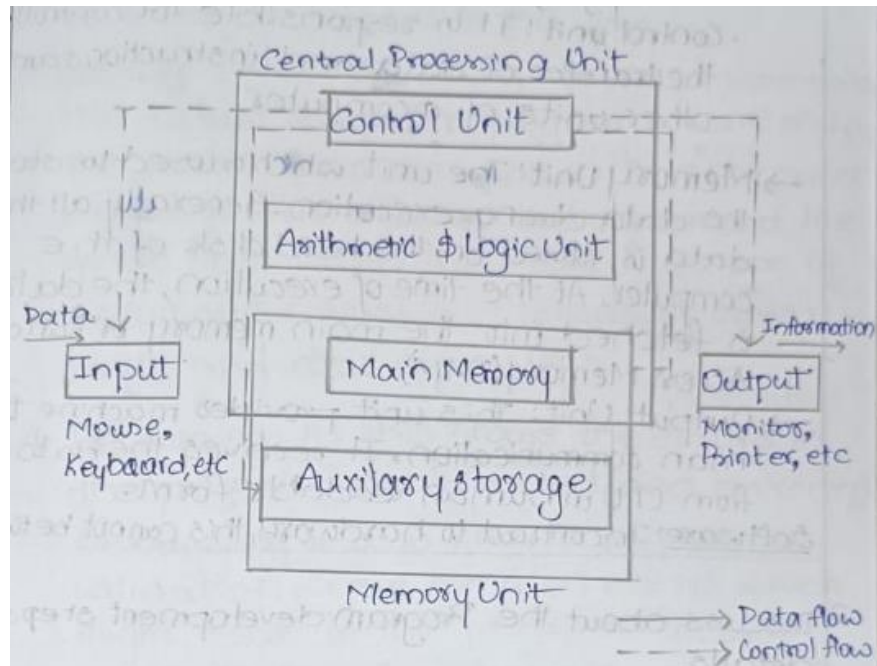


## Unit: 1

### Define Computer and brief about its Hardware and Software?

A Computer is an **electronic device** that stores, manipulates and retrieves the data.

We can also refer computer computes the data supplied to it and generates the results



Hardware, and software are components of computer.

~~Hardware~~ Hardware of computer system can be referred as anything which we can touch and feel. It is mainly classified into three types following:

- Input unit: The unit which accepts the user input to the program.
- CPU: It performs all types of data processing operations. It stores data, intermediate results and instructions.
- Arithmetic Logic Unit: Function of arithmetic section is to perform operations like addition,

subtraction, multiplication, and division.

Function of logic section is to perform logic operations such as comparing, selecting, matching and merging of data.

- Control unit: It is responsible for controlling the transfer of data and instructions among other units of a computer.

Memory Unit: The unit which is used to store the data during execution. Generally all the data is stored on the hard disk of the computer. At the time of execution, the data is fetched into the main memory or Random Access Memory (RAM).

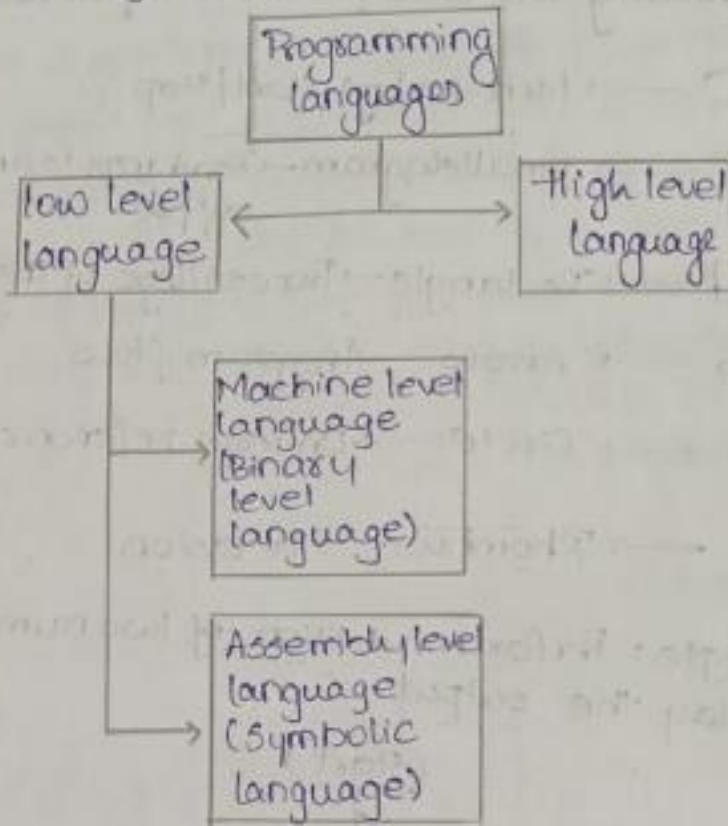
- Output Unit: This unit provides machine to man communication. It receives the data from CPU in human readable form.

Software: In contrast to hardware, this cannot be touched

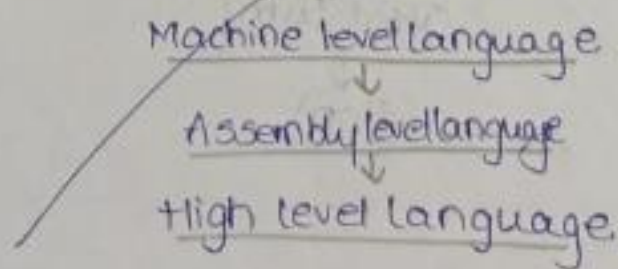
Explain about  
Programming

the types of  
Languages?

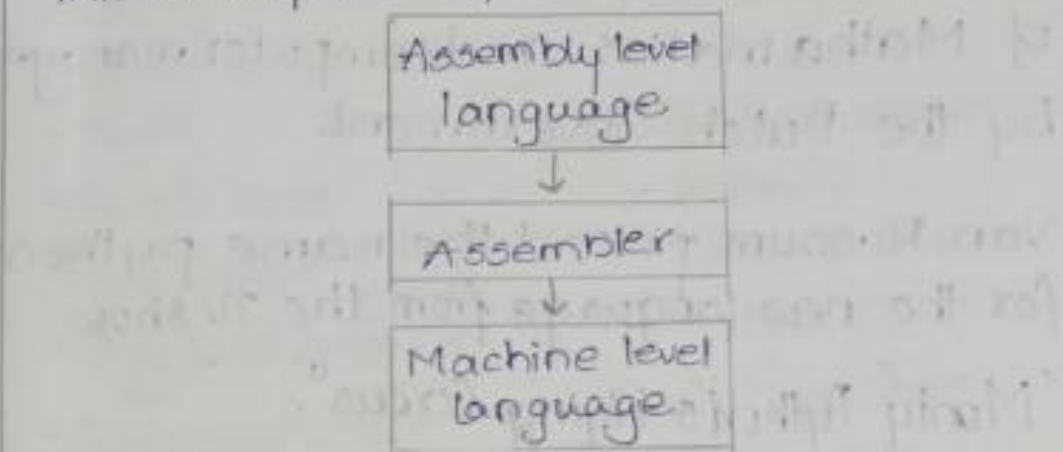
and uses.  
⇒ To write a program for computer, we must use a computer language. Generally humans are communicating to the computer using programming languages.



