Unit-5

Object-Oriented Concepts

Syllabus:

OOP principles: classes, objects, 'self' variable, methods, constructor method, inheritance, overriding methods, data hiding.

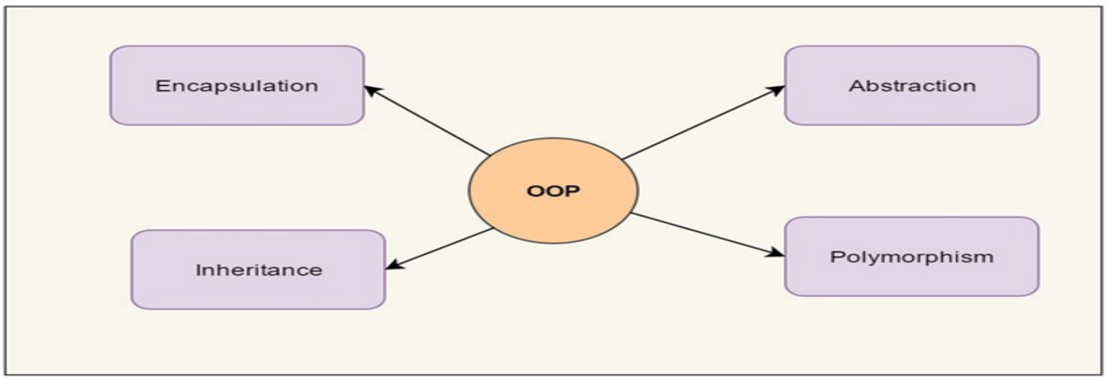
Learning Outcomes

At the end of the unit student will able to learn

- Describe how classes, objects, methods and Constructors are created and applied in java.
- Apply different types of Inheritance and can develop simple programs using Inheritance.
- Differentiate between Method overloading and Method Overriding

Principles of Object-Oriented Programming

1.Abstraction.
2.Encapsulation.
3.Inheritance.
4.Polymorphism.

