=10,b,c):
    print(a+b+c)
add(20,30)
```
OP:

```
SyntaxError: non-default argument follows default argument
```

Note: In Python functions, it is possible to provide both Default Arguments and Keyward Arguments in single function, but, first we have to provide keyward argument then we have to provide default arguments.
EX:

```
def add(a,b=20,c=30):
    print("ADD :",(a+b+c))
add(a=10)
```

OP:

```
ADD : 60
```

EX:

```
def add(a=10,b,c):
    print("ADD :",(a+b+c))
add(20,30)
```
Status: SyntaxError: non-default argument follows

default argument

4. Variable length Arguments:

In Python applications, if we define any function with 'n' no of positional arguments then we must access that function by passing the same n no of argument values. It is not possibnle to access that function by passing n+1 no of arguments values and n-1 no of argument values to the positionsl arguments.

As per the requirement, if we want to access any function by passing variable no of argument values then we have to use Variable Length Argument.

Syntax:
```
def function(*n):
             ----
```

If we access the function which is having variable length arguments by passing n no of argument values then all the argument values will be stored in a tupe and it is refered by variable length argument name.
EX:

```
def add(*n):
    print("No Of Arguments    : ",len(n))
    print("Argument List      : ",end="")
    result = 0
    for x in n:
        print(x,end="  ")
        result = result + x;
    print()
    print("Addition            :",result)
    print("------------------------------")

add()
add(10)
add(10,20)
add(10,20,30)
OP:
----
No Of Arguments    :  0
Argument List      :
Addition           : 0
------------------------------
No Of Arguments    :  1
Argument List      : 10
Addition           : 10
------------------------------
No Of Arguments    :  2
Argument List      : 10  20
Addition           : 30
------------------------------
```

```
No Of Arguments    :  3
Argument List      : 10  20  30
Addition           : 60
-------------------------------
```

--> In the case of Variable length arguments, we are unable to provide positional arguments after the variable length argument, but, we are able to provide keyword arguments and Default arguments after variable length arguments in a function.

EX:

```
def add(*n,a=100,b=200):
    print(n,a,b)
add(10,20)
```
OP:

```
(10, 20) 100 200
```

EX:

```
def add(*n,a,b):
    print(n,a,b)
add(10,20,a=100,b=200)
```
OP:

```
(10, 20) 100 200
```

EX:

```
def add(*n,a,b):
    print(n,a,b)
add(10,20,100,200)
```
OP:

```
TypeError: add() missing 2 required keyword-only
arguments: 'a' and 'b'
```

-->In a function, positional arguments and Default Arguments are possible before variable length arguments, but, Keyword Arguments are not possible before variable length argument.

EX:

```
def add(a,b,*n):
    print(a,b,n)
add(10,20,100,200)
```
OP:

```
10 20 (100, 200)
```

EX:

```
def add(a,b,*n):
    print(a,b,n)
add(a=10,b=20,100,200)
```

OP:

```
SyntaxError: positional argument follows keyword argument
```

EX

```
def add(a=10,b=20,*n):
    print(a,b,n)
add(100,200,300,400)
```
OP:

```
100 200 (300, 400)
```

Q)Is it possible to provide more than one variable length arguments in single function?

Ans:

No, it is not possible to provide more than one Variable length arguments in single function.

EX:

```
def add(*n1,*n2):
    print(n1,n2)
```

OP:

```
SyntaxError: invalid syntax
```

In Python, there are two types of Variables.
1. Local Variables
2. Global variables

1. Local Variables:
--> If we declare any variable inside a function then that variable is called as Local variable, where Local variables are having scope upto the respective function.
EX:

```
def add():
    a = 10
    b = 20
    print("a   :",a)
    print("b   :",b)
    print("ADD :",(a+b))
def sub():
    a = 20
    b = 10
    print("a   :",a)
    print("b   :",b)
    print("SUB :",(a-b))
def mul():
    a = 5
    b = 10
    print("a   :",a)
    print("b   :",b)
    print("MUL :",(a*b))
add()
print()
sub()
print()
mul()
```

OP:

```
a   : 10
b   : 20
ADD : 30

a   : 20
b   : 10
SUB : 10

a   : 5
b   : 10
MUL : 50
```

2. Global variables
--> If we declare any variable in out side of the functions then that variable is called as Global variable, where Global variables are having scope through out the program , that is, in all the functions which are existed in the present file.
EX:

```
a = 10
b = 5
print("a   :",a)
print("b   :",b)
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
add()
sub()
mul()
```

OP:

```
a   : 10
```

```
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

--> In python program, if we have variables at local and at global with the same name and with different values then if we access that variable in current function then PVM will access local variable value, PVM will not access Global variable value, but, in this context, if we want to access Global variable value then we have to use a predefined function like globals()['VarName'] .
EX:

```
a = 10
def function():
    a = 20
    print("Local Variable Value  : ",a)
    print("Global Variable Value : ",globals()['a'])

function()
```
OP:

```
Local Variable Value  :  20
Global Variable Value :  10
```

Note: If we access any variable in a function then PVM will search for that variable at local first, if that variable is not existed at local then PVM will search for it at Global.

--> In Python applications, if we want to make available gloval variable inside a function for modifications then we have to use 'global' keyword.
EX:

```
a = 10
def function1():
    a = 20
    print(a)
def function2():
    print(a)

function1()
function2()
```

OP:

```
20
10
```

In the above program, a variable is teated as Global variable initially, in function1() a variable is trated as local variable, if we perform modification on a variable in function1 then modification will be performed on local variable only, not in Global variable.

In the above context, if we want to perform modifications over the glolbal variables inside the function then we have to use 'global' keyword and the respective global statement must be used as first statement.

EX:

```
a = 10
def function1():
    global a
    a = 20
    print(a)
def function2():
    print(a)

function1()
function2()
```

OP:

```
20
20
```

--> If we access a function with in the same function then that function is called as Recursive Function.

EX:

```
def factorial(no):
    result = 1
    if no == 0:
        result = 1
    else:
        result =  no * factorial(no-1)
    return result
fact = factorial(5)
print("Factorial of 5 :",fact)
```
OP:

```
Factorial of 5 : 120
```

Anonymous Functions:

--> Name less Function is called as Anonymous Function.
--> Anonymous Functions are also called as Lambdas.
--> In general, we will use Lambdas when we want to pass one function as parameter   to another function.
--> In python applications, Lambdas are able to provide less code and more   Redability.
--> To use lambdas in python applications we have to use the following two steps.

1. Create Lambda

    2. Access Lambda

1. Create lambda:
   var = lambda argList: Expression

2. Access Lambda:
   var(argValues)

EX:

```
def wish(message):
    print("Hello User,",message)
wish("Good Afternoon.")

print()
wishUser = lambda message: print("Hello 
User,",message)
wishUser("Good Evening.")
```
OP:

```
Hello User, Good Afternoon.
Hello User, Good Evening.
```

--> In Python applications, in lambdas , bydefault, expression results are returned, it is not required to use 'return' statements.
EX:

```
def wish(message):
    return "Hello User,"+message
print(wish("Good Afternoon."))

wishLambda = lambda message: "Hello User,"+message
print(wishLambda("Good Evening."))
```

OP:

```
Hello User, Good Afternoon.
Hello User,Good Evening
```

EX:

```
def add(a,b):
    return (a+b)
print("ADD :",add(10,20))

add_Lambda = lambda a,b: a+b
print("ADD :",add_Lambda(10,20))
```

OP:

```
ADD : 30
ADD : 30
```

EX:

```
---
def biggest(a,b):
    if a < b:
        return b
    else:
        return a
print("Biggest :",biggest(10,20))

biggestLambda = lambda a,b: b if a < b else a
print("Biggest :",biggestLambda(10,20))
```

In general, we will use Lambdas in Python applications when we want to pass one function as parameter to another function.

EX:

```
---
#With out Lambdas
def bigger(a,b):
    return a if a > b else b

def compare(a,b,big=bigger):
    print("Biggest : ",big(a,b))
compare(10,20)
compare(20,30)
```

OP:

```
Biggest :  20
Biggest :  30
```

EX:

```
#With Lamdas
def compare(a,b,big=lambda x,y: x if x > y else y):
    print("Biggest : ",big(a,b))
compare(10,20)
compare(20,30)
```
OP:

```
Biggest :  20
Biggest :  30
```

In Python, some of the predefined functions required functions as parameters.
EX: filter(), map(), reduce(),...

filter()
--> It will take a function as parameter where the parameter function will return True or False value as return value.
--> Filter function will take sequence as parameter and it will filter some values of the sequence.
Syntax:
----

filter(function, sequence)

Where filter will take value by value from the provided sequence , filter will check each and every value against to the conditional expression which we provided as parameter function and filter will return all the matched values.

EX:

```
---
list = [5,10,15,20,25,30,35,40,45,50]
val = filter(lambda x: x%2==0, list)
for x in val:
    print(x)
```

OP:

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

EX:

```
---
list = [5,10,15,20,25,30,35,40,45,50]
val = filter(lambda x: x%2 != 0, list)
for x in val:
    print(x)
```

```
OP:
----
5
15
25
35
45

EX:
---
list =
[5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,
95,100]
result = filter(lambda x: x<=50 , list)
for x in result:
    print(x)
OP:
---
5
10
15
20
25
30
35
40
```

```
45
50
```

EX:

```
list =
[5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,
95,100]
result = filter(lambda x: x >= 50, list)
for x in result:
    print(x)
```
OP:

```
50
55
60
65
70
75
80
85
90
95
100
```

map()
-->It will take an expression in the form of a function

and a sequence to perform an operation over all the elements of the sequence as an input and it will return all the elements which are having manipulations.
map(function, sequence)

EX:

```
list = [5,10,15,20,25,30,35,40,45,50]
result = map(lambda x: x+5 , list)
for x in result:
    print(x)
```
OP:

```
10
15
20
25
30
35
40
45
50
55
```
EX:

```
list = [5,10,15,20,25,30,35,40,45,50]
print(list)
```

```
result = map(lambda x: 2*x, list)
for x in result:
    print(x,end=" ")
```
OP:

```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
10 20 30 40 50 60 70 80 90 100
```

reduce():
---> It will reduce the provided sequence of elements into single value.
Syntax: reduce(function, sequence)
EX:

```
from functools import *
list = [1,2,3,4,5,6,7,8,9,10]
result = reduce(lambda x,y:x+y, list)
print(result)
```
OP:

```
55
```

EX:

```
from functools import *
list = [1,2,3,4,5]
result = reduce(lambda x,y:x*y, list)
```

```
print(result)
```

OP:

```
120
```

In Python applications, we are to provide alias names to the functions. To perform functiona aliasing we have to assign existed function name to another name.
Note: After providing alias name to the function we are able to use both new name and old name to access the function.
EX:

```
def calculate(a,b):
    print("ADD :", (a + b))
    print("SUB :", (a - b))
    print("MUL :", (a * b))

calculate(10,5)
print()
operate = calculate
operate(20,10)
print()
calculate(30,20)
```

--> In Python applications, after defining a function we

are able to delete that function from Python Application
Syntax: del functionName
EX:

```
def add(a,b):
    print("ADD :",(a+b))
add(10,20)
print()
del add
add(10,20) --> Error
```
OP:

```
ADD : 30
NameError: name 'add' is not defined
```

--> In Python, it is possible to declare a function inside another function, here the inner functuion is called as nested function.
--> If we declare nested functions in an outer function then nested functions are having scope upto the respective outer function only, if we want to access netsed functions then we have to access them inside the respective outer function.
EX:

```
def doTransaction():
    def deposit():
        print("Deposit Success")
    def withdraw():
        print("Withdraw Success")
    def transferFunds():
        print("Transfer Funds Success")
    deposit()
    withdraw()
    transferFunds()
doTransaction()
```
OP:

```
Deposit Success
Withdraw Success
Transfer Funds Success
```

--> In Python applications, it is possible to return a function or functions from another function, but, the respective function must be the nested function.
EX:

```
def doTransaction():
    def deposit():
        print("Deposit Success")
    def withdraw():
        print("Withdraw Success")
    def transferFunds():
        print("Transfer Funds Success")
    return deposit, withdraw, transferFunds
tx1,tx2,tx3 = doTransaction()
tx1()
tx2()
tx3()
```

OP:

```
Deposit Success
Withdraw Success
Transfer Funds Success
```

Modules:

Def: Module is the collection of variables, functions, classes,....
Def: Module is a python file contains variables, functions, classes,....

In Python applications, Modules are providing the followlwing advantages.
1. Modularization
2. Abstraction
3. Security
4. Sharability
5. Reusability

In Python applications, there are two types of Modules.

1. Pre Defined Modules
2. User defined Modules

1. Pre Defined Modules:

These modules are defined by Python programming language and which are provided along with Python Software.
EX:
functool
math
numpy
random
tkinter

---

2. User defined Modules:

These modules are defined by the developers as per their application requirements.

To use modules in python applications we have to use the following two steps.
1. Create Module.
2. Access members from module

1. Create Module:

In Python applications, Every python file is treated as module,so to create module we have to create a python file with .py extension.
EX: mymath.py

```
a = 10
b = 20
def add():
  print(a+b)
def sub():
  print(a-b)
```

2. Access Members from module

To access members from module we have to use "import" statement.
Syntax-1:
import moduleName
--> It able to import all the members from the specified module, where if we want to use the members then we must use moduleName explicitly.
EX:
---
cal.py

```
a = 10
b = 5
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
```

test.py

```
import cal
print("a   :",cal.a)
print("b   :",cal.b)
cal.add()
cal.sub()
cal.mul()
```

```
D:\python10>py test.py
a   : 10
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

2. from moduleName import memberName

--> It able to import only the specified member from the

specified modcule, where to access the member it is not required to use module name.
EX:
---
cal.py

```
a = 10
b = 5
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
```

test.py

```
from cal import a
from cal import b
from cal import add
from cal import sub
from cal import mul

print("a   :",a)
print("b   :",b)
add()
sub()
```

```
mul()
D:\python10>py test.py
a   : 10
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

--> In Python applications, it is possible to import more than one individual member by using simple import statement.
Syntax: from moduleName import member1,member2,...member-n
EX:
---
cal.py

```
a = 10
b = 5
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
```

```
test.py
--------
from cal import a,b,add,sub,mul

print("a   :",a)
print("b   :",b)
add()
sub()
mul()
```

```
D:\python10>py test.py
a   : 10
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

Note: By using single import statement we are able to import all the members of the specified module by using * notation also.
Syntax: from moduleName import *
EX:

```
cal.py
------
a = 10
b = 5
```

```
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
```

test.py

```
from cal import *

print("a   :",a)
print("b   :",b)
add()
sub()
mul()
```

```
D:\python10>py test.py
a   : 10
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

Q)What is the difference between the following two import statements?
----

1. import cal
2. from cal import *

Ans:

To import members from cal module if we use "import cal" syntax then we must use module name to access members.

To import members from cal module if we use "from cal import *" syntax then it is not required to use module name to access members.

--> In Python applications, it is possible to define alias names to the module names by using 'as' , in this context, once if we provide alias names to the module names then we are able to use alias names only to access the members , it is not possible to use original module name.
EX:
---
cal.py

```
a = 10
b = 5
def add():
    print("ADD :",(a+b))
```

```
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))
```

test.py

```
---------
import cal as math
print("a   :",math.a)
print("b   :",math.b)
math.add()
math.sub()
math.mul()
```

```
D:\python10>py test.py
a   : 10
b   : 5
ADD : 15
SUB : 5
MUL : 50
```

--> In Python applications, we are able to provide alias names to the module members also.
Syntax: from moduleName import memberName1 as aliasName1, memberName1 as aliasName2,..
EX:

```
cal.py
------
a = 10
b = 5
def add():
    print("ADD :",(a+b))
def sub():
    print("SUB :",(a-b))
def mul():
    print("MUL :",(a*b))

test.py
-------
from cal import a as x,b as y, add as sum, sub as diff,
mul as prod
print("a   :",x)
print("b   :",y)
sum()
diff()
prod()
```

Q)Find the valid python import statements from the following list?
----
1. from cal ---> Invalid
2. from cal * ---> Invalid
3. import cal ----> Valid

4. import cal * ----> Invalid
5. from cal import * ---> Valid
6. from cal import a ---> valid
7. from cal import a,b,add,sub,mul ----> valid

-->In Python applications, when we execute modules , PVM will compile the modules and PVM will generate a seperate compiled file at the location "ApplicationFolder\ __pycache__\ fileName.cpython-37.pyc

EX:D:\python10\app05\__pycache__\cal.cpython-37.pyc

--> In Python applications, a particular module will be loaded only one time even though we have provided multiple times import statements.
EX:

```
hello.py
--------
print("Hello, This is from hello module")

test.py
--------
import hello
import hello
import hello
import hello
import hello
print("Hello, This is from test module")
```

```
D:\python>py test.py
Hello, This is from hello module
Hello, This is from test module
```

--> As per the application requirement, if we want to perform modifications frequently in a module and if we want to get all these modifications in our present python application then we have to reload module time to time . To reload module we have to use a predefined function reload(--) from "importlib" module.

EX:

```
hello.py
--------
msg = "Good Morning"

test.py
--------
from importlib import reload
import hello

print(hello.msg)
print(hello.msg)
print("Application is in Pausing state, perform
modifications on hello module")
val = input()
reload(hello)
print(hello.msg)
print(hello.msg)
print(hello.msg)
```
OP:

```
Good Morning
Good Morning
Application is in Pausing state, perform modifications on
hello module
Good Afternoon
Good Afternoon
Good Afternoon
```

EX:

TicketReservation.py
```
---------------------
no_Of_Vacancies = 3
customers_And_SeatNos = {}
def bookTocket(seatNo, customerName):
    global no_Of_Vacancies
    if no_Of_Vacancies > 0:
        customers_And_SeatNos[seatNo] = customerName
        no_Of_Vacancies = no_Of_Vacancies - 1
        print("Hello",customerName, "Your Ticket is
confirmed and your seat no",seatNo)
    else:
        print("No Tickets are available")

def getStatus():
    if no_Of_Vacancies != 0:
        print("No of Vacancies :",no_Of_Vacancies)
        print("Select Seat number except
",list(customers_And_SeatNos.keys()))
    else:
        print("All seats are Filled")
```

test.py

```
from importlib import reload
import TicketReservation as res
while True:
    res.getStatus()
    seatNo = int(input("Enter Seat No : "))
    customerName = input("Enter Customer Name :")
    res.bookTocket(seatNo,customerName)
    option = input("Onemore Ticket[yes/no]? :")
    if option == "yes":
        if(res.no_Of_Vacancies == 0):
            break
        else:
            continue
    else:
        break
```

--> If we want to get all the members of the present Module we have to use "dir()" predefinede function.
EX:test.py

```
a = 10
b = 20
def f1():
    print("f1-Function")
def f2():
    print("f2-Function")
def f3():
    print("f3-Function")

print(dir())
```

OP:

```
['__annotations__', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'a', 'b', 'f1', 'f2', 'f3']
```

--> IN Python applications, by using dir() function we

are able to get other module members also.
EX:
---
hello.py

```
a = 10
b = 20
def f1():
    print("f1-Function")
def f2():
    print("f2-Function")
def f3():
    print("f3-Function")
```

test.py

```
import hello
print(dir(hello))
```

OP:

```
['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'a',
'b', 'f1', 'f2', 'f3']
```

Note: In Python applications, when we execute any module or any file then PVM will add the following predefined variables to the python file internally.

'__annotations__'
'__builtins__'
'__cached__'
'__doc__'
'__file__'
 '__loader__'
 '__name__'
'__package__'
 '__spec__'

Where '__name__' variable is able to provide information about the present python file is acting as normal python program or a python module. If the python file is acting as normal python program then PVM will assign '__Main__' value to '__name__' variable, if the python file is acting as a module then PVM will assign '__moduleName__' to '__name__' variable.
EX:
---
hello.py

```
def displayMessage():
    if __name__ == "hello":
        print("hello.py is acting as hello module")
    else:
        print("hello.py is acting as normal python
program")

displayMessage()

D:\python10>py hello.py
OP:  hello.py is acting as normal python program
```

EX:
---
hello.py

```
def displayMessage():
    if __name__ == "hello":
        print("hello.py is acting as hello module")
    else:
        print("hello.py is acting as normal python
program")

displayMessage()
```

test.py

```
import hello

D:\python>py test.py
OP: hello.py is acting as hello module
```

==
=======================================

Object Orientation:

In general, to prepare applications we will use the following types of Programming Languages.

1. Unstructered Programming Languages
2. Structered Programming Languages
3. Object Oriented Programming Languages
4. Aspet Oriented Programming Languages

Q)What are the differences between Unstructered Programming Languages and Structered Programming Languages?

Ans:
----
1. Unstructered Programming Languages are outdated Programming Languages, these programming languages are not suitable for our at present Application requirements.
EX: BASIC, FORTRAN

Structered Programming Languages are not out dated Programming Languages, these Programming Languages are suitable for our at present application requirements.
EX: C, PASCAL,...

2. UnStructered Programming Languages are not following any structer to prepare applications.

Structered Programming Languages are following a particular Structer to prepare applications.

3. Unstructered Programming Languages are using menomonic codes, they are available in less number and they will provide less no of features to the applications.

Structered Programming Languages are using High level syntaxes, they are avaiable in more no and they will provide more no of features to the applications.

4. Unstructered Programming Languages are using only "goto" statement to defined flow of execution, but, it is not good.

Structered Programming Languages are using more no of flow controllers to defined flow of execution.

5. Unstructered Programming Languages are not using

functions features, Code Redundency will be increased.

Structered Programming Languages are using "functions" features, which will increase code reusability.

Q)What are the differences between Structered Programming Languages and Object Oriented Programming Languages?

Ans:
-----
1. Structered Programming Languages are providing difficult approach to prepare applications.

Object Oriented Programming Languages are providing simplified approaches to prepare applications.

2. Structered Programming Langiages are not having Good Modularization.

  Object Orientged Programming Langvuages are Having Good Modularization.

3. Structered Programming Languages are not providing very good Abstraction Levels.
   Object Oriented Programming languages are providing

very good Abstraction levels.

4. Structered Programming Languages are not having good Security for the data.
   Object Oriented Programming Laqnguages are having very good security for the data.

5. Structered Programming Languages are not having very good Code Reusability.
   Object Oriented Programmnig Languages are having very good Code Reusability.

6. Structered Programming Languages are not having very good Sharability.
   Object Oriented Programming Languages are providing very good Sharability.

Note: Aspect Orientation is a methodology or a set of rules and regulations which are applied on Object Oriented Programming inorder to improve Sharability and Reusability.

Note: C is Structered come Procedure Oriented Programming Language, C++ and Java are Object Oriented Programming Languages, but, Python is both Structered or Procedure Oriented and Object Oriented Programming Langiage.

Object Oriented Features or Object Oriented Principles:

To describe the nature of Object Orientation, Object Oriention has provided a set of features.

1. Class
2. Object
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism

Q)What are the differences between class and Object?

Ans:
----

1. Class is a group of elements having common properties and common behaviours.

  Object is an individual element amoung the group of elements having physical   properties and physical behaviours.

2. Class is virtual.
   Object is physical or real

3. Class is virtual encapsulation of properties and behaviours.
   Object is Physical encapsulation of the properties and behaviours.

4. Class is Generalization.
   Object is Specialization.

5. Class is a model or blue print or a plan for the objects.
   Object is an instance of the class.

Q)What is the difference between Encapsulation and Abstraction?

Ans:

The process of combining or binding data and coding part is called as "Encapsulation".

The process of hiding unneccessary implementations and the process of showing neccessary implementations is called as "Abstraction".

The main advantage of "Encapsulation" and "Abstration" is "Security".
Encapsulation + Abstraction = Security.

Inheritance:

--> Inheritance is a relation between entity classes, where inheritance relation will bring variables and methods from one class[Super class/ Parent CLass/ Base Class] to another class[Sub Class / Chaild Class / Derived Class].

The main advantage of Inheritance is "Code Reusability".

Polymorphism:

Polymorphism is a Greak word, where poly means Many and Morphism means forms or Structers.

If one thing is existed in more than one form then it is called as Polymorphism.

The main advantage of Polymorphism is "Flexbility" in application development.

classes in python:

The main intention of classes is to represent all real world entities in python programming.
EX: Student, Account, Transaction, Customer,

Employee,.....

Syntax:

```
class ClassName:
    ----docstring----
    ----Consrtuctor----
    ----Variables------
    ----Methods----
```

Procedure to use Classes in python Applications:
----
1. Declare a class by using "class" keyword.
2. Declare variables and methods inside the class.
3. Create object for the class.
4. Access Class Members.

EX:

```
class Employee:
    eno = 111
    ename = "Durga"
    esal = 50000
    eaddr = "Hyd"

    def getEmployeeDetails(self):
        print("Employee Details")
        print("--------------------")
        print("Employee Number   :",self.eno)
        print("Employee Name     :",self.ename)
        print("Employee Salary   :",self.esal)
        print("Employee Address  :",self.eaddr)

emp = Employee()
emp.getEmployeeDetails()
```

OP:

```
Employee Details
--------------------
Employee Number   : 111
Employee Name     : Durga
Employee Salary   : 50000
Employee Address  : Hyd
```

'self' variable in Python:

'self' is a default variable or default parameter in all python constructors and python methods, it can be used to represent current class object.

In Python applications, we are able to refer current class variables and current class methods by using 'self' variable.

EX:

```
class Student:
    def setStudentDetails(self, sid, sname, saddr, semail,
smobile):
        self.studentId = sid
        self.studentName = sname
        self.studentAddress = saddr
        self.studentEmail = semail
        self.studentMobile = smobile

    def getStudentDetails(self):
        print("Student Details")
        print("--------------------")
        print("Student Id        :",self.studentId)
        print("Student Name      :",self.studentName)
        print("Student Address   :",self.studentAddress)
        print("Student Email Id  :",self.studentEmail)
        print("Student Mobile No :",self.studentMobile)


std = Student()
std.setStudentDetails("S-111", "Durga", "Hyd",
"durga@durgasoft.com", "91-9988776655")
std.getStudentDetails()
```

OP:

```
Student Details
--------------------
Student Id        : S-111
Student Name      : Durga
Student Address   : Hyd
Student Email Id  : durga@durgasoft.com
Student Mobile No : 91-9988776655
```

EX:

```
class A:
    a = 10
    b = 20
    def m1(self):
        print("m1-Method")
        print(self.a)
        print(self.b)
        self.m2()

    def m2(self):
        print("m2-Method")

a = A()
a.m1()
```

OP:

```
m1-Method
10
20
m2-Method
```

In Python applications, when we have same names to the local variables and to the class level instance variables, where to access class level instance variables over local variables there we will use 'self' parameter.
EX:

```
class A:
    a = 10
    b = 20
    def m1(self, a, b):
        print(a,"   ",b)
        print(self.a,"    ",self.b)
a = A()
a.m1(100,200)
```
OP:

```
100     200
10      20
```

--> In Python applications, 'self' parameter name is not fixed, it is variable, we can use any name for 'self' parameter variable.

EX:

```
class A:
    def __init__(this):
        print(this)
        print("A-Con")
a = A()
```
OP:

```
<__main__.A object at 0x000001AC744CEDC8>
A-Con
```

--> In Python constructors, Python methods, which parameter we provided as first parameter then that parameter is treated as 'self' parameter and all the remaining parameters are treated as explit parameters even though the parameter name is 'self'.
EX:

```
class A:
    def __init__(a,b,self):
        print("a value :",a)
        print("b value :",b)
        print("self value :",self)
a = A(10, 20)
```

OP:

```
a value : <__main__.A object at 0x00000264FFBEEF08>
b value : 10
self value : 20
```

docstring:

--> "docstring" is a string data in every python class, it will provide some description about the present class.
--> "dostring" is optional in python classes, we are able to write python classes with out docstring , but, if we want to write docstring in python class then we must provide that docstring as first statement in python class.
--> To access "docstring" from a class we have to use a predefined variable like '__doc__' by using class Name or by using help() predefined function.
EX:

```
class Account:
    "This is Account class"
print(Account.__doc__)
print(help(Account))
```
OP:

```
This is Account class
Help on class Account in module __main__:
```

```
class Account(builtins.object)
 |  This is Account class
 |
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

None
```

Q)Is it possible to provide more than one docstring in single python class?

Ans:

No, It is not possible to provide more than one docstring in single python class, because, in python classes, docstring must be the first statement. If we provide more than one docstring in single python class then PVM will recognize the first docstring as original docstring and remaining docstrings are treated as normal string data.
EX:

```
class Account:
    "This is Account class"
    "It will be used in Bank Applications"
print(Account.__doc__)
print(Account.__doc__)
OP:
---
This is Account class
This is Account class
```

Procedure to create Object in Python:
----
1. When PVM executes Object creation statement, PVM will load the respective class bytecode to the memory.
2. PVM will recognize all instance variables which are declared at class level and which are initialized with 'self' variable inside the constructor[__init__(self)] and PVM will calculate memory size on the basis of instance variables data types.
3. PVM will submit a request to Memory Manager about to create an object.
4. Memory Manager will search and create required block of memory as an object.
5. Memory Manager will create an unique identity in the form of a constant integer value called as Object Id / Object reference value.

6. Memory Manager will send Object Id value to PVM, where PVM will assign that Object Id value to reference variable or Id variable.
7. After generating Id value, PVM will store all instance variables which are declared at class level and which initialized through Constructor and PVM will provide their values inside the object.

To access Id value, Python has provided a predefined function in the form of id(--).
EX:

```
class Employee:
    eno=111
    ename="AAA"
    def __init__(self):
        self.esal=50000
        self.eaddr="Hyd"
    def getEmpDetails(self):
        print("Employee Details")
        print("--------------------")
        print("Employee Number  :", self.eno)
        print("Employee Name    :", self.ename)
        print("Employee Salary  :", self.esal)
        print("Employee Address :", self.eaddr)

emp = Employee()
```

```
print("Employee Object Id :",id(emp))
emp.getEmpDetails()
print()
print("Employee Details")
print("--------------------")
print("Employee Number  :", emp.eno)
print("Employee Name    :", emp.ename)
print("Employee Salary  :", emp.esal)
print("Employee Address :", emp.eaddr)
```

OP:

```
Employee Id : 2951690732744
Employee Details
--------------------
Employee Number  : 111
Employee Name    : AAA
Employee Salary  : 50000
Employee Address : Hyd

Employee Details
--------------------
Employee Number  : 111
Employee Name    : AAA
Employee Salary  : 50000
Employee Address : Hyd
```

Q)What is the difference between Object and Instance?

Ans:

Object is a block of memory to store data.
Instance is a copy or layer of data in an object at a particular point of time.
EX:

```
class A:
    def __init__(self):
        self.i = 0
        self.j = 0
a = A()
for x in range(0,10):
    a.i = a.i + 1
    a.j = a.j + 1
```

In the above program, only one Object is created and 11 no of instances are created.

Constructors In Python:

--> Constructor is a python Feature, it can be used to create Objects.
--> In Object Creation process, the role of the constructor is to provide initial values   to the objects.
--> In Python applications, Constructors are recognized and executed exactly at the time   of creating objects ,

not before creating objects and not after creating objects.
--> In Python applications, we are able to utilize constructors to provide   initializations for the instance variables and to manage a set of instructions which we want to execute while creating objects for the respective class.
--> In Python, Constructor is a special kind of method and whose name is always

```
     __init__(self)
```

Syntax:

```
def __init__(self[,param_List]):
   ----instructions-----
```

Types of Constructors:

In Python, there are two types of constructors.
1. Default Constructor
2. User Defined Constructor

1. Default Constructor:

Default constructor is a constructor , it will be provided by PVM when we have not provided any construictor in the respective class. If we provide any constructor explicitly then PVM will not provide default constructor.

In Python applications, PVM will provide default constructor with the following syntax.

```
def __init__(self):
   ----
```

EX:

```
class Account:
    accNo = "abc123"
    accHolderName = "Durga"
    accType = "Savings"
    balance = 600000

    def getAccountDetails(self):
        print("Account Details")
        print("---------------------")
        print("Account  No  :", self.accNo)
        print("Account Holder Name
:",self.accHolderName)
        print("Account Type :",self.accType)
        print("Account Balance :",self.balance)

acc = Account()
acc.getAccountDetails()
```

2. User Defined Constructor:

These constructores are defined by the developers as per their application requirements.

In Every constructor "self" parameter is default parameter, so that, there is no feature like 0-arg constructor.

In Python applications, as per the requirement , if we want to provide parameters explicitly in constructors then we can provide the parameters after "self" parameter.