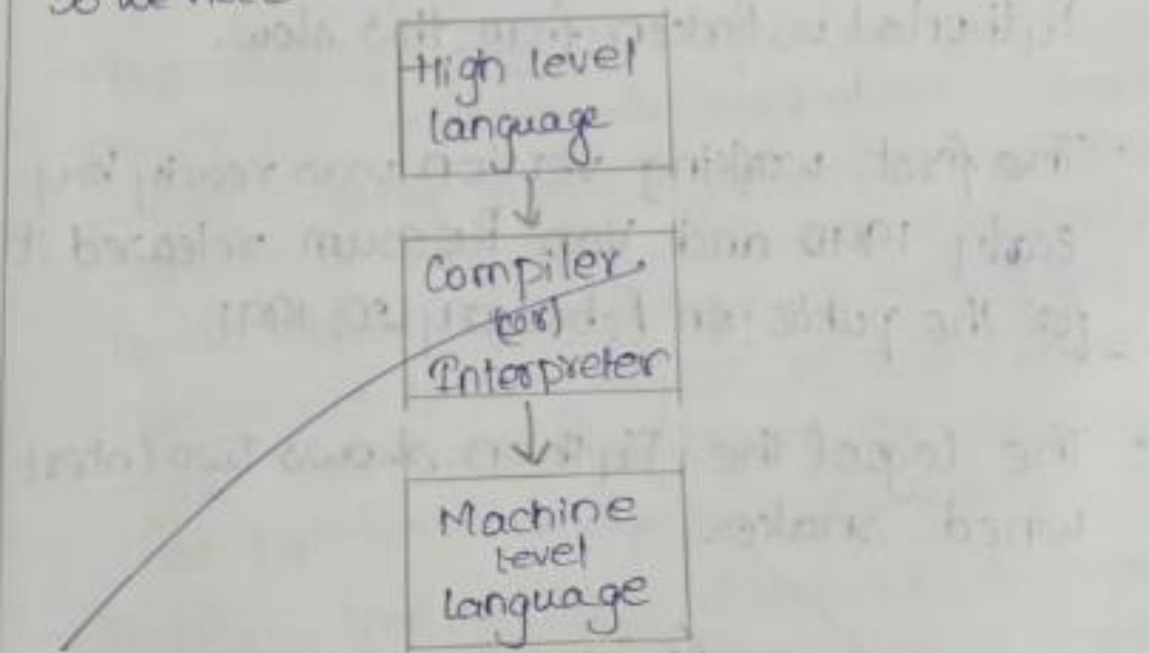
Machine level language: This is made up of streams of 0's and 1's. The instructions in machine language must be in streams of 0's and 1's.



Assembly level language: The Assembly level language is made up mnemonics. But the computer can not understand this language. This is why we require conversion.



High level language: High level language is similar to natural language like English language. But the computer can not understand this language. So we need conversion.



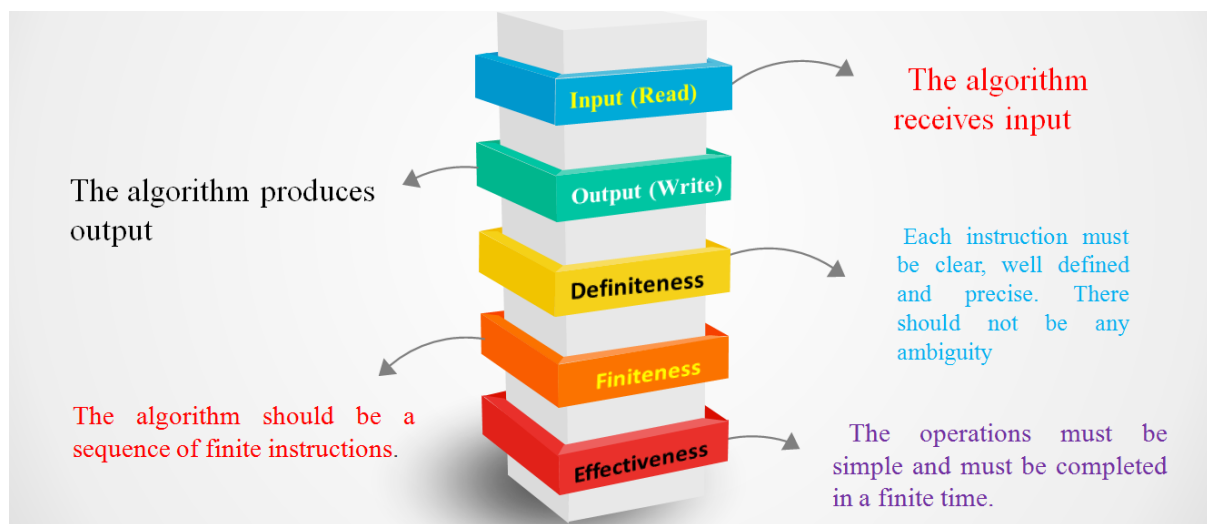
### Uses

- High speed execution
- The computer can understand the instructions
- translation is needed.
- Low development cost.

Explain the steps to write an Algorithm and write an algorithm to find sum of two numbers?

Algorithm is a step by step procedure for solving a particular problem.





Step 1: Start

Step 2: [Read A and B]

Read a,b

Step 3: [Compute]

$\text{Sum} = A + B$







Step 4: [Write Sum]

Print Sum

Step 5: [end] Stop.

**Define Flowchart and brief about the symbols of flowchart with an example program?**

Flowchart is a Symbolic representation of an algorithm. Flowchart is very helpful in writing program and explaining program to others. In flowchart each symbol indicates particular operation. While drawing the flow chart, operation must be written inside the symbols.

Symbol	Symbol Name	Purpose	Description
	Arrow	Flow line	Used to indicate the flow of logic by connecting symbols.
	Oval	Terminal (Stop/Start)	Used to represent start and end of flowchart.
	Parallelogram	Input/Output	Used for input and output operation.
	Rectangle	Processing	Used for arithmetic operations and data- manipulations.
	Rhombus	Decision	Used to represent the operation in which there are two alternatives, true and false.
	Circle	On-page Connector	Used to join different flow line

### Define Python and write short note on features of Python Programming

The Python programming language is an object-oriented language, which means that it can model real-world entities. It is also dynamically-typed because it carries out type-checking at runtime. There are various reasons why python is gaining good popularity in the programming community. The following are some of the important features of python:

- ✓ Simple
- ✓ Easy to Learn
- ✓ Open Source

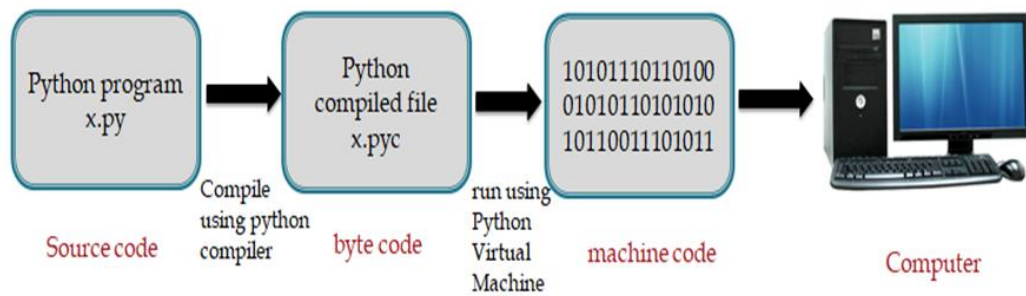
- ✓ High Level Language
  - ✓ Dynamically Typed
  - ✓ Platform Independent
  - ✓ Huge Library
  - ✓ Scripting Language
  - ✓ Database Connectivity
  - ✓ Scalable
  - ✓ Batteries included
  - ✓ Portable
  - ✓ Interpreted
  - ✓ Extensible
  - ✓ Embeddable
  - ✓ Easy to Understandable
- **Dynamically Typed:** In Python, we need not to declare anything. An assignment statement binds a name to an object, and the object can be of any type. If a name is assigned to an object of one type, it may later be assigned to an object of a different type. This is a meaning of the saying that python is a dynamically typed language.
  - **Platform Independent:** When a python program is compiled using a python compiler, it generates byte code. Python's byte code represents a fixed set of instructions that runs on all operating systems and hardware. Using a python virtual machine, anybody can run these byte code instructions on any computer system. Hence, python program are not dependent on any specific operating system.
    - It executes the code line by line in a **REPL (Read-Evaluate-Print-Loop) fashion**
  - **Scripting Language:** A Scripting Language is a programming language that does not use a compiler for executing the source code. Rather, it uses an interpreter to translate the source code into machine code on while running. Generally, scripting languages perform supporting tasks for a bigger applications or software.
  - **Huge Library:** Python has a big library which can be used on any operating system like UNIX, Windows or Macintosh. Programmers can develop programs very easily using the modules available in the Python library.
  - **Batteries included:** The Huge library of python contains several small applications which are already developed and immediately available to programmers. These small packages can be used and maintained easily. Thus the programmers need not to download separate packages or applications in many cases. These libraries are called 'batteries included'.
  - **Interpreted:** A Program code is called source code. After writing a python program we should compile the source code using python compiler. Python compiler translates the python program into an intermediate code called byte code. This byte code is then executed by PVM. Inside the PVM, an interpreter converts the byte code instructions into machine code so that the processor will understand and run that machine code to produce results.
  - **Extensible:** The Programs or pieces of code written in C or C++ can be integrated into python and executed using PVM. This is what we see in standard python that is downloaded from [www.python.org](http://www.python.org). There are others flavors of python where programs from other languages can be integrated into python. For example, Jython is useful to integrate java code into python programs and run on JVM (java virtual machine).
  - **Embeddable:** We can insert python programs into C or C++ program. Several applications are already developed in python which can be integrated into other programming languages like C, C++, PHP, Java and .NET. It means programmers can use these applications for their advantage in various software projects.