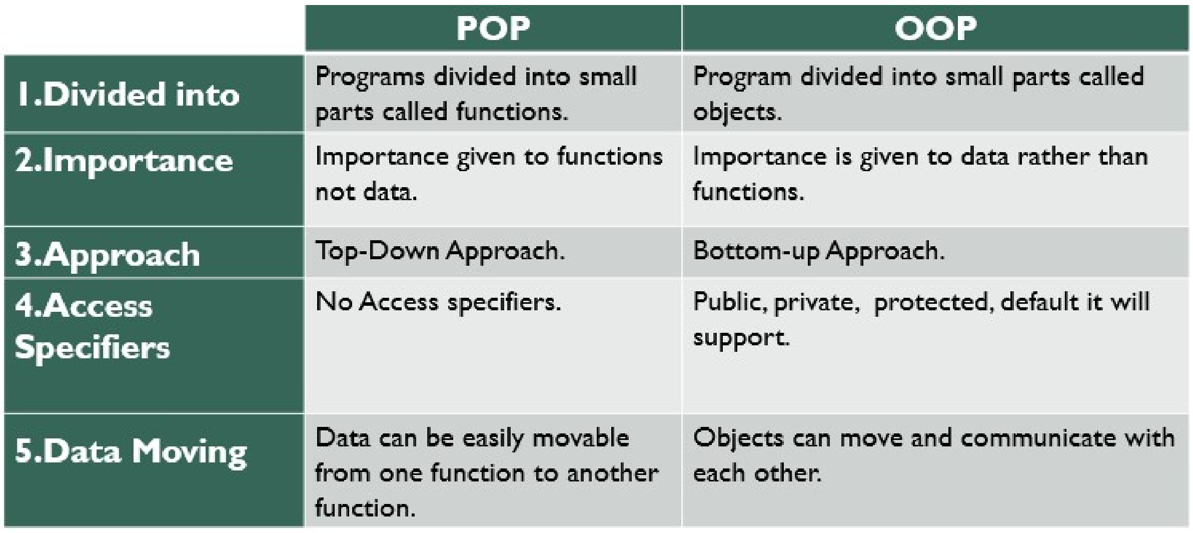


Four Pillars of Object Oriented Programming

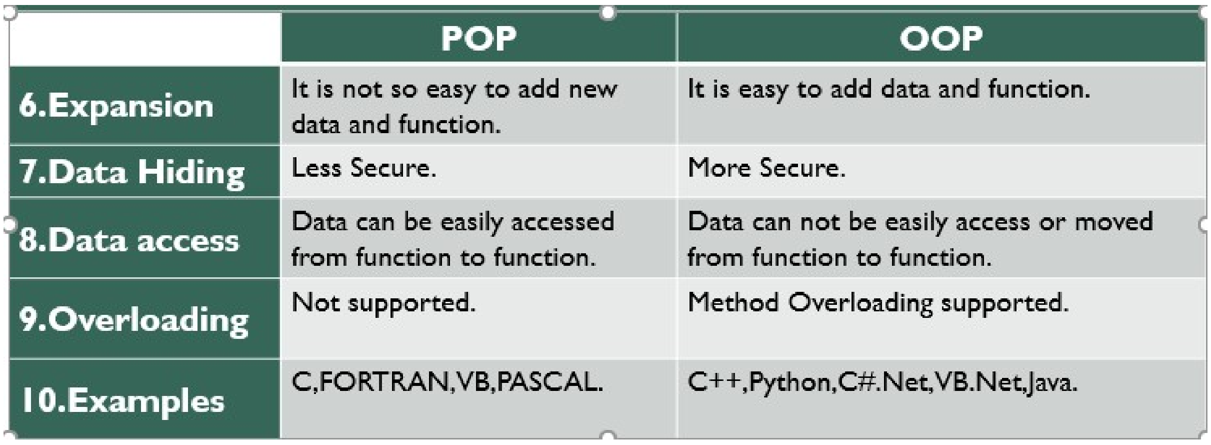
OOP Concepts

1. Class
2. Object
3. Method
4. Inheritance
5.Polymorphism
6.Data Abstraction and
7.Encapsulation.

Procedural Oriented Programming v/s Object Oriented Programming



	POP	OOP
1.Divided into	Programs divided into small parts called functions.	Program divided into small parts called objects.
2.Importance	Importance given to functions not data.	Importance is given to data rather than functions.
3.Approach	Top-Down Approach.	Bottom-up Approach.
4.Access Specifiers	No Access specifiers.	Public, private, protected, default it will support.
5.Data Moving	Data can be easily movable from one function to another function.	Objects can move and communicate with each other.



	POP	OOP
6.Expansion	It is not so easy to add new data and function.	It is easy to add data and function.
7.Data Hiding	Less Secure.	More Secure.
8.Data access	Data can be easily accessed from function to function.	Data can not be easily access or moved from function to function.
9.Overloading	Not supported.	Method Overloading supported.
10.Examples	C,FORTRAN,VB,PASCAL.	C++,Python,C#.Net,VB.Net,Java.

Classes

- Classes and objects are the two main aspects of object oriented programming.
- In fact, a **class is the basic building** block in Python.
- A class creates a new type and object is an instance (or variable) of the class.
- Classes provides a blueprint or a template using which objects are created.
- The syntax of class is:
-

```
class class_name:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Object

- In fact, in Python, everything is an object or an instance of some class.
- For example, all integer variables that we define in our program are actually instances of class int.
- Once a class is defined, the next job is to create an object (or instance) of that class.

```
object_name = class_name()
```

- The object can then access class variables and class methods using the dot operator (.)
- The syntax for accessing a class member through the class object is:

```
object_name.class_member_name
```

Programming Tip: Python does not require the new operator to create an object.

Example : Python program to access class variables

```
class ABC:
    var = 10    # class variable
obj = ABC()
print(obj.var)   # class variable is accessed using class object
```

OUTPUT

```
10
```

Programming Tip: `self` in Python works in the same way as the "`this`" pointer in C++.

Data Abstraction

Classes provide methods to the outside world to provide the functionality of the object or to manipulate the object's data. Any entity outside the world does not know about the implementation details of the class or that method.

Data Encapsulation

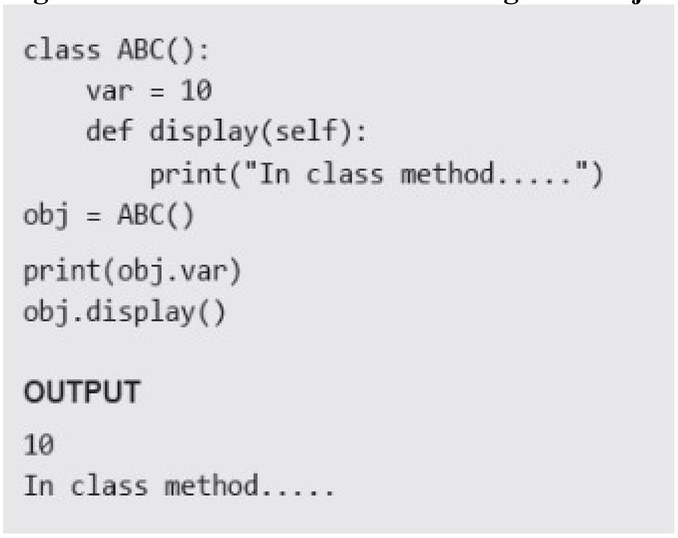
- **Data encapsulation, also called data hiding** organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.
- **Encapsulation** defines different access levels for data variables and member functions of the class. These access levels specify the access rights for example,
    a. Any data or function with access level **public** can be accessed by any function belonging to any class. This is the lowest level of data protection.
    b. Any data or function with access level **private** can be accessed only by the class in which it is declared. This is the highest level of data protection.

Class Methods and Self Arguments

- ***Class methods*** (or functions defined in the class) are exactly same as ordinary functions that we have been defining so far with just one small difference. Class methods must have the first argument named as **self.**

- The **self argument** refers to the object itself. That is, the object that has called the method. This means that even if a method that takes no arguments, should be defined to accept the self.
- Similarly, a **function defined to accept one parameter** will actually take **two-self and the parameter**, so on and so forth.
- Since, the class methods use self, they require an object or instance of the class to be used. For this reason, they are often referred to as ***instance methods***.

Example: Python Program to access class members using class object