EX:

```
class Account:
    def __init__(self):
        self.accNo = "abc123";
        self.accHolderName = "Durga"
        self.accType = "Savings"
        self.balance = 50000
    def getAccountDetails(self):
        print("Account Details")
        print("----------------------")
        print("Account Number     :",self.accNo)
        print("Account Holder Name:",
self.accHolderName)
        print("Account Type        :", self.accType)
        print("Account Balance     :", self.balance)
acc = Account()
acc.getAccountDetails()
```

OP:

```
Account Details
----------------------
Account Number     : abc123
Account Holder Name: Durga
Account Type       : Savings
Account Balance    : 50000
```

EX:

```
class Account:
    def __init__(self, accNo, accHolderName, accType,
balance):
        self.accNo = accNo
        self.accHolderName = accHolderName
        self.accType = accType
        self.balance = balance
    def getAccountDetails(self):
        print("Account Details")
        print("----------------------")
        print("Account Number      :",self.accNo)
        print("Account Holder Name:",
self.accHolderName)
        print("Account Type        :", self.accType)
        print("Account Balance     :", self.balance)

acc1 = Account("a111", "AAA", "Savings", 10000)
acc1.getAccountDetails()
print()
acc2 = Account("a222", "BBB", "Savings", 20000)
acc2.getAccountDetails()
print()
acc3 = Account("a333", "CCC", "Savings", 30000)
acc3.getAccountDetails()
```

OP:

```
---
Account Details
---------------------
Account Number      : a111
Account Holder Name: AAA
Account Type        : Savings
Account Balance     : 10000

Account Details
---------------------
Account Number      : a222
Account Holder Name: BBB
Account Type       : Savings
Account Balance    : 20000

Account Details
----------------------
Account Number     : a333
Account Holder Name: CCC
Account Type       : Savings
Account Balance    : 30000
```

Q)What are the differences between Constructor and Method?

Ans:

----

1. Constructor can be used to create objects.
   Method is a set of instructions, it can be used to represent a particular action.

2. Constructors are used to provide initial values inside the objects.
   Methods are not used to provide initializations inside the objects.

3. Constructor will be executed automatically when Object is created.

Methods are executed the moment when we access them.

4. Constructor name is fixed that is __init__ .
   Method name is not fixed.

5. For one object constructor will be executed only once, that is, at the time of creating object.
   For one object, we are able to access methods in any number of times.

Variables in Python Classes:

In Python, the main intention of variables is to store entities data.

There are three types of variables in Python.
1. Instance Variables
2. Static Variables
3. Local variables

1. Instance Variables:

If any variable values is changed from one instance to another instance of an object then that variaable is called as Instance variable.

In Python applications, if we create multiple objects for single class then a seperate copy of instance variables are created at each and every object. In this context, if we perform modifications on one object over instance variables then that modification is available upto the respective object only, it is not applicable to all the objects.

In Python applications, we are able to declare instance variables in the following locations.

1. Inside Constuctor by using 'self' variable.
2. Inside Instance Method by using self.
3. In Outside of the class by using Object reference variables.

EX:

```
class Employee:
    def __init__(self):
        self.eno = 111
        self.ename = "AAA"

    def setEmpDetails(self):
        self.equal = "BTech"
        self.esal = 50000

emp = Employee()
```

```
print(emp.__dict__)
emp.setEmpDetails()
print(emp.__dict__)
emp.email = "durga@durgasoct.com"
emp.emobile = "91-9988776655"
print(emp.__dict__)
Op:
---
{'eno': 111, 'ename': 'AAA'}
{'eno': 111, 'ename': 'AAA', 'equal': 'BTech', 'esal':
50000}
{'eno': 111, 'ename': 'AAA', 'equal': 'BTech', 'esal':
50000, 'email': 'durga@durgasoct.com', 'emobile':
'91-9988776655'}
```

To access instance variables data we will use the following locations.
1. Inside constructors by using self variable.
2. Inside Instance method by using 'self' variable.
3. In Out side of the class by using object reference variable.

EX:

```
class Employee:
    def __init__(self):
        self.eno = 111
        self.ename = "AAA"
        print("Inside Constructor")
        print("eno value :",self.eno)
        print("ename value :",self.ename)
    def getEmpDetails(self):
        print("Inside Instance Method")
        print("eno Value :",self.eno)
        print("ename Value :",self.ename)

emp = Employee()
emp.getEmpDetails()
print("In Out side of the class")
print("eno Value :",emp.eno)
print("ename Value  :"+emp.ename)
```

OP:

```
---
Inside Constructor
eno value : 111
ename value : AAA
Inside Instance Method
eno Value : 111
ename Value : AAA
In Out side of the class
eno Value : 111
ename Value  :AAA
```

In Python, it is possible to delete instance variables from

Python Class object in the following locations.
1. Inside __init__() by using self variable.
2. In side instance method by using 'self' variable.
3. In out side of the clsas by using object referererence variable.
EX:

```
class Employee:
    def __init__(self):
        self.eno = 111
        self.ename = "AAA"
        self.esal = 50000
        self.eaddr = "Hyd"
        del self.eno
    def getEmpDetails(self):
        del self.ename

emp = Employee()
print(emp.__dict__)
emp.getEmpDetails()
print(emp.__dict__)
del emp.esal
del emp.eaddr
print(emp.__dict__)
```

OP:

```
{'ename': 'AAA', 'esal': 50000, 'eaddr': 'Hyd'}
{'esal': 50000, 'eaddr': 'Hyd'}
{}
```

In Python applications, if we perform modifications on instance variables in an object then that modifications are applicable upto the respective object only, that modifications are not applicable for all the objects of the respective clsas.
EX:

```
class Student:
    def __init__(self):
        self.sid = "S-111"
        self.sname = "AAA"

std1 = Student()
std2 = Student()

print("std1 Data :",std1.sid,"  ",std1.sname)
print("std2 Data :",std2.sid,"  ",std2.sname)

std1.sid = "S-222"
std1.sname = "XXX"

print("std1 Data :",std1.sid,"  ",std1.sname)
print("std2 Data :",std2.sid,"  ",std2.sname)
```

OP:

```
std1 Data : S-111    AAA
std2 Data : S-111    AAA
std1 Data : S-222    XXX
std2 Data : S-111    AAA
```

EX:

```
class Student:
    def __init__(self):
        self.sid = "S-111"
        self.sname = "AAA"

std1 = Student()
std2 = Student()

print("std1 Data :",std1.sid,"  ",std1.sname)
print("std2 Data :",std2.sid,"  ",std2.sname)
del std1.sid
print("std1 Data :",std1.sname)
print("std2 Data :",std2.sid,"  ",std2.sname)
```

OP:

```
std1 Data : S-111    AAA
```

```
std2 Data : S-111    AAA
std1 Data : AAA
std2 Data : S-111    AAA
```

2. Static Variables:

If any variable is not having changes in its value from one instance to another instance then that variable is called as Static variable.

In Python applications, if we create multiple objects for a particular class then a single copy of static variable value will be shared to multiple objects.

In general, in Python applications, we are able to access static variables either by using class name or by using object reference variables.

In python applications, we are able to declare static variables in the following locations.
1. In out side of the methods and inside class.
2. Inside constructor by using class name.
3. Inside Instance method by using class name.
4. Inside classmethod[a Method is decvlared with @classmethod decorator] by using class   name and cls variable.

5. Inside static method[a method is declared with @staticmethod decorator] by using   class name.
6. In out side of the class by using class name.
EX:

```
class A:
    i = 10
    j = 20
    def __init__(self):
        A.k = 30
        A.l = 40
    def m1(self):
        A.m = 50
        A.n = 60

    @classmethod
    def m2(cls):
        A.o = 70
        cls.p = 80

    @staticmethod
    def m3():
        A.q = 90

print(A.__dict__)
a = A()
print(A.__dict__)
```

```
a.m1()
print(A.__dict__)
A.m2()
print(A.__dict__)
A.m3()
print(A.__dict__)
A.r = 100
print(A.__dict__)
OP:
---
{'i': 10, 'j': 20, .... }
{'i': 10, 'j': 20, 'k': 30, 'l': 40 .... }
{'i': 10, 'j': 20,  'k': 30, 'l': 40, 'm': 50, 'n': 60 ....}
{'i': 10, 'j': 20, 'k': 30, 'l': 40, 'm': 50, 'n': 60, 'o': 70, 'p':
80,...}
{'i': 10, 'j': 20, 'k': 30, 'l': 40, 'm': 50, 'n': 60, 'o': 70, 'p':
80, 'q': 90,...}
{'i': 10, 'j': 20, 'k': 30, 'l': 40, 'm': 50, 'n': 60, 'o': 70, 'p':
80, 'q': 90, 'r': 100,.....}
```

In Python applications, we are able to access static variables in the following locations.

1. In out side of the class by using class name and by using refverence variable.
2. Inside the constructor by using Class Name
3. Inside Instance method by using class name and self

variable.
4. Inside classmethod by using class name and by using cls variable.
5. Inside static method by using class name.

EX:

```
class A:
    i = 10
    j = 20
    def __init__(self):
        print("Inside Constructor Class Name  :",A.i,"
",A.j)
    def m1(self):
        print("Inside Instance Method Class Name :",A.i,"
",A.j )
              print("Inside Instance Method Self Var
:",self.i,"   ",self.j )
    @classmethod
    def m2(cls):
        print("Inside class method Class Name :",A.i,"
",A.j)
        print("Inside class method cls var    :",cls.i,"
",cls.j)
    @staticmethod
    def m3():
        print("Inside static method class name :",A.i,"
",A.j)
a = A()
a.m1()
a.m2()
a.m3()
print("Out side of the Class Ref Var :",a.i,"   ",a.j)
print("Out Side of The Class Class Name :",A.i,"   ",A.j)
```

OP:

```
Inside Constructor Class Name  : 10     20
Inside Instance Method Class Name : 10     20
Inside Instance Method self var : 10     20
Inside class method Class Name : 10     20
Inside class method cls var    : 10     20
Inside static method class name : 10     20
Out side of the Class Ref Var : 10     20
Out Side of The Class Class Name : 10     20
```

--> In Python applications, we are able to delete static variables in the following locations.

1. In out side of the class by using class name
2. Inside COnstrucotr by using class name.
3. Inside instance method by using class name.
4. Inside classmethod by using class name and cls variable

5. Inside static method by using class name

EX:

```
class A:
    i = 10
    j = 20
    k = 30
    l = 40
    m = 50
    n = 60
    def __init__(self):
        del A.i
    def m1(self):
        del A.j
    @classmethod
    def m2(cls):
        del A.k
        del cls.l
    @staticmethod
    def m3():
        del A.m
print(A.__dict__)
a = A()
print(A.__dict__)
a.m1()
print(A.__dict__)
```

```
A.m2()
print(A.__dict__)
A.m3()
print(A.__dict__)
del A.n
print(A.__dict__)
```

OP:

```
{'i': 10, 'j': 20, 'k': 30, 'l': 40, 'm': 50, 'n': 60,
{'j': 20, 'k': 30, 'l': 40, 'm': 50, 'n': 60,
{'k': 30, 'l': 40, 'm': 50, 'n': 60, ..}
{'m': 50, 'n': 60, }
{'n': 60, }
{ }
```

In Python applications, if we are trying to perform modifications on Static variables by using object referfence variable then PVM will not give any effect to the static variable, here , PVM will create the same equalent instance variables with the provided values.

Note: In case of static variables, if we perform modifications with class Name then only PVM will perform modifications on Static variables.
EX:

```
class A:
    i = 10
    j = 20
a1 = A()
print("Before Updations Class Name : ",A.i,"   ",A.j)
print("Before Updations Ref Value  : ",a1.i,"   ",a1.j)
print("Before Updations Static variables Dict    :
",A.__dict__)
print("Before Updations Instance Variables Dict  :
",a1.__dict__)
print()
a1.i = 100
a1.j = 200
print("After Updations  Class Name : ",A.i,"   ",A.j)
print("After Updations  Ref Value: ",a1.i,"   ",a1.j)
print("After Updations Static variables Dict    :
",A.__dict__)
print("After Updations Instancd Variables Dict  :
",a1.__dict__)
```

OP:

```
Before Updations Class Name :  10     20
Before Updations Ref Value  :  10     20
Before Updations Static variables Dict    :  {'i': 10, 'j': 20,
}
Before Updations Instance Variables Dict  :  {}
```

```
After Updations  Class Name :  10     20
After Updations  Ref Value:  100     200
After Updations Static variables Dict    :  {'i': 10, 'j': 20, }
After Updations Instancd Variables Dict  :  {'i': 100, 'j':
200}
```

3. Local variables

--> These variables are declared insid methods, these variables are utilized for temporary purpose.
--> Local variables are created the moment when we access the respective method and local variables are destroyed the moment when we complete the execution of the respective method.
--> Loical variables are having scope upto the respective method only, they are not having scope in out side of the methods.
EX:

```
class A:
    def add(self):
        i = 10
        j = 20
        print("ADD : ",(i+j))
    def sub(self):
        i = 10
        j = 5
        print("SUB : ",(i-j))
    def mul(self):
        i = 10
        j = 5
        print("MUL : ",(i*j))
a = A()
a.add()
a.sub()
a.mul()
```

```
OP:
---
ADD :  30
SUB :  5
MUL :  50
```

Methods in Python:

In Python classes, method is a set of instructions representing a particular action or behaviour of an entity.

There are three types of methods in Python.
1. Instance Method.
2. Class Method
3. Static Method

1. Instance Method:

--> If any Python method is performing operations on atleast one instance variable then that method is called as Instance method.

--> Inside instance methods, we need to pass atleast self parameter, because, by using self parameter only we are able to manipulate instance variables.

--> In Python applications, we are able to access instance methods by using object reference variables in out side of the methods, but, inside the methods we can use "self" parameter.
EX:

```
class Employee:
    def __init__(self, eid, ename, esal):
        self.eid = eid
        self.ename = ename
        self.esal = esal
    def setAccountDetails(self, accNo, accHolderName,
accBranch, bankName):
        self.accNo = accNo
        self.accHolderName = accHolderName
        self.accBranch = accBranch
        self.bankName = bankName
    def setAddressDetails(self, hno, street, city, state, 
country):
        self.hno = hno
        self.street = street
        self.city = city
        self.state = state
        self.country = country
    def getEmployeeDetails(self):
        print("Employee Details ")
        print("---------------------------")
        print("Employee Id        :",self.eid)
        print("Employee Name      :",self.ename)
        print("Employee Salary    :",self.esal)
        print()
        print("Account Details")
        print("-------------------")
        print("Account Number       :",self.accNo)
        print("Account Holder Name 
:",self.accHolderName)
        print("Account Branch       :",self.accBranch)
        print("Account Bank         :",self.bankName)
        print()
        print("Address Details")
        print("---------------------")
        print("House Number     :",self.hno)
        print("Street           :",self.street)
        print("City              :",self.city)
        print("State             :",self.state)
        print("Country           :",self.country)

emp = Employee("E-111", "AAA", 5000)
emp.setAccountDetails("abc123", "AAA","Ameerpet",
"ICICI Bank")
emp.setAddressDetails("102, 128/3rt", "MG Road",
"Hyd", "Telangana", "India")
emp.getEmployeeDetails()
OP:
---
Employee Details
---------------------------
Employee Id       : E-111
Employee Name     : AAA
Employee Salary   : 5000

Account Details
-------------------
Account Number      : abc123
Account Holder Name : AAA
Account Branch      : Ameerpet
Account Bank        : ICICI Bank

Address Details
---------------------
```

House Number   : 102, 128/3rt
Street   : MG Road
City   : Hyd
State   : Telangana
Country   : India

In Python, there are two types of instance methods.
1. Mutator Methods
2. Accessor Methods

Q)What is the difference between Mutator Methods and Accessor Methods?

-
Mutator method is a Python method, it can be used to modify or set data in Object.
EX: setXXX() methods in Python applications.

Accessor Methods are Python methods, which are used to access data from Object.
EX: getXxx() methods in Python applications.

EX:

```
class Student:
    def setSid(self, sid):
        self.sid = sid
    def setSname(self, sname):
        self.sname = sname
    def setSaddr(self,saddr):
        self.saddr = saddr

    def getSid(self):
        return self.sid
    def getSname(self):
        return self.sname
    def getSaddr(self):
        return self.saddr

std = Student()
std.setSid("S-111")
std.setSname("AAA")
std.setSaddr("Hyd")
print("Student Details")
print("-------------------")
print("Student Id        :",std.getSid())
print("Student Name      :",std.getSname())
print("Student Address   :",std.getSaddr())
```

OP:

```
Student Details
-------------------
Student Id        : S-111
Student Name     : AAA
Student Address  : Hyd
```

2. Class Method:

--> It is a python method, it will work on class variables[Static Variables].
--> To declare class methods we will use @classmethod decorator.
--> In Python, all class methods are not having "self" parameter, all class methods are   having 'cls' parameter, where ls parameter can be used to access class variables.
--> In Python classes, we are able to access class methods either by using class name or   by using object referfence variable.
EX:

```
class A:
    i = 10
    j = 20

    @classmethod
    def m1(cls):
        print("i value :",cls.i)
        print("j value :",cls.j)
```

```
A.m1()
a = A()
a.m1()
```

OP:

```
i value : 10
j value : 20
i value : 10
j value : 20
```

EX:

```
class Customer:
    count = 0
    def __init__(self):
        Customer.count = Customer.count + 1

    @classmethod
    def getObjectsCount(cls):
        print("No of Objects :", cls.count)

cust1 = Customer()
cust2 = Customer()
cust3 = Customer()
cust4 = Customer()
cust5 = Customer()
```

```
Customer.getObjectsCount()
```

OP:

```
No of Objects : 5
```

3. Static Method:

--> It is a Python method, it will not use instance variables and class variables.
--> To declare static methods wed will use @staticmethod decoarator.
--> Static methods are not having self parameter and cls parameter.
--> In python, we are able to access static methods by using class name directly or by using object reference variable.

EX:

```
class A:
    @staticmethod
    def m1(i, j):
        print("m1()-Method")
        print(i,"   ",j)
A.m1(10,20)
a = A()
```

```
a.m1(30,40)

OP:
---
m1()-Method
10    20
m1()-Method
30    40

EX:
---
class Calculator:

    @staticmethod
    def add(fval, sval):
        print("ADD  :",(fval + sval))

    @staticmethod
    def sub(fval, sval):
        print("SUB  :",(fval - sval))

    @staticmethod
    def mul(fval, sval):
        print("MUL  :",(fval * sval))

Calculator.add(10,5)
Calculator.sub(10,5)
Calculator.mul(10,5)
```

OP:

```
ADD  : 15
SUB  : 5
MUL  : 50
```

--> In Python applications, we are able to pass one class object reference variable as parameter to a constructor or a method in another class.
EX:

```
class Account:
    def __init__(self, accNo, accHolderName, accType, 
balance):
        self.accNo = accNo
        self.accHolderName = accHolderName
        self.accType = accType
        self.balance = balance

class Employee:
    def __init__(self, eid, ename, esal, eaddr, account):
        self.eid = eid
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr
        self.account = account

    def getEmpDetails(self):
        print("Employee Details")
        print("----------------------")
        print("Employee ID      :",self.eid)
        print("Employee Name    :",self.ename)
        print("Employee Salary  :",self.esal)
        print("Employee Address :",self.eaddr)
        print()
        print("Account Details")
        print("----------------------")
        print("Account Number      :",self.account.accNo)
        print("Account Holder Name
:",self.account.accHolderName)
        print("Account Type        :",self.account.accType)
        print("Account Balance     :",self.account.balance)

acc = Account("abc123", "Durga", "Savings", 25000)
emp = Employee("E-111", "Durga", 10000, "Hyd", acc)
emp.getEmpDetails()
```

OP:

```
Employee Details
----------------------
Employee ID      : E-111
Employee Name    : Durga
Employee Salary  : 10000
Employee Address : Hyd

Account Details
----------------------
Account Number      : abc123
Account Holder Name : Durga
Account Type        : Savings
Account Balance     : 25000
```

Inner Classes:

In Python applications, we are able to declare one class in another class, here the internal class is called as Inner class.

Syntax:

```
class Outer:
  ----
  class Inner:
     -----
```

In Python, if we want to access members of inner class then we have to create object for inner class.

Syntax-1:

```
refVar = Outer().Inner()
```

Syntax-2:
```
o = Outer()
i = o.Inner()
```

In Python applications, outer class reference variable can be used to access only outer class members, inner class reference variable can be used to access only inner class memebers.

EX:

```
class A:
    i = 10
    def m1(self):
        print("m1-A")
    class B:
        j = 20
        def m2(self):
            print("m2-B")
a = A()
print(a.i)
a.m1()
#print(a.j) --> Error
#a.m2() ------> Error
b = A().B()
#print(b.i)-->Error
#b.m1() --> Error
```

```
print(b.j)
b.m2()
```

OP:

```
10
m1-A
20
m2-B
```

In Python applications, Outer class members are not available to inner class directly and inner class members are not available to outer class directly.
EX:

```
class A:
    i = 10
    def m1(self):
        #print(j) --> Error
        print("m1-A")
    class B:
        j = 20
        def m2(self):
           # print(self.i)--> Error
            print("m2-B")
b = A().B()
b.m2()
```

```
a = A()
a.m1()
```

OP:

```
m2-B
m1-A
```

--> In Python, we are able to declare any no of inner classes in a single outer class.
EX:

```
class Account:
    def __init__(self):
        print("Account Object is Created")
    class StudentAccount:
        def createStudentAccount(self):
            print("Student Account is Created")
    class EmployeeAccount:
        def createEmployeeAccount(self):
            print("Employee Account is Created")
    class LoanAccount:
        def createLoanAccount(self):
            print("Loan Account is Created")

acc = Account()
stdAccount = acc.StudentAccount()
```

```
stdAccount.createStudentAccount()

empAccount = acc.EmployeeAccount()
empAccount.createEmployeeAccount()

loanAccount = acc.LoanAccount()
loanAccount.createLoanAccount()
```

OP:

```
Account Object is Created
Student Account is Created
Employee Account is Created
Loan Account is Created
```

Garbage Collection:

In general, in Object Oriented Programming Languages data must be represented in the form of Objects.

In Object Oriented Programming Languages to manage data in the form of objects first we have to create objects and then we have to destroy objects atleast at the endc of the applications.

In general, in Object Oriented programming Languages to create objects we will use constructors and to destroy objects we will use destructors.

In C++ programming language, Developers must take responsibility explicitly to create objects and to destroy objects, where in C++ applications, to destroy objects we must use destructors.

In Java programming language, developers are not required to take any responsibility to destroy objects explicitly, because, JAVA has Garbage Collector to destroy objects automatically.

Similarily, in Python, we will use both Garbage Collector and Destructor method to destroy objects.

In Python, when Garbage Collector destroy objects autimatically Destructor method will be executed internally.

In Python, Garbage collector will destroy objects internally and which is enabled bydefault in every python application.

Python has provided Grabage Collector in a seperate module called as "gc" , if we want to use Garbage Collector explicitly then we have to import 'gc' module.

```
import gc
```

To check whether Garbage Collector is enable or disble we have to use the following function from 'gc' module.

```
gc.isenabled()
```

To enable Garbage collector explicitly we have to use the following function from gc module.

```
gc.enable()
```

To disable Garbage Collector explicitly we have to use the following function from GC module.

```
gc.disable()
```

EX:

```
import gc
print(gc.isenabled())
gc.disable()
print(gc.isenabled())
gc.enable()
print(gc.isenabled())
```

OP:

```
---
True
False
True
```

In Python, if we want to destroy any object explicitly then we have to assign None value object reference variable or we have to delete object reference variable from Python program.

In Python applications, when Object is destroying, Garbage Collector will execute destructor method just before destroying object inorder to perfomr clean up operations.

In Python, destructor method must have a fixed name like below.

```
  def __del__(self):
      ----
```

EX:

```
---
class A:
    def __init__(self):
        print("Object Creating....")

    def __del__(self):
        print("Object Destroying....")
```

```
a = A()
a = None
```

OP:

```
Object Creating....
Object Destroying....
```

EX:

```
class A:
    def __init__(self):
        print("Object Creating....")

    def __del__(self):
        print("Object Destroying....")

a = A()
del a
```

OP:

```
Object Creating....
Object Destroying....
```

In Python applications, Garbage Collector will destroy all the objects automatically just before terminating

application execution.
EX:

```
class A:
    def __init__(self):
        print("Object Creating....")

    def __del__(self):
        print("Object Destroying....")

a1 = A()
a2 = A()
a3 = A()
```
OP:

```
Object Creating....
Object Creating....
Object Creating....
Object Destroying....
Object Destroying....
Object Destroying....
```

In Python applications, Garbage Collector will destroy an object when that object does not have any reference value. If any single reference variable is existed then Garbage Collector will not destroy that object.
EX:

```
import time
class A:
    def __init__(self):
        print("Object Creating....")

    def __del__(self):
        print("Object Destroying....")

a = A()
b = a
c = a
d = a
time.sleep(5)
del a
print("deleting a")
time.sleep(5)
del b
print("deleting b")
time.sleep(5)
del c
print("deleting c")
time.sleep(5)
del d
print("deleting d")
```

OP:

```
---
Object Creating....
deleting a
deleting b
deleting c
Object Destroying....
deleting d
```

EX:

```
---
import time
class A:
    def __init__(self):
        print("Object Creating....")

    def __del__(self):
        print("Object Destroying....")

list = [A(),A(),A(),A()]
del list
time.sleep(5)
print("End of Application")
```

OP:

```
---
Object Creating....
Object Creating....
Object Creating....
Object Creating....
Object Destroying....
Object Destroying....
Object Destroying....
Object Destroying....
End of Application
```

In Python, it is possible to find the no of references for an object by using sys.getrefcount(--) function from sys module.

EX:

```
import sys
class A:
    pass
a = A()
b = a
c = a
d = a
print(sys.getrefcount(a))
```

OP:

```
5
```

Relationships in Python:

In Python, the main intention of relationships is
1. To provide less execution
2. To provide Code Reusability
3. To provide less memory utilization.

In Python, there are two types of Relationships are existed.
1. HAS-A Relationship
2. IS-A relationship

Q)What is the difference between HAS-A Relationship and IS-A Relationship?

Ans:

HAS-A relationship is able to define associations between entities, where associations are able to improve communication between entities and data navigation between entities.

IS-A relationship is able to define Inheritance relation between entity classes, where inheritance relation is able to provide Code Reusability in Python applications.

Associations In Python:

The main intention of associations in Python applications is to provide communication between entity classes and it will improve data navigation between entity classes.

There are four types of associations in Python.
1. One-To-One Association
2. One-To-Many Association
3. Many-To-One Association
4. Many-To-Many Association

To define associations in Python applications, we have to declare one or more reference variables of an entity class in another entity class.

EX:

```
class Address:
  def __init__(self, hno, street, city, state):
        self.hno = hno
        self.street = street
        self.city = city
        selft.state = state

class Account:
  def __init__(self, accNo, accHolderName, accType):
       self.accNo = accNo
       self.accHolderName = accHolderName
```

```
      self.accType = accType

class Employee:
   def __init__(self, eno, ename, esal, account, addrList):
      self.eno = eno
      self.ename = ename
      self.esal = esal
      self.account = account #One to One
      self.addrList = addrList #One To Many
   -----

emp = Employee(111, "Durga", 50000.0,[Address(--),
Address(---), Address(----)])
```

1. One-To-One Association:

It is a relation between entities, where one instance of an entity should me mapped with exactly one instance of another entity.
EX: Each and Every Employee has exactly one individual Account.

EX:

```
class Account:
    def __init__(self, accNo, accHolderName, accType,
balance):
        self.accNo = accNo
        self.accHolderName =  accHolderName
        self.accType = accType
        self.balance = balance

class Employee:
    def __init__(self, eno, ename, esal, eaddr, account):
        self.eno = eno
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr
        self.account = account

    def getEmpDetails(self):
        print("Employee Details")
        print("-------------------")
        print("Employee Number  :",self.eno)
        print("Employee Name    :",self.ename)
        print("Employee Salary  :",self.esal)
        print("Employee Address :",self.eaddr)
        print()
        print("Account Details")
        print("-------------------")
        print("Account Number       :",self.account.accNo )
        print("Account Holder Name
:",self.account.accHolderName)
```

```
        print("Account Types         :",self.account.accType)
        print("Account Balance       :",self.account.balance)

acc = Account("abc123", "Durga", "Savings", 25000)
emp = Employee("E-111", "Durga", 10000.0, "Hyd", acc
)
emp.getEmpDetails()
```

OP:

```
Employee Details
-------------------
Employee Number  : E-111
Employee Name    : Durga
Employee Salary  : 10000.0
Employee Address : Hyd

Account Details
-------------------
Account Number       : abc123
Account Holder Name  : Durga
Account Types        : Savings
Account Balance      : 25000
```

2. One-To-Many Association:

It is a relation between entity classes, where one instance of an entity should be mapped with multiple instances of another entity.
EX: Single Department has Multiuple Employees.

EX:

```
class Employee:
    def __init__(self, eno, ename, esal, eaddr):
        self.eno = eno
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr

class Department:
    def __init__(self, did, dname, empList):
        self.did = did
        self.dname = dname
        self.empList = empList

    def getDeptDetails(self):
        print("Department Details")
        print("------------------------")
        print("Department Id     :",self.did)
        print("Department Name  :",self.dname)
        print("ENO\tENAME\tESAL\tEADDR")
        print("----------------------------")
        for emp in self.empList:
            print(emp.eno,end="\t")
            print(emp.ename,end="\t")
            print(emp.esal,end="\t")
            print(emp.eaddr,end="\n")

e1 = Employee(111, "AAA", 500000.0, "Hyd")
e2 = Employee(222, "BBB", 150000.0, "Hyd")
e3 = Employee(333, "CCC", 250000.0, "Hyd")
e4 = Employee(444, "DDD", 350000.0, "Hyd")
empsList = [e1,e2,e3,e4]

dept = Department("D-111", "Admin", empsList)
dept.getDeptDetails()
```

OP:

```
Department Details
------------------------
Department Id     : D-111
Department Name   : Admin
ENO         ENAME     ESAL              EADDR
-------------------------------------
111         AAA       500000.0 Hyd
222         BBB       150000.0 Hyd
333         CCC       250000.0 Hyd
444         DDD       350000.0 Hyd
```

Many-To-One Association:

It is a relation between entity classes, where multiple instances of an entity should be mapped with exaxctly one instances of another entity.
EX: Multiple Students have joined with Single Branch.

EX:

```
class Branch:
    def __init__(self, bid, bname):
        self.bid = bid
        self.bname = bname
class Student:
    def __init__(self, sid, sname, saddr, branch):
        self.sid = sid
        self.sname = sname
        self.saddr = saddr
        self.branch = branch
    def getStudentDetails(self):
        print("Student Details")
        print("----------------------")
        print("Student Id        :",self.sid)
        print("Student Name      :",self.sname)
        print("Student Address   :",self.saddr)
        print("Branch Id         :",self.branch.bid)
        print("Branch Name      :",self.branch.bname)
        print()

branch = Branch("B-111", "CS")
std1 = Student("S-111", "AAA", "Hyd", branch)
std2 = Student("S-222", "BBB", "Hyd", branch)
std3 = Student("S-333", "CCC", "Hyd", branch)
std4 = Student("S-444", "DDD", "Hyd", branch)

std1.getStudentDetails()
std2.getStudentDetails()
std3.getStudentDetails()
std4.getStudentDetails()
```

OP:

```
---
Student Details
----------------------
Student Id       : S-111
Student Name      : AAA
Student Address  : Hyd
Branch Id        : B-111
Branch Name      : CS

Student Details
----------------------
Student Id       : S-222
Student Name     : BBB
Student Address  : Hyd
Branch Id        : B-111
Branch Name      : CS

Student Details
-----------------------
Student Id       : S-333
Student Name     : CCC
Student Address  : Hyd
Branch Id        : B-111
Branch Name      : CS

Student Details
-----------------------
Student Id       : S-444
Student Name     : DDD
Student Address  : Hyd
Branch Id        : B-111
Branch Name      : CS
```

Many_To_Many Association:

It is a relation between entity classes, where multiple instances of an entity should be mapped with mutliple instances of another entity.
EX: Multiple Students have joined multiple Courses.

EX:

```
class Course:
    def __init__(self, cid, cname, ccost):
        self.cid = cid
        self.cname = cname
        self.ccost = ccost

class Student:
    def __init__(self, sid, sname, saddr, coursesList):
        self.sid = sid
        self.sname = sname
        self.saddr = saddr
        self.coursesList = coursesList

    def getStudentDetails(self):
        print("Student Details")
        print("---------------------")
        print("Student Id        :",self.sid)
        print("Student Name      :",self.sname)
        print("Student Address  :",self.saddr)
        print("CID\tCNAME\tCCOST")
        print("------------------------")
        for course in self.coursesList:
            print(course.cid,end="\t")
            print(course.cname,end="\t")
            print(course.ccost,end="\n")
        print()

course1 = Course("C-111", "C", 1000)
course2 = Course("C-222", "C++", 2000)
course3 = Course("C-333", "Java", 5000)
course4 = Course("C-444", "Python", 6000)
coursesList = [course1, course2, course3, course4]

std1 = Student("S-111", "AAA", "Hyd", coursesList)
std2 = Student("S-222", "BBB", "Hyd", coursesList)
std3 = Student("S-333", "CCC", "Hyd", coursesList)

std1.getStudentDetails()
std2.getStudentDetails()
std3.getStudentDetails()
```

OP:

```
Student Details
---------------------
Student Id        : S-111
Student Name      : AAA
Student Address   : Hyd
CID       CNAME    CCOST
------------------------
C-111     C        1000
```

```
C-222      C++        2000
C-333      Java       5000
C-444      Python     6000

Student Details
----------------------
Student Id       : S-222
Student Name     : BBB
Student Address  : Hyd
CID        CNAME    CCOST
-------------------------
C-111      C          1000
C-222      C++        2000
C-333      Java       5000
C-444      Python     6000

Student Details
----------------------
Student Id       : S-333
Student Name     : CCC
Student Address  : Hyd
CID        CNAME    CCOST
-------------------------
C-111      C          1000
C-222      C++        2000
C-333      Java       5000
C-444      Python     6000
```

In Python, Associations are represented in two forms.
1. Composition
2. Aggregation

Q)What are the differences between Composition and Aggregation?