### Why Python is called as an Interpreted Language? Briefly explain about PVM with neat diagram?

Python is called an interpreted language because it does not need to be compiled before it is executed. Instead, Python source code is directly executed by the interpreter line by line, which interprets the code and executes it immediately.

It is therefore necessary to convert the byte code into machine code so that our computer can understand and execute it. For this purpose, we should use PVM(Python Virtual Machine).

## Execution of Python program



### List the steps used to view the byte code of a python program?

- Lets consider the following python program:  

```
#python program to add two numbers
a = b = 10 #take two variables and store 10 in to them
print("Sum= " ,(a+b)) # display their sum
```
- we can type this program in text editor like notepad and then save it as 'first.py'. it means, the first.py file contains the source code.
- Now, lets compile the program using python compiler as:  

```
C:\>python first.py
```
- It will display the results as:  

```
Sum = 20
```
- That is ok. But we do not want the output of the program .
- we want to see the byte code instructions that were created internally by the python compiler before they are executed by the PVM.
- For this purpose, we should specify the **dis** module while using python command as:  

```
C:\>python -m dis first.py
```
- The preceding byte code is displayed by the dis module, which is also known as 'disassembler' that displays the byte code in the human understandable format.
- If we observe the preceding code we can find 5 columns.
  - The left-most or first column represents the line number in our source program (first.py).
  - The second column represents the offset position of the byte code.
  - The third column shows the name of the byte code instructions.
  - The fourth column represents instructions argument and
  - The last column represents constants or name as specified by 4<sup>th</sup> column .

### Distinguish between C and Python programming?

S.N O	Key	C Language	Python Language
1	Definition	C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.	Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
2	Type	Structured type programming language and following Imperative programming model. Also it is statically typed.	Object-oriented type programming language and dynamically typed.
3	Variable Declaration	Variables are need to be declared in C before get used in code further.	While on other hand in Python no need of variable declaration for its use.
4	Compilation	C language is compiled by the compiler hence is also known as compiled language.	On other hand interpreter is used in Python for interpreting the code and hence Python is known as Interpreted language.
5	Functions available	C language has limited number of built-in functions as compared to that in Python language.	On other hand Python has a large library of built-in function as compared to C language.
6	Execution	As mentioned in above point C is a compiled language hence its code is compiled direct to machine code which is executed directly by the CPU.	On other hand in case of Python code is firstly compiled to a byte-code and then it is interpreted by a large C program.

### Brief about comments and docstrings in python program with a sample program?

#### Comments and Docstrings

- Declare comments using an octothorpe (#).
- comments are ignored by the compiler during the execution of the program.
- However, Python does not support multiline comments.

#This is a comment

# sample comment

name ="Welcome"

print(name)

docstrings are documentation strings that help explain the code.

""" This is a docstring """

#### Comments are generally used for the following purposes:

- Code Readability
- Explanation of the code or Metadata of the project
- Prevent execution of code
- To include resources

#### Types of comments

- Single Line Comments: #
- Multi-Line Comments: text enclosed in a delimiter ("""") on each end of the comment

"""

This would be a multiline comment in Python that spans several lines and describes comments.

"""

print("Hello")

- In the Python, the docstring is then made available via the `__doc__` attribute.
- Docstring is an in-built feature of Python, which is used to associate documentation that has been written with Python modules, functions, classes and methods.
- It is added right below the functions, modules or classes to describe what they do.

def multiply(a, b):



```
"""Multiplies the value of a and b"""
return a*b
```

```
# Print the docstring of multiply function
print(multiply.__doc__)
```

**Output:**

Multiplies the value of a and b

- **Declaring Docstrings:**

The docstrings are declared using

'''triple single quotes''' or

"""triple double quotes""" just below the class, method or function declaration.

## **Unit: 2**

Classify the built-in data types based on mutable and immutable. Explain any two data types of each with example program.

Mutable:

Object can be changed after created is called mutable. Example: list, dictionary, set

List:

Lists are collections of items (strings, integers, or even other lists). It is enclosed with help of [].

Each element is assigned with an index number.

Example: list = [10, -20, 15.5, 'vijay', "mary"]

List=("Anjana","Navya","Appu")

list.append('Varun')

print(list)

Immutable:

Object cannot be changed once created. Example: int, float, string, tuple

Tuple:

A tuple is a collection which is ordered and unchangeable. It is enclosed with help of (). It is also ordered.

Example: (1,2,3)

tuple1=("apple","banana","cherry")

tuple1.append("mango")

print(tuple1)

Explain the following built-in data types:

>None:

- The None keyword is used to define a null variable or an object.
- None keyword is an object, and it is a data type of the class NoneType
- None is used to define a null value. It is not the same as an empty string, False, or a zero.
- Assigning a value of None to a variable is one way to reset it to its original, empty state.

print(type(None))

**Output**

<class 'NoneType'>

>Numeric:

- int – holds signed integers of non-limited length.
- float- holds floating precision numbers and it's accurate up to 15 decimal places.
- complex- holds complex numbers.

intgr = 3

flt = 2.1

type(intgr)

type(flt)

Output:

<class 'int'>

<class 'float'>

### >Bool:

Boolean type is one of the built-in data types provided by Python, which are defined by the True or False keywords. It is used to represent the truth values of the expressions.

```
b=False
```

```
type(b)
```

Output:

```
<class 'bool'>
```

### >String:

A string is represented by a group of characters. It is enclosed between `""`. The representation of it is done by `str`.

```
strng="Hello world"
```

```
print(type(string))
```

output:

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

### Describe the list comprehension with an example program?

- List comprehensions provide a concise way to create lists.
- It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.
- The expressions can be anything, meaning you can put in all kinds of objects in lists.
- The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it.
- The list comprehension always returns a result list.

```
x = [i for i in range(10)]
```

```
print(x)
```

output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

### List few operations and methods on Tuple and dictionaries with an example programs?

Tuple:

#### Check if Item Exists

To determine if a specified item is present in a tuple use the `in` keyword:

```
thistuple = ("apple", "banana", "cherry")
```

```
if "apple" in thistuple:
```

```
    print("Yes, 'apple' is in the fruits tuple")
```

#### Tuple Length

To determine how many items a tuple has, use the `len()` method:

```
thistuple = ("apple", "banana", "cherry")
```

```
print(len(thistuple))
```

#### Loop Through a Tuple

You can loop through the tuple items by using a for loop.

```
thistuple = ("apple", "banana", "cherry")
```

```
for x in thistuple:
```

```
    print(x)
```

#### Adding and Removing items id not possible as it tuple is unchangeable.

Dictionary:

#### Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

```
x = thisdict.get("model")
```

#### Change Values

You can change the value of a specific item by referring to its key name:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

```
thisdict["year"] = 2018
```

```
print(thisdict)
```