```
class ABC():
    var = 10
    def display(self):
        print("In class method.....")
obj = ABC()

print(obj.var)
obj.display()
```

OUTPUT

```
10
In class method.....
```

Class Variables vs Instance Variables

Basically, these variables are of two types-

1. class variables and
2. object variables

- *Class var*iables are owned by the class and *object variables* are owned by each object.
- If a class has n objects, then there will be n separate copies of the object variable as each object will have its own object variable.
- The object variable is not shared between objects. A change made to the object variable by one object will not be reflected in other objects. On the other hand, if a class has one class variable, then there will be one copy only for that variable. All the objects of that class will share the class variable.

Note: -Since there exists a single copy of the class variable, any change made to the class variable by an object will be reflected to all other objects.

Example:

# Python program to differentiate between class and instance variable

```
class Student:
   cv=0        # class variable
   def __init__(self,var):
      self.iv=var      #instance variable or object variable
      Student.cv=Student.cv+1
   def displaystudent(self):    #instance method
```

```
        print("The value of class variable is",Student.cv)
        print("The value of instance variable is",self.iv)
obj1=Student(10)
obj1.displaystudent()
obj2=Student(20)
obj2.displaystudent()
obj3=Student(30)
obj3.displaystudent()

'''output: -
The value of class variable is 1
The value of instance variable is 10
The value of class variable is 2
The value of instance variable is 20
The value of class variable is 3
The value of instance variable is 30'''
```

Destructor-__del__() method

- The __del__() method does just the opposite work of __init__() method.
- The __del__() method is automatically called when an object is going out of scope.
- This is the time when object will no longer be used and its occupied resources are returned back to the system so that they can be reused as and when required. You can also explicitly do the same using the del keyword.

Example:

# Python program to delete the object memory

```
class Student:
    cv=0# class variable
    def __init__(self,var):# constructor
        self.iv=var#instance variable or object variable
        Student.cv=Student.cv+1
def displaystudent(self):#instance method
        print("The value of class variable is",Student.cv)
        print("The value of instance variable is",self.iv)
    def __del__(self):#destructor
        print("The object refered",self.iv," is deleted")
obj1=Student(10)
obj1.displaystudent()
obj2=Student(20)
obj2.displaystudent()
obj3=Student(30)
obj3.displaystudent()
del obj1
del obj2
del obj3
'''output:
The value of class variable is 1
The value of instance variable is 10
The value of class variable is 2
The value of instance variable is 20
The value of class variable is 3
The value of instance variable is 30
The object refered 10  is deleted
The object refered 20  is deleted
The object refered 30  is deleted'''
```

Public, Private and Protected variables

- ***Public variables*** are those variables that are defined in the class and can be accessed from anywhere in the program, of course using the dot operator.

- ***Private variables,*** on the other hand, are those variables that are defined in the class with a double score prefix (__). These variables can be accessed only from within the class.

- ***Protected variable*** of a class is accessible from within the class and are also available to its sub-classes. The variable is declared with single underscore (_) symbol.

Example:

#Python program to demonstrate the usage of public, private and protected variables

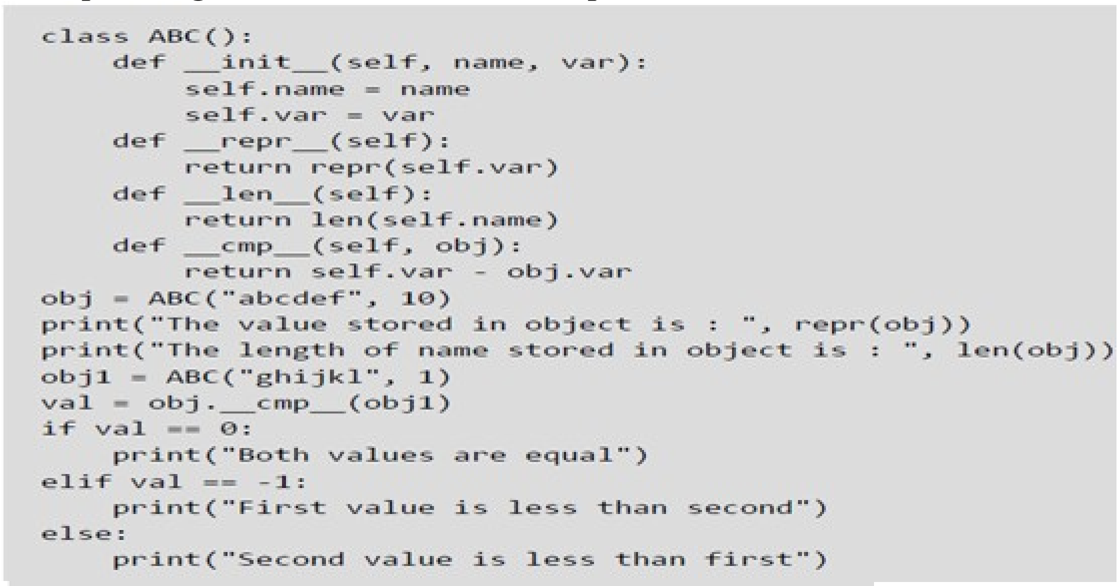
```
class CSEB:
    def __init__(self):
        self.var=10 #public
        self._var1=20 #protected
        self.__var2=30  #private
    def displaycseb(self):
        print("The value of public variable var is ",self.var)
        print("The value of protected variable var is",self._var1)
        print("The value of private variable var is",self.__var2)
obj=CSEB() # Creating Object
obj.displaycseb()
print("Outside of the class the public data is",obj.var)
print("Outside of the class the protected data is",obj._var1)
print("Outside of the class the private data is",obj.__var2)