Ans:
----
1. Strong association between entity classes is called as Composition.
   Weak Association between entity classes is called as Aggregation.

2. In case of Composition, if we destroy container object then COntained object is also   be destroyed, conatined object will not be existed after destroying container object.

  In case Aggregation, even if we destroy Container object then Contained object is   existed.
3.In case of Composition, the life time of Contained object is almost all same as   Container object lifetime.

  In case of Aggregation, the life time of Contained object is not same as Contaier   object.

EX: Association between Library and Books is Composition.
    Association between Library and Students is Aggregation.

Dependency Injection:

Injecting dependent object in another object is called as Dependency Injection.

IN Python applications, we are able to achieve dependency injection in the following two ways.
1. Constructor Dependency Injection.
2. Setter Method Dependency Injection.

1. Constructor Dependency Injection:

If we inject dependent object in an object through Constructor then this it is called as Consructor Dependency Injection.
EX:

```
class Address:
    def __init__(self, hno, street, city, state, country):
        self.hno = hno
        self.street = street
        self.city = city
        self.state = state
        self.country = country

class Course:
    def __init__(self, cid, cname, ccost):
        self.cid = cid
        self.cname = cname
        self.ccost = ccost

class Student:
    def __init__(self, sid, sname, address , coursesList):
        self.sid = sid
        self.sname = sname
        self.address = address
        self.coursesList = coursesList

    def getStudentDetails(self):
        print("Student Details")
        print("--------------------")
        print("Student Id       :",self.sid)
        print("Student Name     :",self.sname)
        print()
        print("Student Address Details")
        print("------------------------")
        print("House Number :",self.address.hno)
        print("Street       :",self.address.street)
        print("City          :",self.address.city)
        print("State         :",self.address.state)
        print("Country       :",self.address.country)
        print()
        print("Student Courses Details")
        print("--------------------------")
        print("CID\tCNAME\tCCOST")
        print("-----------------------")
        for course in self.coursesList:
            print(course.cid,end="\t")
            print(course.cname,end="\t")
            print(course.ccost,end="\n")

course1 = Course("C-111", "C", 1000)
course2 = Course("C-222", "C++", 2000)
course3 = Course("C-333", "Java", 5000)
course4 = Course("C-444", "Python", 6000)
coursesList = [course1, course2, course3, course4]

address = Address("202,23/3rt","MG Road", "Hyd",
"Telangana", "India")

std = Student("S-111", "AAA", address, coursesList)

std.getStudentDetails()
```

OP:

```
---
Student Details
---------------------
Student Id       : S-111
Student Name      : AAA

Student Address Details
-------------------------
House Number : 202,23/3rt
Street       : MG Road
City         : Hyd
State        : Telangana
Country      : India

Student Courses Details
---------------------------
CID        CNAME     CCOST
------------------------
C-111      C         1000
C-222      C++       2000
C-333      Java      5000
C-444      Python    6000
```

2. Setter Method Dependency Injection.

If we inject dependent object in an object through setter method then it is called as setter method Dependency

Injection.
EX:

```
class Address:
    def setHno(self, hno):
        self.hno = hno
    def setStreet(self, street):
        self.street = street
    def setCity(self, city):
        self.city = city
    def setState(self, state):
        self.state = state
    def setCountry(self, country):
        self.country = country

    def getHno(self):
        return self.hno
    def getStreet(self):
        return self.street
    def getCity(self):
        return self.city
    def getState(self):
        return self.state
    def getCountry(self):
        return self.country

class Course:
    def setCid(self,cid):
        self.cid = cid
    def setCname(self,cname):
        self.cname = cname
    def setCcost(self,ccost):
        self.ccost = ccost

    def getCid(self):
        return self.cid
    def getCname(self):
        return self.cname
    def getCcost(self):
        return self.ccost

class Student:
    def setSid(self,sid):
        self.sid = sid
    def setSname(self,sname):
        self.sname = sname
    def setAddress(self,address):
        self.address = address
    def setCoursesList(self,coursesList):
        self.coursesList = coursesList

    def getStudentDetails(self):
        print("Student Details")
        print("--------------------")
        print("Student Id        :",self.sid)
        print("Student Name      :",self.sname)
        print()
        print("Student Address Details")
        print("------------------------")
        print("House Number :",self.address.getHno())
        print("Street        :",self.address.getStreet())
        print("City          :",self.address.getCity())
        print("State         :",self.address.getState())
        print("Country       :",self.address.getCountry())
        print()
        print("Student Courses Details")
        print("---------------------------")
        print("CID\tCNAME\tCCOST")
        print("-----------------------")
        for course in self.coursesList:
            print(course.getCid(),end="\t")
            print(course.getCname(),end="\t")
            print(course.getCcost(),end="\n")

course1 = Course()
course1.setCid("C-111")
course1.setCname("JAVA")
course1.setCcost(10000)

course2 = Course()
course2.setCid("C-222")
```

```
course2.setCname("Python")
course2.setCcost(15000)

course3 = Course()
course3.setCid("C-333")
course3.setCname("Oracle")
course3.setCcost(16000)

coursesList = [course1, course2, course3]

address = Address()
address.setHno("202/123")
address.setStreet("MG Road")
address.setCity("Hyd")
address.setState("Telangana")
address.setCountry("India")

std = Student()
std.setSid("S-111")
std.setSname("Durga")
std.setAddress(address)
std.setCoursesList(coursesList)

std.getStudentDetails()
```
OP:
```
---
Student Details
```

```
---------------------
Student Id       : S-111
Student Name     : Durga

Student Address Details
-------------------------
House Number : 202/123
Street       : MG Road
City         : Hyd
State        : Telangana
Country      : India

Student Courses Details
---------------------------
CID       CNAME    CCOST
------------------------
C-111     JAVA      10000
C-222     Python    15000
C-333     Oracle    16000
```

Note: In Python applications, when we have less no of dependencies then it is suggestible to use setter method dependency Injection, If we have more no of dependencies then it is suggestible to use constructor dependency injection.

Note: Constructor dependency injection is not providing more readability, but, setter method dependency injection will provide more readability.

Inheritance In Python:

It is a relation between classes, it will bring variables and methods from one clsas[Parent class / Super class/ Base Class] to another class[chaild class / sub class/ Derived class].

The main advantage of Inheritance is "Code reusability", Declare variables and methods one time in super class then access that variables and methods in any no of times in sub classes.

To specify super classes for a sub class then we have to use () along with sub class name with the super class names specification with , seperator.
Syntax:

```
class ClassName(SuperClass1,
SuperClass2,...SuperClass-n)
{

}
```

EX:

```
class Person:
    def __init__(self,pname,page,paddr):
        self.pname = pname
        self.page = page
        self.paddr = paddr
    def getPersonDetails(self):
        print("Name    :",self.pname)
        print("Age     :",self.page)
        print("Address :",self.paddr)
class Employee(Person):
    def __init__(self,eid,esal,edes,pname,page,paddr):
        self.eid=eid
        self.esal=esal
        self.edes=edes
        self.pname = pname
        self.page = page
        self.paddr = paddr

    def getEmployeeDetails(self):
        print("Employee Details")
        print("-------------------")
        self.getPersonDetails()
        print("EID    :",self.eid)
        print("ESAL   :",self.esal)
        print("EDES   :",self.edes)
emp = Employee("E-111", 10000.0,"Manager", "Durga",
28, "Hyd")
emp.getEmployeeDetails()
```

OP:

```
Employee Details
--------------------
Name    : Durga
Age     : 28
Address : Hyd
EID    : E-111
ESAL   : 10000.0
EDES   : Manager
```

Note: In the above application, we have provided super class instance variables initialization in sub class constructor.

EX:

```
class Person:
    def __init__(self,pname,page,paddr):
        self.pname = pname
        self.page = page
        self.paddr = paddr
    def getPersonDetails(self):
        print("Name    :",self.pname)
        print("Age      :",self.page)
        print("Address :",self.paddr)
class Employee(Person):
    def __init__(self,eid,esal,edes,pname,page,paddr):
        super(Employee, self).__init__(pname,page,paddr)
        self.eid=eid
        self.esal=esal
        self.edes=edes


    def getEmployeeDetails(self):
        print("Employee Details")
        print("--------------------")
        self.getPersonDetails()
        print("EID     :",self.eid)
        print("ESAL    :",self.esal)
        print("EDES    :",self.edes)
emp = Employee("E-111", 10000.0,"Manager", "Durga",
28, "Hyd")
emp.getEmployeeDetails()
```

OP:

```
Employee Details
--------------------
Name     : Durga
Age      : 28
```

```
Address : Hyd
EID    : E-111
ESAL   : 10000.0
EDES   : Manager
```

Note: in the above application, we have provided initialization for super class instance variables by calling super class constructor.

In Python applications, when we create object for sub class then sub class constructor will be executes, here super class constructor will not be executed directly, in this context, if we want to access super class constructor from sub class constructor then we have to use using super().

EX:

```
class A:
    def __init__(self):
        print("Super Class Constructor")
class B(A):
    def __init__(self):
        print("Sub Class Constructor")

b = B()
```

OP:

Sub Class Constructor

EX:

```
class A:
    def __init__(self):
        print("Super Class Constructor")
class B(A):
    def __init__(self):
        super(B, self).__init__()
        print("Sub Class Constructor")

b = B()
```

OP:

```
Super Class Constructor
Sub Class Constructor
```

In Python, if we create sub class object then sub class object id value will be shared to all the self parameters in both sub class and the respective super classes.
EX:

```
class A:
    def __init__(self):
        print("Super Class Constructor :",id(self))
    def m1(self):
        print("m1() :",id(self))
class B(A):
    def __init__(self):
        super(B, self).__init__()
        print("Sub Class Constructor :",id(self))
    def m2(self):
        print("m2() :",id(self))
b1 = B()
print("b1 ref value :",id(b1))
b1.m2()
b1.m1()
print()
b2 = B()
print("b2 ref value :",id(b2))
b2.m2()
b2.m1()
```

OP:

```
C:\Python38\python.exe D:/python10/oops/oops.py
Super Class Constructor : 2143009913296
Sub Class Constructor : 2143009913296
b1 ref value : 2143009913296
m2() : 2143009913296
m1() : 2143009913296
```

```
Super Class Constructor : 2143010014544
Sub Class Constructor : 2143010014544
b2 ref value : 2143010014544
m2() : 2143010014544
m1() : 2143010014544
```

In Python applications, by using super class id variable we are able to access only super class members, we are unable to access sub class own members, but, by using sub class id variable we are able to access both super class members and sub class members.
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m2(self):
        print("m2-B")

a = A()
a.m1()
#a.m2() --> Error
b = B()
b.m1()
b.m2()
```

OP:

```
m1-A
m1-A
m2-B
```

--> Python is not allowing cyclic inheritance, extending same class is called as cyclic inheritance.
EX:

```
class A(A):
    pass
```

Status: Error
EX:

```
class A(B):
    pass
class B(A):
    pass
```
Status: Error

Types of Inheritances:

In Python there are five types of inheritances.
1. Single Inheritance
2. Multiple Inheritance

3. Multi Level inheritance
4. Hierachical Inheritance
5. Hybrid Inheritance

In Python, all the above inheritances are possible.

1. Single Inheritance:

It is a relation between classes, it will bring variables and methods from only one super class to one or more no of sub classes.
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m2(self):
        print("m2-B")

a = A()
a.m1()
b = B()
b.m1()
b.m2()
```

2. Multiple Inheritance:

It is a relation between classes, it will bring variables and methods from more than one super class to one or more no of sub classes.
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B:
    def m2(self):
        print("m2-B")
class C(A,B):
    def m3(self):
        print("m3-C")
        self.m1()
        self.m2()
a = A()
a.m1()
b = B()
b.m2()
c = C()
c.m1()
c.m2()
c.m3()
```

OP:

```
---
m1-A
m2-B
m1-A
m2-B
m3-C
m1-A
m2-B
```

--> In case of multiple inheritance, if we declare same variable with different value and same method with different implementation in both the super classes and if we access that variable and that method in the respective sub class then which super class variable will be accessed and which super class method will be accessed is completly depending on the super classes order which we provided along with sub class.

In the above context, PVM will search for super class variables and methods at the super classes in which order we provided all super classes at sub class declaration.

EX:

```
---
class A:
    i = 10
    def m1(self):
        print("m1-A")
class B:
    i = 20
    def m1(self):
        print("m1-B")
class C(A,B):
    def m2(self):
        print(self.i)
        self.m1()

c = C()
c.m2()
```

OP:

```
10
m1-A
```

EX:

```
class A:
    i = 10
    def m1(self):
        print("m1-A")
class B:
    i = 20
    def m1(self):
        print("m1-B")
class C(B,A):
    def m2(self):
        print(self.i)
        self.m1()

c = C()
c.m2()
```

OP:

```
20
m1-B
```

EX:

```
class A:
    i = 10

class B:
    i = 20
    def m1(self):
        print("m1-B")
class C(A,B):
    def m2(self):
        print(self.i)
        self.m1()
c = C()
c.m2()
```

OP:

```
10
m1-B
```

3. Multi Level inheritance:

It is the combination of single inheritances in more than one level.
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m2(self):
        print("m2-B")
class C(B):
    def m3(self):
        print("m3-C")
a = A()
a.m1()
b = B()
```

```
b.m1()
b.m2()
c = C()
c.m1()
c.m2()
c.m3()
```

OP:

```
m1-A
m1-A
m2-B
m1-A
m2-B
m3-C
```

4. Hierachical Inheritance

It is the combination of single inheritances in a particular structer.
EX:

```
class A:
    def m1(self):
        print("m1-A")

class B(A):
    def m2(self):
        print("m2-B")

class C(A):
    def m3(self):
        print("m3-C")

class D(B):
    def m4(self):
        print("m4-D")

class E(B):
    def m5(self):
        print("m5-E")

class F(C):
    def m6(self):
        print("m6-F")

class G(C):
    def m7(self):
        print("m7-G")

a = A()
a.m1()

b = B()
```

```
b.m1()
b.m2()

c = C()
c.m1()
c.m3()

d = D()
d.m1()
d.m2()
d.m4()

e = E()
e.m1()
e.m2()
e.m5()

f = F()
f.m1()
f.m3()
f.m6()

g = G()
g.m1()
g.m3()
g.m7()
```

OP:

```
m1-A
m1-A
m2-B
m1-A
m3-C
m1-A
m2-B
m4-D
m1-A
m2-B
m5-E
m1-A
m3-C
m6-F
m1-A
m3-C
m7-G
```

5. Hybrid Inheritance

It is the combination of Single inheritance , Hierarchical Inheritance, Multip level and Multiple Inheritance
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m2(self):
        print("m2-B")
class C(A):
    def m3(self):
        print("m3-C")
class D(B,C):
    def m4(self):
        print("m4-D")

a = A()
a.m1()

b = B()
b.m1()
b.m2()

c = C()
c.m1()
c.m3()

d = D()
d.m1()
d.m2()
d.m3()
```

```
d.m4()

OP:
---
m1-A
m1-A
m2-B
m1-A
m3-C
m1-A
m2-B
m3-C
m4-D
```

In case of inheritance, we are able to get the classes tracing part while accessing members of the sub classes by using mro() predefined function.

MRO:

--> MRO algorithm is also known as C3 algorithm.
--> MRO algorithm was praposed by Samuele Pedroni.
--> MRO algorithm follows DLR principle, that is, Depth First Left to Right.
--> In Depth First Left To Right principle, Chaild will get more priority and left   chaild will get more priority.
--> In MRO algorithm, to get Method Resolution Order

we will uyse the following formula.
    MRO(x) = x + Merge(MRO(p1),MRO(p2),.....,parent)
--> In finding MRO, we have to decide Head and tail element at each and every element.
--> If we consider "ABCD" are the elements we find in sub level of MRO, then, A is head   and BCD is tails.
--> To calculate Merge elements, we have to use the following algorithms.
    1. Take Head element of first list.
    2. If Head is not available in tail part of other lists then add this Head to   result and remove it from the list in Merge.
    3. If Head is existed in the tail part of any other list then consider the head   element of the next list and continue the same.

--> In Python, to calculate list by using MRO, Python has provided a predefined method   like mro().
     print(ClassName.mro())

EX:

```
class A:
   pass
class B(A):
    pass
class C(A):
    pass
class D(B,C):
    pass

print(D.mro())
```

OP:

```
[<class '__main__.D'>, <class '__main__.B'>, <class
'__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

EX:

```
class A:
   pass
class B(A):
    pass
class C(A):
    pass
class D(C,B):
    pass

print(D.mro())
```

OP:

```
[<class '__main__.D'>, <class '__main__.C'>, <class
'__main__.B'>, <class '__main__.A'>, <class 'object'>]
```

EX:

```
class A:
    pass
class B:
    pass
class C:
    pass
class X(A,B):
    pass
class Y(B,C):
    pass
class P(X,Y,C):
    pass
print(P.mro())
```

OP:

```
[<class '__main__.P'>, <class '__main__.X'>, <class
'__main__.A'>, <class '__main__.Y'>, <class
'__main__.B'>, <class '__main__.C'>, <class 'object'>]
```

EX:

```
class A:
    pass
```

```
class B:
    pass
class C:
    pass
class X(B,A):
    pass
class Y(C,B):
    pass
class P(X,Y,C):
    pass
print(P.mro())
```

OP:

```
[<class '__main__.P'>, <class '__main__.X'>, <class
'__main__.Y'>, <class '__main__.C'>, <class
'__main__.B'>, <class '__main__.A'>, <class 'object'>]
```

EX:

```
class D:
    pass
class E:
    pass
class F:
    pass
class B(D,E):
    pass
```

```
class C(D,F):
    pass
class A(B,C):
    pass

print(A.mro())
```

OP:

```
[<class '__main__.A'>, <class '__main__.B'>, <class
'__main__.C'>, <class '__main__.D'>, <class
'__main__.E'>, <class '__main__.F'>, <class 'object'>]
```

EX:

```
class D:
    pass
class E:
    pass
class F:
    pass
class B(E,D):
    pass
class C(F,D):
    pass
class A(B,C):
    pass
```

```
print(A.mro())
```

OP:

```
[<class '__main__.A'>, <class '__main__.B'>, <class
'__main__.E'>, <class '__main__.C'>, <class
'__main__.F'>, <class '__main__.D'>, <class 'object'>]
```

super() Method:
----
super() is an inbuilt function in Python, it can be used to perform the following actions.

1. To refer super class constructor.
2. To refer super class method.
3. Torefer super class variables.

1. To refer super class constructor.

If we want to refer super class constructor from sub class constructor then we have to use the following syntax.

```
   super().__init__()
```

EX:

```
class A:
    def __init__(self):
        print("A-Con")
class B(A):
    def __init__(self):
        print("B-Con")
        super().__init__()

b = B()
```

OP:

```
B-Con
A-Con
```

In Python, it is possible to access super class constructor from sub class normal methods by using super() method.
ex:

```
class A:
    def __init__(self):
        print("A-Con")
class B(A):
    def __init__(self):
        print("B-Con")
    def m1(self):
        print("m1-B")
        super().__init__()
b = B()
b.m1()
```
OP:

```
B-Con
m1-B
A-Con
```

2. To refer super class method:

If we want to refer super class methods from sub classes then we have to use below syntax.

```
super().methodName([ParamList])
```

EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m2(self):
        print("m2-B")
        super().m1()
```

```
b = B()
b.m2()
```

OP:

```
m2-B
m1-A
```

3. To refer super class variables.

To refer super class variable by using super() method we will use the following syntax.

```
  super().varName
```

EX:

```
class A:
    i = 10
class B(A):
    def m1(self):
        print("m1-A")
        print(super().i)
b = B()
b.m1()
```
OP:

```
m1-A
```

```
10
```

In Python applications, by using super() method we are able to access only static variabnles of the parent class from sub class, it is not possible to access instance variables of the parent class from sub classes.
EX:

```
class A:
    i = 10
    def __init__(self):
        self.j = 20

class B(A):
    def m1(self):
        print("m1-A")
        print(super().i)
        #print(super().j) ---> Error
b = B()
b.m1()
```

In Python applications, by using super() method we are able to access static variable, instance method, static method and class method of parent class from sub class constructor.
EX:

```
---
class A:
    i = 10
    def m1(self):
        print("Instance Method")

    @staticmethod
    def m2():
        print("Static Method")

    @classmethod
    def m3(cls):
        print("Class Method")
class B(A):
    def __init__(self):
        print("B-Con")
        print(super().i)
        super().m1()
        super().m2()
        super().m3()
b = B()
```

OP:

```
---
B-Con
10
Instance Method
```

Static Method
Class Method

In Python applications, we are able to access parent class static variable, instance method, static method, class method from sub class instance method by using super() method.
EX:

```
class A:
    i = 10
    def m1(self):
        print("Instance Method")

    @staticmethod
    def m2():
        print("Static Method")

    @classmethod
    def m3(cls):
        print("Class Method")
class B(A):
    def m1(self):
        print("B-Con")
        print(super().i)
        super().m1()
        super().m2()
        super().m3()
b = B()
b.m1()
```

```
OP:
---
B-Con
10
Instance Method
Static Method
Class Method
```

In Python applications, we are unable use to super() function in sub class static method.
EX:

```
class A:
    i = 10
    def m1(self):
        print("Instance Method")

    @staticmethod
    def m2():
        print("Static Method")

    @classmethod
    def m3(cls):
        print("Class Method")
class B(A):
    @staticmethod
    def meth():
        print("B-meth")
        #print(super(B).i) --> Error
        #super().m1() --> Error
        #super().m2() --> Error
        #super().m3() ---> Error
        A.m2() ---> Valid
B.meth()
OP:
---
B-meth
Static Method
```

In Python applications, by using super() method , we are able to access super class static variable, instance method, static method and class method from sub class class method . To access super class instance method from sub class class method by using super() then we have to pass cls parameter to instance method of parent class.

EX:

```
class A:
    i = 10
    def m1(self):
        print("Instance Method")

    @staticmethod
    def m2():
        print("Static Method")

    @classmethod
    def m3(cls):
        print("Class Method")
class B(A):
    @classmethod
    def meth(cls):
        print("B-class-method")
        print(super().i)
        super().m1(cls)
        super().m2()
        super().m3()

B.meth()
```

OP:

```
---
B-class-method
10
Instance Method
Static Method
```

Class Method

In Python applications, of we want to access a particular super class method then we have to use class className directly or super() method.

1. A.m1(self): It will access A class m1() method will be executed.
2. super(SubClassName,self).methodName()

EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m1(self):
        print("m1-B")
class C(B):
    def m1(self):
        print("m1-C")
class D(C):
    def m1(self):
        print("m1-D")
        B.m1(self)
        super().m1()
        super(C,self).m1()
        super(B,self).m1()
d = D()
d.m1()
```

OP:

```
m1-D
m1-B
m1-C
m1-B
m1-A
```

In the above example, if we use super(C,self).m1() then PVM will access B class m1() method, because, class B is super class to class C.

Polymorphism:

Polymorphism is a Greak word, where Poly means Many and Morphism means Structers or forms.

If one thing is existed in more than one form then it is called as Polymorphism.

The main advantage of Polymorphism is "Flexibility" to develop applications.

In general, we are able to achieve Polymorphism in the following three ways.

1.Duck Typing Philosophy of Polymorphism.
2. Overloading
3. Overriding

1.Duck Typing Philosophy of Polymorphism.

In Python, we are unable to specify the types explicitly, based on the provided value explicitly we will use find type at runtime, so, Pythos is Dynamically Typed Programming language.

EX:

```
def f1(obj):
 obj.talk()
```

In the above example, what is the type of obj?, we are unable to decide initially, at runtime we are able to pass any type of obj and we are able to decide its type like below.

At runtime, if it walks like duch and talks like duck then it will be duct, Python follows this at principle and it is called as Duck Typing Philosophy of Python.

EX:

```
class Duck:
    def talk(self):
        print("Quack..Quack...")


class Dog:
    def talk(self):
        print("Bow Bow....")


class Cat:
    def talk(self):
        print("Moew Moew,...")


class Goat:
    def talk(self):
        print("Myaa Myaa..")


def fn(obj):
    obj.talk()


list = [Duck(), Dog(), Cat(), Goat()]
```

```
for x in list:
    fn(x)
```

```
OP:
----
Quack..Quack...
Bow Bow....
Moew Moew,...
Myaa Myaa..
```

In the above approach, if talk() method is not available in the provided object then PVM will raise an error like Attribute Error.

EX:

```
class Duck:
    def talk(self):
        print("Quack..Quack...")
class Dog:
    def bark(self):
        print("Bow Bow....")
def fn(obj):
    obj.talk()

list = [Duck(), Dog()]
for x in list:
    fn(x)
```

Status: AttributeError: 'Dog' object has no attribute 'talk'

To overcome the above problem we have to check whether the provided attribute is existed or not in the provided object, for this, we will use a predefined function like hasattr()
EX:

```
class Duck:
    def talk(self):
        print("Quack..Quack...")
class Dog:
    def bark(self):
        print("Bow Bow....")
def fn(obj):
    if hasattr(obj, 'talk'):
        obj.talk()
    else:
        obj.bark()

list = [Duck(), Dog()]
for x in list:
    fn(x)
```
OP:

```
Quack..Quack...
Bow Bow....
```

2. Overloading:

The process of extending existed functionality upto some new functionality is called as Overloading.

In general , there are three types of overloadings in Object Orientation.
1. Constructor Overloading
2. Method Overloading
3. Operator Overloading

1. Constructor Overloading

If we declare more than one Constructor with the same name and with the different parameter list then it is called as Constructor overloading.

Python does not support Constructor overloading, if we provide more than one constructor with the same name and with the different parameter list then PVM will consider only the last provided constructor while creating objects.
EX:

```
class Employee:
    def __init__(self):
        print("0-arg-con")
    def __init__(self, eno):
        print("1-arg-con")
    def __init__(self,eno, ename):
        print("0-arg-con")
emp = Employee()
```

Status: TypeError: __init__() missing 2 required positional arguments: 'eno' and 'ename'

2. Method Overloading:

If we declare more than one method with the same name and with different parameter list then it is called as Method overloading.

Python does not support Method Overloading, if we provide more than one method with the same name and with different parameter list then PVM will consider only the last provided method when we access that method.

EX:

```
class A:
    def add(self, i):
        print("1-param-add :",i)
    def add(self, i,j):
        print("2-param-add :",i,j)
    def add(self, i,j,k):
        print("3-param-add :",i,j,k)
a = A()
a.add(10)
OP:
---
Status: TypeError: add() missing 2 required positional
arguments: 'j' and 'k'
```

In Python, we are able to achieve Constructor overloading and method overloading by using either default arguments or variable length arguments.
EX:

```
class A:
    def __init__(self, i=0, j=0, k=0):
        self.i = i
        self.j = j
        self.k = k
    def add(self):
        print("ADD  :",self.i+self.j+self.k)
a1 = A()
a1.add()
a2 = A(10)
a2.add()
```

```
a3 = A(10,20)
a3.add()
a4 = A(10,20,30)
a4.add()
```

OP:

```
ADD  : 0
ADD  : 10
ADD  : 30
ADD  : 60
```

EX:

```
class A:
    def add(self,*n):
        print("ADD  :",sum(n))
a = A()
a.add()
a.add(10)
a.add(10,20)
a.add(10,20,30)
```
OP:

```
ADD  : 0
ADD  : 10
ADD  : 30
```

ADD : 60

3. Operator Overloading

If we declare any operator with more than one functionality then it is called as Operator Overloading.
EX:

+ ---> Arithmetic Addition and String concatination
* ---> Arithmetic Multiplication and Repeat Concatination operation
% ---> Arithmetic Modulo operation and Formatted output

In Python, every operator has its own Magic method internally and it will be executed when we use that respective operator in python applications.
EX:

+ ------> object.__add__(self,other)
- ------> object.__sub__(self,other)
* ------> Object.__mul__(self,other)
/ ------> Object.__div__(self,other)
% ------> Object.__mod__(self,other)
**------> Object.__pow__(self,other)
//------> Object.floordiv__(self,other)
+=------> Object.__iadd__(self,other)

-=------> Object.__isub__(self,other)
*=------> Object.__imul__(self,other)
/=------> Object.__idiv__(self,other)
%=------> Object.__imod__(self,other)
**=------> Object.__ipow__(self,other)
//=------> Object.__ifloordiv__(self,other)
==------> Object.__eq__(self,other)
!=------> Object.__nq__(self,other)
<------> Object.__lt__(self,other)
>------> Object.__gt__(self,other)
<=------> Object.__le__(self,other)
>=------> Object.__ge__(self,other)

If we want to perform operator overloadig in pytho applications then we have to override the respective magic method as per our requirement.

In general, we will use + operator between numebric values, it is not possible to use + operator between objects, if we use + operator between objects then we will get an error.

EX:

```
class Book:
    def __init__(self, pages):
        self.pages = pages
b1 = Book(100)
b2 = Book(200)
```

```
b = b1 + b2
print(b)
```

Status: TypeError: unsupported operand type(s) for +: 'Book' and 'Book'

If we overload + operator with new functionality like to add two objects then we will get any error.

EX:

```
class Book:
    def __init__(self, pages):
        self.pages = pages

    def __add__(self, other):
        return self.pages + other.pages

b1 = Book(100)
b2 = Book(200)
b = b1 + b2
print(b)
```

OP:

```
300
```

EX:

```
class Student:
    def __init__(self,sid,  smarks):
        self.sid = sid
        self.smarks = smarks
    def __eq__(self, other):
        if self.smarks == other.smarks:
            return "Students are Equal at Education"
        else:
            return "Students are not equals at Eduction"
std1 = Student("S-111", 78)
std2 = Student("S-222", 78)
print(std1 == std2)

std3 = Student("S-333", 68)
std4 = Student("S-444", 79)
print(std3 == std4)
```

EX:

```
Students are Equal at Education
Students are not equals at Eduction
```

EX:

```
class Student:
    def __init__(self,sid, sname, saddr, smarks):
        self.sid = sid
        self.sname = sname
        self.saddr = saddr
        self.smarks = smarks
    def __lt__(self, other):
        return self.smarks < other.smarks
    def getStdDetails(self):

print(self.sid,"\t",self.sname,"\t",self.saddr,"\t",self.smar
ks)

std1 = Student("S-111", "AAA", "Hyd", 78)
std2 = Student("S-222", "BBB", "Hyd", 88)
std3 = Student("S-333", "CCC", "Hyd", 67)
std4 = Student("S-444", "DDD", "Hyd", 96)
std5 = Student("S-555", "EEE", "Hyd", 58)
stdList = [std1,std2,std3,std4,std5]

while True:
    count = 0
    for x in range(0,len(stdList)-1):
        if stdList[x] < stdList[x+1]:
            pass
        else:
            count = count + 1
            temp = stdList[x]
            stdList[x] = stdList[x+1]
            stdList[x+1] = temp
    if count == 0:
        break
    else:
        continue

print("SID\tSNAME\tSADDR\tSMARKS")
print("----------------------------------")
for std in stdList:
    std.getStdDetails()
```

EX:

```
import sys
class Employee:
    sortingKey = "ENO"
    def __init__(self,eno, ename, esal, eaddr ):
        self.eno = eno
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr
    def __lt__(self, other):
        if self.sortingKey == "ENO":
            return self.eno < other.eno
        elif self.sortingKey == "ENAME":
            return self.ename < other.ename
        elif self.sortingKey == "ESAL":
            return self.esal < other.esal
        elif self.sortingKey == "EADDR":
            return self.eaddr < other.eaddr
        else:
            pass
emp1 = Employee(111, 'Durga', 5000, 'Hyd')
emp2 = Employee(222, 'Kish', 8000, 'Chennai')
emp3 = Employee(333, "Anil", 6000, 'Pune')
emp4 = Employee(444, "Raju", 7000, "Delhi")
emp5 = Employee(555, "Sonu", 9000, "Mumbai")

def sort(list):
    while True:
        count = 0
        for x in range(0, len(list) - 1):
            if list[x] < list[x + 1]:
                pass
            else:
                count = count + 1
                temp = list[x]
                list[x] = list[x + 1]
                list[x + 1] = temp
        if count == 0:
            break
        else:
            continue

list = [emp1, emp2, emp3, emp4, emp5]
```

```
while True:
    print("1.Sorting By ENO")
    print("2.Sorting By ENAME")
    print("3.Sorting By ESAL")
    print("4.Sorting By EADDR")
    print("5.EXIT")
    option = int(input("Your Option : "))
    if option == 1:
        Employee.sortingKey = "ENO"
        sort(list)
    elif option == 2:
        Employee.sortingKey = "ENAME"
        sort(list)
    elif option == 3:
        Employee.sortingKey = "ESAL"
        sort(list)
    elif option == 4:
        Employee.sortingKey = "EADDR"
        sort(list)
    elif option == 5:
        sys.exit()
    else:
        print("Provide the number from 1 to 5")
    print("ENO","ENAME","ESAL","EADDR")
    print("------------------------------")
    for emp in list:
        print(emp.eno,emp.ename,emp.esal,emp.eaddr)
```

    print()

Overridding:

In Python, Overridding is the process providing replacement for the existed funfctionality.