#### Removing Items

There are several methods to remove items from a dictionary:

The `pop()` method removes the item with the specified key name:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

```
thisdict.pop("model")
```

```
print(thisdict)
```

The del keyword removes the item with the specified key name:

```
del thisdict["model"]
```

```
print(thisdict)
```

The clear() keyword empties the dictionary:

```
thisdict.clear()
```

```
print(thisdict)
```

### Brief about Set and frozenset with an example program?

Set:

- The set type is mutable - the contents can be changed using methods like add() and remove().
- Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set.

```
my_set = {1, 2, 3, 4, 5}
```

```
print(my_set)
```

```
my_set.add(6)
```

```
print(my_set)
```

Output:

```
{1,2,3,4,5}
```

```
{1,2,3,4,5,6}
```

Frozenset:

- The frozenset type is immutable and hashable - its contents cannot be altered after it is created; it can, therefore, be used as a dictionary key or as an element of another set.

```
my_set = {1, 2, 3, 4, 5}
```

```
my_frozen_set = frozenset(my_set)
```

```
print(my_frozen_set)
```

```
my_frozen_set.add(6)
```

```
# try to add a value to the frozen set (will raise an error)
```

### State the use of constants, identifiers, reserve words and naming conventions?

Constants:

- A constant is a type of variable whose value cannot be changed.
- It is helpful to think of constants as containers that hold information which cannot be changed later.
- You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag.

Identifiers:

- An identifier is a user-defined name to represent a variable, a function, a class, a module, or any other object.
- It is a programmable entity in Python- one with a name.
- It is a name given to the fundamental building blocks in a program.

Reserve words:

- Python Server Side Programming Programming Reserved words (also called keywords) are defined with predefined meaning and syntax in the language.
- These keywords have to be used to develop programming instructions.
- Reserved words can't be used as identifiers for other programming elements like name of variable, function etc.

Naming conventions:

- Naming conventions are a set of rules or best practices that are suggested to be followed while naming an identifier.
- The benefits of following the naming conventions are: They help us in understanding what a particular identifier is.

### Demonstrate the following with an example program?

>Assignment operator:

Assignment operators are used in Python to assign values to variables.

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
=	x  = 5	x = x   5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

```
x=2
```

```
x+=5
```

```
print(x)
```

Output:

7

### >Logical Operation:

Logical operators are the and, or, not operators. The output is always True or False.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

```
x = True
```

```
y = False
```

```
print(x and y)
```

```
print(x or y)
```

```
print(not x)
```

Output:

False

True

False

### >Bitwise operator:

Bitwise operators act on operands as if they were strings of binary digits.

Operator	Meaning	Example
&	Bitwise AND	x & y
	Bitwise OR	x   y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x >> 2
<<	Bitwise left shift	x << 2

x=4

y=10

print(x&y)

print(x|y)

print(x^y)

print(~x)

print(2>>x)

print(2<<x)

Demonstrate the following with an example programs?

### >Relational Operator:

Relational operators are used to compare values.

It returns either **True** or **False** according to the condition.

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

a=10

b=20

print(a>b)

print(a<b)

print(a==b)

print(a!=b)

print(a>=b)

print(a<=b)



Output:

False

True

False

True

False

True

### > Membership operator:

in and not in are the membership operators in Python.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

```
x = 'Hello world'
print('H' in x)
print('Hello' not in x)
```

Output:

True

False

### > Identity Operator:

They are used to check if two values (or variables) are located on the same part of the memory. is and is not are the identity operators in Python.

Operator	Meaning	Example
is	True if the operands are <b>identical</b> (refer to the same object)	x is True
is not	True if the operands are <b>not identical</b> (do not refer to the same object)	x is not True

```
x=10
y=20
print(x is y)
print(x is not y)
```

Output:

False

True

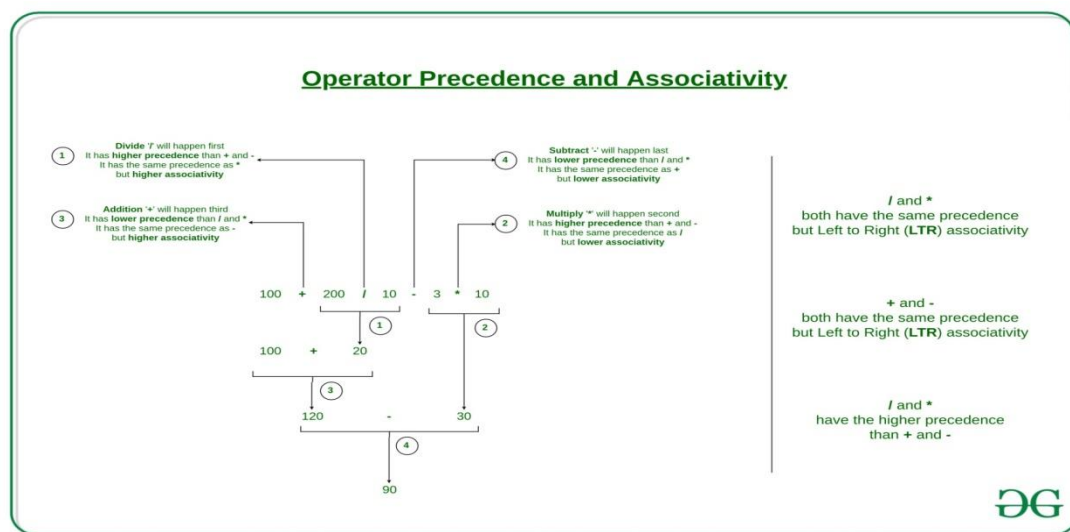
### >Operator precedence and associativity:

The combination of values, variables, operators, and function calls is termed as an expression. To evaluate these types of expressions there is a rule of precedence in Python. It guides the order in which these operations are carried out.

The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.



Describe about input and output statements with a sample program for each type of statement?

Input statements:

- The data given to the computer is called input
- To accept input from keyboard, python provides the input () functions.
- This function takes a value from the keyboard and returns it as a string
- Once the value comes into the variable 'str', it can be converted into 'int' or 'float' etc. this is useful to accept number as
- `x=int(input('enter a number:'))`  
`print(x)`  
output:  
125

Output statements:

- The results returned by the computer are called output.
- The display output or results, python provides the print () function.
- The print () Statement**

When the print() function is called simply. It will throw the cursor to the next line. It means that a blank line will be displayed.

- **The print () Statement**

When the print() function is called simply. It will throw the cursor to the next line. It means that a blank line will be displayed.

- **'\n' indicates new line. '\t' represents tab space.**
- **We can use repetition operator (\*) to repeat the strings in the output**
- **To display a string, we can use %s in the formatted string. To display an integer, we can use %i in the formatted string. To display float, we can use %f in the formatted string.**

**What is a command line argument? Explain briefly how to implement it in python programming with an examples?**

- Python Command line arguments are input parameters passed to the script when executing them.
- They are passed to the program from outside the program.
- All the arguments should be entered from the keyboard separating them by a space.
- These arguments are stored by default in the form of strings in a list with the name 'argv' which is available in sys module.