'''output:-
The value of public variable var is  10
The value of protected variable var is 20
The value of private variable var is 30
Outside of the class the public data is 10
Outside of the class the protected data is 20
Traceback (most recent call last):
  File "D:/Pythonprogramscseb/publicprivateprotected.py", line 15, in <module>
 print("Outside of the class the private data is",obj.__var2)
AttributeError: 'CSEB' object has no attribute '__var2''''
```

Other Special Methods

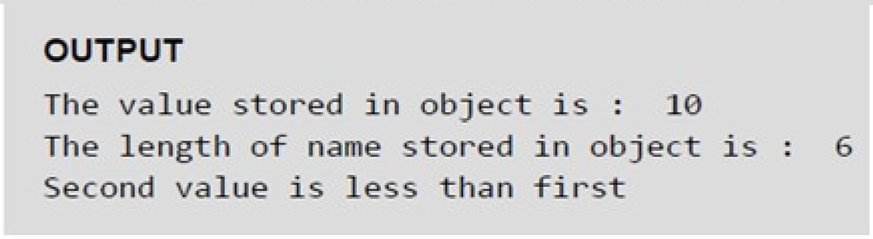
1. **__repr__()**: __ The __repr__() function is a built-in function with syntax repr(object). It returns a string representation of an object. The function works on any object, not just class instances.

2. **__cmp__()**: The __cmp__() function is called to compare two class objects.
3. **__len__()**: The __len__() function is a built-in function that has the syntax, len(object). It returns the length of an object.

Example :Program to illustrate the use of special methods



```
class ABC():
    def __init__(self, name, var):
        self.name = name
        self.var = var
    def __repr__(self):
        return repr(self.var)
    def __len__(self):
        return len(self.name)
    def __cmp__(self, obj):
        return self.var - obj.var
obj = ABC("abcdef", 10)
print("The value stored in object is : ", repr(obj))
print("The length of name stored in object is : ", len(obj))
obj1 = ABC("ghijkl", 1)
val = obj.__cmp__(obj1)
if val == 0:
    print("Both values are equal")
elif val == -1:
    print("First value is less than second")
else:
    print("Second value is less than first")
```



```
OUTPUT
The value stored in object is :  10
The length of name stored in object is :  6
Second value is less than first
```

Calling class method from another method within same class

- Instance methods are built functions into the class definition of an object and require an instance of that class to be called. To call the method, you need to qualify function with **self.**

Example: -Python program to call class method from another class method within same class.

```
class cse:
   def csea(self):
      print("Hi CSEA")
   def cseb(self):
      print("Hi CSEB")
   def csec(self):
      print("Hi CSEC")
   def csed(self):
      print("Hi CSED")
   def welcome(self):
self.csea()
      self.cseb()
      self.csec()
      self.csed()
obj=cse()
```

```
obj.welcome()
'''output:-
Hi CSEA
Hi CSEB
Hi CSEC
Hi CSED'''
```

Private Methods

- Like **private attributes**, you can even have **private methods** in your class.
- Usually, we keep those methods as private which have implementation details. So, like private attributes, you should also not use private method from anywhere outside the class. However, if it is very necessary to access them from outside the class, then they are accessed with a small difference.
- A private method can be accessed using the **object name as well as the class name** from outside the class.
- The syntax for accessing the private method in such a case would be.

objectname._classname__privatemethodname

Example: Python program to access the private methods outside of the class.