Therer are two types of Overriddings in python.
1. Constructor Overridding.
2. Method Overridding.

1. Constructor Overridding.

In general, when we provide inheritance relation between two classes and when we create object for sub class then Super class constructor will come to sub class , where at sub class, super class constructor will be overriden with sub class constructor.

To perform Constructor overridding in python applications we have to declare constructors at both super class and sub class explicitly and create object for sub class, where we will get output from sub class constructor.

In the above constext, if sub class constructor is not existed then PVM will execute super class constructor as

super class constructor is inherited to sub class.
EX:

```
class A:
    def __init__(self):
        print("A-Con")
class B(A):
   pass
b = B()
```
OP:

```
A-Con
```

EX:

```
class A:
    def __init__(self):
        print("A-Con")
class B(A):
  def __init__(self):
      print("B-Con")
b = B()
```
OP:

```
B-Con
```

In the above example, when we define inheritance relation betweem class A and class B , PVM will bring Super class[Class A] constructor to sub class, in this context, if PVM is identifying sub class constructor explicitly then PVM will override super class constructor with sub class constructor and PVM will execute sub class constructor as per sub class object creation.

2. Method Overriding:

In general, in python applications, when we define inheritance relation between two classes , PVM will bring super class methods to sub class internally, in this context, if PVM identifies any super class method at sub class then explicitly then PVM will override super class method with sub class method, in this context, if we acess that method then PVM will execute sub class method only, not super class method.
EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    def m1(self):
        print("m1-B")
b = B()
b.m1()
```

OP:

```
m1-B
```

In the above program, if m1() method is not existed in sub class then PVM will execute super class m1() method directly.

EX:

```
class A:
    def m1(self):
        print("m1-A")
class B(A):
    pass
b = B()
b.m1()
```
OP:

```
m1-A
```

EX:

```
class Account:
    def getMinBal(self):
      return 10000
```

```
class EmployeeAccount(Account):
    def getMinBal(self):
        return 5000

class StudentAccount(Account):
    def getMinBal(self):
        return 0

class LoanAccount(Account):
    pass

empAccount = EmployeeAccount()
print("Employee Account Min Bal :
",empAccount.getMinBal())
stdAccount = StudentAccount()
print("Student Account Min Bal  :
",stdAccount.getMinBal())
loanAccount = LoanAccount()
print("Loan Account Min Bal     :
",loanAccount.getMinBal())
```

OP:

```
---
Employee Account Min Bal :  5000
Student Account Min Bal  :  0
Loan Account Min Bal     :  10000
```

Abstract Method:

In Python, if we declare any method with out implementation then that method is called as an abstract method.

To declare abstract methods, we have to use @abstractmethod decorator for the abstract method.
EX:

```
@abstrasctmethod
def m1(self):
   pass
```

--> In the case of abstract methods, if we declare a method with @abstractmethod decorator in normal class and if we provide body for the abstarct method then PVM will treate that method as normal Python method, not as abstract method.
EX:

```
from abc import *
class A:
    @abstractmethod
    def m1(self):
        print("m1-A")
```

```
a = A()
a.m1()
```
OP:

```
m1-A
```

Abstract class:

In general, in Python applications, if any class is extended from ABC class from abc module then that class is treated an abstract class.
EX:

```
from abc import *
class A(ABC):
  pass
```

In Pythgon applications, if we declare any abstract class with abstract methods then we must provide implementation for abstract methods by taking a sub class.
EX:

```
from abc import *
class A(ABC):
    @abstractmethod
    def m1(self):
        pass
    @abstractmethod
    def m2(self):
        pass
    @abstractmethod
    def m3(self):
       pass
class B(A):
    def m1(self):
        print("m1-B")
    def m2(self):
        print("m2-B")
    def m3(self):
        print("m3-B")
b = B()
b.m1()
b.m2()
b.m3()
OP:
---
m1-B
m2-B
m3-B
```

In abstract clases, we are able to provide both concreate methods and abstract methods.

EX:

```
from abc import *
class A(ABC):
    @abstractmethod
    def m1(self):
        pass
    @abstractmethod
    def m2(self):
        pass
    def m3(self):
        print("m3-A")
class B(A):
    def m1(self):
        print("m1-B")
    def m2(self):
        print("m2-B")
b = B()
b.m1()
b.m2()
b.m3()
```
OP:

```
m1-B
m2-B
m3-A
```

--> In Python applications, It is possible to declare an abstract class with out abstract methods , in this case, we can create object for abstract class and we can access concreate methods by using abstract class object reference variable if any concreate methods are existed , but, if any abstract method is existed in abstract class then it is not possible to create object for abstract class.
EX:

```
from abc import *
class A(ABC):
    def m1(self):
        print("m1-A")
    def m2(self):
        print("m2-A")
    def m3(self):
        print("m3-A")
a = A()
a.m1()
a.m2()
a.m3()
```
OP:

```
m1-A
m2-A
m3-A
```

EX:

```
from abc import *
class A(ABC):
    @abstractmethod
    def m1(self):
        print("m1-A")
    def m2(self):
        print("m2-A")
    def m3(self):
        print("m3-A")
a = A()
a.m1()
a.m2()
a.m3()
```

Status: TypeError: Can't instantiate abstract class A with abstract methods m1

In Python, if we declare abstract class with abstract method and if we have not provided implementation for abstract method in the sub class then sub class is converted as abstract class automatically, where Object is not possible for both super abstract class and sub class.
EX:

```
from abc import *
class A(ABC):
    @abstractmethod
    def m1(self):
        print("m1-A")

class B(A):
    pass
b = B()
b.m1()
Status: TypeError: Can't instantiate abstract class B with
abstract methods m1
```

Note: IN pythoin applications, if we want to declare methods and if we want to give option to some other to provide implementation for methods then we have to declare methods as abstract methods and classes as abstract classes.

EX:

```
from abc import *
class Account(ABC):
    def getMinBal(self):
        pass
class SavingsAccount:
    def getMinBal(self):
        return 10000
class CurrentAccount:
    def getMinBal(self):
        return 50000

sa = SavingsAccount()
print("Savings Account Min Bal :",sa.getMinBal())
ca = CurrentAccount()
print("Current Account Min Bal :",ca.getMinBal())
```
OP:

```
Savings Account Min Bal : 10000
Current Account Min Bal : 50000
```

Interface:

In Python, if any abstract class is declared with only abstract methods then that abstract class is called as an interface.
EX:

```
from abc import *
class FordCar(ABC):
    @abstractmethod
    def getCarType(self):
        pass
    @abstractmethod
    def getSeatingCapacity(self):
        pass

class FordFiesta(FordCar):
    def getCarType(self):
        print("Ford Fiesta Model 2012")
    def getSeatingCapacity(self):
        print("3 Passangers +  Driver")

class EchoSports(FordCar):
    def getCarType(self):
        print("Ford EchoSports Model 2015")
    def getSeatingCapacity(self):
        print("4 Passangers +  Driver")

class Endeavour(FordCar):
    def getCarType(self):
        print("Ford Endeavour Model 2017")
    def getSeatingCapacity(self):
        print("7 Passangers +  Driver")

fiesta = FordFiesta()
fiesta.getCarType()
fiesta.getSeatingCapacity()
print()
```

```
echosports = EchoSports()
echosports.getCarType()
echosports.getSeatingCapacity()
print()
endeavour = Endeavour()
endeavour.getCarType()
endeavour.getSeatingCapacity()
```

```
OP:
---
Ford Fiesta Model 2012
3 Passangers +  Driver

Ford EchoSports Model 2015
4 Passangers +  Driver

Ford Endeavour Model 2017
7 Passangers +  Driver
```

globals()[name] function in Python:

In Python applications, if we take any input by using input() function then that input values will be returned in the form of String only, here if we want to get inputy values in int, float,.... then we have to perform type casting by using predefined functions like int(),

float(),....

In the above context, if we want to convert any string data as class name then we have to use globals()[name] predefined function.
EX:

```
from abc import *
class Fruit(ABC):
    @abstractmethod
    def getPrice(self):
        pass

class Apple(Fruit):
    def getPrice(self):
        return 200

class Banana(Fruit):
    def getPrice(self):
        return 50

class Guava(Fruit):
    def getPrice(self):
        return 70

for x in range(0,3):
    className = input("Enter Class Name : ")
    obj = globals()[className]()
    print(className,"1 Dozen :",obj.getPrice())
```

OP:

```
Enter Class Name : Apple
Apple 1 Dozen : 200
Enter Class Name : Guava
Guava 1 Dozen : 70
Enter Class Name : Banana
Banana 1 Dozen : 50
```

importance of __str__() function in Python

In general, in Python applications, if we pass any object reference variable as parameter to print() function then PVM will execute __str__(self) function internally .

__str__() function was provided in Object class initially, it can was implemented in such a way to return a string contains "<__main__.Student object at Hex_Decimal_Val>".
EX:

```
class Student:
    def __init__(self,sid,sname,saddr):
        self.sid = sid
        self.sname = sname
        self.saddr = saddr

std = Student("S-111", "Durga", "Hyd")
print(std)
OP:
---
<__main__.Student object at 0x000001FE44C14520>
```

In Python applications, when we pass object reference variable as parameter to print() function if we want to display our own details instead of object reference value then we have to override __str__() function.

EX:

```
class Student:
    def __init__(self,sid,sname,saddr):
        self.sid = sid
        self.sname = sname
        self.saddr = saddr
    def __str__(self):
        print("Student Details")
        print("---------------------")
        print("Student Id       :",self.sid)
        print("Student Name     :",self.sname)
        print("Student Address  :",self.saddr)
        return ""
std = Student("S-111", "Durga", "Hyd")
print(std)
```

```
OP:
---
Student Details
----------------------
Student Id       : S-111
Student Name     : Durga
Student Address  : Hyd
```

Declaring private ,protected and public attributes in Python:

In Python there are three scopes for the attributes.
1. private
2. protected
3. public

In Python, bydefault, every attribute is public, we can access public variables from anywere in our application.
EX: name = "Durga"

In Python applications, we are able to declare protected attributes by usisng _ [Single Underscor]symbol and it is

having scope upto the present class and in the chaild classes.
EX: _name = "Durga"
Note: Protected scope is a convention, it is not existed really in python.

In Python applications, we are able to declare private attributes by using __ [Double underscore] symbols and it is having scope upto the present class.
EX: __name = "Durga"

EX:

```
class A:
    i = 10
    _j = 20
    __k = 30
    def m1(self):
        print(A.i)
        print(A._j)
        print(A.__k)
a = A()
a.m1()
print(A.i)
print(A._j)
#print(A.__k)---> Type Error: type object 'A' has no
attribute '__k'
```

OP:

```
10
20
30
10
20
```

Exception Handling:

Q)What is the difference between Error and Exception?

Ans:

Error is a problem , it will not allow to run Python applications.

There are two types of errrors in Python.

1. Basic Errors / Fundamental Errors
2. Runtime Errors

1. Basic Errors / Fundamental Errors

There are three types of basic Errors.
1. Lexical Errors:

---> Mistakes in lexemes or in tokens
EX1:

```
if i == 10:
  print("i value is 10")
```
Status: Valid
EX2:

```
fi i == 10:
  print("i value is 10")
```

Status: Invalid

2. Syntax Errors:

Mistakes in Python syntaxes.
EX1:
----
i = 10 --> Valid

EX2:
----
i 10 = ----> Invalid

3. Semantic Errors:

Performing operations with incompatible types.

EX1:
```
i = 10
j = 20
k = i + j
Status: valid
```

EX:2

```
i = 10
str = "abc"
result = i - str
Status: Invalid
```

2. Runtime Errors/Explicit Errors:

These errors occurred at runtime which are not having any solution programatically.
EX: Insufficient Main memory
    PVM Internal Problems
    Unavailability of IO Components.

Exception:

Exception is an unexpected event occurred at Runtime, it may be provided by the users while entering dynamic input in Python programs, it may be provided by Database Engine while executing sql queries in PDBC

applications, It may be provided by the Network when we establish connection between client and Server in distributed applications,.... causes abnormal termination to the applications.

In Python, there are two types of Terminations are existed.
1. Smooth Terminatrion
2. Abnormal Termination

1. Smooth Terminatrion:
---> If any program is terminated at the end of code then that termination is called as Smooth Termination.

2. Abnormal Termination:

--> If any Program is terminated in middle of the program then that termination is called as Abnormal Termination

In general, in applications execution, abnormal terminations may provide the followong peoblems
1. It may crash local OS.
2. It may provide Hanged out situation for Network based applications.
3. It may collaps the database which we are using in Python applications

  -----

To overcome the above problems we have to handle excerptions properly, for this, we have to use a set of mechanisms explicitly called as Exception Handling mechanisms.

In Python, teher are two types of Exceptions are existed.

1. Predefined Exceptions
2. User Defined Exceptions

1. Predefined Exceptions:

These Exceptions are defined by Python programming language.

In Python, all the Errors and Exceptions are sub types to BaseException either directly or indirectly.

In general, almost all the Exceptions are sub classes to Exception class.

To handle Exceptions in python applications, we have to use try-except-else-finally block
Syntax:

```
try:
   ---instructions---
except ExceptionName:
   ---instructions----
else:
   ---instructions----
finally:
   ---instructions----
```

try block:

--> It include as set of instructions which may raise exceptions
--> It will include doubtfull code to get an exception, where doubtfull code may or may   not generate exception.
--> If any exceptyion is generated in try block then PVM will bypass flow of execution   to except block then finally block by skipping remaining instructons in ry block.
--> if no exception is identified in try block then PVM will execute the comllete try   block, at the end of try block PVM will bypass flow of execution to else block.

except block:

--> Its main intention is to catch exception from try

block and to provide exception details on console.
--> Along with except keyword we have to provide Exception name and we have to defined an alias name[Reference variable] to Exception name.
--> In Python applications, if any exception is raised in try block there PVM will execute except block, if no exception is raised in try block then PVM will not execute except block.

EX:

```
try:
    a = 100
    b = 0
    f = a / b
except ZeroDivisionError as e:
    print(e)
```

If we want to provide except block to handle any type of exception then we have to use default except block.
EX:

```
try:
    a = 100
    b = 0
    f = a / b
except:
    print("Exception in try")
```

else block:

--> It will be executed when no exceptions are generated in try block, it is an alternative to except block.
--> If any exception is identified in try block then PVM will execute except block only, PVM will not execute else block.

EX:

```
try:
    print("Inside try block")
except:
    print("in side except block")
else:
    print("Inside else block")
```
OP:

```
Inside try block
Inside else block
```

EX:

```
try:
    print("Inside try block")
    a = 100 / 0
except:
    print("In side except block")
else:
    print("Inside else block")
```

OP:

```
Inside try block
In side except block
```

finally block:

--> finally block is able to include a set of instructions which must be executed   irrespective of executing try block, except block and else block.
--> In general, in Python applications, we will use resources , we will create resources   inside try block and we will close these resources in side finally, because, finally   block is giving guarantee for execution irrespective getting exceptions in try   block.
EX:

```
try:
    print("Inside try block")
except:
    print("In side except block")
```

```
else:
    print("Inside else block")
finally:
    print("Inside finally block")
OP:
---
Inside try block
Inside else block
Inside finally block

EX:
---
try:
    print("Inside try block")
    a = 100 / 0

except:
    print("In side except block")
else:
    print("Inside else block")
finally:
    print("Inside finally block")
OP:
---
Inside try block
In side except block
Inside finally block
```

EX:

```
print("Before try")
try:
    print("Inside try , Before Exception")
    a = 100 / 0
    print("Inside try, After Exception")
except:
    print("In side except block")
else:
    print("Inside else block")
finally:
    print("Inside finally block")
print("After finally block")
```
OP:

```
Before try
Inside try , Before Exception
In side except block
Inside finally block
After finally block
```

EX:

```
print("Before try")
try:
    print("Inside try")
except:
    print("In side except")
else:
    print("Inside else")
finally:
    print("Inside finally")
print("After finally")
```

OP:

```
Before try
Inside try
Inside else
Inside finally
After finally
```

Q)Is it possible to write try block with out except block?

Ans:

Yes, it is possible to write try block with out except block , but, we must provide finally block.
EX:

```
try:
    print("Inside try")
```

```
finally:
    print("Inside finally")
```
OP:

```
Inside try
Inside finally
```

EX:

```
try:
    print("Inside try")
    a = 100 /0
finally:
    print("Inside finally")
```

OP:

```
Inside try
Inside finally
ZeroDivisionError: division by zero
```

Q)Is it possible to provide try block with out finally block?

Yes, it is possible to provide try block with out finally block, but, we must provide except block.
EX:

```
try:
    print("Inside try")
    a = 100 /0
except :
    print("Inside except")
```

OP:

```
Inside try
Inside except
```

Q)Is it possible to provide try-except-finally in side try block, inside except block and inside finally block?

Ans:

Yes, It is possible to provide try-excepti-finally block inside try block, inside except block and inside finally block.

EX:

```
try:
    print("Inside try")
    try:
        print("Inside nested try")
    except:
        print("Inside nested except")
    finally:
        print("Inside nested finally")
except :
    print("Inside except")
finally:
    print("Inside finally")
```

OP:

```
Inside try
Inside nested try
Inside nested finally
Inside finally
```

EX:

```
try:
    print("Inside try")
    a = 100 / 0
except :
    print("Inside except")
    try:
        print("Inside nested try")
    except:
        print("Inside nested except")
    finally:
        print("Inside nested finally")
finally:
    print("Inside finally")
OP:
----
Inside try
Inside except
Inside nested try
Inside nested finally
Inside finally
EX:
---
try:
    print("Inside try")
    a = 100 / 0
except :
    print("Inside except")
finally:
    print("Inside finally")
    try:
        print("Inside nested try")
    except:
        print("Inside nested except")
    finally:
        print("Inside nested finally")

OP:
---
Inside try
Inside except
Inside finally
Inside nested try
Inside nested finally
```

In Python applications, we are able to provide more than one except block for a single try block, where if we provide any default except block then it must be provided as last statement .

EX:

```
try:
    print("Inside try")
    a = 100 / 0
except ZeroDivisionError as e:
    print(e)
except AttributeError as e:
    print(e)
except NameError as e:
    print(e)
except:
    print("Unknown Error")
```
OP:

```
Inside try
division by zero
```

2. User Defined Exceptions

These exceptions are defined by the developers as per their applpication requirements.

To prepare User defined exceptions we have to use the following steps.
1. Define User defined Exception class.
2. Raise and handle User defined Exception.

EX:

```
class InsufficientFundsException(Exception):
    def __init__(self, exceptionDescription):
        self.exceptionDescription = exceptionDescription

class Transaction:
    def __init__(self, accNo, accHolderName, accType,
balance):
            self.accNo = accNo
            self.accHolderName = accHolderName
            self.accType = accType
            self.balance = balance
    def withdraw(self, wdAmt):
        print("Transaction Details")
        print("-----------------------")
        print("Account Number       :", self.accNo)
        print("Account Holder Name :",
self.accHolderName)
        print("Account Type          :",self.accType)
        print("Transaction Type     : WITHDRAW")
        print("Withdraw Amount      :",wdAmt)
        try:
            if wdAmt > self.balance:
                print("Total Balance        :", self.balance)
                print("Transaction Status  : FAILURE")
                raise InsufficientFundsException("Reasone :
Funds are not Sufficient in Your Account")
            else:
                self.balance = self.balance - wdAmt
                print("Total Balance        :", self.balance)
                print("Transaction Status  : SUCCESS")
        except InsufficientFundsException as ex:
            print(ex)
        finally:
            print("******ThanQ, Visit Again********")

tx1 = Transaction("abc123", "Durga", "Savings", 10000)
tx1.withdraw(5000)
print()
tx1 = Transaction("xyz123", "Anil", "Savings", 10000)
tx1.withdraw(15000)
```

OP:
```
---
Transaction Details
------------------------
Account Number      : abc123
Account Holder Name : Durga
Account Type        : Savings
Transaction Type    : WITHDRAW
Withdraw Amount     : 5000
Total Balance       : 5000
Transaction Status  : SUCCESS
******ThanQ, Visit Again********

Transaction Details
------------------------
Account Number      : xyz123
Account Holder Name : Anil
Account Type        : Savings
Transaction Type    : WITHDRAW
Withdraw Amount     : 15000
Total Balance       : 10000
Transaction Status  : FAILURE
Reasone : Funds are not Sufficient in Your Account
******ThanQ, Visit Again********
```

Packages:

Def1: Package is the collection related modules and sub packages.
Def2: Package is a folder contains .py files[Modules] , sub folders[Sub Packages] and a special file named as __init__.py
Def3: If any folder contains __init__.py file then that folder is treated as Python package.

Python packages are providing the following advantages.
1. Modularity
2. Abstraction
3. Security
4. Sharability
5. Reusability

1. Modularity:

In general, in python application development, by using packages we are able to provide the logical seperation over the complete requirement set, here each and every logical seperation is called as application module, hewre to represent application level module we will use python package, hence, Python packages are able to provide Modularity in application development.

2. Abstraction:

In python applications, if we declare modules or sub packages ,... in a package then package is able to hide its internal declarations , therefore , Package are able to improve "Abstraction".

3. Security:

In Python applications, Python packages are able to improve both encapsulation and abstraction , so these encapsulation and abstraction are able to provide "Security" in Python applications.

Encapsulation + Abstraction = Security

4. Sharability:

In Python applications, if we define a package with no of modules and sub packages then we are able to share that single copy of package to any no applications at time, so that, Package are able to improve Sharability.

5.Reusability:

In Python applications, if we define a package with no of modules one time then we are able to reuse that package any no of times either in the same application

or in other applications, therefore, Packaages are able to improve reusability.

There are two types of packages.
1. Predefined Packages
2. User Defined Packages.

1. Predefined Packages:

These packages are defined by Python programming language and which are provided along with Prothon Software.
EX: TKinter, Http, Html,Email.....

Note: All predefined packages are existed under "Lib" folder under Python Software.

2. User Defined Packages:

These packages are defined by the developers as per their application requirements.

In Python, no predefined syntax or instruction is existed to create a package, to create packages in python we have to create folders manually and we have to prepare __init__.py files at each and every folder.

In Python applications, to access members of a particular package we have to use import statement.

Case-1: If the required modules are existed at current location where our python file is existed then we are able to import the modules of that package directly and we are able to use the members of that module.

Case-2: If the required packages are not existed at current location, existed at different location where our python file is existed then we have to append the package location to the present python file by using the following instruction.

```
    import sys
    sys.path.append("---Package Location---")
```

In the above two cases , to use module members in the present python file we have to use the following syntaxes.

1. import moduleName
--> It imports all members of the specified module , but, to access module members we have to use moduleName.

2. from moduleName import memberName
--> It able to import only the specified member from the

specified module, but, to access the member, it is not required use moduleName.

Example-1:

D:\python10\apps\app1

```
pack1
|---- __init__.py
|---- welcome.py
|---- test.py
```

welcome.py

```
def sayWelcome(name):
            print("Welcome ",name,"!")
```

test.py

```
import welcome
welcome.sayWelcome("Durga")
```

test1.py

```
from welcome import *
sayWelcome("Durga")
```

```
D:\python10\apps\app1\pack1>py test.py
Welcome  Durga !

D:\python10\apps\app1\pack1>py test1.py
Welcome  Durga !
```

Example-2:

D:\abc

```
std
|---__init__.py
|---student.py
```

D:\python10\apps\app2
----
test.py

student.py

```
sid = "S-111"
sname = "Durga"
squal = "BTech"
saddr = "Hyd"
def getStudentDetails():
            print("Student Details")
```

```
        print("-------------------------")
        print("Student Id   :",sid)
        print("Student Name             :",sname)
        print("Student Qual :",squal)
        print("Student Addr :",saddr)
```

```
test.py
--------
import sys
sys.path.append("D:/abc/std")
from student import *
getStudentDetails()
```

```
D:\python10\apps\app2>py test.py
Student Details
-------------------------
Student Id   : S-111
Student Name    : Durga
Student Qual : BTech
Student Addr : Hyd
```

Example-3:

```
C:\abc
-------
emp
```

```
|---- __init__.py
|---- employee.py

E:\xyz
------
acc
|--- __init__.py
|---account.py

D:\python10\apps\app3
----------------------
test.py

employee.py
------------
eno = 111
ename = "Durga"
esal = 50000
eaddr = "Hyd"
def getEmployeeDetails():
            print("Employee Details")
            print("--------------------");
            print("Employee Number    : ",eno)
            print("Employee Name      : ",ename)
            print("Employee Salary    : ",esal)
            print("Employee Address   : ",eaddr)
```

```
account.py
-----------
accNo = "abc123"
accHolderName = "Durga"
accType = "Savings"
accBalance = 50000
def getAccountDetails():
            print("Account Details")
            print("--------------------")
            print("Account Number        : ",accNo)
            print("Account Holder Name   : 
",accHolderName)
            print("Account Type          : ",accType)
            print("Account Balance       : ",accBalance)

test.py
--------
import sys
sys.path.append("C:/abc/emp")
sys.path.append("E:/xyz/acc")
import employee
from account import *
employee.getEmployeeDetails()
print()
getAccountDetails()
```

```
D:\python10\apps\app3>py test.py
Employee Details
--------------------
Employee Number    :  111
Employee Name      :  Durga
Employee Salary    :  50000
Employee Address   :  Hyd

Account Details
---------------------
Account Number        :  abc123
Account Holder Name   :  Durga
Account Type          :  Savings
Account Balance       :  50000
```

Example-4:[PyCharm]

```
packagesapp
|---cust
|    |---- __init__.py
|    |----customer_module
|
|---prd
|    |---- __init__.py
|    |----product_module.py
|---test.py
```

```
customer_module.py
-----------------
class Customer:
    def __init__(self,cid,cname,caddr):
        self.cid = cid
        self.cname = cname
        self.caddr = caddr

    def getCustomerDetails(self):
        print("Customer Details")
        print("-----------------------")
        print("Customer Id        :",self.cid)
        print("Customer Name      :",self.cname)
        print("Customer Address  :",self.caddr)

product_module.py
-----------------
class Product:
    def __init__(self,pid,pname,pcost):
        self.pid = pid
        self.pname = pname
        self.pcost = pcost

    def getProductDetails(self):
        print("Product Details")
        print("---------------------")
        print("Product Id     :",self.pid)
        print("Product Name    :",self.pname)
        print("Product Cost    :",self.pcost)

test.py
--------
from cust.customer_module import Customer
from prd.product_module import Product

cust =  Customer("C-111", "AAA", "Hyd")
cust.getCustomerDetails()
print()
prd = Product("P-111", "BBBB", 5000)
prd.getProductDetails()

OP:
----
Customer Details
------------------------
Customer Id        : C-111
Customer Name      : AAA
Customer Address   : Hyd

Product Details
----------------------
Product Id      : P-111
Product Name    : BBBB
Product Cost    : 5000
```

File Handling:

In enterprise application development , to manage data about the org we need some storage areas.

There are two type of Storage areas.
1. Temporary Storage Areas.
2. Permanent Storage Areas.

1. Temporary Storage Areas:
--> These storage areas are able to store data temporarily.
EX: Bufferes, Python Objects

2. Permanent Storage Areas
--> These storage areas are able to store data permanently.
EX: File System, DBMS, Datawarehouses

File System:

It is a parmanent Storage Area, it ablle to manage data permanently and it is managed by local Operating system.

If we want to use files in python applications we have to use the following steps.

1. Create file
2. Write Data in Files / Read data from Files
3. Close File.

1. Create File:
   file = open(fileName, mode)
   EX: file = open("E:/abc/xyz/welcome.txt","w")
   Note: If use "name" attribute from file we are able to get absolute path of the file.
      print(file.name)
2. Get Name of the File:
   name = os.path.basename(file.name)

3. Get Absolute path of the FIle:
   print(file.name)
   print(os.path.abspath(file.name))

4. Get Parent location :
   os.path.dirname(file.name)

5. To get Mode of the file.
   print(file.mode)

6. To check readable file
   print(file.readable())

7. To check Writable
```
   print(file.writable())
```
EX:

```
import os
file = open("E:/abc/xyz/hello.txt","w")
print("File Mode   :",file.mode)
print("File Name   :",os.path.basename(file.name))
print("File Dir    :",os.path.dirname(file.name))
print("Abs Path    :",file.name)
print("Abs Path    :",os.path.abspath(file.name))
print("Is Readable :",file.readable())
print("Is Writable :",file.writable())
```
OP:

```
File Mode   : w
File Name   : hello.txt
File Dir    : E:/abc/xyz
Abs Path    : E:/abc/xyz/hello.txt
Abs Path    : E:\abc\xyz\hello.txt
Is Readable : False
Is Writable : True
```

If we want to Write Data in a File we have to use the following two functions.

1. write()
2. writelines()

Q)What is the difference between write() and writelines() function?

Ans:
----
write() function is able to write single line of data.
EX:

```
import os
file = open("E:/abc/xyz/hello.txt","w")
str = "Welcome To Durgasoft"
file.write(str)
print("data Send to E:/abc/xyz/hello.txt")
```

writelines() function is able to write more than one line which are existed in list to a file.

EX:

```
import os
file = open("E:/abc/xyz/hello.txt","w")
str1 = "Welcome To Durgasoft!\n"
str2 = "Hello User!\n"
str3 = "Hai User!\n"
```

```
list = [str1,str2,str3]
file.writelines(list)
print("data Send to E:/abc/xyz/hello.txt")
```

Bydefault, files are existed in overriden mode, that is, at each and every write operation previous data will be overridden with new data. If we want to append new data to the old data which is existed in file previously we have to use 'a' as filemode.
EX:

```
import os
file = open("E:/abc/xyz/hello.txt","a")
file.write(" Ameerpet")
```

```
abc.txt
--------
Durga Software Solutions Ameerpet
```

If we want to read data from a particular file then we havhe to use the following methods.

1. read()
2. readline()
3. readlines()
4. read(no_Of_Characters)

1. read(): It can be used to read the complete data

which is existed in the specified file.
EX:

```
file = open("E:/abc/xyz/hello.txt","r")
data = file.read()
print(data)
```
OP:

```
Durga
Software
Solutions
Ameerpet
```

2.readline(): It able to read only one line of data in the form of String.
EX:

```
file = open("E:/abc/xyz/hello.txt","r")
data = file.readline()
print(data)
```

OP:

```
Durga
```

3. readlines():It able to read data in the form multiple lines in a list.

EX:

```
file = open("E:/abc/xyz/hello.txt","r")
list = file.readlines()
print(list)
for data in list:
    print(data,end="")
```
OP:

```
['Durga \n', 'Software \n', 'Solutions\n', 'Ameerpet\n']
Durga
Software
Solutions
Ameerpet
```

4. read(no_Of_Characters):It able to read the specified no of characters from the file.
EX:

```
file = open("E:/abc/xyz/hello.txt","r")
data = file.read(15)
print(data)
```
OP:

```
Durga Software
```

In File System , we are able to use the following modes to perform operations with files.

'r' --> only read operation, not write operation
'w' --> Only Write operation with override, not read operation
'a' --> Append new data to the old data.
'r+'-->Read and Write, but, it will override the existed content upto the required part.
'a+'--> Append and Read operations
'w+'--> Write then read operation.
'x' --> It will create file, if file is existed then it will generate error.

In Python applications, if we create any file then it is convention to close that file, for closing files we will use close() funtion.

To Check whether a FILE is closed or not we have to use "closed" variable from File.
EX:

```
file = open("E:/abc/xyz/hello.txt","w+")
file.write("Welcome To Durgasoft")
file.seek(0)
print(file.read())
print(file.closed)
file.close()
```

```
print(file.closed)
```

OP:

```
Welcome To Durgasoft
False
True
```

In general, in python applications, of we open the file then we have to close that file at the end of python application explicitly, in this context, developers may or may not close the files explicitly, if we want to close the files with out failure then we have to go for an alternative where PVM must close the files at the end of appliactions. To achieve this rejquired we have to create file with 'with' keyword.
EX:

```
with open("E:/abc/xyz/hello.txt","w+") as file:
    file.write("Welcome To Durgasoft")
    file.seek(0)
    print(file.read())
    print("File Closed? :",file.closed)
print("File Closed? :",file.closed)
```

OP:

```
Welcome To Durgasoft
File Closed? : False
File Closed? : True
```

To get current location of the file pointer we will use tell() function.
To move file pointer to a particular location we will use seek(position) function.
EX:

```
file = open("E:/abc/xyz/hello.txt","r")
print(file.tell())
file.seek(8)
print(file.tell())
print(file.read(13))
print(file.tell())
```
OP:

```
0
8
To Durgasoft
22
```

Q)Write a Python program to take file name as dynamic input and to display file content?

Ans:

```
fileName = input("Enter File Name : ")
file = open(fileName,'r')
data = file.read()
print(data)
```
OP:

```
File Name  : D:\Python4\test.py
sid = input("Student Id    : ")
sname = input("Student Name  : ")
sage = int(input("Student Age  : "))
squal = eval(input("Student Qualifications in List : "))
ssubjects = eval(input("Student Subjects in Tuple : "))
stechs = eval(input("Student Technologies in set : "))
ssubjects_and_marks = eval(input("Student Subjects
and Marks in Dict  : "))
print()
print("Student Details")
print("---------------------")
print("Student Id                    :",sid)
print("Student Name                  :",sname)
print("Student Age                   :",sage)
print("Student Qual                  :",squal)
print("Student Subjects              :",ssubjects)
print("Student Techs                 :",stechs)
print("Student Subjects and Marks
```

:",ssubjects_and_marks)

Q)Write a python program to take file name as dynamic input and to display no of lines, no of words and no of characters of the respective file?

Ans:

```
fileName = input("Enter File Name : ")
file = open(fileName,'r')
#data = file.read()
linesCount = 0
wordsCount = 0
charsCount = 0

for line in file:
    linesCount = linesCount + 1
    wordsCount = wordsCount + len(line.split())
    for word in line.split():
        charsCount = charsCount + len(word)
print("No of Lines :",linesCount)
print("No of Words :",wordsCount)
print("No of Chars :",charsCount)
```

OP:

```
----
Enter File Name : E:/abc/xyz/hello.txt
No of Lines : 3
No of Words : 11
No of Chars : 65
```

EX:

```
fileName = input("Enter File Name : ")
file = open(fileName,'r')
data = file.read()
linesCount = 0
wordsCount = 0
charsCount = 0
linesCount = data.count("\n")
wordsCount = len(data.split())
charsCount = len(data)
print("Lines Count :",linesCount)
print("Words Count :",wordsCount)
print("Chars Count :",charsCount)
```

OP:

```
Enter File Name : E:/abc/xyz/hello.txt
Lines Count : 3
Words Count : 11
Chars Count : 76
```

Q)Write a python program to transfer data from one file to another file?

Ans:

```
sourceFile = open("E:/abc/xyz/welcome.txt",'r')
targetFile = open("D:/abc/xyz/welcome_new.txt",'w')
data = sourceFile.read()
targetFile.write(data)
print("Data transfered from E:/abc/xyz/welcome.txt to
D:/abc/xyz/welcome_new.txt")
```

If we want to work with binary data like images,...then we have to open the files with the file modes like 'rb', 'wb',.....

Q)Write a Python program to transfer image data from one file to another file?