- `n = len (sys.argv)` #n is the number of arguments

`args = sys.argv` # args lists contains arguments

`print('No.of command line args=', n)`

`print ('The args are: ', args)`

`print('The args one by one: ')`

`for a in args:`

`print(a)`

output:

`c:\python cmd.py 10 Aishwarya Shetty 22500.75`

No.of command line args= 5

The args are: ['cmd.py', '10', 'Aishwarya', 'Shetty', '22500.75']

The args one by one:

cmd.py

10

Aishwarya

Shetty

22500.75

## **Unit: 3**

**Explain elif ladder in python programming with a sample program?**

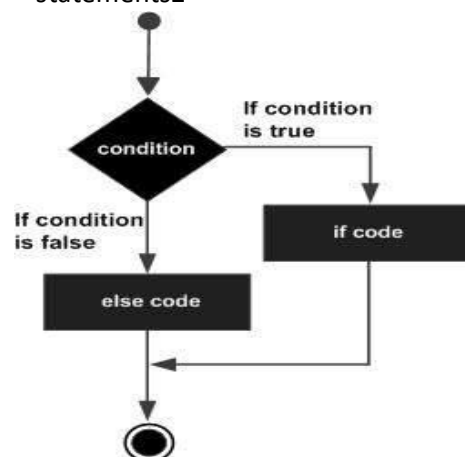
- The if....else statement execute a group of statements when a condition is true. Otherwise, it will execute another group of statements.
- The syntax of if...else statement is given below:

if condition:

statements1

else:

statements2



- `x=10`  
`if x%2==0:`  
`print(x, "is a even number")`  
`else:`  
`print(x, "is a odd number")`

### Illustrate for and while loop with an example programs?

- A loop statement allows us to execute a statement or group of statements multiple times.
- for loop: Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.  
the syntax of the for loop is given below:  
`for var in sequence:`  
`Statements`

Example:

```
for x in range(1, 6):
    print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

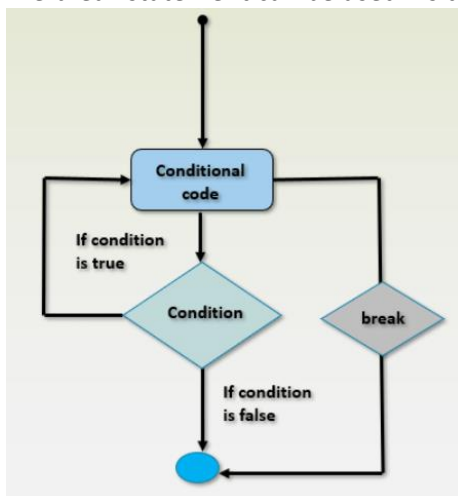
- While loop: Executes a group of statements as long as a condition is True.  
Syntax:  
`while condition:`  
`statements`

```
x = 1
while x<=10:
    print(x)
    x+=1
print ("End")
```

### Explain the following statements with flowchart and sample program

#### >break

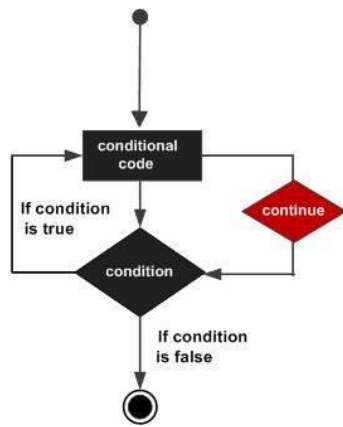
The break statement can be used inside a for loop or while loop to come out of the loop.



```
if search ==element:
    print('element found in group1')
    break #come out of for loop
else:
    print('element not found in group1') #this is else suite
```

#### >continue:

The continue statement is used in a loop to go back to the beginning of the loop.

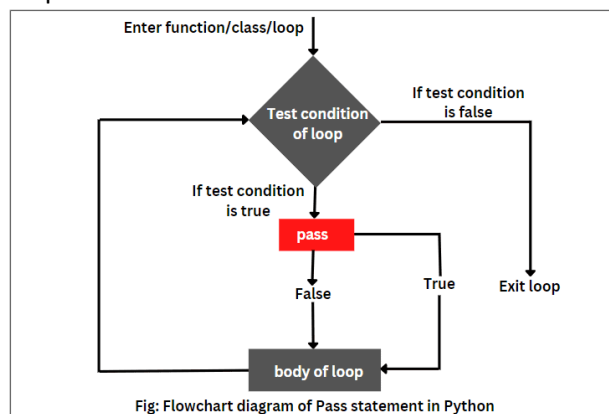


```

x=0
while x<10:
    x+=1
    if x>5:
        continue
    print('x=',x)
print("out of loop")
  
```

**>pass:**

The pass statement does not do anything. It is used with 'if' statement or inside a loop to represent no operation.

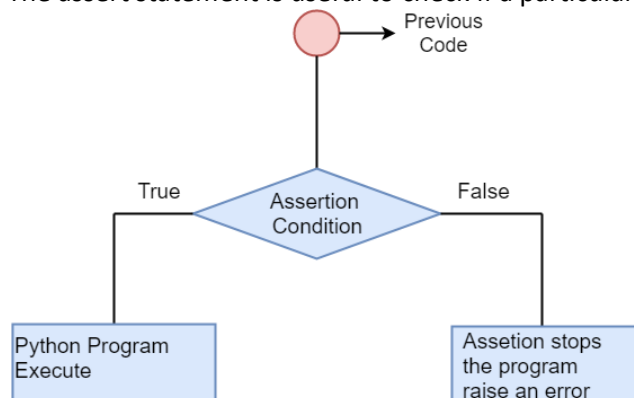


```

num = [1,2,3,-4,-5,-6,-7,-8,9]
for i in num:
    if i>0:
        pass
    else:
        print(i)
  
```

**>assert:**

The assert statement is useful to check if a particular condition is fulfilled or not.



```

x = int(input('Enter a number greater than 0: '))
assert x>0, "Wrong input entered"
print('U entered: ',x)
  
```

What is an array? Can we use List as a substitute of arrays? Justify.



- An array is a collection of items stored at contiguous memory locations
- The idea is to store multiple items of the same type together
- Arrays are used to store multiple values in one single variable
- Arrays can be multidimensional, and all elements in an array need to be of the **same type**, all integers or all floats
- Example:  

```
import array
a=array.array('d', [1, 3.5, "Hello"])
print(a)
```

Yes, lists can be used as a substitute for arrays in many cases. Here are some justifications:

**Dynamic resizing:** One of the biggest advantages of lists over arrays is that lists can dynamically resize themselves as needed, while arrays have a fixed size. This means that you don't need to know how many elements you will need to store in advance, which makes lists more flexible in certain situations.

**Built-in functionality:** Lists come with built-in functionality, such as sorting, appending, inserting, and removing elements. While these operations can also be performed on arrays, it usually requires more code to do so.

**More readable code:** Lists can make your code more readable and easier to understand, especially when dealing with complex data structures. For example, you can use list comprehensions to create new lists based on existing ones, which can be more concise and expressive than using arrays.

**Pythonic:** In Python, lists are the default data structure for storing sequences of elements. Most Python libraries and packages also expect lists as input or output, so using lists can make your code more Pythonic and easier to integrate with other code.

Explain the following with a simple program

> **Creating an array:**

- Array is created in Python by importing **array** module to the python program.
- Then the array is declared as shown below.

```
from array import *
arrayName = array(typecode, [Initializers])
or
import array
arrayName = array.array(type code, [array,items])
```

> **Accessing array elements:**

- We can access each element of an array using the index of the element.
- The below code shows how

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

> **Different ways to add an element in to array:**

- we add a data element at the middle of the array using the python in-built **insert()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.insert(1,60)
for x in array1:
    print(x)
```

- **append()** is also used to add the value mentioned in its arguments at the end of the array.

```
import array as arr
numbers = arr.array('i', [1, 2, 3])
numbers.append(4)
print(numbers)
Output: array('i', [1, 2, 3, 4])
```

### > Different ways to delete an element from array:

- we remove a data element at the middle of the array using the python in-built **remove()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.remove(40)
array1[index]
for x in array1:
    print(x)
```

- **pop()** function can also be used to remove and return an element from the array

```
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]
print(number)
Output:
array('i', [1, 2, 3, 4])
```

### > Searching an element in array:

- You can perform a search for an array element based on its value or its index

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1.index(40))
```

### > Slicing on array:

- There are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation.

```
import array as arr
numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)
print(numbers_array[2:5])
Output:
array('i', [62, 5, 42])
```

### Mention the advantages of using Numpy module instead of array module?

Here are some advantages of using NumPy instead of the array module:

- It consumes **less memory**:  
NumPy uses much less memory to store data and it provides a mechanism of specifying the data types. This allows the code to be optimized even further.
- It is **fast** as compared to the array:  
NumPy arrays are faster and more efficient than arrays created with the built-in array module. This is because NumPy arrays are implemented in C and use optimized algorithms and memory layouts.
- It is **convenient to use**:  
NumPy arrays are faster and more efficient than arrays created with the built-in array module. This is because NumPy arrays are implemented in C and use optimized algorithms and memory layouts.