```
#Python program to access private methods
class cseb:
   def __init__(self,name,number):
      self.name=name
      self.__number=number
   def __displaycseb(self): #private method
      print("The name of the student is",self.name)
      print("The number of the student is",self.__number)
obj=cseb('hari',9294959697)
obj._cseb__displaycseb()  # Access private method\
'''output:-
The name of the student is hari
The number of the student is 9294959697'''
```

Nested classes

- It is also possible to define inner class in outer class.
- The syntax of creating inner class object is

Object=Outerclass.Innerclass()

Example Program: Python Program to demonstrate Nested classes

```
#Python program to demonstrate the nested class
class Outer:
   def __init__(self):
      self.var1=10
   def displayouter(self):
      print("The of outer variable is ",self.var1)
   class Inner:
      def __init__(self):
         self.var2=20
```

```
        def displayinner(self):
            print("The inner class variable is",self.var2)
obj=Outer()
obj.displayouter()
obj1=Outer.Inner()
obj1.displayinner()
'''output:-
The of outer variable is  10
The inner class variable is 20'''
```

Inheritance

A New class is derived from old class is said to be inheritance
Types of Inheritance

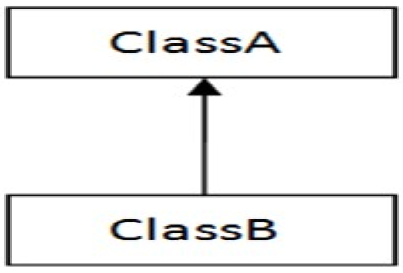
1. **Single Inheritance**
2. **Mutilevel Inheritance**
3. **Hierarchical Inheritance**
4. **Multiple Inheritance**
5. **Hybrid Inheritance**

Advantages of Inheritance in Python

- Python Inheritance provides Code reusability, readability, and scalability.
- Python Inheritance reduces the code repetition. ...
- By dividing the code into multiple classes, the applications look better, and the error identification is easy.

1.Single Inheritance

Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

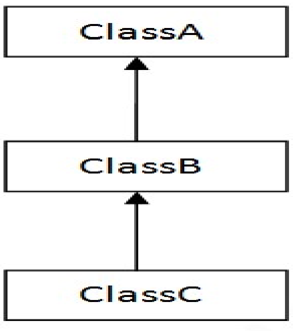


1) Single

```
#Python Program to implement single inheritance
class Student():
    def __init__(self,roll,name):
        self.roll=roll
        self.name=name
    def displaystudent(self):
        print("rollno",self.roll)
        print("name",self.name)
class StudentInfo(Student):
    def __init__(self,roll,name,Address):
        Student.__init__(self,roll,name)
        self.Address=Address
    def displaystudentinfo(self):
        print("Address",self.Address)
cobj=StudentInfo(501,'Ajay','Gudlavalleru')
cobj.displaystudent()
cobj.displaystudentinfo()

'''output:-
rollno 501
name Ajay
Address Gudlavalleru
'''
```

2.Multi-Level Inheritance

Def: The Derived class inherits another derived class is said to be multilevel inheritance.

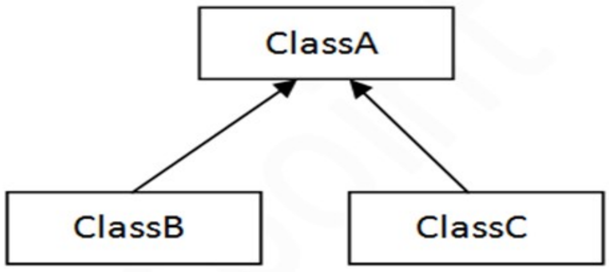


2) Multilevel

```
#Python program to implement multilevel inheritance
class Student: #parent class
    def __init__(self,roll,name):
        self.roll=roll
        self.name=name
    def displaystudent(self):
        print("The roll is",self.roll)
        print("The name is",self.name)
class StudentInfo(Student):#Childclass
    def __init__(self,roll,name,address):
        Student.__init__(self,roll,name)
        self.address=address
    def displaystudentinfo(self):
        print("The address is",self.address)
class StudentPhone(StudentInfo):#sub child class
    def __init__(self,roll,name,address,pno):
        StudentInfo.__init__(self,roll,name,address)
        self.pno=pno
    def displaystudentphone(self):
        print("The pno is",self.pno)
sobj=StudentPhone(501,'ajay','gudlavalleru',9848033338)
sobj.displaystudent()
sobj.displaystudentinfo()
sobj.displaystudentphone()