```
sourceFile = open("E:/images/Java_Python.jpg",'rb')
targetFile = open("D:/images/banner.jpg",'wb')
bytes = sourceFile.read()
targetFile.write(bytes)
print("Image Data Transfered from
E:/images/Java_Python.jpg to D:/images/banner.jpg")
sourceFile.close()
targetFile.close()
```

CSV Files:

CSV --> Coma Seperated Values

CSV files are able to manage semi structered data, to manipulate CSV files data python has provided a seperate module in the form of 'csv'.

If we want to write data into CSV files, first we have to get Writer object from file then we have to write data to CSV file through Writer object.
EX

```
f = open(fileName,'w')
writer = csv.writer(f)
writer.writerow([Col1,col2....])
```

If we want to get data from CSV file then we have to get reader object from the source file by using reader() method.

Write Data To CSV File:

EX1:

```
import csv
targetFile =
open("E:/documents/emp.csv",'w',newline='')
writer = csv.writer(targetFile)
writer.writerow(["ENO", "ENAME", "ESAL", "EADDR"])
writer.writerow([111, "AAA", 5000, "Hyd"])
writer.writerow([222, "BBB", 6000, "Hyd"])
writer.writerow([333, "CCC", 7000, "Hyd"])
writer.writerow([444, "DDD", 8000, "Hyd"])
print("Data written to E:/documents/emp.csv")
targetFile.close()
```

Note: If we write data in CVS file, bydefault, data will be stored in CSV file with a line space between rows, if we want to remove line spaces between rows we have to use 'newline' attribute in open() function.

EX2:

```
import csv
targetFile =
open("E:/documents/student.csv",'w',newline='')
writer = csv.writer(targetFile)
writer.writerow(["SID", "SNAME", "SQUAL", "SADDR",
"SEMAIL", "SMOBILE"])
while True:
    sid = input("Student ID     : ")
    sname = input("Student Name   : ")
    squal = input("Student Qual   : ")
    saddr = input("Student Addr   : ")
    semail = input("Student Email  : ")
    smobile = input("Student Mobile : ")

writer.writerow([sid,sname,squal,saddr,semail,smobile])
    print("Student",sid,"Inserted Successfully")
    option = input("Onemore Student[yes/no]?  : ")
    if option == "yes":
        continue
    else:
        break

targetFile.close()
```

Reading Data from CSV:

```
import csv
sourceFile = open("E:/documents/emp.csv",'r')
reader = csv.reader(sourceFile)
data = list(reader)
for row in data:
    for column in row:
        print(column,"\t",end="")
    print()
```

```
sourceFile.close()
```

EX:

```
import csv
sourceFile = open("E:/documents/student.csv",'r')
reader = csv.reader(sourceFile)
data = list(reader)
for column in data[0]:
    if column == "SEMAIL":
        print(column,"\t\t",end="")
    else:
        print(column,"\t",end=" ")
print()
for row in data[1::]:
    for column in row:
        print(column,"\t",end="")
    print()
sourceFile.close()
```

Creating ZIP file with no of files:

Zipping : Gatherring all files and folder from the current location.
Unzipping: Unpacking all the files and folders from Zip file to current location.

Advantages:
1. SImplifies Files transfermation from one location to location.
2. Optimize Memory utilization

```
from zipfile import *

zip_File =
ZipFile("D:/images/images.zip",'w',ZIP_DEFLATED)
zip_File.write("E:/images/Java_Python.jpg")
zip_File.write("E:/images/Nagoor.jpg")
zip_File.write("E:/images/NagoorBabu.jpg")
zip_File.write("E:/images/my_Image.jpg")
zip_File.close()
print("Zip file is created Successfully")
```

EX:

```
import os
from zipfile import *

f1 = open("E:/abc/welcome.txt",'w')
f1.write("Welcome User!")
f2 = open("E:/abc/hello.txt",'w')
f2.write("Hello User!")
f3 = open("E:/abc/hai.txt",'w')
```

```
f3.write("Hi User!")

zip_File =
ZipFile("E:/documents/docs.zip",'w',ZIP_DEFLATED)
zip_File.write("E:/abc/welcome.txt")
zip_File.write("E:/abc/hello.txt")
zip_File.write("E:/abc/hai.txt")
zip_File.close()
print("Zip file is created Successfully")
```

Creating ZIP File, Extracting ZIP File and Displaying Data from ZIP File:

```
------------------------------------------------------------------------
-----
from zipfile import *
zip_File1 =
ZipFile("E:/documents/docs.zip",'w',ZIP_DEFLATED)
zip_File1.write("E:/abc/welcome.txt")
zip_File1.write("E:/abc/hello.txt")
zip_File1.write("E:/abc/hai.txt")
zip_File1.close()

zip_File2 =
ZipFile("E:/documents/docs.zip",'r',ZIP_STORED)
zip_File2.extract("abc/welcome.txt",path="E:/documents
")
zip_File2.extract("abc/hello.txt",path="E:/documents")
```

```
zip_File2.extract("abc/hai.txt",path="E:/documents")
zip_File2.close()

f1 = open("E:/documents/abc/welcome.txt",'r')
print(f1.read())
f2 = open("E:/documents/abc/hello.txt",'r')
print(f2.read())
f3 = open("E:/documents/abc/hai.txt",'r')
print(f3.read())
```

Pickling and Unpickling:

The process of seperating data from an object is Pickling.
The process reconstructing an object on the basis of data is called as Unpickling.

To perform Pickling and Unpickling Python has provided predefined library in thye form of 'pickle' module.

To perform pickling we have to use dump() function from pickle module.
Toperform unpickloing we have to use load() function from pickle module.

Example on Pickling:

```
import pickle
class Employee:
    def __init__(self,eno,ename,esal,eaddr):
        self.eno = eno
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr
    def getEmpDetails(self):
        print("Employee Details")
        print("----------------------")
        print("Employee Number     :",self.eno)
        print("Employee Name       :",self.ename)
        print("Employee Salary     :",self.esal)
        print("Employee Address    :",self.eaddr)
f = open("E:/abc/xyz/emp.txt","wb")
emp1 = Employee(111,"AAA",5000,"Hyd")
print("Employee Details Before Pickling")
emp1.getEmpDetails()
pickle.dump(emp1,f)
print("Pickling is perfomred Successfully")
```

Ex On Unpickling:

```
import pickle
class Employee:
    def __init__(self,eno,ename,esal,eaddr):
        self.eno = eno
        self.ename = ename
        self.esal = esal
        self.eaddr = eaddr
    def getEmpDetails(self):
        print("Employee Details")
        print("----------------------")
        print("Employee Number    :",self.eno)
        print("Employee Name      :",self.ename)
        print("Employee Salary    :",self.esal)
        print("Employee Address   :",self.eaddr)
f2 = open("E:/abc/xyz/emp.txt","rb")
obj = pickle.load(f2)
print("Employee Details After Unpickling")
obj.getEmpDetails()
```

Manipulations with Directories:

--> To get Current Working directory we have to use getcwd() function fron 'os' module.
EX:
```
import os
dir = os.getcwd()
print(dir)
```

OP:

```
D:\Python4\helloproject
```

--> To create a directory we have to use mkdir() function from os module.
EX:

```
import os
os.mkdir("E:/abc/xyz/emp")
```

--> To create Multiple directories in nested fashion we will use makedirs() function.

EX:

```
import os
os.makedirs("E:/abc/xyz/emp/Employee")
```

--> To remove a directory from the specified location we will use rmdir() function.
EX:

```
import os
os.rmdir("E:/abc/xyz/emp/Employee")
```

--> To delete more multiple directories we will use removedirs() function.
EX:

```
import os
os.removedirs("D:/abc/xyz/emp")
```

--> To rename a directory we will use rename() function:
EX:

```
import os
os.rename("E:/abc/xyz/emp","E:/abc/xyz/student")
```

--> To rename multiple directories we will use renames() function.
EX:

```
import os
os.renames("D:/abc/xyz/emp","D:/a/x/customer")
```

--> To get the content of a particular location we will use listdir() function.
EX:

```
import os
list = os.listdir("D:/java7")
for item in list:
    print(item)
```

Note: listdir() is able to provide only files and folders of the specified location with out nested folders.

--> If we want to get files and folders including nested folders we will use walk() function.
EX:

```
import os
list = os.walk("D:/java7")
for item in list:
    print(item)
```

EX:

```
import os
f =
os.walk("E:/abc",topdown=True,onerror=None,followlink
s=False)
for dirpath, dirnames, filenames  in f:
    print("Dir Path  :",dirpath)
    print("dir names :",dirnames)
    print("filenames :",filenames)
    print("------------------------------------")
```

OP:

```
Dir Path  : E:/abc
dir names : ['xyz']
```

```
filenames : []
---------------------------------------
Dir Path  : E:/abc\xyz
dir names : []
filenames : ['emp.txt', 'hello.txt', 'welcome.txt']
---------------------------------------
```

Multi Threadding:

Q)What is the difference between process , procedure and processor?

Ans:

Procedure is a set of instructions to represent a particular task.

Process is a flow of execution to execute a set of instruction inorder to perform a particular task.

processor is an H/W component to generate processes.

There are two models are existed to execute applications.
1. Single Process Model or Single Tasking

2. Multi Process Model or Multi Tasking

1. Single Process Model or Single Tasking:

--> It able to allow only one process at a time to execute application
--> It able to allow sequential execution in our application
--> It able to increase application execution time
--> It able to reduce application performance.

2. Multi Process Model or Multi Tasking

--> It allows more than one process to execute applications.
--> It follows parallel execution.
--> It reduces application execution time.
--> Its improves application performance.

In general, To execute Python program, We need a flow of execution from PVM , here the flow of execution generated by PVM is called as Thread. In Python Software, PVM is a program, to execute PVM program, Operating System has to assign a flow of execution called as Process.

Q)What is the difference between process and Thread?

Ans:

Process is heavy weight, to handle process, System has to consume more no of resources, that is, more memory and execution time, It will reduce application performance.
Note: In general, Processes are managed by Operating System.

Thread is Light Weight, to handle threads, System has to consume less no of resources, that is , less memory and less execution time, it will increase application perfomrance.
Note: In general, Threads are managed by Python Software or PVM.

There are two thread models to execute applications.
1. Single Thread Model
2. Multi Thread Model

1. Single Thread Model:

--> It able to allow only one thread to execute application
--> It able to allow sequential execution in our application

--> It able to increase application execution time
--> It able to reduce application performance.

2. Multi Thread Model

--> It able to allow more than one thread to execute applications.
--> It allows parallel execution in our applications.
--> It able to reduce application execution time.
--> It able to increase application perfomance.

Python is able to follow Multi Thread Model, because, Python is able to provide very good environment to create and execute more than one thread.

To create threads and to execute threads, Python has provided seperate predefined module in the form of "threading" module.

To create Threads, "threading" module has provided a predefined class in the form of "Thread".

Q)What is Thread and in how many ways we are able to create Threads in Python?

Ans:

Thread is flow of execution to perform a particular task.

As per the predefined library, there are two ways to create threads.
1. By Extending Thread class.
2. With out Extending Thread class.

1. By Extending Thread class:
----
1. Declare an user defined class as chaild class to Thread class.
2. Override Thread class run() method.
3. Provide application logic in run() method which we want to execute by creating new	Thread.
4. Create Object for User defined Thread class.
5. Access Thread class start() method by using user defined thread class reference	variable.
EX:

```
from threading import Thread
import time
class WelcomeThread(Thread):
    def run(self):
        for x in range(0,10):
            time.sleep(1)
            print("Welcome Thread :",x)
```

```
wt = WelcomeThread()
wt.start()
for x in range(0,10):
    time.sleep(1)
    print("Main Thread ",x)
OP:
---
Main Thread  0
Welcome Thread : 0
Main Thread  1
Welcome Thread : 1
Main Thread  2
Welcome Thread : 2
Main Thread  3
Welcome Thread : 3
Main Thread  4
Welcome Thread : 4
Main Thread  5
Welcome Thread : 5
Main Thread  6
Welcome Thread : 6
Main Thread  7
Welcome Thread : 7
Main Thread  8
Welcome Thread : 8
Main Thread  9
```

```
Welcome Thread : 9
```

Internal Flow:
----
1. When we execute the above application, PVM will create a thread to execute python application called as "Main Thread".

2. When Main Thread encounter start() method then start() method will create a new thread and start() method will bypass new thread to User defined thread class provided run() method.

3. In the above context, User Thread executes run() method implementation and Main Thread executes remaining part of the Python file parallely, so the above python application is able to generate mixed output.

Drawbacks: In the above approach, we must extend Thread class and we must override run() method, it is not possible to replace run() method with any other method.

2. With out Extending Thread class.
----
1. Declare a function with application logic which we

want to execute by creating new thread.
2. Create Thread class object with "target" attribute in Thread class constructor.
   Note: To the target attribute we must provide method name which we want to execute by creating new thread.
3. Access start() method by using Thread class reference variable.

EX:

```
import time
from threading import Thread

def welcome():
    for x in range(0,10):
        time.sleep(1)
        print("User Thread Says Welcome",x)

t = Thread(target=welcome)
t.start()
for x in range(0,10):
    time.sleep(1)
    print("Main Thread Says Hello",x)
```
OP:

```
Main Thread Says Hello 0
```

```
User Thread Says Welcome 0
Main Thread Says Hello 1
User Thread Says Welcome 1
Main Thread Says Hello 2
User Thread Says Welcome 2
Main Thread Says Hello 3
User Thread Says Welcome 3
Main Thread Says Hello 4
User Thread Says Welcome 4
Main Thread Says Hello 5
User Thread Says Welcome 5
Main Thread Says Hello 6
User Thread Says Welcome 6
Main Thread Says Hello 7
User Thread Says Welcome 7
Main Thread Says Hello 8
User Thread Says Welcome 8
Main Thread Says Hello 9
User Thread Says Welcome 9
```

Internal Flow:
----

1. When we execute the above application, PVM will create a thread called as Main Thread   to execute the above python file.
2. When Mian thread encounter start() method, start() method will create new thread and   start() method will

bypass new thread to a method which we specify along with "target" attribute.
3. In the above context, User thread executes user defined function and main thread executes remaining part of Python file parallely, it will provide mixed output.

In Python applications, we arre able to create thread and we are able to submit that thread to a method which is available in a particular class.
EX:

```
import time
from threading import Thread

class Task:
    def doTask(self):
        for x in range(0,10):
            time.sleep(1)
            print("User Thread :",x)
t = Task();
t = Thread(target=t.doTask)
t.start()
for x in range(0,10):
    time.sleep(1)
    print("Main Thread :",x)
```

In Python applications, if the target method contains explicit parameters then we are able to pass that parameter values by using "args" attribute in Thread class constructer.
EX:

```
import time
from threading import Thread

class Bank:
    def displayCustmerNames(self,customerNames):
        for customerName in customerNames:
            time.sleep(1)
            print(customerName)

bank = Bank()
t =
Thread(target=bank.displayCustmerNames,args=(["AAA
", "BBB", "CCC", "DDD","EEE", "FFF"],))
t.start()
```

OP:

```
AAA
BBB
CCC
DDD
```

EEE
FFF

EX:

```
import time as tm
from threading import Thread
def sayWelcome():
    for x in range(0,10):
        tm.sleep(1)
        print("Welcome User!")
def sayHello():
    for x in range(0,10):
        tm.sleep(1)
        print("Hello User!")
def sayHai():
    for x in range(0,10):
        tm.sleep(1)
        print("Hi User!")
t1 = Thread(target=sayWelcome)
t2 = Thread(target=sayHello)
t3 = Thread(target=sayHai)
t1.start()
t2.start()
t3.start()
```

IN Python applications, whern we create a thread

automatiocally an identity number will be create that is called as Thread Identity Number, to access Thread Identity number we have to use "ident" attribute oin Thread reference.
EX:

```
from threading import Thread
t1 = Thread()
t2 = Thread()
t3 = Thread()

t1.start()
t2.start()
t3.start()

print("t1 Identity :",t1.ident)
print("t2 Identity :",t2.ident)
print("t3 Identity :",t3.ident)
```

OP:

```
t1 Identity : 16104
t2 Identity : 16516
t3 Identity : 16884
```

IN Python applications, each and every thread has its own internal name in the form of Thread-1, Thread2,.....

, If we want to get Name of the thread we have to use getName() method and if we want to set a particular name to the thread we have to use setName().
EX:

```
from threading import Thread

t = Thread()
print(t.getName())
t.setName("AAA")
print(t.getName())
```
OP:

```
Thread-1
AAA
```

IN Python applications, it is possible to get currently executed thread by using current_thread() function.
EX:

```
from threading import Thread, current_thread
import time

def fn():
    for x in range(0,10):
        time.sleep(2)
        print(current_thread().getName())
```

```
t1 = Thread(target=fn)
t2 = Thread(target=fn)
t3 = Thread(target=fn)
t1.start()
t2.start()
t3.start()
```

OP:

```
---
Thread-1
Thread-2
Thread-3
Thread-1
Threead-2
Thread-3
---
---
```

In Python applications, we are able to get the no of threads which are active in present python applications by using active_count().

EX:

```
---
from threading import Thread, current_thread,
active_count
import time
```

```
def fn():
    for x in range(0,10):
        time.sleep(2)
t1 = Thread(target=fn)
t2 = Thread(target=fn)
t3 = Thread(target=fn)
t1.start()
t2.start()
t3.start()
print(active_count())
```

OP:

```
4
```

In Python applications, we are able to check whether thread is in live or not by using is_Alive() function.
EX:

```
from threading import *
import time
def fn():
    time.sleep(1)
t = Thread(target=fn)
print(t.is_alive())
t.start()
print(t.is_alive())
```

OP:

```
False
True
```

Note: In general, Thread is in live when we access start() method, before acessing start() method thread is not in live.

IN Python applications, we are able to get all the active threads references in a list by using enumerate() function.
EX:

```
from threading import *
import time
def fn():
    time.sleep(1)
t1 = Thread(target=fn)
t2 = Thread(target=fn)
t3 = Thread(target=fn)
t1.start()
t2.start()
t3.start()
list = enumerate()
for thread in list:
    print(thread.getName())
```

OP:

```
MainThread
Thread-1
Thread-2
Thread-3
```

In Python applications, if we want to pause a thread to complete other thread and if we want to contnue the paused thread after completion of the other thread we have to use join() method.

EX:

```
from threading import *
def wish(name):
    for x in range(0,10):
        print("Hello",name,"!")

t = Thread(target=wish, args=("Durga",))
t.start()
t.join()
for x in range(0,10):
    print("Hai Durga")
```

OP:

```
from threading import *
def wish(name):
    for x in range(0,10):
        print("Hello",name,"!")

t = Thread(target=wish, args=("Durga",))
t.start()
t.join()
for x in range(0,10):
    print("Hai Durga")
```

OP:

```
---
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hello Durga !
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
Hai Durga
```

Daemon Thread:

Daemon thread is a thread, it will be executed internally and it will provide some services to some other thread and these threads are terminated automatically along with the thread which is taking services.

In Python applications, to make a thread as daemon thread we have to access
setDaemon(True) method on thread reference variable.

Note: If we want to access setDaemon(True) method on a thread reference then we must access it before starting thread only, not after starting thread. If we access setDaemon(True) method after calling start() method then PVM is able to provide an error like "RuntimeError: cannot set daemon status of active thread".

To check whether a thread is daemon thread or not we have to use isDaemon() method.
EX:

```
from threading import *
class GarbageCollectorThread(Thread):
    def run(self):
        while True:
            print("Garbage Collector Thread")

gct = GarbageCollectorThread()
gct.setDaemon(True)
gct.start()
#gct.setDaemon(True)
for x in range(0,10):
    print("PVM Thread")
print(gct.isDaemon())
```

OP:

```
Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread
Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadPVM Thread

Garbage Collector ThreadTrue

Garbage Collector Thread
```

Synchronization:

In Multi Threadding, if we execute more than one thread on single data item or on single Program then that threads are called as Concurrent threads and that process is called as Threads concurrency.

IN Threads concurrentcy, when more than one thrtead is accessing data from single object ,then, there may be a chance to get Data Inconsistency, here to preserve data consistency we have to use "Synchronization".

Synchronization is a mechanism, it able to allow only one thread at a time to access data , it will not allow more than one thread at a time, after completion of the present thread only other threads are allowed.

IN Python , Synchronization is running on the top of Locking Mechanisms only.

In Python applications, we are able to achieve Synchronization in the following three ways.

1. Lock [Simple Lock]
2. RLock[Re-Entrent Lock]
3. Semaphores

1. Synchronization through Simple Lock:

If any thread aqcuire lock then that thread is eligible to execute program.
Once the program is executed by a thread which has Lock already then that thread must release Lock. After releasing Lock by a Thread then the available lock will be assigned to another thread.

To reprepsent Simple Lock in Python Applications,

Python has provided a predefined class that is Lock
EX: l = Lock()

To acquire a lock we have to use a function like acquire().
To release lock we have to use a function like release().
EX:

```
from threading import current_thread, Thread, Lock

l = Lock()
def wish():
    l.acquire()
    for x in range(0,10):
        print(current_thread().getName())
    l.release()
t1 = Thread(target=wish)
t1.setName("AAA Thread")
t2 = Thread(target=wish)
t2.setName("BBB Thread")
t3 = Thread(target=wish)
t3.setName("CCC Thread")

t1.start()
t2.start()
t3.start()
```

Drawback:
----
1. It is not supporting for acquiring lock by same thread again and again.
2. It is not suitable for Recursive functions.
EX:

```
from threading import Thread, Lock

l = Lock()
def factorial(n):
    l.acquire()
    result = n
    if n == 0:
        result = 1
    else:
        result = result * factorial(n-1)
    l.release()
    return result

def fn(n):
    print("Factorial of ",n,"is ",factorial(n))

t1 = Thread(target=fn, args=(5,))
t2 = Thread(target=fn, args=(6,))
t1.start()
t2.start()
```

OP:

No Output

Synchronization through RLock:

RLock is called as Re-Entrent Lock, it is very much suitable for acquiring lock by the same thread again and again and it is very much suitable in Recursive functions.

To reprepsent Re-Entrent Lock , Python has provided a predefined class in the form of RLock.

EX:

```
from threading import current_thread, Thread, RLock

l = RLock()
def factorial(n):
    l.acquire()
    result = n
    if n == 0:
        result = 1
    else:
        result = result * factorial(n-1)
    l.release()
    return result

def fn(n):
    print("Factorial of ",n,"is ",factorial(n))

t1 = Thread(target=fn, args=(5,))
t2 = Thread(target=fn, args=(6,))
t1.start()
t2.start()
```

OP:

```
Factorial of  5 is  120
Factorial of  6 is  720
```

Synchronization by Semaphores:

Lock and RLock are able to allow only one thread at a time, after completing the present thread execution only other threads are allowed.

As part of Synhcornization, if n no of threads are trying to execute program, where if we want to allow a fixed no of threads [Not all the threads] then we have to use Semaphore.

To represent Semaphore, Python has provided a

predefined class in the form of Semaphore.
Syntaxes:
----
s = Semaphore()
--> It able to allow 1 thread at a time in Synchronized area.

s = Semaphore(count)
--> It able to allow the specified no of threads in Synchronized Area.
EX:

```
from threading import *
import time
s = Semaphore(2)
def wish():
    s.acquire()
    for x in range(0,10):
        time.sleep(1)
        print(current_thread().getName())
    s.release()
t1 = Thread(target = wish)
t2 = Thread(target = wish)
t3 = Thread(target = wish)
t4 = Thread(target = wish)
t5 = Thread(target = wish)
```

```
t1.setName("AAA")
t2.setName("BBB")
t3.setName("CCC")
t4.setName("DDD")
t5.setName("EEE")

t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

```
OP:
----
AAA
BBB
AAA
BBB
AAA
BBB
AAA
BBB
AAA
BBB
AAA
BBB
AAA
```

BBB
AAA
BBB
AAA
BBB
AAA
BBB
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
CCC
DDD
DDDCCC

```
EEE
EEE
EEE
EEE
EEE
EEE
EEE
EEE
EEE
EEE
```

IN Normal Semaphores, it is not at all mandatory condition to match no of acquire() functions and no release() functions calls.

EX:

```
from threading import *
s = Semaphore()
s.acquire()

s.release()
s.release()
print("End of Program")
```

OP:

```
End of Program
```

In Python applications, if we want to make no of acquire() functions calls and no of releas() function as equal then we have to use "BoundedSemaphore".
EX:

```
from threading import *
s = BoundedSemaphore()
s.acquire()

s.release()
s.release()
print("End of Program")
```
OP:

```
ValueError: Semaphore released too many times
```

EX:

```
from threading import *
s = BoundedSemaphore()
s.acquire()

s.release()
print("End of Program")
```

OP:

End of Program

Note: In case of Semaphores, if any thread access acquire() method then the provided threads count value will be decremented by 1, if any thread access release() function then threads count value is incremented by 1. in this process, if count value is 0 then Semaphore will make the next threads in waiting state until the completion of threads.

Note: In Python applications, Synchronization is able to allow only one thread at a time, it followis sequential execution of the threads, it will increase application execution time, it will reduce application performance, due to this reason, in python applications , it is not suggestible to use "Synchronization", if we are not thinking about the data consistency then it is suggestible to remove Synchronization in python applications.

Inter Thread Communication:

The process of providing communication between more than one thread then it is called as Inter Thread Communication.

To provide inter thread communication between threads Python has provided a set of predefined classes.

1. Event
2. Condition
3. Queue

In general, Inter Thread Communication is able to provide solutions for the problems like "Producer-Consumer" problems.

In Producer-Consumer problem, both Producer and Consumer are two threads, where Producer Thread has to produce an item and COnsumer Thread has to consume that item, the same sequence has to manage upto infinite no of times.

Inter thread Communication through Event object:

Event is an object, it will provide the methods inorder to make a thread to wait and to send notification to some other threads to active which are existed in waiting state.

To create an Even object we have to use the following instrution.
event = threadding.Event()

Note: Event has an internal flag that we can set and clear depending on our requirement.

Methods in Event:
1. set(): It can be used to set True value to internal flag and it will give notification to other threads which are available in waiting state.

2. clear(): It can be used to set False value to internal flag and it will give notification to other threads not to active which are available in waiting state.

3. isSet(): It ca be used to check whether event is set or not.

4. wait() / wait(seconds): It will make current thread to wait.

EX:

```
from threading import *
import time
count = 0
flag = True
def produce():
    global count, flag
    while True:
        if flag == True:
            count = count + 1
            print("Producer Produced  :",count)
            flag = False
            event.set()
            event.wait()
        else:
            event.wait()
def consume():
    global count, flag
    while True:
        if flag == False:
            print("Consumer Consumed  :",count)
            flag = True
            event.set()
            event.wait()
        else:
            event.wait()

event = Event()
producer = Thread(target = produce)
consumer = Thread(target = consume)
producer.start()
consumer.start()
```

OP:

```
Producer Produced  : 1
```

```
Consumer Consumed  : 1
Producer Produced  : 2
Consumer Consumed  : 2
Producer Produced  : 3
Consumer Consumed  : 3
Producer Produced  : 4
Consumer Consumed  : 4
Producer Produced  : 5
Consumer Consumed  : 5
---
---
---
---
```

EX:

```
from threading import *
import time
def trafficPolice():
    while True:
        print("Traffic Police : GREEN Signal")
        event.set()
        time.sleep(10)
        event.clear()
        print("Traffic Police : RED Signal")
        event.wait()
def vehicles():
    event.wait()
    num = 0
    while True:
        time.sleep(1)
        if event.isSet() == True:
            num = num + 1
            print("Vehicle",num,"Moving....")
        else:
            num = 0
            print("All Vehicles are STOPPED")
            event.wait(2)
            event.set()

event = Event()
t1 = Thread(target=trafficPolice)
t2 = Thread(target=vehicles)
t1.start()
t2.start()
```

OP:

```
---
Traffic Police : GREEN Signal
Vehicle 1 Moving....
Vehicle 2 Moving....
Vehicle 3 Moving....
Vehicle 4 Moving....
Vehicle 5 Moving....
Vehicle 6 Moving....
Vehicle 7 Moving....
Vehicle 8 Moving....
Vehicle 9 Moving....
Traffic Police : RED Signal
All Vehicles are STOPPED
Traffic Police : GREEN Signal
Vehicle 1 Moving....
Vehicle 2 Moving....
Vehicle 3 Moving....
Vehicle 4 Moving....
Vehicle 5 Moving....
Vehicle 6 Moving....
---
---
```

Inter Thread Communication By Using Condition:

Condition is more advanced than Event, It able to manage a Condition internally and it will change the state on the basis of the condition.

In Producer-Consumer problem, we will use Condition to make threads wait and to send notification when condition happend.

Condition has the following methods.

1. acquire() --> To acquire a thread before producing or consuming item, that is, Thread   acquire lock internally.
2. release() --> To release condition object after producing or consuming item, that is,
Thread release lock internally.
3. wait() -----> To make a thread to wait.
4. notify() ---> To give notification to other thread which is in waiting state.
5. notifyAll()-> To give notification to all threads which are in waiting state.

EX:

```
from threading import *
import time
count = 0
def produce():
    global count
    while True:
        c.acquire()
        time.sleep(2)
        count = count + 1
        print("Producer Produced Item :",count)
        c.notify()
        c.wait()
        c.release()
def consume():
    global count
    while True:
        c.acquire()
        print("Consumer Consumed Item :",count)
        c.notify()
        c.wait()
        c.release()

c = Condition()
producer = Thread(target=produce)
consumer = Thread(target=consume)
producer.start()
consumer.start()
```

Inter Thread Communication By Using Queue:

Queue is most enhanced Inter Thread Communication mechanism to establish communication between threads and to share data between threads.