### Write short notes on the following using numpy with an example programs

**i) arrange:** The arrange () function of Python numpy class returns an array with equally spaced elements as per the interval where the interval mentioned is half opened, i.e. [Start, Stop).

```
import numpy as np
A = np.arange(4)
print('A =', A)
B = np.arange(12).reshape(2, 6)
print('B =', B)
Output: A = [0 1 2 3]
        B = [[ 0 1 2 3 4 5]
              [ 6 7 8 9 10 11]]
```

ii) **reshape**: Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
output: [[ 1 2 3]
         [ 4 5 6]
         [ 7 8 9]
         [10 11 12]]
```

iii) **array of zeros and ones**: produce the ones array ( np.ones) produce the zeros array ( np.zeros)

```
import numpy as np
zeros_array = np.zeros( (2, 3) )
print(zeros_array)
Output: [[0. 0. 0.]
        [0. 0. 0.]]
ones_array = np.ones( (1, 5), dtype=np.int32 )
print(ones_array)
Output: [[1 1 1 1 1]]
```

iv) **dimensions**:

One Dimension: Many arrays have only one dimension, such as the number of people of each age.

Two Dimensions: Some arrays have two dimensions, such as the number of offices on each floor of each building on a...

Three Dimensions: A few arrays have three dimensions, such as values in three-dimensional space. Such an array uses...

More than Three Dimensions. Although an array can have as many as 32 dimensions, it is rare to have more than three.

v) **slicing**: Slicing of a Matrix Slicing of a one-dimensional NumPy array is similar to a list.

Example:

```
import numpy as np
letters = np.array([1, 3, 5, 7, 9, 7, 5])
print(letters[2:5])
Output: [5, 7, 9]
```

Write a program to perform the following operations on 2 dimension matrix of 3x3 size using array

>**Addition**:

```
import numpy as np
x=np.array([[1,2,3],[4,5,6],[7,8,9]])
y=np.array([[1,1,1],[1,1,1],[1,1,1]])
print(x+y)
output: [[ 2 3 4]
         [ 5 6 7]
         [ 8 9 10]]
```

>**Multiplication**:

```
import numpy as np
x=np.array([[1,2,3],[4,5,6],[7,8,9]])
y=np.array([[1,1,1],[1,1,1],[1,1,1]])
z=np.matmul(x,y)
print(z)
output: [[ 6 6 6]
         [15 15 15]
         [24 24 24]]
```

## Unit: 4

Define function with an example program?

- A function is a block of organized, reusable code that is used to perform a single, related action.
- It is a block of code which only runs when it is called.

- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- Python gives you many **built-in functions** like print(), etc. but you can also create your own functions. These functions are called **user-defined functions**.
- Example: bool(), bytes(), bytearray(), print()

**def** function\_name(parameters):

"""docstring"""

statement(s)

**return** <expression>

**Brief about formal and actual arguments with an example program?**

When function is defined it has some parameters. These parameters are used to receive values from outside the function.

→ formal arguments.

When we call the function, we should pass data and values in the function.

→ actual arguments.

Ex:

```
def sum(a, b)
    c = a + b
    print(c)
```

Here a, b → formal

```
x = 10
y = 15
sum(x, y)
```

x, y → actual

**Demonstrate the various types of arguments in functions with a sample program?**

Q. What are the categories of arguments in function?

→ Default Parameter value:  
 → If we call function without parameter name, it uses the default parameter value.  
 → def my\_function(country = "Norway"):  
     print("I am from" + country)  
 my\_function("India")

Required Arguments:  
 → Are the arguments to function in correct positional order  
 → def sum(a, b):  
     return (a + b)  
 a = int(input("value:"))  
 b = int(input("value:"))  
 print("sum" = sum(a, b))

### Keyword Arguments:

Used to skip arguments and place them out of order.

```
def student(fn, ln):  
    print(fn, ln)  
student(fn='Mani', ln='Pat')  
student(ln='Pat', fn='Mani')
```

### Variable length Arguments:

More than one argument

```
def printinfo(arg1, *vartuple):  
    print("Output is")  
    print(arg1)  
    for var in vartuple:  
        print(var)  
  
    return  
printinfo(10)
```

**In python functions are always called by passing reference, Justify.**

- Python functions are not always called by passing reference. Whether a function call passes by value or reference depends on the data type of the argument being passed.
- For immutable data types such as integers, strings, and tuples, the function call is passed by value, which means a copy of the argument is made and passed to the function. This is because the original value cannot be modified in the function, so it is safer to make a copy.
- For mutable data types such as lists and dictionaries, the function call is passed by reference, which means the original object is passed to the function. This is because the original value can be modified in the function.
- The behavior of Python function calls is determined by the language's design, which is based on a combination of factors including efficiency, ease of use, and consistency with other programming languages.
- In summary, Python functions are not always called by passing reference. The behavior of a function call depends on the data type of the argument being passed.

### **Explain built-in functions with suitable example programs?**

Built-in functions in programming refer to the pre-defined functions that are available in a programming language's standard library. These functions perform common tasks and can be used directly in the code without the need to write them from scratch. Here are some examples of built-in functions in Python with suitable example programs:

**print():** The print() function is used to output data to the console.

Example program:

```
print("Hello, world!")
```

output:

Hello, world!

**input():** The input() function is used to accept user input from the console.

Example program:

```
name = input("What is your name? ")
```

```
print("Hello, " + name + "!")
```

Output:

What is your name? John

Hello, John!



**len():** The len() function is used to get the length of a string, list, or tuple.

Example program:

```
word = "Python"
```

```
print(len(word))
```

Output:

6

**range():** The range() function is used to generate a sequence of numbers.

Example program:

```
for i in range(1, 6):
```

```
    print(i)
```

Output:

1

2

3

4

5

**sum():** The sum() function is used to get the sum of all the values in a list.

Example program:

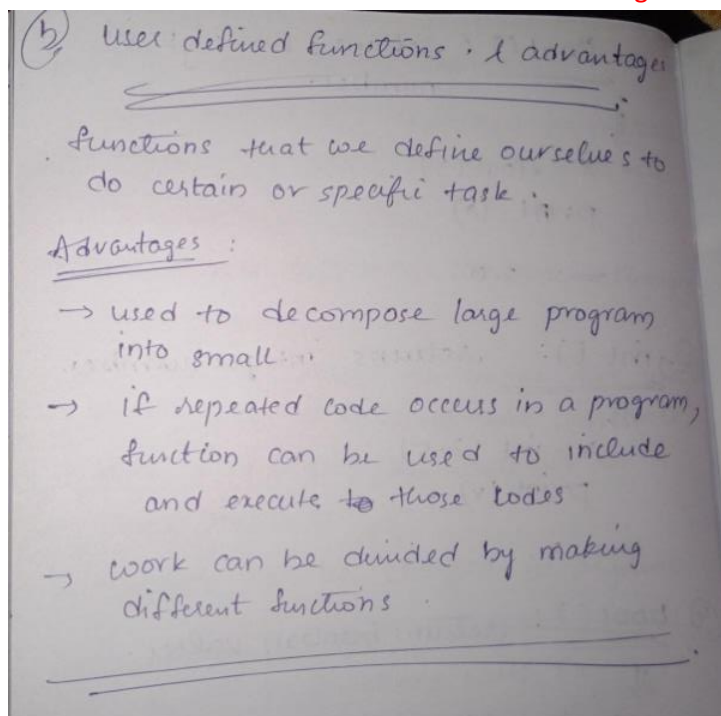
```
numbers = [5, 2, 8, 1, 9]
```

```
print(sum(numbers))
```

Output:

25

**Define user defined function and list out the advantages of user defined functions?**



**Brief about local and global variable with an example program?**

A local variable is a variable that is declared within a specific block or function in a computer program and has local scope, which means it can only be accessed and used within that block or function.

A global variable is a variable that is declared outside of any specific function or block in a computer program and has global scope, which means it can be accessed and used anywhere within the program.

Example:

```
x = 10 # global variable
```

```
def function1():
```

```
    x = 5 # local variable
```

```
    print("Inside function1, x =", x)
```

```
def function2():
```

```
print("Inside function2, x =", x)
function1() # Output: Inside function1, x = 5
function2() # Output: Inside function2, x = 10
Output:
Inside function1, x = 5
Inside function2, x = 10
```

Define python Recursion function? Write a python program of factorial using recursion?

Recursion:  
 It means a defined function can call itself it is called as Recursion function.  
 Used to solve problems which can be divided into smaller subproblems which are similar to original problem.

Program:

```
x = int(input("Enter the number: "))
def fact(x):
    if x == 0:
        return 1
    else:
        return x * fact(x-1)
print("The factorial of {} is {}".format(x, fact(x)))
```

Output:  
 Enter the number: 7  
 The factorial of 7 is 5040

Write a Python function that takes two lists and returns True if they have at least one common member?