'''output:-
The roll is 501
The name is ajay
The address is gudlavalleru
The pno is 9848033338'''
```

3.Hierarchical Inheritance

Def: More than one derived class inherits the properties from single base class is said Hierarchical inheritance.

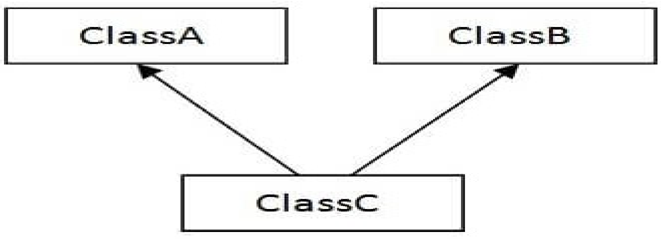


3) Hierarchical

```
#Python program to implement hierarchical Inheritance
class Details:   #parent class
   def __init__(self,name,address):
      self.name=name
      self.address=address
   def displaydetails(self):
      print("The name is",self.name)
      print("The address is",self.address)
class Student(Details):#Child class1
   def __init__(self,name,address,sid):
      Details.__init__(self,name,address)
      self.sid=sid
   def displaystudent(self):
      print("The student id is",self.sid)
class Employee(Details):#child class2
   def __init__(self,name,address,empid):
      Details.__init__(self,name,address)
 self.empid=empid
   def displayemployee(self):
      print("The employee id is",self.empid)
s=Student('hari','MTM',501)
e=Employee('Bhaskar','gudur','cse501')
print("The student details are")
s.displaydetails()
s.displaystudent()
print("The employee details are")
e.displaydetails()
```

```
e.displayemployee()


'''output:-
The student details are
The name is hari
The address is MTM
The student id is 501
The employee details are
The name is Bhaskar
The address is gudur
The employee id is cse501'''
```

4.Multiple Inheritance
Def: A Single derived class can inherit properties from more than one base class is said to be multiple inheritance.

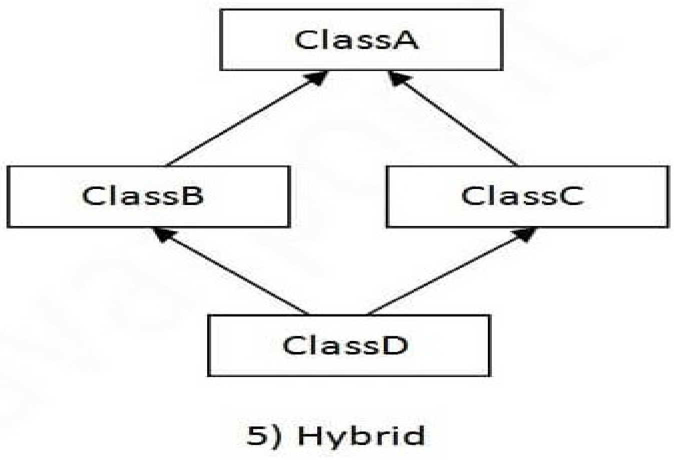


4) Multiple

```
#Python program to implement multiple inheritance
class AcademicMarks:#Parent1
   am=450
def displayacademicmarks(self):
      print("The academic marks",self.am)
class SportsMarks:#Parent2
   sm=50
   def displaysportsmarks(self):
      print("The sports marks ",self.sm)
class StudentMarks(AcademicMarks,SportsMarks):#Child class
   def __init__(self):
      self.tm=0
   def displaystudentmarks(self):
      self.tm=AcademicMarks.am+SportsMarks.sm
      print("The total marks of the student is",self.tm)
s=StudentMarks()
s.displayacademicmarks()
s.displaysportsmarks()
s.displaystudentmarks()

 '''output:
The academic marks 450
The sports marks 50
The total marks of the student are 500'''
```

5.Hybrid Inheritance

Def: A Combination of more than one type of inheritance is said to be hybrid inheritance.



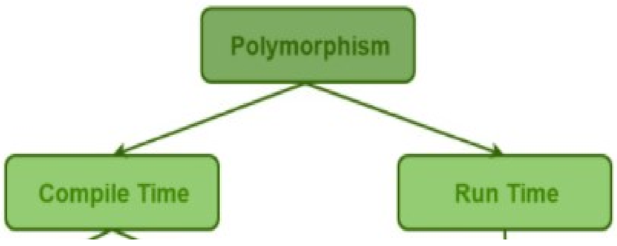
5) Hybrid

```
#Python program to implement hybrid inheritance
class StudentInfo:
   def __init__(self):
      self.name='Aravind'
      self.rollno='20481A05P9'
      self.address='Gudivada'
class Academicmarks(StudentInfo):
   def academic(self):
      self.am=50
      return self.am
class SportsMarks:
   def sport(self):
      self.sm=50
      return self.sm
class Student(Academicmarks,SportsMarks):
   def displaystudent(self):
          self.tm=self.academic()+self.sport()
        print("The name of the student is",self.name)
        print("The rollno of student is",self.rollno)
        print("the address of the student is",self.address)
        print("The marks of the student is",self.tm)
s=Student()
s.displaystudent()