Queue internally has condition and a lock , it will set automatically when we put elements and retrieving elements from Queue.

Queue is an element provided by queue module, if we want to use Queue in Python application then we have to import queue module.

Methods of Queue:
----
1. put(): It will ad an element in Queue.
2. get(): It will remove and return an element from Queue.

Producer Thread will use put() method to insert Item in Queue , put() method has logic to acquire lock before inserting item in queue and it will release lock after inserting item.

put() method will check whther Queue is full or not, if Queue is full then Producer Thread will come to waiting state.

Consumer thread will use get() method to remove and return element from Queue, it has internal logic to acquire lock beforer removing element and to release lock after removing element.

get() method will check first whether item is existed or not, if no item is existed then get() method will keep consumer thread in waiting state.

EX:

```
import queue as q
from threading import *
import time

count = 0
def produce():
    global count
    while True:
        count = count + 1
        print("Producer Produced Item",count)
        queue.put(count)
        time.sleep(2)
def consume():
    while True:
        print("Consumer Consumed Item",queue.get())
        time.sleep(2)

queue = q.Queue()
producer = Thread(target=produce)
consumer = Thread(target=consume)
producer.start()
consumer.start()
```

OP:

```
---
Producer Produced Item 1
Consumer Consumed Item 1
```

```
Producer Produced Item 2
Consumer Consumed Item 2
Producer Produced Item 3
Consumer Consumed Item 3
Producer Produced Item 4
Consumer Consumed Item 4
Producer Produced Item 5
Consumer Consumed Item 5
Producer Produced Item 6
Consumer Consumed Item 6
Producer Produced Item 7
Consumer Consumed Item 7
----
----
----
```

There are three types of Queues in python.

1. FIFO Queue:

It is default behaviour of Queue, it will retrieve all the elements in the same order in which we entered.

EX:

```
import queue as q
queue = q.Queue()
queue.put("AAA")
queue.put("DDD")
queue.put("BBB")
queue.put("CCC")

for x in range(0,queue.qsize()):
    print(queue.get())
```

OP:

```
AAA
DDD
BBB
CCC
```

LIFO Queue:

It able to retrieve all elements in Last In First Out Manner.
EX:

```
import queue as q
queue = q.LifoQueue()
queue.put("AAA")
queue.put("DDD")
queue.put("BBB")
queue.put("CCC")
```

```
for x in range(0,queue.qsize()):
    print(queue.get())
```

OP:

```
CCC
BBB
DDD
AAA
```

PriorityQueue:

It allows elements on the basis of Priorities.
EX:

```
import queue as q
queue = q.PriorityQueue()
queue.put(10)
queue.put(20)
queue.put(30)
queue.put(40)

for x in range(0,queue.qsize()):
    print(queue.get())
```

OP:

```
10
20
30
40
```

Python Applications with Visual Studeo IDE:
----

1. Open "https://visualstudio.microsoft.com/" url in Browser.
2. Select "Download Visual Studeo" and select "Community 2019".
3. Copy the downloaded file into Softwares dump.
4. Double click on Setup file.
5. Click on "Yes".
6. Click on "Continue".
7. Select "Python Development" and Click on "Install" button
8. Click on "Launch" Button
9. Select "Not Now, No Thanks".
10. Select "light" and "Start Visual Studeo".
11. Select "Create New Project"
12. Select "Python Application" and "Next" button.
13. Provide project details
    Project Name: app01
    Project Location: D:\vsapps

14. Click on "Create" button.

15.Write Python Code

PDBC[Python Database Connectivity]

In general, in enterprise applications,it is convention to manage data about the organizations like Employees details, Services details, products details,....

In enterprise application development , to manage these details we need to use Storage Areas.

There are two types of Storage Areas.
1. Temporary Storage Areas
2. Permanent Storage Areas

1. Temporary Storage Areas:

These Storage Areas are able to store data temporarily.
EX: Buffers, Pyuthon Objects

2. Permanent Storage Areas:

These Storage Areas are able to store data permanently.

There are three types of Permament Storage Areas.

1. File Systems
2. Database Management Systems
3. Datawarehouses.

1. File Systems:

--> File System is a system , it will be managed by local Operating System, it is platform dependent and it is not suitable for Platform independent Programming languages like Python, Java,....
--> File System is able to store less data.
--> File System is able to provide less security.
--> File System is able to increase Data Redundency.
--> File Systems are not having Query Languages support.

2. Database Management Systems

--> Database Management System is a Software System, it able to manage data by storing it and by retrieving it from Database.
--> Database Management System is able to store data more data when compared with File Systems, but, Database Management System is able to store less data when compared with Datawarehouses.
--> Database Management System is very good while storing data , updating data, deleting data,....but it is not

good

3. Datawarehouses:

--> It able to store more data when compared with File Systems and DBMS.
--> It is very good while retrieving data from Databases, because, it has Data Mining Tech.

Q)What is the difference between Database and Database Management System?

Ans:
-----
1. Database is memory storage to store data.
   Database Management System is a Software System, it able to manage data by storing it   and by retrieving it from Database.

2. Database is the collection of interrelated data.
   Database Management System is the collection of interrelated data and a set of rules   and regulations to access data.

There are three types of Database Management Systems.

1. Relational Database Management System
2. Object Oriented Database Management System
3. Object Relational Database Management System

1. Relational Database Management System:

--> It able to manage data in the form of Tables.
--> It able to use SQL2 as query language to manage data manipulations.

2. Object Oriented Database Management System:

--> It able to manage data in the fomr of Objects.
--> It able to use OQL as query language to manage data manipulations.

3. Object Relational Database Management System:

--> It able to manage data in the form of tables and objects.
--> It able to manage SQL3 as query language to manipulate data.

Query Processing System:

If we submit an SQL query to Database, Where at database , Database Engine will take query and

Database will execute sql query by following the following actions.

1. Query Tokenization
2. Query Parsing
3. Query Optimization
4. Query Execution

1. Query Tokenization:

It able to devide the provided sql query into no of tokens and it will generate stream of tokens as an output.

2. Query Parsing:

The main intention of this pahse is to check the syntax errors in the provided sql query.

This phase will take the stream of tokens as an input and it will constructor a Tree called as Query Truee, if query Tree is success then no syntax errors are existed in the provided sql query. If query tree is not success then there are some syntax errors.

3. Query Optimization:

This phase will apply no of optimization algorithms on query tree to optimize the query tree inorder to reduce execution time and inorder to reduce memory utilization.

4. Query Execution

This phase is able to execute sql queries by using an interpretor and it will send results to the respective client.

Python Database Connectivity:

-->PDBC is the process of iteracting with database from Python program in order to perform database operations from python applications.

-->PDBC is a step by step by process to connect with Database from Python programs inorder to perform database operations from python applications.

--> PDBC is a library contains predefined functions, classes,... it can be used to iteract with database from Python programs inorder to perform database operations from python applications.

In general, in PDBC applications, we will write database logic in Python programs as per python repersentations and we have to submit that database logic to database, where Database has to execute database logic and database has to send the results to Python application.

In the above context, when we submit python represented database logic to database trhen Database will not execute python represented database logic , because, Database is unable to execute python representations.

In the above context, to execute PDBC applications we must use a translator to map Python API calls to Database API calls and database API calls to Python API calls, here the translator is called as "Driver".

Driver is an interface existed in between Python program and database, it can be used to map Python Representations to Database Representations and Database representations to Python Representations.

To provide Drivers Python has provided a seperate module for each and every database which includes PDBC library and Driver logic.
EX1: cx_Oracle for Oracle Database.
EX2: mysql.connector for MySQL Database
EX3: pyodbc for all databases from python applications.

If we want to use cx_Oracle module in Python applications, first , we have to install cx_Oracle explicit library then we have to import cx_Oracle module.

If we use the IDE like PyCharm then we have to add cx_Oracle or cx_Oracle_ctypes to project interpretor.

file --> settings --> project iterpretor --> + [add] --> search cx_Oracle / cx_Oracle_ctypes --> Install Libraries --> Ok button.

If we use the IDE like Visual Studeo the it is not required to add cx_Oracle module, because, cx_Oracle is bydefault provided by Visual Studeo.

If we are not using any IDE and if we want to prepare applications by using Editplus, IDLE... then we have to install cx_Oracle by using pip command.

```
pip install cx_Oracle
```

In Python file, to use cx_Oracle module, we must import cx_Oracle.

```
import cx_Oracle
```

Steps to prepare PDBC Applications:

----
1. Install Oracle Database.
2. In Python Appliactions
   a)import cx_Oracle module
      import cx_Oracle as cxo
   b)Create Connection between Python application and Database
      con =
cxo.connect("uname/password@DBServerIPAddrtess:DBPort/DBName")
      EX: con =
cxo.connect("system/durga@localhost:1521/xe")
   c)Create Cursor inorder to hold result of the SQl query execution.
      cursor = con.cursor()
   d)Write and execute SQl query
      cursor.execute(Query)
      EX: cursor.execute("select * from emp1")
   e)If we execute non select sql queries like create or insert or update or delete   or alter or drop then we must perform either commit or rollback operation.
        con.commit() --> To store manipulations Permanently.
        con.rollback() --> To remove manipulations and to get back previous state.
   f)If we execute select sql query then get Results from Cursor

```
        data1 = cursor.fetchone() --> To get only one row.
        data2 = cursor.fetchmany(n) --> To get no of
rows.
        data3 = cursor.fetchall() --> To get all rows.
```

g)Close the connection and cursor

```
        cursor.close()
        con.close()
```

EX1: PDBC Application to create table in DB:

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("create table emp1(ENO number(3)
primary key,ENAME varchar2(10),ESAL float(5),EADDR
varchar2(10))")
print("Table emp1 created Successfully")
con.commit()
cursor.close()
con.close()
```

OP:

```
Table emp1 created Successfully
```

EX2: PDBC Application to create table by taking table name as Dynamic Name

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
tname = input("Table Name  : ")
query = "create table "+tname+"(ENO number(3)
primary key, ENAME varchar2(10), ESAL float(5), EADDR
varchar2(10))"
cursor.execute(query)
con.commit()
print("Table",tname,"Created Successfully")
cursor.close()
con.close()
```

OP:

```
Table Name : emp2
Table emp2 Created Successfully
```

EX3: PDBC Application to insert records into Database table:

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
while True:
    eno = int(input("Employee Number  : "))
    ename = input("Employee Name   : ")
    esal = float(input("Employee Salary   : "))
    eaddr = input("Employee Address  : ")
    cursor.execute("insert into emp1 
values(%i,'%s',%f,'%s')"%(eno,ename,esal,eaddr))
    print("Employee ",eno," Inserted Successfully")
    option = input("Onemore Employee[yes/no]? : ")
    if option == "yes":
        continue
    else:
        break

con.commit()
cursor.close()
con.close()
```

EX4: PDBC Application to perform Updations on database table:

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("update emp1 set ESAL = ESAL + 500 
where ESAL < 10000")
con.commit()
print("Employee Records Updated Successfully")
```

```
cursor.close()
con.close()
```

EX5: PDBC Application to delete records from Database:

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("delete from emp1 where ESAL <
10000")
con.commit()
print("Employee records Deleted Successfully")
cursor.close()
con.close()
```

EX6: PDBC Application to drop table from DB:

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("drop table emp1")
con.commit()
print("emp1 table dropped from Db successfully")
cursor.close()
con.close()
```

EX7: PDBC Application to retrieve Data from Database table by using fetchone():

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("select * from emp1")
data = cursor.fetchone()
print("Employee Details")
print("--------------------")
print("Employee Number   : ",data[0])
print("Employee Name     : ",data[1])
print("Employee Salary   : ",data[2])
print("Employee Address  : ",data[3])
cursor.close()
con.close()
```

EX8: PDBC Application to retrieve Data from Database table by using fetchmany():

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("select * from emp1")
data = cursor.fetchmany(3)
print("ENO\tENAME\tESAL\tEADDR")
print("-----------------------------")
for row in data:
    print(row[0],end="\t")
    print(row[1],end="\t")
    print(row[2],end="\t")
    print(row[3],end="\n")
cursor.close()
con.close()
```

EX9: PDBC Application to retrieve Data from Database table by using fetchall():

```
import cx_Oracle as cxo
con = cxo.connect("system/durga@localhost:1521/xe")
cursor = con.cursor()
cursor.execute("select * from emp1")
data = cursor.fetchall()
print("ENO\tENAME\tESAL\tEADDR")
print("------------------------------")
for row in data:
    print(row[0],end="\t")
    print(row[1],end="\t")
    print(row[2],end="\t")
    print(row[3],end="\n")
cursor.close()
con.close()
```

Regular Expression:

In general, in applications, if we want to check or compare a string w.r.t a particular pattern or format then we have to use "Regular Expressions".
EX: Checking Whether a Mobile is valid mobile number of not.
EX: Checking whether an Email ID is valid or not.
EX: Checking whether Date of Birth value is a valid Date or not.

In general, we will use Regular Expressions in the following areas.
1. Data Validations
2. Pattern matching Applications
3. Network Applications
4. AI and Nueral Networks Applications
5. Communication Protocols TCP/IP, UDP,....
6. Compilers and Interpretors

To implement regular Expressions related things in python applications , Python has provided predefined library in the form of an inbuilt module "re".

To implement Regular Expressions in Python applications, we have to use the following steps.
1. Prepare pattern by using compile()

```
pattern = re.compile("ab")
```

2. Find all matches of the pattern over the target string by using finditer() function.
   matcher = pattern.finditer("ababab ab")
   Where finditer() function will identify all matches of the pattern over the target   striung and it will create Match object for each and every match and it will return an Iterator with all match objects.
3. get Start index , end index of each and every match and the matched group of   characters from match object.
     start() : To get start index of thematch
     end() : To get end_index+1 of the match
     group(): sequence of characters of the match

EX:

```
import re
pattern = re.compile("ab")
matcher = pattern.finditer("abababab")
for match in matcher:

print(match.start(),"---->",match.end(),"---->",match.gr
oup())
```
OP:

```
0 ----> 2 ----> ab
```

```
2 ----> 4 ----> ab
4 ----> 6 ----> ab
6 ----> 8 ----> ab
```

Note: We are able to provide pattern string in finditer() function with out using compile() function.
EX:

```
import re
matcher = re.finditer("ab","abababab")
count = 0
for match in matcher:
    count = count + 1

print(match.start(),"---->",match.end(),"---->",match.group())

print("No of Occurences  :",count)
```
OP:

```
0 ----> 2 ----> ab
2 ----> 4 ----> ab
4 ----> 6 ----> ab
6 ----> 8 ----> ab
No of Occurences  : 4
```

Q)Find the no of occurences of the word Durga and its

starting and ending positions in String data ?

Ans:

```
import re
data = "Durga Software Solutions is very good Org for
IT Courses, Durga Software Solutions has no of
branches, Durga Software Solutions has very good
trainer in the form of Durga sir"
pattern = "Durga"
matcher = re.finditer(pattern,data)
count = 0
for match in matcher:
    count = count + 1

print(match.start(),"--->",match.end()-1,"--->",match.gr
oup())
print("No Of occurences  :",count)
```

OP:

```
0 ---> 4 ---> Durga
58 ---> 62 ---> Durga
103 ---> 107 ---> Durga
165 ---> 169 ---> Durga
No Of occurences  : 4
```

Q)Find the no of occurences of the word Durga and its starting and ending positions in String data existed in a text file?

Ans:

```
import re
file = open("E:/abc/xyz/data.txt","r")
pattern = "Durga"
count = 0
for line in file:
    matcher = re.finditer(pattern,line)
    for match in matcher:
        count = count + 1

print(match.start(),"--->",match.end()-1,"--->",match.group())
print("No of Occurences  :",count)
```

OP:

```
0 ---> 4 ---> Durga
58 ---> 62 ---> Durga
103 ---> 107 ---> Durga
165 ---> 169 ---> Durga
```

No of Occurences : 4

Character Classes:

The main intention of Character classes is to search for group of characters in tarhet string.
EX:
a ----> It will find the match with exactly 'a'
[abc] ----> It will find the match for either a or b or c
[^abc] ---> It will find the match for the elements except a, b anc c.
[a-z] ----> It will find the match for All lower case characters
[A-Z] ----> It will find the match for all Upper case characters
[a-zA-Z]---> It will find the match for all lowerr case and upper case matches.
[0-9] -----> It will find the match for all digits.
[a-zA-Z0-9] ---> It will find the match for all lower case , upper case characters and   digits.
. -----------> It will find the match for all characters , dogots and special symbols.

EX1:

```
import re
data = "abc123xyz@durgasoft.com"
matcher = re.finditer("a",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
0 ---> 0 ---> a
14 ---> 14 ---> a
```

EX2:

```
import re
data = "abc123bca@durgasoft.com"
matcher = re.finditer("[abc]",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
6 ---> 6 ---> b
```

```
7 ---> 7 ---> c
8 ---> 8 ---> a
14 ---> 14 ---> a
20 ---> 20 ---> c
```

EX:

```
import re
data = "abc123bca@durgasoft.com"
matcher = re.finditer("[a-z]",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
6 ---> 6 ---> b
7 ---> 7 ---> c
8 ---> 8 ---> a
10 ---> 10 ---> d
11 ---> 11 ---> u
12 ---> 12 ---> r
13 ---> 13 ---> g
14 ---> 14 ---> a
15 ---> 15 ---> s
```

```
16 ---> 16 ---> o
17 ---> 17 ---> f
18 ---> 18 ---> t
20 ---> 20 ---> c
21 ---> 21 ---> o
22 ---> 22 ---> m
```

EX:

```
import re
data = "abc123ABC@durgasoft.com"
matcher = re.finditer("[A-Z]",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
6 ---> 6 ---> A
7 ---> 7 ---> B
8 ---> 8 ---> C
```

EX:

```
import re
data = "abc123ABC@durgasoft.com"
matcher = re.finditer("[0-9]",data)
```

```
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
3 ---> 3 ---> 1
4 ---> 4 ---> 2
5 ---> 5 ---> 3
```

EX:

```
import re
data = "abc123ABC@durgasoft.com"
matcher = re.finditer("[^0-9]",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
6 ---> 6 ---> A
7 ---> 7 ---> B
```

```
8 ---> 8 ---> C
9 ---> 9 ---> @
10 ---> 10 ---> d
11 ---> 11 ---> u
12 ---> 12 ---> r
13 ---> 13 ---> g
14 ---> 14 ---> a
15 ---> 15 ---> s
16 ---> 16 ---> o
17 ---> 17 ---> f
18 ---> 18 ---> t
19 ---> 19 ---> .
20 ---> 20 ---> c
21 ---> 21 ---> o
22 ---> 22 ---> m
```

EX:

```
import re
data = "abc123ABC@durgasoft.com"
matcher = re.finditer("[a-zA-Z0-9]",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
----
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
3 ---> 3 ---> 1
4 ---> 4 ---> 2
5 ---> 5 ---> 3
6 ---> 6 ---> A
7 ---> 7 ---> B
8 ---> 8 ---> C
10 ---> 10 ---> d
11 ---> 11 ---> u
12 ---> 12 ---> r
13 ---> 13 ---> g
14 ---> 14 ---> a
15 ---> 15 ---> s
16 ---> 16 ---> o
17 ---> 17 ---> f
18 ---> 18 ---> t
20 ---> 20 ---> c
21 ---> 21 ---> o
22 ---> 22 ---> m
```

In Regular Expressions, we are able to use the following predefined characters to prepare patterns.

\s ------> Represents Space

\S ------> Represents all except space
\d ------> Represents all digits
\D ------> Represents all except digits
\w ------> Represents all word characters [a-zA-Z0-9]
\W ------> Represents all except word characters

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\s",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
3 ---> 3 --->
7 ---> 7 --->
```

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\S",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
---
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
4 ---> 4 ---> 1
5 ---> 5 ---> 2
6 ---> 6 ---> 3
8 ---> 8 ---> A
9 ---> 9 ---> B
10 ---> 10 ---> C
11 ---> 11 ---> @
12 ---> 12 ---> d
13 ---> 13 ---> u
14 ---> 14 ---> r
15 ---> 15 ---> g
16 ---> 16 ---> a
17 ---> 17 ---> s
18 ---> 18 ---> o
19 ---> 19 ---> f
20 ---> 20 ---> t
21 ---> 21 ---> .
22 ---> 22 ---> c
23 ---> 23 ---> o
24 ---> 24 ---> m
```

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\d",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
4 ---> 4 ---> 1
5 ---> 5 ---> 2
6 ---> 6 ---> 3
```

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\D",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
0 ---> 0 ---> a
```

```
1 ---> 1 ---> b
2 ---> 2 ---> c
3 ---> 3 --->
7 ---> 7 --->
8 ---> 8 ---> A
9 ---> 9 ---> B
10 ---> 10 ---> C
11 ---> 11 ---> @
12 ---> 12 ---> d
13 ---> 13 ---> u
14 ---> 14 ---> r
15 ---> 15 ---> g
16 ---> 16 ---> a
17 ---> 17 ---> s
18 ---> 18 ---> o
19 ---> 19 ---> f
20 ---> 20 ---> t
21 ---> 21 ---> .
22 ---> 22 ---> c
23 ---> 23 ---> o
24 ---> 24 ---> m
```

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\w",data)
```

```
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
---
0 ---> 0 ---> a
1 ---> 1 ---> b
2 ---> 2 ---> c
4 ---> 4 ---> 1
5 ---> 5 ---> 2
6 ---> 6 ---> 3
8 ---> 8 ---> A
9 ---> 9 ---> B
10 ---> 10 ---> C
12 ---> 12 ---> d
13 ---> 13 ---> u
14 ---> 14 ---> r
15 ---> 15 ---> g
16 ---> 16 ---> a
17 ---> 17 ---> s
18 ---> 18 ---> o
19 ---> 19 ---> f
20 ---> 20 ---> t
22 ---> 22 ---> c
23 ---> 23 ---> o
24 ---> 24 ---> m
```

EX:

```
import re
data = "abc 123 ABC@durgasoft.com"
matcher = re.finditer("\W",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
3 ---> 3 --->
7 ---> 7 --->
11 ---> 11 ---> @
21 ---> 21 ---> .
```

Quantifiers:

The main intention of Quantifiers is to repersents the no of occurences of a particular character ot string,....
EX:

```
a -----> Exactly one a
a+ ----> One or more no of a's
a* -----> 0 or more no of a's
a? -----> Atmost one a
a{m} ---> Exactly m no of a's
```

a{m,n}--> Minimum m and maximum n no of a's

EX:

```
import re
data = "abc123ABC@durgasoft.com"
matcher = re.finditer("a",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
0 ---> 0 ---> a
14 ---> 14 ---> a
```

EX:

```
import re
data = "abcaadaab@durgasoft.com"
matcher = re.finditer("a+",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
0 ---> 0 ---> a
3 ---> 4 ---> aa
```

```
6 ---> 7 ---> aa
14 ---> 14 ---> a
```

EX:

```
import re
data = "abcaadaab@durgasoft.com"
matcher = re.finditer("a*",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```
OP:

```
0 ---> 0 ---> a
1 ---> 0 --->
2 ---> 1 --->
3 ---> 4 ---> aa
5 ---> 4 --->
6 ---> 7 ---> aa
8 ---> 7 --->
9 ---> 8 --->
10 ---> 9 --->
11 ---> 10 --->
12 ---> 11 --->
13 ---> 12 --->
14 ---> 14 ---> a
15 ---> 14 --->
```

```
16 ---> 15 --->
17 ---> 16 --->
18 ---> 17 --->
19 ---> 18 --->
20 ---> 19 --->
21 ---> 20 --->
22 ---> 21 --->
23 ---> 22 --->
```

EX:

```
import re
data = "abcaadaab@durgasoft.com"
matcher = re.finditer("a?",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

OP:

```
0 ---> 0 ---> a
1 ---> 0 --->
2 ---> 1 --->
3 ---> 3 ---> a
4 ---> 4 ---> a
5 ---> 4 --->
6 ---> 6 ---> a
```

```
7 ---> 7 ---> a
8 ---> 7 --->
9 ---> 8 --->
10 ---> 9 --->
11 ---> 10 --->
12 ---> 11 --->
13 ---> 12 --->
14 ---> 14 ---> a
15 ---> 14 --->
16 ---> 15 --->
17 ---> 16 --->
18 ---> 17 --->
19 ---> 18 --->
20 ---> 19 --->
21 ---> 20 --->
22 ---> 21 --->
23 ---> 22 --->
```

EX:

```
import re
data = "abcaaadaaabaaaa@durgasoft.com"
matcher = re.finditer("a{3}",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

```
OP:
---
3 ---> 5 ---> aaa
7 ---> 9 ---> aaa
11 ---> 13 ---> aaa
```

EX:

```
import re
data = "abcaaadaaaabaaaaaaa@durgasoft.com"
matcher = re.finditer("a{2,5}",data)
for match in matcher:
    print(match.start(),"--->",
match.end()-1,"--->",match.group())
```

```
OP:
---
3 ---> 5 ---> aaa
7 ---> 10 ---> aaaa
12 ---> 16 ---> aaaaa
17 ---> 18 ---> aa
```

're' module functions:
----
1. fullmatch(pattern, str)
--> It will check strictly str is as per the specified pattern or not, if any single letter or digit mistake is available

then it will return None value, if the provided str is as per the specified pattern then it will return Match object.
EX:

```
import re
pattern = input("Enter Pattern : ")
str = "abababab"
match = re.fullmatch(pattern,str)
if match != None:
    print(pattern,"is matched with",str)
else:
    print(pattern,"is not matched with",str)
```
OP:

```
Enter Pattern : ab
ab is not matched with abababab
```

OP:

```
Enter Pattern : abababab
abababab is matched with abababab
```

2. match(pattern,str)
--> It will return Match object atleast the provided pattern is same as the prefix of the target string, if the provided pattern is not matched with the prefix of the target string then match() function will return None

value.
EX:

```
import re
pattern = input("Enter Pattern : ")
str = "abababab"
match = re.match(pattern,str)
if match != None:
    print(pattern,"is matched with",str)
else:
    print(pattern,"is not matched with",str)
```
OP:

```
Enter Pattern : ab
ab is matched with abababab
```

OP:

```
Enter Pattern : abababab
abababab is matched with abababab
```

OP:

```
Enter Pattern : xyz
xyz is not matched with abababab
```

3. search(pattern, str)

-->

EX:

```
import re
pattern = input("Enter Pattern : ")
str = "abcxyz123"
match = re.search(pattern,str)
if match != None:
    print(pattern,"is existed in ",str,"at start
Index",match.start(),"upto the End
Index",match.end()-1)
else:
    print(pattern,"is not existed in",str)
```

OP:

```
Enter Pattern : abc
abc is existed in  abcxyz123 at start Index 0 upto the
End Index 2
```
OP:

```
Enter Pattern : xyz
xyz is existed in  abcxyz123 at start Index 3 upto the
```

```
End Index 5
```

OP:

```
Enter Pattern : 123
123 is existed in  abcxyz123 at start Index 6 upto the
End Index 8
```

OP:

```
Enter Pattern : 456
456 is not existed in abcxyz123
```

4. findall()
--> It will return a list contains all the occurences of the speiciied pattern.
EX:

```
import re
pattern = input("Enter Pattern : ")
str = "abcxyzabcxyzabcxyz"
list = re.findall(pattern,str)
print(list)
```

OP:

```
Enter Pattern : abc
['abc', 'abc', 'abc']
```

OP:

```
Enter Pattern : xyz
['xyz', 'xyz', 'xyz']
```

5. finditer()
--> It able to find all occurences of the specified pattern in the string returns all these occurences in the form of multiple Match objects in an iterator.
EX:

```
import re
pattern = input("Enter Pattern : ")
str = "abcxyzabcxyzabcxyz"
iterator = re.finditer(pattern,str)
for match in iterator:

print(match.start(),"--->",match.end()-1,"---->",match.g
roup())
```

OP:

```
Enter Pattern : abc
0 ---> 2 ----> abc
6 ---> 8 ----> abc
12 ---> 14 ----> abc
```

OP:

```
Enter Pattern : xyz
3 ---> 5 ----> xyz
9 ---> 11 ----> xyz
15 ---> 17 ----> xyz
```

6. sub(str1,str2,str)
-->It will replace str1 elements in str with str2.
EX:

```
import re
str = "abc123xyz456"
data1 = re.sub("[a-z]","#",str)
print(data1)
data2 = re.sub("[0-9]","X",str)
print(data2)
```
OP:

```
###123###456
abcXXXxyzXXX
```

7.subn(str1,str2,str)
--> It able to perform replacement operation and it will return no of substitutions and the resultent string as a tuple.

EX:

```
import re
str = "abc123xyz456"
tuple1 = re.subn("[a-z]","#",str)
print(tuple1)
tuple2 = re.subn("[0-9]","*",str)
print(tuple2)
```

OP:

```
('###123###456', 6)
('abc***xyz***', 6)
```

8. split()
--> It will split the provided string into no of sub strings on the basis of the provided regular expression.
EX:

```
import re
str = "Durga Software Solutions"
list = re.split(" ",str)
print(list)
```
OP:

```
['Durga', 'Software', 'Solutions']
```

9. ^ symbol:
--> It can be used to check whether the string prefixed with ^ is existed in target string as first string or not.
EX:

```
import re
pattern = "Durga"
str = "Durga Software Solutions"
match1 = re.search("^"+pattern,str)
if match1 != None:
    print(str,"starts with",pattern)
else:
    print(str,"not starts with",pattern)
```
OP:

```
Durga Software Solutions starts with Durga
```

10. $ Symbol:
--> It can be used to check whether target string ends with the specified string or not.
EX:

```
import re
pattern = "Solutions"
str = "Durga Software Solutions"
match1 = re.search(pattern+"$",str)
if match1 != None:
    print(str,"Ends with",pattern)
else:
    print(str,"not Ends with",pattern)
```

OP:

```
Durga Software Solutions Ends with Solutions
```

Q)Write a Regular Expression to check whether a string is Durgasoft Student ID or not?

Durgasoft Student ID: DSS-rollNum
EX:

```
import re
sid = input("Student ID : ")
pattern = "DSS-[0-9]+"
match = re.fullmatch(pattern,sid)
if match == None:
    print(sid," Is Not Valid Student ID")
else:
    print(sid,"Is Valid Student ID")
```
OP:

```
Student ID : DSS-111
DSS-111 Is Valid Student ID
```

```
OP:
---
Student ID : DSS123
DSS123  Is Not Valid Student ID
```

Q)Write a Regular Expression to check whether a string is valid Email ID or not?

```
import re
emailId = input("Email Id : ")
pattern = "[a-zA-Z0-9][a-z0-9A-Z_.]*@gmail.com"
match = re.match(pattern,str(emailId))
if match == None:
    print(emailId," Is Invalid Email Id")
else:
    print(emailId,"Is Valid Email Id")
```

Write a Regular Expression to check whether a string is mobile no or not?

```
 import re
mobileNo = input("Mobile Number : ")
pattern = "91-\d{10}"
```

```
match = re.match(pattern,mobileNo)
if match == None:
    print(mobileNo," Is Not Valid Mobile Number")
else:
    print(mobileNo,"Is Valid Mobile Number")
```

Web Scrapping:

The process of getting some information about the websites is called as Web Scapping.

To perform web scapping we have to use some predefined library in the form of urllib and urllib.request.
EX:

```
import re, urllib
import urllib.request
url =
urllib.request.urlopen("http://durgasoft.com/contact.asp"
)
data = url.read()
title = re.findall("<title>.*</title>",str(data),re.I)
print(title)
nos=re.findall("[0-9-]{7}[0-9-]+",str(data),re.I)
for n in nos:
    print(n)
```

Decorator:

Decorator is a function, it will take a function as parameter and it will extend its functionality and it will return the modified function .

The main intention of Decorator is to extend the existed method functionality with out changing its code.

If we want to use Decorator function in Python applications then we have to use the following steps.

1. Define decorator function.
2. Apply @decor function to methods
3. Access function.

1. Define decorator function:
----
a)Declare decor function with func argument.
b)Define an inner function with arguments of the actual method.
c)Define new functionality.
d)Return new functionality.
EX:

```
def decor(func)
  def newFunc(paramList)
```

```
      --logic for new functionality--
  return nweFunc
```

2. Apply @decor function to methods:

Provide @decor decorator just above of the method declaration.

```
@decor
def methodname(ParamList):
    ---implementation---
```

3. Access method .

```
methodname(paramList)
methodname(paramlist)
```

EX:

```
def decor(func):
    def inner(name):
        if name == "Nag":
            print("Hello",name,"Bad Morning")
        else:
            func(name)
    return inner
```

```
@decor
def wish(name):
    print("Hello",name,"Good Morning!")

wish("Durga")
wish("Nag")
wish("Anil")
```

EX:

```
def smart_division(func):
    def newFunc(a,b):
        if b == 0:
            print("Division is not Possible")
        else:
            func(a,b)
    return newFunc

@smart_division
def div(a,b):
    print("Division :",(a/b))

div(10,5)
div(10,0)
```

Decorator Chaining:

The process of applying more than one decorator to a single function is called as Decorator Chaining.
EX:

```
@decor1
@decor2
@decor3
def num()
   ------
```

In the above num() function three decorators are applied, first inner decorator will be executed then outer decorator will be executed.