```
def common(a,b):
    c=len(set(a).intersection(b))
    if c>=1:
        return True
    else:
        return False
list1=[1,2,3,4]
list2=[5,2,1,8]
common(list1,list2)
```

True

What are Anonymous functions in python?

Anonymous functions in Python are functions that are defined without a name. They are also known as lambda functions because they are defined using the lambda keyword.

It can take many number of arguments but only have single expression. Contains small pieces of code.

Syntax:

lambda arguments:expression

Example:

x=lambda a:a+10

print("Sum:",x(20))

Output: 30

Demonstrate the use of Lambda functions with a sample code.

Lambda is an ~~small~~ anonymous function, which means a function without a name. Contains  
It can take many number of arguments but only have single expression. Contains small piece of code.  
Syntax: lambda arguments: expression.

Characteristics:

- Anonymous.
- Single Expression.
- Arguments.

Example:

```
x = lambda a: a + 10  
print("Sum:", x(20))
```

Output:

Sum: 30

What are Python OOPs Concepts?

⇒ OOPs is abbreviation of Object-Oriented Programming System. It is a methodology to design a program using classes and object objects.

It simplifies software development and maintenance by providing few concepts:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

1. Object: It is a real time entity (instance of class)

2. Class: Structure or plan or blueprint or model to define an object.

3. Inheritance: Creating new class from existing class

4. Abstraction: handling complexity, to avoid excess information from user.

5. Polymorphism: ability of a message to be displayed in more than one form.

6. Encapsulation: bundling data, along with the methods working on data to make it a single unit.

Write the syntax to create a class and an object with an example program?

Syntax for class:

class <ClassName>:

<statement1>

<statement2>

.

.

<statementN>

Syntax for object:

<object-name> = <class-name>(<arguments>)

```
class student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
stu1 = student('Latheef', 18)
stu2 = student('Ashish', 17)
print(stu1.name)
print(stu2.name)
```

Latheef

Ashish

Define inheritance and brief about different types of inheritances with an example program?

Ans. **INHERITANCE**:-

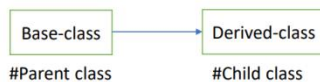
- Inheritance is the process of creating a new class from an already existing class.
- Inheritance is also defined as an object of one class that has the properties of an object of another class.

The inheritances are:-

- Single inheritance
- Multi-level inheritance
- Multiply inheritance
- Hierarchical inheritance

➤ **SINGLE INHERITANCE**:-

- In single inheritance a derived class is created using a single-based class.



#single inheritance code-

```
class parent():
    def __init__(self):
        self.value="Am Inside parent class"
    def show(self):
        print(self.value)
class child(parent):
    def __init__(self):
        self.value="Am inside child class"
    # def show(self):
    #     print(self.value)
o3=parent()
o4=child()
o3.show()
o4.show()
```

Am Inside parent class  
Am inside child class

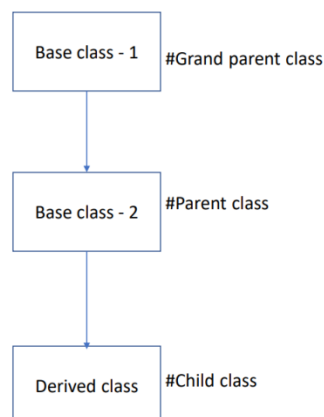
➤ **MULTI LEVEL INHERITANCE**:-

- In this every new class is derived from a single-parent class.

#code for multi-level inheritance

```
class Grandparent:
    def gpf(self):
        print('This is grand parent class')
class Parent(Grandparent):
    def pf(self):
        print('This is parent class')
class Child(Parent):
    def cf(self):
        print('This is child class')
ch=Child()
ch.cf()
ch.pf()
ch.gpf()
```

This is child class  
This is parent funtion  
This is grand parent class



Activate  
Go to Settings

### ➤ MULTIPLE-LEVEL INHERITANCE:-

- In Multiple inheritances a derived class is created by using two or more base classes.

#optional code for studying

```
class child_one(parent):
    def funA(self):
        print("This is class one.")
class child_two(parent):
    def funB(self):
        print("This is class two")
class parent(child_one,child_two):
    def func(self):
        print("This is derived class.")
p=parent
p.funA(1)
p.funB(1)
p.func(1)
```

```
This is class one
This is class two
This is derived class.
```



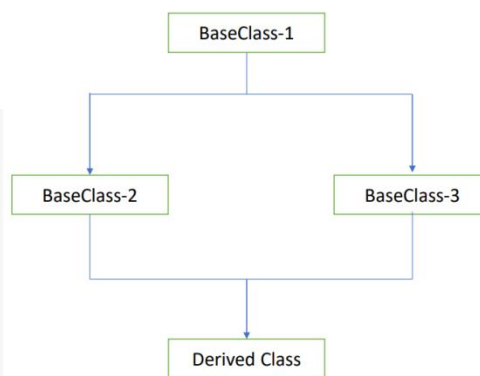
### ➤ HIERARCHICAL INHERITANCE:-

- Hierarchical inheritance is the process of creating a child class from a single-parent class and also multiple-base classes.

#optional code for studying -

```
class parent(child_one,child_two):
    def funA(self):
        print("This function is in the parent class.")
class child_one(parent):
    def funB(self):
        print("This function is in class child_one.")
class child_two(parent):
    def func(self):
        print("This function is in class child_two.")
p=parent
p.funA(1)
p.funB(1)
p.func(1)
```

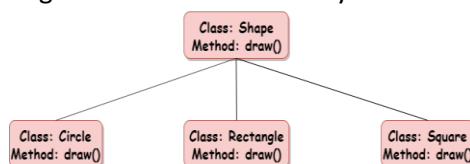
```
This function is in the parent class.
This function is in class child_one.
This function is in class child_two.
```



## Brief about Polymorphism, Encapsulation and Abstraction?

### Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways.



### Encapsulation:

Encapsulation refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many programming languages use encapsulation frequently in the form of classes.

### Abstraction:

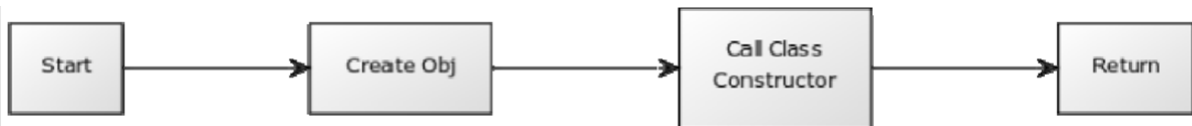
Abstraction refers to the act of representing essential features without including the background details or explanations. since the classes use the concept of data abstraction ,they are known as abstraction data type(ADT).

## Define constructor and Method overriding?

Constructor:

- The constructor is a method that is called when an object is created. This method is defined in the class and can be used to initialize basic variables.
- If you create four objects, the class constructor is called four times.
- Every class has a constructor, but its not required to explicitly define it.
- Each time an object is created a method is called. That methods is named the **constructor**.
- The constructor is created with the function **init**.
- As parameter we write the self keyword, which refers to itself (the object).