'''output: -
===============          RESTART:          D:/Pythonprogramscseb/hybridInheritance.py
===============
The name of the student is Aravind
The rollno of student is 20481A05P9
the address of the student is Gudivada
The marks of the student is 100
```

'''

Polymorphism

Polymorphism is derived from Greek where poly means many and morphism means shapes I.e., One function or methods behaves like several forms.



Method Overloading: Compile time Polymorphism

When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

```
#PYTHON program to implement method overloading
class Area:
    def area(self,s):
        print("The area of square is",s*s)
    def area(self,l,b):
        print("The area of rectangle is",l*b)
a=Area();
a.area(4,5)
'''output:-
The area of rectangle is 20'''
```

Method Overriding: Runtime Polymorphism

Function overriding on the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

```
#Python program to implement method overriding
class Base:
    def fun(self):
        print("This is base class")
class Derived(Base):
    def fun(self):
        print("This is Derived class")
class Derived1(Derived):
    def fun(self):
        print("This id Derived class1")
d1=Derived1()
d1.fun()
b=Base()
b.fun()
```

'''**output:**

```
This id Derived class1
This is base class
>>> '''
```

Data Hiding

Data hiding in Python is the method to prevent access to specific users in the application. Data hiding in Python is done by using a double underscore before (prefix) the attribute name. This makes the attribute private/ inaccessible and hides them from users.

1. **private variables(__)**
2. **Protected variables(_)**

#Example

```
class S:
    def __init__(self):
        self.a=10#public var
        self._var=20#protected var
        self.__var1=30#private var
class C(S):
    def displayc(self):
        print("The value of a is",self.a)
        print("The value of var is",self._var)
class D(C):
    def displayd(self):
        print("The value of a is",self.a)
        print("The value of var is",self._var)
        print("The value of var1 is",self.__var1)#private cannot access
d=D()
d.displayc()
d.displayd()

"""output-
The value of a is 10
The value of var is 20
The value of a is 10
The value of var is 20
Traceback (most recent call last):
  File "D:/Pythonprogramscseb/Datahiding.py", line 17, in <module>
    d.displayd()
  File "D:/Pythonprogramscseb/Datahiding.py", line 14, in displayd
    print("The value of var1 is",self.__var1)
AttributeError: 'D' object has no attribute '_D__var1'
>>>
"""
```

Assignment cum Tutorial Questions

A) *Objective Questions*

1) In Python, a *class* is ___________ for a concrete object. [ ]
   A. a distraction
   B. a blueprint
   C. a nuisance
   D. an instance

2)class Dog:

```
        def __init__(self, name, age):
            self.name = name
            self.age = age
```

The correct way to instantiate the above Dog class is: [ ]
   A. Dog.create("Rufus", 3)
   B. Dog()
   C. Dog("Rufus", 3)
   D. Dog.__init__("Rufus", 3)

3) Object and class attributes are accessed using ___ operator in Python.

4) Which of the following is correct with respect to OOP concept in Python? [ ]

A. Objects are real world entities while classes are not real.
B. Classes are real world entities while objects are not real.
C. Both objects and classes are real world entities.
D. Both object and classes are not real.

5)How many objects and reference variables are there for the given Python code?
[ ]

```
class A:
    print("Inside class")
A()
A()
obj=A()
```

A. 2 and 1
B. 3 and 3
C. 3 and 1
D. 3 and 2

6)Which of the following is False with respect Python code? [ ]

```
class Student:
        def __init__(self,id,age):
            self.id=id
            self.age=age
            std=Student(1,20)
```

A. "std" is the reference variable for object Student (1,20)
B. id and age are called the parameters.
C. Every class must have a constructor.
D. None of the above

7) Which of the following is correct? [ ]

```
class A:
    def __init__(self,name):
         self.name=name
  a1=A("john")
  a2=A("john")
```

A. id(a1) and id(a2) will have same value.
B. id(a1) and id(a2) will have different values.
C. Two objects with same value of attribute cannot be created.
D. None of the above

8) In python, what is method inside class? [ ]
A. attribute
B. object
C. argument
D. function

9) What will be the output of below Python code? [ ]