EX:

```
def square(func):
    def inner():
        x = func()
        return x*x
    return inner

def double(func):
    def inner():
        x = func()
        return 2*x
    return inner

@square
@double
def num():
    return 10

print(num())
```

OP:
```
400
```

Generator Function:

Generator is a function which is responsible to generate a sequence of values.
In general, we will write Generator function just like normal function wit yield keyword.

EX:

```
def mygen():
    yield "A"
    yield "B"
    yield "C"

g = mygen()
```

```
print(type(g))

print(next(g))
print(next(g))
print(next(g))

OP:
---
<class 'generator'>
A
B
C

EX:
----
def countdown(num):
    print("Start Countdown")
    while(num>0):
        yield num
        num = num - 1

values = countdown(5)
for x in values:
    print(x)

OP:
---
```

```
Start Countdown
5
4
3
2
1
```

--> It is possible to convert Generator to a list by using list() function.

EX:

```
def first_n(num):
    n = 1
    while n <= num:
        yield n
        n = n + 1



values = first_n(5)
l = list(values)
print(l)
```

OP:

```
[1, 2, 3, 4, 5]
```

EX: Program to generate Fibonacci Numbers.

```
def fib():
    a,b = 0,1
    while True:
        yield a
        a,b = b,a+b

for x in fib():
    if x > 100:
        break
    print(x)
```

OP:

```
---
0
1
1
2
3
5
8
13
21
34
55
89
```

Python - Logging:

In real time application development, we may get no of problems or bugs or exceptions while executing or testing the applications, To identify the problems and their locations then we have to trace the applications flow of execution.

To trace applications flow of execution we will use print() at basic level.

EX:

```
class Transaction:
    def deposit(self):
        print("Logic for deposit")
    def withdraw(self):
        print("Logic for Withdraw")
    def transfer(self):
        print("Logic for transfer")

tx = Transaction()
print("Before deposit() call")
tx.deposit()
print("After deposit() call")
print("Before withdraw() call")
tx.withdraw()
print("After withdraw() call")
print("Before transfer() call")
tx.transfer()
print("After transfer() call")
```

OP:

```
Before deposit() call
Logic for deposit
After deposit() call
Before withdraw() call
Logic for Withdraw
After withdraw() call
Before transfer() call
Logic for transfer
After transfer() call
```

If we use print() function in applications to trace then we are able to get the following problems.

1. print() function is able to display data on console, it is not for sending messages to the output devices like files , databases, network,.....

2. In Applications, it is not suggestible to use too many no of print() functions , because, it will reduce application performance.

3. print() function is very much usefull in development environment only, it is not suitable in production environment, because, if we use print() function in server side applcations then that messages will be displayed at server consle only, not at client side.

4. If we use print() function will display messages on console only, it will not show difference between warning messages and error messages, normal messages,....

5. print() function is suitable in standalone applications only, not in enterprise applications.

To overcome all the problems while tracing applications we have to use Logging.

Logging: It is the process of writting log messages in a central place , it allows us to report and persist error and warning messages as well as info messages inorder to retrieve and analyze later on.

In general, Logging required.
1. To Understand flow of execution in application.
2. To manage exception messages when exceptions or Errors occurred in python applications.
3. To manage Event-Notifications messages in file

system.

To perform Logging in python applications, python has provided a seperate module in the form of "Logging".

as part of Logging, we have to generate some Log messages as per the following Logging Levels.

There are 6 types of Logging levels.
1. CRITICAL --> 50
--> It repersents serious problem that needs high attention.

2. ERROR --> 40
--> It represents a serious error

3. WARNING --> 30
--> It represnts warning messages , some caution needed, it alerts programmer.

4. INFO --> 20
--> Represnts message with some important information

5. DEBUG --> 10
--> Represents a message with debugging information.

6. NOTEST --> 0

--> Represents no level is set.

To implement Logging, first we have to create a file to store messages and we have to specify which level of messages to store.

To implement the above we have to use basicConfig() function of logging module.
EX:
logging.basicConfig(filename="log.txt",level=logging.WARNING)

The above instruction will create a file log.txt and we can store either WARNINg level or Hiher level.

After creating log file, we can write messages to that file by using the following messages.

```
logging.debug(message)
logging.info(message)
logging.warning(message)
logging.error(message)
logging.critical(message)
```

EX:

```
import logging
```

```
logging.basicConfig(filename="log.txt",level=logging.WA
RNING)
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
WARNING:root:warning Message
ERROR:root:error Message
CRITICAL:root:critical Message
```

note: In the above applipcation, we are able to get WERNING and above messages only, if we set DEBUG then we are able to get all messages.
EX:

```
import logging
logging.basicConfig(filename="log.txt",level=logging.DE
BUG)
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
DEBUG:root:Debugging Messages
INFO:root:info message
WARNING:root:warning Message
ERROR:root:error Message
CRITICAL:root:critical Message
```

If we want to configure logi file in over write mode we have to use "filemode" argument in basicConfig() function.

There are three modes for filemode in basicConfig()
1. 'a' : to append new messages to old messages in log file.
2. 'w' : to over write old messages to new messages in log file.

In basicConfig() function,
1. The default value for 'filemode' is 'a', that is, Mode.
2. The default value for 'level' argument is 'WARNING'.
3. The default value for 'filename' is console.

EX:

```
import logging
```

```
logging.basicConfig()
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
WARNING:root:warning Message
ERROR:root:error Message
CRITICAL:root:critical Message
```

In logging , we are able to format messages by using 'format' argument n basicConfig() .

1. To display only level name:
   ```
   logging.basicConfig(format='%(levelname)s')
   ```

EX:

```
import logging
logging.basicConfig(format='%(levelname)s')
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
WARNING
ERROR
CRITICAL
```

2. To Display levelname and message:

```
logging.basicConfig(format='%(levelname)s:%(message
)s')
```

EX:

```
import logging
logging.basicConfig(format='%(levelname)s:%(message
)s')
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
WARNING:warning Message
ERROR:error Message
```

```
CRITICAL:critical Message
```

3. To add time stamp in Logging Message:

```
logging.basicConfig(format='%(asctime)s:%(levelname)
s:%(message)s')
```

EX:

```
import logging
logging.basicConfig(format='%(asctime)s:%(levelname)
s:%(message)s')
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```

OP:

```
2020-02-02 00:23:15,767:WARNING:warning Message
2020-02-02 00:23:15,767:ERROR:error Message
2020-02-02 00:23:15,767:CRITICAL:critical Message
```

To change date and time format in log messages then we have to use 'datefmt' argument in basicConfig() function.

```
logging.basicConfig(format='%(asctime)s:%(levelname)
s:%(message)s', datefmt='%d/%m/%y %I:%M:%S
%p')
```

EX:

```
import logging
logging.basicConfig(format='%(asctime)s:%(levelname)
s:%(message)s',datefmt='%d/%m/%y %I:%M:%S
%p')
logging.debug("Debugging Messages")
logging.info("info message")
logging.warning("warning Message")
logging.error("error Message")
logging.critical("critical Message")
```
OP:

```
02/02/20 12:28:20 AM:WARNING:warning Message
02/02/20 12:28:20 AM:ERROR:error Message
02/02/20 12:28:20 AM:CRITICAL:critical Message
```

%d ---> Date
%m ---> month
%y ---> year
%I ---> hour in 12 hours format.
%H ---> hour in 24 hours format.
%M ---> Minute

%S ---> Second
%p ---> AM or PM

EX: Application to write Exception in log file

```
import logging
logging.basicConfig(filename='log.txt',level=logging.INFO,format='%(asctime)s:%(levelname)s:%(message)s',datefmt='%d/%m/%y %I:%M:%S %p')
logging.info("A New Request Came")
try:
    val1 = int(input("First Value  : "))
    val2 = int(input("Second Value : "))
    result = val1 / val2
    print("Result  :",result)
except ZeroDivisionError as msg:
    print("Cam divide with Zero")
    logging.exception(msg)
except ValueError as msg:
    print("Provide int value only")
    logging.exception(msg)

logging.info("Request Processing Completed")
```

OP:

```
---
First Value  : 10
Second Value : 2
```

```
Result  : 5.0

OP:
---
First Value  : 10
Second Value : 0
Cam divide with Zero

OP:
---
First Value  : ten
Provide int value only

log.txt
--------
02/02/20 12:43:57 AM:INFO:A New Request Came
02/02/20 12:44:01 AM:INFO:Request Processing
Completed
02/02/20 12:44:36 AM:INFO:A New Request Came
02/02/20 12:44:42 AM:ERROR:division by zero
Traceback (most recent call last):
  File
"D:/nag/python/practice/Operators/com/durgasoft/opera
tors.py", line 7, in <module>
    result = val1 / val2
ZeroDivisionError: division by zero
02/02/20 12:44:42 AM:INFO:Request Processing
```

```
Completed
02/02/20 12:45:00 AM:INFO:A New Request Came
02/02/20 12:45:06 AM:ERROR:invalid literal for int()
with base 10: 'ten'
Traceback (most recent call last):
  File
"D:/nag/python/practice/Operators/com/durgasoft/opera
tors.py", line 5, in <module>
    val1 = int(input("First Value  : "))
ValueError: invalid literal for int() with base 10: 'ten'
02/02/20 12:45:06 AM:INFO:Request Processing
Completed
```

In general, Logging module is using "Root Logger", with Root Logger we are able to get the following problems.

1. Once if we set configuration then that configuration is final and we are unable to change.
2. It allows only one configuration , it does not allows more than one configuration like console   handler, file handler,.....
3. It is not possible to configure logger with different configurations at different levels.
4. We are unable to specify multiple log files for multiple modules / classes / methods.

To overcome the above problem we have to create our own Loggers, that is, Custom Logger.

Logger is advanced logging module , it is more advanced than basic logging and it will provide more advanced features.

To use Logger in Python applications we will use the following steps.

1. create logger object and set log level:
```
   logger = logging.getLogger("testlogger")
   logger.setLevel(logging.INFO)
```

2. Create Handler objects like FileHandler, ConsoleHandler,.....
```
   consoleHandler = logging.ConsoleHandler()
   consoleHandler.setLevel(logging.INFO)
```

3. Create Formatter object:
```
   formatter =
logging.Formatter('%(asctime)s-%(loglevel)s-%(levelna
me)s: %(message)s', datefmt='    %d/%m/%y
%I:%M:%S %p'))
```

4. Add Formatter nto Handler:
```
   consoleHandler.setFormatter(formatter)
```

5. Add Handler to Logger:
```
   logger.addHabdler(consoleHandler)
```

6. Write messages by using logger object:
```
   logger.debug('debug message')
   logger.info('info message')
   logger.warn('Warn Message')
   logger.error('error message')
   logger.critical('criticaql Message')
```

note: Bydefault, log level is WARNING level, but, it is possible to set our own level.
```
      logger = logging.getLogger('testlogger')
      logger.setLevel(logging.INFO)
```

EX on ConsoleHandler:
```
---------------------
import logging
class ConsoleLoggerEx:
    def testLog(self):
        logger = logging.getLogger('testlogger')
        logger.setLevel(logging.INFO)

        consoleHandler = logging.StreamHandler()
        consoleHandler.setLevel(logging.INFO)

        formatter =
logging.Formatter('%(asctime)s-%(name)s-%(levelname
)s:%(message)s', datefmt='%m/%d/%y %I:%M:%S
%p')
        consoleHandler.setFormatter(formatter)
        logger.addHandler(consoleHandler)

        logger.debug('debug message')
        logger.info('info message')
        logger.warn('Warn Message')
        logger.error('error message')
        logger.critical('criticaql Message')

demo = ConsoleLoggerEx()
demo.testLog()
```

OP:

```
02/02/20 02:25:36 AM-testlogger-INFO:info message
02/02/20 02:25:36 AM-testlogger-WARNING:Warn
Message
02/02/20 02:25:36 AM-testlogger-ERROR:error message
02/02/20 02:25:36 AM-testlogger-CRITICAL:criticaql
Message
```

EX on FileHandler:

```
import logging
```

```
class FileLoggerEx:
    def testLog(self):
        logger = logging.getLogger('testlogger')
        logger.setLevel(logging.INFO)

        consoleHandler =
logging.FileHandler('log.txt',mode='a')
        consoleHandler.setLevel(logging.INFO)

        formatter =
logging.Formatter('%(asctime)s-%(name)s-%(levelname
)s:%(message)s', datefmt='%m/%d/%y %I:%M:%S
%p')
        consoleHandler.setFormatter(formatter)
        logger.addHandler(consoleHandler)

        logger.debug('debug message')
        logger.info('info message')
        logger.warn('Warn Message')
        logger.error('error message')
        logger.critical('critical Message')

demo = FileLoggerEx()
demo.testLog()

OP:
---
```

```
02/02/20 02:30:51 AM-testlogger-INFO:info message
02/02/20 02:30:51 AM-testlogger-WARNING:Warn
Message
02/02/20 02:30:51 AM-testlogger-ERROR:error message
02/02/20 02:30:51 AM-testlogger-CRITICAL:critical
Message
```

===
===
================

Python Modules
----
1. random Module:

--> The main intention of random module is to provide some predefined functions to generate random values.
--> In any application, if generate a particular unpredictable value from a range then that value is called as Random Value.
--> In general, we will use random numbers in the application requirements like generating one time password or generating secreate numbers,.....

1. random()
--> It will generate random values of type 'float'.
--> It will generate random values from the range 0 to 1.
EX:

```
import random as r
for x in range(0, 10):
    print(r.random())
```
OP:

```
0.6899582364624999
0.4451755379730873
0.6017774619786778
0.992030959948054
0.3985563128868601
0.2615419679991724
0.5007200094150529
0.8164055389061251
0.07693402570213115
0.49280787024190964
```

2. randint(start,end)
--> It able to generate random values of type int.
--> It will generate random values from the provided start value and upto end value, here both start value

and end values are included.

```
import random as r
for x in range(0, 10):
    print(r.randint(1,10))
OP:
---
9
1
9
6
2
6
9
6
8
1
```

3. uniform(startValue, endValue)
--> It able to generate random values of type float.
--> It will generate random values from the specified range from startValue and upto the endValue, where startVaqlue and EndValues are excluded.
EX:

```
import random as r
for x in range(0,10):
    print(r.uniform(10,20))
OP:
---
12.12485593957307
17.040625436218033
16.04203544198898
10.596479188358215
19.20299599831955
10.078097349450951
15.03077439899118
16.111817388544313
11.72495804863155
14.781072620649283
```

4. randrange(startValue, endValue, stepLength)
--> It will generate random values of type int from the provided startValue and upto the specified endValue on the basis of stepLength.
Ex1:

```
import random as r
for x in range(0,10):
    print(r.randrange(10))
OP:
---
9
1
```

```
3
1
4
7
9
3
2
5
```

StartValue : 0
EndValue: 10
StepLength: 1

EX:

```
import random as r
for x in range(0,10):
    print(r.randrange(1,10))
```

OP:

```
2
1
1
6
5
1
```

```
3
7
6
9
```

StartValue : 1
EndValue: 10
StepLength: 1
EX:

```
import random as r
for x in range(0,10):
    print(r.randrange(1,10,2))
```

OP:

```
3
5
1
9
5
5
9
5
3
9
```

StartValue: 1

EndValue: 10
StepLength: 2

5. choice()
--> It able to generate random values from the provided list.
EX:

```
import random as r
list = ["AAA", "BBB", "CCC", "DDD", "EEE", "FFF"]
for x in range(0, 10):
    print(r.choice(list))
```
OP:

```
DDD
BBB
BBB
AAA
EEE
DDD
BBB
CCC
DDD
CCC
```

datetime module:

--> This module has provided predefined variables and functions to rerpsent date and time in Python applications.
--> In general, we will use date and time to perform operations on the basis of a particular date and a particular time.
EX: To display our system date and time in Python applications.
    To generate count down values for a particular event on a particular date and time.
    To fetch data from a particular start date to end date.
--> In datetime module there are five variables or objects mainly.

1. date
2. time
3. datetime
4. timedelta
5. timezone

--> Where 'date' variable is representing Date from datetime.
--> To get current system date we have to use today() function.
--> In date object,

1. year : To represent year value.
2. month : To represent month value [1 to 12]
3. day : To represent day value [1 to 31 depending

on Month]
EX:

```
from datetime import date
current_Date = date.today()
print(current_Date)
print(current_Date.year)
print(current_Date.month)
print(current_Date.day)
print("To Day
:",current_Date.day,"/",current_Date.month,"/",current_
Date.year)
```

OP:

```
2019-10-15
2019
10
15
To Day : 15 / 10 / 2019
```

In Python applications, we are able to represent a particular date by using date(year, month, day) function
EX:

```
from datetime import date
dob = date(year=1998, month=12, day=12)
```

```
print(dob)
print(dob.year)
print(dob.month)
print(dob.day)
print("DOB :",dob.day,"/",dob.month,"/",dob.year)
```
OP:

```
1998-12-12
1998
12
12
DOB : 12 / 12 / 1998
```

--> In python applications, datetime module is able to represent date in its default format "Year-Month-Day", but, if we want to represent date in our own format then we have to use strftime(date,fmt_in_str) function
EX:

```
from datetime import date
dob = date(year=1998, month=12, day=12)
print(dob)
dob1 = date.strftime(dob,"%d/%m/%y")
print(dob1)
```
OP:

```
1998-12-12
```

```
12/12/98
```

Note: By using strftime() function we are able to convert date value date type to str type.
EX:

```
from datetime import date
dob = date(year=1998, month=12, day=12)
print(dob,"---->",type(dob))
dob1 = date.strftime(dob,"%d/%m/%y")
print(dob1,"------>",type(dob1))
```
OP:

```
1998-12-12 ----> <class 'datetime.date'>
12/12/98 ------> <class 'str'>
```

--> IN Python applications, if we want to convert date value from String to dfatetime type then we have to use a predefined function strptime(str,str_for_Format)
EX:

```
from datetime import datetime
str = "1998-12-10"
print("str Date :",str,"[",type(str),"]")
dt = datetime.strptime(str,"%Y-%m-%d")
date = dt.date()
print("date Date :",date,"[",type(date),"]")
```

OP:

```
str Date : 1998-12-10 [ <class 'str'> ]
date Date : 1998-12-10 [ <class 'datetime.date'> ]
```
Note: IN formation of Dates we have to use the following symbols.

%Y ---> Year
%m ---> Month
%d ---> Day
%H ---> Hour
%M ---> Minute
%S ---> Seconds
%f ---> Micro seconds
%B ---> Month Name

--> By using datetime module, we are able to perform the operations like +, -, <, >,... over the date values.

EX:

```
from datetime import date
today = date.today()
dob = date(year=1998, month=12, day=12)
print("Today :",today)
print("DOB   :",dob)
print("Days  :",(today - dob))
```
OP:

```
Today : 2019-10-15
DOB   : 1998-12-12
Days  : 7612 days, 0:00:00
```

EX:

```
from datetime import date
p1_dob = date(year=2000, month=10, day=12)
p2_dob = date(year=2004, month=7, day=28)
days = p1_dob-p2_dob
print("p1 DOB  :",p1_dob)
print("p2 DOB  :",p2_dob)
print("Age Gap :",days)
if p1_dob < p2_dob:
    print("P1 is Older than P2")
elif p1_dob > p2_dob:
    print("P1 is Younger than P2")
else:
    print("Both are at Same age")
```

OP:

```
p1 DOB  : 2000-10-12
p2 DOB  : 2004-07-28
Age Gap : -1385 days, 0:00:00
P1 is Older than P2
```

--> In python applications, to represent no of days inorder to add or subtract to a particular date we have to use timedelta.
EX:

```
from datetime import date, timedelta
today = date.today()
print("To Day :",today)
days = timedelta(days=100)
afterDate = today + days
print("After 100 Days Date :",afterDate)
beforeDate = today - days
print("Before 100 Days Date :",beforeDate)
```
OP:

```
To Day : 2019-10-15
After 100 Days Date : 2020-01-23
Before 100 Days Date : 2019-07-07
```

time:

--> It object can be used to represent time value in python applications, where time value includes hours, minutes, seconds, micro seconds.
--> To represent a particular time in Python applications we we will use a predefined function time(hr, mnt, sec, microsecond)

EX:

```
t = time(hour=10, minute=35,
second=23,microsecond=100000)
print(t)
```
OP:

```
10:35:23.100000
```

--> IN Python applipcations, we are able to get individual elements in time value like hour, minute, second and microsecond by using the variables like hout, minute, second, microsecond from time object.

EX:

```
from datetime import time
t = time(hour=10, minute=35,
second=23,microsecond=100000)
print(t)
print("Hour :", t.hour)
print("Minut :",t.minute)
print("Second :",t.second)
print("Micro Second :",t.microsecond)
```
OP:

```
10:35:23.100000
```

```
Hour : 10
Minut : 35
Second : 23
Micro Second : 100000
```

--> IN Python applications, we are able to provide time value in our own format by using strftime() function.

EX:

```
from datetime import time
t = time(hour=12, minute=10, second=25,
microsecond=100000)
print(t)
t1 = time.strftime(t,"%H-%M-%S-%f")
print(t1)
```

OP:

```
12:10:25.100000
12-10-25-100000
```

--> In Python applications, we are able to convert time value from time type to str type by using strftime(time, str_For_Formations)

EX:

```
from datetime import time
t = time(hour=10, minute=35,
second=23,microsecond=100000)
print(t,"----->", type(t))
str = time.strftime(t,"%H:%M:%S:%f")
print(str,"----->",type(str))
```
OP:

```
10:35:23.100000 -----> <class 'datetime.time'>
10:35:23:100000 -----> <class 'str'>
```

--> In Python applications, we are able to convert time value from string type to datetime type by using strptime(str, str_For_Format)

EX:

```
from datetime import time, datetime
str = "10:35:20:100000"
print(str,"------>",type(str))
dt = datetime.strptime(str,"%H:%M:%S:%f")
time = dt.time()
print(time,"----->",type(time))
```
OP:

```
10:35:20:100000 ------> <class 'str'>
10:35:20.100000 -----> <class 'datetime.time'>
```

3.datetime

--> It able to represent date and time at a time in python applications.
--> To get current date and time in python applications we have to use now() function from datetime object.
EX:

```
from datetime import datetime
dt = datetime.now()
print(dt)
```
OP:

```
2019-10-16 11:23:48.920714
```

--> In Python applications we are able to get date time values individually by using the variables like year, month, day, hour, minute, second, microsecond from datetime object.
EX:

```
from datetime import datetime
dt = datetime.now()
print("Date Time :",dt)
print("Year :",dt.year)
print("Month :",dt.month)
print("Day :",dt.day)
print("Hour :",dt.hour)
print("Minute :",dt.minute)
print("Second :",dt.second)
print("Micro Second :",dt.microsecond)
```

OP:

```
Date Time : 2019-10-16 11:27:46.069162
Year : 2019
Month : 10
Day : 16
Hour : 11
Minute : 27
Second : 46
Micro Second : 69162
```

--> In Python applications, we are able to provide timedelta operations to datetime object like adding days and hours to the datetime and subtracting days and hours from the datetime.
EX:

```
from datetime import datetime, timedelta
dt = datetime.now()
td = timedelta(days=10, hours=10)
print("Present Date and Time :",dt)
print("After Date And Time   :",(dt+td))
print("Before Date And Time  :", (dt-td))
```

OP:

```
Present Date and Time : 2019-10-16 11:34:13.458291
After Date And Time   : 2019-10-26 21:34:13.458291
Before Date And Time  : 2019-10-06 01:34:13.458291
```

--> In Python applications, by using strftime() function we are able to represent date time values in our own fiormat.
EX:

```
from datetime import datetime
dt = datetime.now()
str_datetime = datetime.strftime(dt, "%Y/%m/%d
%H-%M-%S-%f")
print(str_datetime)
str_dt = datetime.strftime(dt, "%d %B,%Y
%H-%M-%S-%f")
print(str_dt)
```
OP:

```
2019/10/16  11-48-01-683787
16 October,2019  11-48-01-683787
```

--> In python applications, by using datetime we are able to convert date and time from datetime to str time by using strftime() function and we are able to convert

date time valuue from string type to datetime type by using strptime() functions.
EX:

```
from datetime import datetime, timedelta
dt = datetime.now()
print(dt,"----->", type(dt))
str_datetime = datetime.strftime(dt, "%y-%m-%d
%H:%M:%S:%f")
print(str_datetime,"----->", type(str_datetime))
```

OP:

```
2019-10-16 11:40:01.119545 -----> <class
'datetime.datetime'>
19-10-16  11:40:01:119545 -----> <class 'str'>
```

EX:

```
from datetime import datetime, timedelta
str_datetime = "1998-12-10 7:22:35:100000"
print(str_datetime,"---->",type(str_datetime))
dt_datetime = datetime.strptime(str_datetime,
"%Y-%m-%d  %H:%M:%S:%f")
print(dt_datetime,"---->",type(dt_datetime))
```
OP:

```
1998-12-10 7:22:35:100000 ----> <class 'str'>
1998-12-10 07:22:35.100000 ----> <class
'datetime.datetime'>
```

timezone:

-->In general date and time values are varied from one region to another region, to represent date and time values as per the region we have to use timezones in python applications.
Note: Python has provided timezone and tzinfo classes to repersent timezones, but, these are not provide date and time values accurately. To get Date and time values accurately we have to use third party libraries like "pytz".

To use pytz in our applications we have to use the following steps.
1. Download pytz from internet.
   a)search for get-pip.py in google.
   b)Click on get-pip.py link in any website.
   c)Save file at any location ion out computer by clicking on ctr-s.
   d)Goto the location where we saved get-pip.py file and double click on it.
2. INstall Pytz in PyCharm:
   a)Open Terminal in pyCharm.
   b)Use the following command to get Pytz.

```
        pip install pytz
```

To use a particular timezone in our python applications then we have to use the following steps.

1.Create Timezone object.
```
   tz = timezone("time_zone_value")
```
2.Apply timezone to our date and time by using astimezone() function.
```
   date = dt.astimezone(tz);
```

EX:

```
from datetime import datetime
from pytz import timezone
dt = datetime.today()
print("Default Timezone Time :",dt)
tz1 = timezone("Asia/Kolkata")
print("Asia/Kolkata timezone Time :",dt.astimezone(tz1))
tz2 = timezone("US/Eastern")
print("US/Eastern Timezone Time :",dt.astimezone(tz2))
tz3 = timezone("US/Mountain")
print("US/Mountain Timezone Time
:",dt.astimezone(tz3))
```

OP:

```
Default Timezone Time : 2019-10-16 12:15:48.247228
Asia/Kolkata timezone Time : 2019-10-16
12:15:48.247228+05:30
US/Eastern Timezone Time : 2019-10-16
02:45:48.247228-04:00
US/Mountain Timezone Time : 2019-10-16
00:45:48.247228-06:00
```

EX:

```
import pytz
for x in pytz.all_timezones:
    print(x,end="  ")
```

OP:

```
America/Tijuana  America/Toronto  America/Tortola
America/Vancouver  America/Virgin  America/Whitehorse
 America/Winnipeg  America/Yakutat
America/Yellowknife  Antarctica/Casey  Antarctica/Davis
Antarctica/DumontDUrville  Antarctica/Macquarie
Antarctica/Mawson  Antarctica/McMurdo
Antarctica/Palmer  Antarctica/Rothera
Antarctica/South_Pole  Antarctica/Syowa
Antarctica/Troll  Antarctica/Vostok  Arctic/Longyearbyen
Asia/Aden  Asia/Almaty  Asia/Amman  Asia/Anadyr
Asia/Aqtau  Asia/Aqtobe  Asia/Ashgabat  Asia/Ashkhabad
```

Asia/Atyrau Asia/Baghdad Asia/Bahrain Asia/Baku
Asia/Bangkok Asia/Barnaul Asia/Beirut Asia/Bishkek
Asia/Brunei Asia/Calcutta Asia/Chita Asia/Choibalsan
Asia/Chongqing Asia/Chungking Asia/Colombo
Asia/Dacca Asia/Damascus Asia/Dhaka Asia/Dili
Asia/Dubai Asia/Dushanbe Asia/Famagusta Asia/Gaza
Asia/Harbin Asia/Hebron Asia/Ho_Chi_Minh
Asia/Hong_Kong Asia/Hovd Asia/Irkutsk Asia/Istanbul
Asia/Jakarta Asia/Jayapura Asia/Jerusalem Asia/Kabul
Asia/Kamchatka Asia/Karachi Asia/Kashgar
Asia/Kathmandu Asia/Katmandu Asia/Khandyga
Asia/Kolkata Asia/Krasnoyarsk Asia/Kuala_Lumpur
Asia/Kuching Asia/Kuwait Asia/Macao Asia/Macau
Asia/Magadan Asia/Makassar Asia/Manila Asia/Muscat
Asia/Nicosia Asia/Novokuznetsk Asia/Novosibirsk
Asia/Omsk Asia/Oral Asia/Phnom_Penh Asia/Pontianak
Asia/Pyongyang Asia/Qatar Asia/Qostanay
Asia/Qyzylorda Asia/Rangoon Asia/Riyadh Asia/Saigon
Asia/Sakhalin Asia/Samarkand Asia/Seoul
Asia/Shanghai Asia/Singapore Asia/Srednekolymsk
Asia/Taipei Asia/Tashkent Asia/Tbilisi Asia/Tehran
Asia/Tel_Aviv Asia/Thimbu Asia/Thimphu Asia/Tokyo
Asia/Tomsk Asia/Ujung_Pandang Asia/Ulaanbaatar
Asia/Ulan_Bator Asia/Urumqi Asia/Ust-Nera
Asia/Vientiane Asia/Vladivostok Asia/Yakutsk
Asia/Yangon Asia/Yekaterinburg Asia/Yerevan
Atlantic/Azores Atlantic/Bermuda Atlantic/Canary

Atlantic/Cape_Verde Atlantic/Faeroe Atlantic/Faroe
Atlantic/Jan_Mayen Atlantic/Madeira Atlantic/Reykjavik
Atlantic/South_Georgia Atlantic/St_Helena
Atlantic/Stanley Australia/ACT Australia/Adelaide
Australia/Brisbane Australia/Broken_Hill
Australia/Canberra Australia/Currie Australia/Darwin
Australia/Eucla Australia/Hobart Australia/LHI
Australia/Lindeman Australia/Lord_Howe
Australia/Melbourne Australia/NSW Australia/North
Australia/Perth Australia/Queensland Australia/South
Australia/Sydney Australia/Tasmania Australia/Victoria
Australia/West Australia/Yancowinna Brazil/Acre
Brazil/DeNoronha Brazil/East Brazil/West CET
CST6CDT Canada/Atlantic Canada/Central
Canada/Eastern Canada/Mountain
Canada/Newfoundland Canada/Pacific
Canada/Saskatchewan Canada/Yukon Chile/Continental
Chile/EasterIsland Cuba EET EST EST5EDT Egypt
Eire Etc/GMT Etc/GMT+0 Etc/GMT+1 Etc/GMT+10
Etc/GMT+11 Etc/GMT+12 Etc/GMT+2 Etc/GMT+3
Etc/GMT+4 Etc/GMT+5 Etc/GMT+6 Etc/GMT+7
Etc/GMT+8 Etc/GMT+9 Etc/GMT-0 Etc/GMT-1
Etc/GMT-10 Etc/GMT-11 Etc/GMT-12 Etc/GMT-13
Etc/GMT-14 Etc/GMT-2 Etc/GMT-3 Etc/GMT-4
Etc/GMT-5 Etc/GMT-6 Etc/GMT-7 Etc/GMT-8
Etc/GMT-9 Etc/GMT0 Etc/Greenwich Etc/UCT Etc/UTC
Etc/Universal Etc/Zulu Europe/Amsterdam

Europe/Andorra Europe/Astrakhan Europe/Athens
Europe/Belfast Europe/Belgrade Europe/Berlin
Europe/Bratislava Europe/Brussels Europe/Bucharest
Europe/Budapest Europe/Busingen Europe/Chisinau
Europe/Copenhagen Europe/Dublin Europe/Gibraltar
Europe/Guernsey Europe/Helsinki Europe/Isle_of_Man
Europe/Istanbul Europe/Jersey Europe/Kaliningrad
Europe/Kiev Europe/Kirov Europe/Lisbon
Europe/Ljubljana Europe/London Europe/Luxembourg
Europe/Madrid Europe/Malta Europe/Mariehamn
Europe/Minsk Europe/Monaco Europe/Moscow
Europe/Nicosia Europe/Oslo Europe/Paris
Europe/Podgorica Europe/Prague Europe/Riga
Europe/Rome Europe/Samara Europe/San_Marino
Europe/Sarajevo Europe/Saratov Europe/Simferopol
Europe/Skopje Europe/Sofia Europe/Stockholm
Europe/Tallinn Europe/Tirane Europe/Tiraspol
Europe/Ulyanovsk Europe/Uzhgorod Europe/Vaduz
Europe/Vatican Europe/Vienna Europe/Vilnius
Europe/Volgograd Europe/Warsaw Europe/Zagreb
Europe/Zaporozhye Europe/Zurich GB GB-Eire GMT
GMT+0 GMT-0 GMT0 Greenwich HST Hongkong
Iceland Indian/Antananarivo Indian/Chagos
Indian/Christmas Indian/Cocos Indian/Comoro
Indian/Kerguelen Indian/Mahe Indian/Maldives
Indian/Mauritius Indian/Mayotte Indian/Reunion Iran
Israel Jamaica Japan Kwajalein Libya MET MST

MST7MDT Mexico/BajaNorte Mexico/BajaSur
Mexico/General NZ NZ-CHAT Navajo PRC PST8PDT
Pacific/Apia Pacific/Auckland Pacific/Bougainville
Pacific/Chatham Pacific/Chuuk Pacific/Easter
Pacific/Efate Pacific/Enderbury Pacific/Fakaofo
Pacific/Fiji Pacific/Funafuti Pacific/Galapagos
Pacific/Gambier Pacific/Guadalcanal Pacific/Guam
Pacific/Honolulu Pacific/Johnston Pacific/Kiritimati
Pacific/Kosrae Pacific/Kwajalein Pacific/Majuro
Pacific/Marquesas Pacific/Midway Pacific/Nauru
Pacific/Niue Pacific/Norfolk Pacific/Noumea
Pacific/Pago_Pago Pacific/Palau Pacific/Pitcairn
Pacific/Pohnpei Pacific/Ponape Pacific/Port_Moresby
Pacific/Rarotonga Pacific/Saipan Pacific/Samoa
Pacific/Tahiti Pacific/Tarawa Pacific/Tongatapu
Pacific/Truk Pacific/Wake Pacific/Wallis Pacific/Yap
Poland Portugal ROC ROK Singapore Turkey UCT
US/Alaska US/Aleutian US/Arizona US/Central
US/East-Indiana US/Eastern US/Hawaii
US/Indiana-Starke US/Michigan US/Mountain
US/Pacific US/Samoa UTC Universal W-SU WET Zulu

Math Module:

Math module is providing predefined variables and functions inorder to perform mathematgical and

Scientific calculations in Python applications.
EX: Calculating Trignometric Operations.
    Calculating Squere root for numbers.
    Calculating Factorial functions for numbers

Variables:
----

1. math.pi
--> It will represent mathetical constrant 'π' and its value is 3.141592653589793

2. math.e
--> It will represent mathematical constant 'e' and its value is 2.718281828459045

3. math.tau
--> It will represent mathematcal constant 'τ' and its value is 6.283185307179586, it is equals to 2π

EX:

```
import math
print(math.pi)
print(math.e)
print(math.tau)
```

OP:

```
3.141592653589793
2.718281828459045
6.283185307179586
```

Functions:
----
1. math.ceil(x)
--> It will return a smallest integer which is greater than or equals to x.
EX:

```
import math
x = 7.3
print(math.ceil(x))
```
OP:

```
8
```

2. math.floor(x)
--> It wil return largest integer which is less than or equals to X.
EX:

```
import math
x = 7.3
```

```
print(math.floor(x))
```
OP:

```
7
```

3. math.factorial(x)
--> It will return factorial value of x.
EX:

```
import math
x = 4
print(math.factorial(x))
```
OP:

```
24
```

factorial(4) = 1*2*3*4 = 24

4. math.fabs(x)
--> it will return absolute value of x.
EX:

```
import math
x = -4
y = 5
print(math.fabs(x))
print(math.fabs(y))
```

```
OP:
---
4
5
```

Note: Absolute value is the distance from 0 upto the specified value in numbers scale irrespective of the direction.

5. math.fmod(x,y)
--> It will perform modulo operation like x%y

```
EX:
---
import math
x = 10
y = 3
print(math.fmod(x,y))
OP:
---
1.0
```

6. math.fsum(list)
--> It will add all the elements of the provided list and return the result value.