```

class Human:
def __init__(self):
    self.legs = 2
    self.arms = 2
bob = Human()
print(bob.legs)
  
```

Method overriding:

- Method overriding or function overriding is a type of polymorphism in which we can define a number of methods with the same name but with a different number of parameters as well as parameters can be of different types.

```

class Square:
    side =5
    def calculate_area(self):
        return self.side * self.side

class Triangle:
    base =5
    height =4
    def calculate_area(self):
        return 0.5* self.base * self.height

sq = Square()
tri=Triangle()
print("Area of square:",sq.calculate_area())
print("Area of triangle:",tri.calculate_area())
  
```

Output:

Area of square: 25  
Area Of triangle: 10.0

## **Unit: 5**

**Define the error. Distinguish between Compile Time Errors, Run-time Error and Logical errors with an example program.**

The errors in the software are called 'bugs'.

- In general, we can classify errors in a program into one of these three types:
- Compile-time errors
- Run-time errors
- Logical errors

### **Compile-time errors**

There are syntactical errors found in the code, due to which a program fails to compile. For example, forgetting a colon in the statements like if, while, for, def etc. will result in compile-time error. Such errors are detected by python compiler and the line number along with error description is displayed by the python compiler.

#example for compile-time error

```

x = 1
If x == 1
print('where is colon?')
  
```

### **Runtime Errors**

When PVM cannot execute the byte code, it flags runtime error. For example, insufficient memory to store something or inability of the PVM to execute some statement come under runtime errors. Runtime errors are not detected by the python compiler. They are detected by the PVM, only at runtime.

# runtime error

```

animal = ['dog','cat','horse','donkey']
print(animal[4])
  
```

## Logical Errors

These errors depict flaws in the logic of the program. The programmer might be using a wrong formula or the design of the program itself. Logical errors are not detected either by python compiler or PVM. The programmer is solely responsible for them.

#logical error

```
def increment(sal):
```

```
    sal = sal * 15/100 # sal =sal +sal * 15/100
```

```
return sal
```

#call increment() and pass salary

```
sal = increment(5000.00)
```

```
print('Incremented Salary: %.2f'%sal)
```

## Why exception handling is more important in Python? Briefly explain try except-else-finally block?

Exception handling is crucial in Python due to its dynamic typing, interpreted nature, and extensive standard library. Handling exceptions enables programmers to make their code more reliable, debug runtime errors, and prevent the interpreter from stopping program execution.

**try:** The try block encloses the code that might raise an exception.

**except:** The except block is executed if an exception is raised in the try block. It contains the code to handle the exception.

**else:** The else block is executed if no exception is raised in the try block skipping the except block.

**finally:** The finally block is executed after the try and except blocks. It contains code that will be executed regardless of whether an exception was raised or not.

Syntax of exception handling:

try:

```
    #statement(s) where error may exist except
```

ExceptionName:

```
    #statement(s) to be executed when exception raises
```

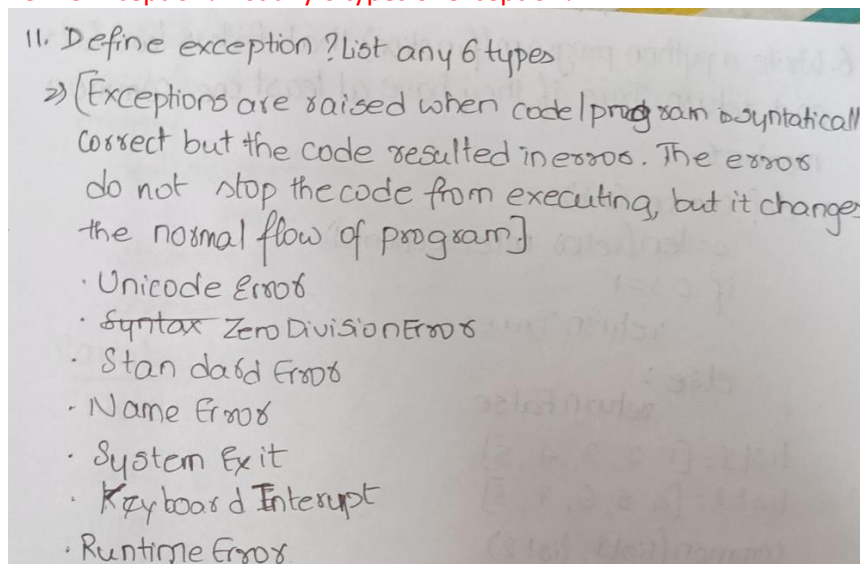
else:

```
    #statement(s) to be executed if no exception is raised
```

finally:

```
    #statement(s) that are always executed
```

## Define Exception? List any 6 types of exception?



## Define file and explain the two categories of files?

A file is a collection of data stored on a computer's storage device with a unique name and location. In Python, a file can be opened using the open function and can be read, written, or appended to, depending on the mode in which it was opened.

Two main categories of files in Python:

**Text files:** Text files contain human-readable characters and are used to store textual data such as strings, letters, numbers, and symbols. In Python, text files can be opened and manipulated using the built-in open() function and various methods such as read(), write(), and close().

Binary files: Binary files contain non-human-readable data such as images, videos, audio files, and program executables. In Python, binary files can also be opened and manipulated using the `open()` function with the appropriate file mode such as "rb" for reading binary data or "wb" for writing binary data.

### List out different types of file modes in python

Different types of file modes in Python:

>'r' (Read-only mode): Open the file for reading. This is the default mode.

>'w' (Write mode): Open the file for writing. If the file already exists, its contents are overwritten. If the file does not exist, a new file is created.

>'a' (Append mode): Open the file for writing and append data to the end of the file. If the file does not exist, a new file is created.

>'x' (Exclusive creation mode): Open the file for writing, but only if it does not already exist. If the file exists, an error is raised.

>'b' (Binary mode): Open the file in binary mode for reading or writing binary data.

>'t' (Text mode): Open the file in text mode for reading or writing text data. This is the default mode.

### Write a short note on:

#### >Advantages of storing data in a files:

Storing data in files in Python offers advantages such as data persistence, portability, flexibility, ease of use, and efficiency. Files can be read and written in various formats and processed in custom ways, making them suitable for large volumes of data and sharing between different systems and programs.

#### >Importance of closing file:

Closing a file in Python is essential to release system resources and ensure data integrity. If a file is not closed properly, it can result in data corruption, loss of data, or even system crashes. Moreover, the operating system imposes limits on the number of files that can be open at a time, so failing to close a file can cause performance issues and prevent other programs from accessing the file.

#### >seek ()function:

The `seek()` function in Python is used to change the current position of the file pointer within a file. The function takes two arguments: the offset value and the reference point. The offset value indicates the number of bytes to move the file pointer, while the reference point indicates the position from where the offset value should be applied. The `seek()` function is useful for random access to files and is commonly used with binary files.

#### >tell () function:

The `tell()` function in Python is used to determine the current position of the file pointer within a file. The function returns an integer value indicating the current byte offset from the beginning of the file. The `tell()` function is useful for tracking the progress of file operations, such as reading or writing data to a file. It is commonly used in conjunction with the `seek()` function to navigate to specific positions within a file.

What are the different ways of creating a new file and writing data into a file by giving an example program in Python?

- **Using write along with open() function:**

```
with open('file.txt', 'r') as f:
    file_contents = f.read()
    print(file_contents)
```

- **Using write along with with() function:**

```
with open('file.txt', 'w') as f:
    f.write('This is some sample data.\n')
    f.write('This is another line of data.')
    print('Data written to file.')
```

Write a Python program to know whether a file exists or not, if it is existed display the content of a file with an example programs.

```
import os
```

```
if os.path.exists('file.txt'):
    with open('file.txt', 'r') as f:
        file_contents = f.read()
```

```
    print(file_contents)
else:
    print('File does not exist.')
```

-THE END-