```
EX:
---
import math
list = [1,2,3,4,5,6]
```

```
print(math.fsum(list))
Op:
--
21.0
```

7. math.gcd(a,b)
--> It will return the greatest common divisor of the integers a and b.
EX:

```
import math
x = 8
y = 10
print(math.gcd(x,y))
OP:
---
2
```

8.math.sqrt(x)
--> It will retun sque root of x.
EX:

```
import math
x = 4
print(math.sqrt(x))
OP:
---
```

```
2.0
```
Note: sqrt(4) = $4^{1/2}$

9. math.remainder(x,y)
--> It will return remainder of x/y.
Ex:
```
----
import math
x = 10
y = 3
print(math.remainder(x,y))
OP:
---
1.0
```

10.math.trunc(x)
--> It will return truncated value , that is, integral value of the provided float value.
EX:
```
---
import math
x = 5.9
print(math.trunc(x))
OP:
---
5
```

11. math.exp(x)
--> It will return exponential of x, that is, e^x
EX:

```
import math
x = 3
print(math.exp(x))
```
OP:

```
20.085536923187668
```

12.math.log(x,base)
--> It will return log value of x with the provided base.
EX:

```
import math
x = 10
print(math.log(x,2))
print(math.log(x,3))
print(math.log(x,10))
```
OP:

```
3.3219280948873626
2.095903274289385
1.0
```

Note:We can use the functins like log1p(x), log2(x), log10(x) to accurate values.
EX:
```
---
import math
x = 10
print(math.log(x,2))
print(math.log(x,3))
print(math.log(x,10))
print()
print(math.log1p(x))
print(math.log2(x))
print(math.log10(x))
```
OP:
```
---
3.3219280948873626
2.095903274289385
1.0

2.3978952727983707
3.321928094887362
1.0
```

13.math.pow(x,y)   y
-->It will calculate power operation between x and y, that is, x
EX:

```
---
import math
x = 2
y = 4
print(math.pow(x,y))
```
OP:
```
---
16.0
```

14.math.degrees(x)
-->It will return the value in the form of degree.
1 degree = pi/180 radians

15.math.radians(x)
--> it will return in the form of radians.
EX:
```
---
import math
print(math.radians(30))
print(math.degrees(0.5235987755982988))
```
OP:
```
---
0.5235987755982988
29.999999999999996
```

16. sin(x), cos(x), tan(x)
--> These will perfor Trignoimetric functions.

EX:

```
import math
print(math.sin(30))
print(math.cos(30))
print(math.tan(30))
```

OP:

```
-0.9880316240928618
0.15425144988758405
-6.405331196646276
```

NumPy Module

--> The main intetion of NumPy module to provide Multi dimentional array object and   tools for working with these arrays.
--> NumPy is core library or module for scientific Arrays computing in Python.
--> NumPy contains N-Dimensional arrays and tools for integrating with C and C++
--> NumPy is used for linear Alzeebra with random numbers capabilities.
--> NumPy can be used for Multi Dimensaionl Container for generic data.

Array: It is a data structer, it able to allow group of elements of same data type.
IN general, two types of arrays.
1. Single Dimensional Arrays
2. Multi Dimensional Arrays.

Single Dimensional Array: It is an array ,it contains only one Row.
EX:
```
   0 1 2 3 4
0 [1,2,3,4,5]
    A[0,0] = 1
    A[0,1] = 2
    A[0,2] = 3
    A[0,3] = 4
    A[0,4] = 5
```

Multi Dimensional Array:It conatains data in more than one domension.
EX:2 Dimensional Array
```
   0 1 2 3 4
0 [1,2,3,4,5]
1 [2,3,4,5,6]
2 [3,4,5,6,7]
3 [4,5,6,7,8]
```

A[0,3] = 4

A[1,2] = 4
A[3,3] = 7

To install NumPy in PyCharm we have to use the following steps.
a)Select "File"
b)Select "Settings".
c)Select "Project"
d)Select "Project Interpretor".
e)Click on '+' symbol at right side.
f)Type 'NumPy' in Search bar.
g)Click "Install Package" button
h)Close "packages" window
i)Click On "ok" button.

-->To Use Numpy in Python apllications, first, we have to import Numpy module to our python file.
EX: import numpy as np

-->To create an array by using numpy we will use a function array().
EX:
```
import numpy as np
a = np.array([1,2,3,4])
print(a)
```
OP:

```
[1 2 3 4]
```

--> To create Multi dimentional arrays we have to use tuples as dimentions in list.

EX:2-Dimensional Array

```
import numpy as np
a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6),(4,5,6,7)])
print(a)
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]]
```

EX: 3-Dimensional Array

```
import numpy as np
a =
np.array([[(1,2,3),(2,3,4)],[(2,3,4),(3,4,5)],[(3,4,5),(4,5,
6)],[(4,5,6),(5,6,7)]])
print(a)
```

OP:

```
[[[1 2 3]
  [2 3 4]]

 [[2 3 4]
  [3 4 5]]

 [[3 4 5]
  [4 5 6]]

 [[4 5 6]
  [5 6 7]]]
```

Q)To provide array of elements we have already List, tuple,range,... then what is the requirement to use NumPy?

Ans:

Numpy is more powerfuill than List, tuple,range,.... because of the following three reasons.

1. Less Memory
2. Faster
3. Convenient

1. Numpy will take "Less Memory" when compared with List, tuple, range,....
EX:

```
import numpy as np, sys

a = range(1000)
print(sys.getsizeof(a)*len(a))

b = np.arange(1000)
print(b.size*b.itemsize)
```

OP:

```
48000
4000
```

Where "sys.getsizeof(a)" will return size of single element.
Where "sys.getsizeof(a)*len(a)" will return all the elements size which are available in range.
Where "b.itemsize" variable is able to return single item size.
Where "b.size" will return the no of elements in range.
Where "b.size*b.itemsize" will return the size of all the elements available in range.

2. Numpy is more Faster than list, tuple, range,.....
EX:

```
import numpy as np, time

size = 10000000
r1 = range(size)
r2 = range(size)

range_Start_Time = time.time()
range_result = [(x,y) for x,y in zip(r1,r2)]
range_End_Time = time.time()

print("Range Processing Time : ", (range_End_Time -
range_Start_Time)*1000)

a1 = np.arange(size)
a2 = np.arange(size)

np_Start_Time = time.time()
np_Result = a1 + a2
np_End_Time = time.time()

print("Numpy Processing Time : ", (np_End_Time -
np_Start_Time)*1000)
```

OP:

```
Range Processing Time :  1984.4019412994385
Numpy Processing Time :  31.253576278686523
```

3.NumPy is more Convenient than the data types like List, Tuple, Range,...., because, NumPy had very good predefined library manipulate elements like getting different shapes, identifying elements types,.....

--
--> In Numpy, dimensions are called as "axis"
EX:

```
[[1,2,3]
 [2,3,4] ]
```

In the above array, array has 2 axis, where first axis has three elements they are 1,2,3 and second axis has three elements they are 2,3,4 .

Variables:
----
1. np.ndim:

--> In NumPy , to get no of axis of an array we have to use np.ndim

EX:

```
import numpy as np
a = np.array([(1,2,3),(2,3,4)])
print(a)
print(a.ndim)
```
OP:

```
[[1 2 3]
 [2 3 4]]
2
```

2.np.shape:

--> In NumPy, to get dimensions of an array or matrix like (n,m) no of rows and no of columns we have to use np.shape.
EX:

```
import numpy as np
a = np.array([(1,2,3),(2,3,4)])
print(a)
print(a.shape)
```

OP:

```
[[1 2 3]
```

```
 [2 3 4]]
(2, 3)
```

3.np.size:

--> In NumPy, to get total no of elements in an array we have to use np.size.
EX:

```
import numpy as np
a = np.array([(1,2,3),(2,3,4),(3,4,5)])
print(a)
print(a.size)
```
OP:

```
[[1 2 3]
 [2 3 4]
 [3 4 5]]
9
```

np.dtype

--> To get data type of the elements inside array we have to use np.dtype variable.
EX:

```
import numpy as np
```

```
a = np.array([(1,2,3),(2,3,4),(3,4,5)])
print(a)
print(a.dtype)
```
OP:

```
[[1 2 3]
 [2 3 4]
 [3 4 5]]
int32
```

Note: Numpy has provided its own data types for the elements in arrays, they are, int32, int64, float64,...
EX:

```
import numpy as np
a = np.array([(1.0,2.0,3.0),(2.0,3.0,4.0),(3.0,4.0,5.0)])
print(a)
print(a.dtype)
```
OP:
```
[[1. 2. 3.]
 [2. 3. 4.]
 [3. 4. 5.]]
float64
```

Note: In Numpy, we can provide our own data types to the array elements by using np.dtype property in array() function.

EX:

```
import numpy as np
a = np.array([(1,2,3),(2,3,4),(3,4,5)], dtype=np.int64)
print(a)
print(a.dtype)
```
OP:

```
[[1 2 3]
 [2 3 4]
 [3 4 5]]
int64
```

np.itemsize

--> In Numpy, we are able to get individual item sizes from an array by using np.itemsize variable.
int32,complex32 ----> 4 [32/8]
int 64, float 64 ---> 8 [64/8]
EX:

```
import numpy as np
a = np.array([(1,2,3),(2,3,4),(3,4,5)], dtype=np.int32)
print(a.itemsize)

b = np.array([(1,2,3),(2,3,4),(3,4,5)], dtype=np.int64)
print(b.itemsize)
```

```
c = np.array([(1,2,3),(2,3,4),(3,4,5)], dtype=np.float)
print(c.itemsize)

d = np.array([(1,2,3),(2,3,4),(3,4,5)],
dtype=np.complex)
print(a.itemsize)
```

OP:

```
---
4
8
8
4
```

Creating Arrays in Numpy:

In Numpy we are able to create arrays in the following five ways.

1. By using array() function
2. By using zeros() function
3. By using ones() function
4. By Using empty() function
5. By Using arange() function
6. By using linespace() function
7. By using fromfunction() function

1. By using array() function:

--> It will create an element wit hthe specified elements.
EX:
```
import numpy as np
a = np.array([(1,2,3),(2,3,4),(3,4,5)])
print(a)
```
OP:

```
[[1 2 3]
 [2 3 4]
 [3 4 5]]
```

2. By using zeros() function

--> It will create an array with only zeros as elements.
EX:
```
import numpy as np
a = np.zeros((3,3))
print(a)
```
OP:

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Note: zeros() function will create arrays with 0s of type float, but, if we want to provide in our own data type then we have to 'dtype' variable in zeros() function.
EX:

```
import numpy as np
a = np.zeros((3,3),dtype=np.int32)
print(a)
```
OP:

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

EX:

```
import numpy as np
a = np.zeros((3,3),dtype=np.complex)
print(a)
```

OP:

```
[[0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j]]
```

3. By using ones() function

--> It able to create an array with onlt 1's as elements as per the provided shape.

EX:

```
import numpy as np
a = np.ones((3,3))
print(a)
```

OP:

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Note: By default, ones() function is able to generate an array with 1's as elements of type float, but, if we want to provide our own type then we have to use 'dtype' variable in ones() function.

EX:

```
import numpy as np
a = np.ones((3,3), dtype=np.int64)
print(a)
```

OP:

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

EX:

```
import numpy as np
a = np.ones((3,3), dtype=np.complex)
print(a)
```

OP:

```
[[1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j]
 [1.+0.j 1.+0.j 1.+0.j]]
```

4. By Using empty() function:

--> It can be used to create an array with empty values like 0.0000000 .

EX:

```
import numpy as np
a = np.empty((3,3))
print(a)
```

OP:

```
[[0.00000000e+000 0.00000000e+000
0.00000000e+000]
```

```
 [0.00000000e+000 0.00000000e+000
2.05531309e-321]
 [1.42410974e-306 2.22522596e-306
4.94065646e-322]]
```

Note: If we run the above program multiple times then Output will be varied.

5. By Using arange() function

--> It can be used to generate array as per the specified startvalue , endvalue and stepLength.
a)np.arange(enValue)
--> It will create an array with the elements from default startValue 0 to endValue-1 with the defazult stepLength 1.
EX:

```
import numpy as np
a = np.arange(10)
print(a)
```

OP:

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

b)np.arange(startValue,endValue)
--> It will create an array with the elements from

startValue to endValue-1 with the default stepLength 1
EX:

```
import numpy as np
a = np.arange(1,10)
print(a)
```
OP:

```
[1 2 3 4 5 6 7 8 9]
```

c)np.arange(startValue, endValue, stepLength)
--> It will create an array with the elements from startValue to endValue-1 with the provided step length.
EX:

```
import numpy as np
a = np.arange(1,10,2)
print(a)
```
OP:

```
[1 3 5 7 9]
```

6. By using linspace() function

--> It is same as arange() function , but, linespace() function will not take stepLength as third parameter, it will take no_of_elements as third parameter.

Syntax: np.linespace(startValue, endValue, no_Of_Elements_between_startValue_and_endValue)
EX:

```
import numpy as np
a = np.linspace(1,20,10)
print(a)
```
OP:

```
[ 1.          3.11111111  5.22222222  7.33333333
9.44444444 11.55555556 13.66666667 15.77777778
17.88888889 20.        ]
```

7. By using fromfunction() function:

```
import numpy as np

def getElement(x, y):
    return x*y

a = np.fromfunction(getElement, (5, 5), dtype=np.int32)
print(a)
```

OP:

```
[[ 0  0  0  0  0]
```

```
 [ 0  1  2  3  4]
 [ 0  2  4  6  8]
 [ 0  3  6  9 12]
 [ 0  4  8 12 16]]
```

Displaying array elements:

To display elements from array, Numpy will use the following order.
1. Last axis is printed from left to right.
2. The second to last axis is printed from tom to bottom.
3. Remaing axis are printed from top to bottom.

In Numpy, we are able to display the array elements in the following forms.
1. Single Dimensional Array elements will be displayed in single row as list.
2. 2-Dimensional arrays elements are displayed in Rows and Columns , where each row is displayed as list.
3. 3-Dimensional arrays elements are displayed in list of matrices.

np.reshape() function:

--> It can be used to create array in our own shape.

EX:

```
import numpy as np
a = np.arange(6)
print(a)
```
OP:

```
[0 1 2 3 4 5]
```

EX:

```
import numpy as np
a = np.arange(6).reshape(2,3)
print(a)
```
OP:

```
[[0 1 2]
 [3 4 5]]
```

EX:

```
import numpy as np
a = np.arange(15).reshape(3,5)
print(a)
```
OP:

```
[[ 0  1  2  3  4]
```

```
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

EX:

```
import numpy as np
a = np.arange(24).reshape(2,3,4)
print(a)
```
OP:

```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Note: In array, if the elements are more then NumPy will display all the elemnts with .... in middle.
Ex:

```
import numpy as np
a = np.arange(10000).reshape(100,100)
print(a)
```
op:

```
[[   0    1    2 ...   97   98   99]
 [ 100  101  102 ...  197  198  199]
 [ 200  201  202 ...  297  298  299]
 ...
 [9700 9701 9702 ... 9797 9798 9799]
 [9800 9801 9802 ... 9897 9898 9899]
 [9900 9901 9902 ... 9997 9998 9999]]
```

--> If we want to arrange all the elements of an array with fixed no of rows then we have to use -1 arguiment in reshape() function.

EX:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6),(4,5,6,7)])
print(a)
print()
print(a.reshape(2,-1))
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]]
```

```
[[1 2 3 4 2 3 4 5]
 [3 4 5 6 4 5 6 7]]
```

--> If we want to get Transpose array or matrices[interchange rows to columns and columns to rows] then we have to use 'T' operator.
EX:

```
import numpy as np

a =
np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12),(13,14,15,16)]
)
print(a)
print("a Dimension : ",a.shape)

print()
print(a.T)
print("a.T Dimension : ",a.T.shape)
```
OP:

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
a Dimension :  (4, 4)
```

```
[[ 1  5  9 13]
 [ 2  6 10 14]
 [ 3  7 11 15]
 [ 4  8 12 16]]
a.T Dimension :  (4, 4)
```

Q)What is the difference between resize() function and reshape() function?

Ans:

1.resize() function will change size on the existed array, but, reshape() will change the size on different   array, not on the same array.

2. resize() function will return None type, but, reshape() function will return array type.

EX:

```
import numpy as np

a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print(a,"    Shape :",a.shape,"    id :",id(a))
print()
b= a.resize(2,6)
```

```
print(b)
print(a)

OP:
----
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]     Shape : (3, 4)     id : 2228842448496

None
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]

EX:
---
import numpy as np

a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print("a Data : ",a,"   a Shape :",a.shape,"   a id :",id(a))
print()
b= a.reshape(2,6)
print("b Data : ",b,"   b Shape :",b.shape,"   b id :",id(b))
print()
print("a Data : ",a,"   a Shape :",a.shape,"   a id :",id(a))

OP:
---
```

```
a Data :  [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]    a Shape : (3, 4)    a id :
1379653830256

b Data :  [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]    b Shape : (2, 6)    b id :
1379659295920

a Data :  [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]    a Shape : (3, 4)    a id :
1379653830256
```

--> To get all the elements in single row, that is , flattened, then we have to use ravel() function.
EX:

```
import numpy as np

a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print("a Data : ",a,"   a Shape :",a.shape,"   a id :",id(a))
print()
b= a.ravel()
print("b Data : ",b,"   b Shape :",b.shape,"   b id :",id(b))
```
op:

```
a Data :  [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]    a Shape : (3, 4)    a id :
2749935830640

b Data :  [ 1  2  3  4  5  6  7  8  9 10 11 12]    b Shape :
(12,)    b id : 2749941296304
```

Array Stacking Numpy:

The process of combining more than one array into single arry is called as Array Stacking.
There are Two types of stackings in numpy.
1. Horizontal Stacking
2. Vertical Stacking
3. Row Stacking
4. Column Stacking
1. Horizontal Stacking

It will take all corresponding axis elements from botrh the arrays and generate as row.
To perform Horizontal stacking we will use the functions like hstack()

EX:

```
a = [a,b]
    [c,d]
b = [e,f]
    [g,h]

c = np.hstack(a,b)
Where c is[ [a,b,e,f]
            [c,d,g,h] ]
```

2. Vertical Stacking

It will take corresponding axis elments from both the arrays and generate as columns.
To perform Vertical Stacking we will use the functions like vstack().
EX:

```
a = [a,b]
    [c,d]
b = [e,f]
    [g,h]

c = np.vstack(a,b) or np.row_stack(a,b)
Where c is [ [a,b]
             [c,d]
```

```
            [e,f]
            [g,h] ]
```

Note: Vertical Stacking And row_stacking are same.

4.Column Stacking:

```
a = [a,b]
    [c,d]
b = [e,f]
    [g,h]

c = np.column_stack(a,b)
Where c is [ [a,b,e,f]
             [c,d,g,h]]
```

EX:

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
print(np.hstack((a,b)))
print()
print(np.vstack((a,b)))
print()
print(np.column_stack((a,b)))
print()
print(np.row_stack((a,b)))
```

OP:

```
[1 2 3 4 5 6 7 8]

[[1 2 3 4]
 [5 6 7 8]]

[[1 5]
 [2 6]
 [3 7]
 [4 8]]

[[1 2 3 4]
 [5 6 7 8]]
```

EX:

```
import numpy as np

a = np.array([(10,20,30,40), (50,60,70,80)])
b = np.array([(5,15,25,35),(45,55,65,75)])
```

```
print(np.hstack((a,b)))
print()
print(np.vstack((a,b)))
print()
print(np.column_stack((a,b)))
print()
print(np.row_stack((a,b)))
```

OP:

```
---
[[10 20 30 40  5 15 25 35]
 [50 60 70 80 45 55 65 75]]

[[10 20 30 40]
 [50 60 70 80]
 [ 5 15 25 35]
 [45 55 65 75]]

[[10 20 30 40  5 15 25 35]
 [50 60 70 80 45 55 65 75]]

[[10 20 30 40]
 [50 60 70 80]
 [ 5 15 25 35]
 [45 55 65 75]]
```

Spring single Array into no of smaller arrays:

To split single array into no of smaller arrays, we have to use the following functions.

1. hsplit()
2. vsplit()

1. hsplit():

It will split horizontally.

EX:

```
import numpy as np

a =
np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12),(13,14,15,16),
(17,18,19,20)])
print(a)
print()
print(np.hsplit(a,4))
```

OP:

```
[array([[ 1],
       [ 5],
       [ 9],
       [13],
       [17]]), array([[ 2],
       [ 6],
       [10],
       [14],
       [18]]), array([[ 3],
       [ 7],
       [11],
       [15],
       [19]]), array([[ 4],
       [ 8],
       [12],
       [16],
       [20]])]
```

2. vsplit()

It will split vertically.

EX:

```
import numpy as np

a =
np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12),(13,14,15,16),
(17,18,19,20)])
print(a)
print()
print(np.vsplit(a,5))
OP:
---
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]

[array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]]), array([[ 9, 10,
11, 12]]), array([[13, 14, 15, 16]]), array([[17, 18, 19,
20]])]
```

Arrays Copying:

IN Numpy, it is possible to copy elements from one array to another array with the following cases.

1. Assign one array reference variable to another reference variable.
2. By using view() function
3. By using copy() function

1. Assign one array reference variable to another reference variable:

-

If we assign one array reference to another array reference then new array object is not created, where the same array reference will be shred to another reference variable.

```
import numpy as np
EX:
---
a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print(a)
print(id(a))
print()
b = a
print(b)
print(id(b))
```

OP:

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
1763291481712

[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
1763291481712
```

2. By using view() function

IN array copying, view() function will create new array with the same elements.
view() function will perform Shallow copy, that is, both the arrays are having same base.
EX:
```
import numpy as np

import numpy as np

a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print(a)
print(id(a))
print()
b = a.view()
print(b)
print(id(b))

print(b.base is a)
```

OP:

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
3094910584432
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
3094916050096
True
```

3. By using copy() function

To compy array, if we use copy() function then it will copy the elements from one array to another array but it will provide seperate copy of base and its elements in another array, this type of copy is called as Deep Copy.

```
import numpy as np

a = np.array([(1,2,3,4),(5,6,7,8),(9,10,11,12)])
print(a)
print(id(a))
print()
b = a.copy()
print(b)
print(id(b))

print(b.base is a)
```

OP:

```
---
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
2284921734768

[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
2284922154160
False
```

Retrieving elements from arrays in Numpy:

In Numpy, we are able to retrieve elements from Arrays by using the following ways.

1. By using indexes.
2. By using Slice operator.
3. By Using Iterator.

1. By using indexes:

To retrieve elements from arrays in Numpy we are able to use both +ve index values and -ve index values, in

forward direction [+ve] index values are started from 0 and in backward direction[-ve] index values are started with -1.

arra_Ref[indexValue]
EX:

```
import numpy as np

a = np.array([1,2,3,4])
print(a)
print(a[0])
print(a[1])
print(a[-2])
print(a[-3])
```
OP:

```
[1 2 3 4]
1
2
3
2
```

EX:

```
import numpy as np
```

```
a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6)])
print(a)
print(a[0,1])
print(a[1,2])
print(a[-2,3])
print(a[-3,2])
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
2
4
5
3
```

EX:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6)])
print(a)
print()
for x in a:
    for y in x:
        print(y, end=" ")
    print()
OP:
---
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]

1 2 3 4
2 3 4 5
3 4 5 6
```

EX:

```
import numpy as np
a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6)])
print(a)
print()
for x in range(0,len(a)):
    for y in range(0,len(a[x])):
        print(a[x,y], end=" ")
    print()
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
```

```
1 2 3 4
2 3 4 5
3 4 5 6
```

2. By using Slice operator

To retrieve the elements by using slice operator we have to use the following formats.
1. 1-Dimensional array: arrayRef[start:end:stepLength]
EX:

```
import numpy as np
a = np.arange(1,10)
print(a[2:8:1])
print(a[1:10:2])
print(a[3:7:])
print(a[2::])
print(a[::])
print(a[3:-3:1])
print(a[-1:-8:-1])
print(a[-1::-1])
print(a[::-1])
print(a[-1:2:-1])
```
OP:

```
[3 4 5 6 7 8]
```

```
[2 4 6 8]
[4 5 6 7]
[3 4 5 6 7 8 9]
[1 2 3 4 5 6 7 8 9]
[4 5 6]
[9 8 7 6 5 4 3]
[9 8 7 6 5 4 3 2 1]
[9 8 7 6 5 4 3 2 1]
[9 8 7 6 5 4]
```

2. 2-Dimensional Array:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6)])
print(a)
print(a[0:2:1,1:3:1])
print(a[0:-1,0:3])
print(a[-1:-4:-1,-1:-4:-1])
print(a[-1:-3:-1,-1:-3:-1])
print(a[::-1, ::-1])
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
[[2 3]
```

```
 [3 4]]
[[1 2 3]
 [2 3 4]]
[[6 5 4]
 [5 4 3]
 [4 3 2]]
[[6 5]
 [5 4]]
[[6 5 4 3]
 [5 4 3 2]
 [4 3 2 1]]
```

In Slice operator , we can use ... notation to get the remaining indecies.
EX:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6)])
print(a)
print(a[1,...])
```

OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
```

```
[2 3 4 5]
```

3. By Using Iterator:

In Numpy, we are able to iterate elements from an array in the following ways.

1. In the form of rows depending on indeces.
EX:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6),(4,5,6,7)])
print(a)
print()
for row in a:
    print(row)
```
OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]]

[1 2 3 4]
[2 3 4 5]
[3 4 5 6]
```

```
[4 5 6 7]
```

2. In the form of direct elements by using 'flat' variable.
EX:

```
import numpy as np

a = np.array([(1,2,3,4),(2,3,4,5),(3,4,5,6),(4,5,6,7)])
print(a)
print()
for x in a.flat:
    print(x)
```
OP:

```
[[1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]
 [4 5 6 7]]

1
2
3
4
2
3
4
5
```

```
3
4
5
6
4
5
6
7
```

Operations over arrays in Numpy

In Numpy , we are able to perform operations over array elements like matrices by using the operators like +, -, *,......
EX:

```
import numpy as np
a = np.array([(1,2),(2,3)])
b = np.array([(5,6),(7,8)])
print(a+b)
```

EX:

```
import numpy as np

a = np.array([(1,2),(3,4)])
b = np.array([(5,6),(7,8)])
```

```
print(a+b)
print(a-b)
print(a*b)
print(a@b)
```

OP:

```
---
[[ 6  8]
 [ 9 11]]
[[-4 -4]
 [-5 -5]]
[[ 5 12]
 [14 24]]
[[19 22]
 [31 36]]
```

Where '@' operator can be used to perform Matrices multiplecation like below.

```
[a,b]   @   [e,f]   =====>    [a*e+b*g , a*f+b*h]
[c,d]       [g,h]             [c*e+c*g , c*f+d*h]
```

In Numpy, we can perform the operations over the elements like below.

1. sqrt(a)

--> It will perfrom sqrt() operation over all the elements of the array.
EX:

```
import numpy as np
a = np.array([(4,16),(25,64)])
print(a)
print(np.sqrt(a))
```
OP:

```
[[ 4 16]
 [25 64]]
[[2. 4.]
 [5. 8.]]
```

2.num+a, num-a, num*a, a/num, a%num
EX:

```
import numpy as np
a = np.array([(10,20),(30,40)])
print(a)
print(a+10)
print(a-5)
print(a*10)
print(a/2)
print(a%3)
print(a**2)
print(pow(a,3))
```

OP:

```
[[10 20]
 [30 40]]
[[20 30]
 [40 50]]
[[ 5 15]
 [25 35]]
[[100 200]
 [300 400]]
[[ 5. 10.]
 [15. 20.]]
[[1 2]
 [0 1]]
[[ 100  400]
 [ 900 1600]]
[[ 1000  8000]
 [27000 64000]]
```

--> In Numpy, we can perform the comparision operators like ==, !=, <, >, <=, >=,... between elements from two arrays.
EX:

```
import numpy as np
```

```
a = np.array([(10,20),(30,40)])
b = np.array([(50,60),(70,80)])

print(a == b)
print(a != b)
print(a < b)
print(a > b)
print(a <= b)
print(a >= b)

OP:
---
[[False False]
 [False False]]
[[ True  True]
 [ True  True]]
[[ True  True]
 [ True  True]]
[[False False]
 [False False]]
[[ True  True]
 [ True  True]]
[[False False]
 [False False]]
```

--> IN Numpy, we are able to get max value and min value from an array by using max() function and min()

function
EX:

```
import numpy as np
a = np.array([(10,20),(30,40)])
print(a.max())
print(a.min())
```

OP:

```
40
10
```

--> In Numpy, it is possible to get average value of all the elements by using average() function and it is possible to get floor value and ceiling value for each and every element by using floor() and ceil() functions.
EX:

```
import numpy as np
a = np.array([(10.5,20.5),(30.5,40.5)])
print(np.average(a))
print(np.floor(a))
print(np.ceil(a))
```

OP:

```
25.5
[[10. 20.]
 [30. 40.]]
[[11. 21.]
 [31. 41.]]
```