```
  class A:
        def __init__(self,num):
              num=3
              self.num=num
        def change(self):
              self.num=7
  a=A(5)
  print(a.num)
  a.change()
  print(a.num)
```

A. 5
   7
B. 5
   5
C. 3
   3
D. 3
   7

10) _____ represents an entity in the real world with its identity and behavior. [ ]
A. A method
B. An object
C. A class
D. An operator

11) _____ is used to create an object. [ ]
A. class
B. constructor
C. User-defined functions
D. In-built functions

12) What will be the output of the following Python code? [ ]
```
class test:
   def __init__(self,a):
      self.a=a
     def display(self):
    print(self.a)
  obj=test()
 obj.display()
```

A. Runs normally, doesn't display anything
B. Displays 0, which is the automatic default value
C. Error as one argument is required while creating the object
D. Error as display function requires additional argument

13) What is Instantiation in terms of OOP terminology? [ ]
A. Deleting an instance of class
B. Modifying an instance of class
C. Copying an instance of class
D. Creating an instance of class

14) What will be the output of the following Python code? [ ]
```
class Demo:
    def __init__(self):
       pass
     def test(self):
    print(__name__)
obj = Demo()
 obj.test()
```

A. Exception is thrown
B.__main__
C. Demo
D. test

15) Which of the following Python code creates an empty class? [ ]

A. class A:
    return
B. class A:
    pass
C. class A:
D. It is not possible to create an empty class

16) Which of the following best describes inheritance? [ ]

A. Ability of a class to derive members of another class as a part of its own definition
B. Means of bundling instance variables and methods in order to restrict access to certain class members
C. Focuses on variables and passing of variables to functions
D. Allows for implementation of elegant software that is well designed and easily modified

17) Which of the following statements is wrong about inheritance? [ ]

A. Protected members of a class can be inherited
B.The inheriting class is called a subclass
C. Private members of a class can be inherited and accessed
D. Inheritance is one of the features of OOP

18) What will be the output of the following Python code? [ ]

```
class Test:

  def __init__(self):
      self.x = 0

class Derived_Test(Test):
     def __init__(self):
     self.y = 1
   def main():
      b = Derived_Test()
print(b.x,b.y)
main()
```

A. 0 1
B. 0 0
C. Error because class B inherits A but variable x isn't inherited
D. Error because when object is created, argument must be passed like Derived_Test(1)

19) What will be the output of the following Python code? [ ]

```
class A():
    def disp(self):
          print("A disp()")

class B(A):
      pass
```

```
 obj = B()
 obj.disp()
```

A. Invalid syntax for inheritance
B. Error because when object is created, argument must be passed
C. Nothing is printed
D. A disp()

20) Suppose B is a subclass of A, to invoke the __init__ method in A from B, what is the line of code you should write? [ ]
A. __init__(self)
B. __init__(self)
C.__init__(B)
D. __init__(A)

21) What type of inheritance is illustrated in the following Python code? [ ]

```
class A():
   pass
 class B():
   pass
 class C(A,B):
   pass
```

A. Multi-level inheritance
B. Multiple inheritance
C. Hierarchical inheritance
D. Single-level inheritance

22) What type of inheritance is illustrated in the following Python code? [ ]

```
class A():
      pass
 class B(A):
    pass
 class C(B):
        pass
```

A. Multi-level inheritance
B. Multiple inheritance
C. Hierarchical inheritance
D. Single-level inheritance

23) Which of the following statements isn't true? [ ]
A. A non-private method in a superclass can be overridden
B. A derived class is a subset of superclass
C. The value of a private variable in the superclass can be changed in the subclass
D. When invoking the constructor from a subclass, the constructor of superclass is automatically invoked

24) What will be the output of the following Python code? [ ]

```
        class A:
            def __init__(self):
                self._x = 5
        class B(A):
           def display(self):
           print(self._x)
        def main():
           obj = B()
          obj.display()
  main()
```

A. Error, invalid syntax for object declaration
B. Nothing is printed
C. 5
D. Error, private class member can't be accessed in a subclass

25) Which of the following best describes polymorphism? [ ]

A. Ability of a class to derive members of another class as a part of its own definition
B. Means of bundling instance variables and methods in order to restrict access to certain class members
C. Focuses on variables and passing of variables to functions
D. Allows for objects of different types and behavior to be treated as the same general type

B) Subjective Questions

1. Discuss about the OOP principles. [L2]
2. What is a class, how to define it? What are the class members and write a program to access them? [L2]
3. Explain class instantiation, and significance of constructor, __init__() method with an example. [L2]
4. What does the *self* argument signify in the class methods and constructor? [L2]
5. With the help of examples explain the concept of *instance, class and static* methods. [L2]
6. Differentiate a constructor and a method. [L3]
7. Explain data hiding with an example. Differentiate between public and private methods. [L2]
8. Define a class 'OOPDemo' and display the area of rectangle using an instance method, and areas of triangle using the concept of class method, and display your roll number using a static method. (NOTE: Implement using relevant variable types: self, cls, local. Initialize the instance variables b, h in a constructor and declare roll number as a local variable. [L3]

9. Write a python program that uses classes to store the name and marks of students. [L3]
10. Write a class that has a list of integers as data members and read (), display(), find_largest(), find_smallest(), sum() and find_mean() as its member functions. [L4]
11. Write a program with class Employee that keeps a track of the number of employees in an organization and also stores their names, designation and salary details. [L3]
12. Write a program that has a class Circle. Use a class variable to define the value of constant PI. Use the class variable to calculate area and circumference of a circle with specified radius. [L4]
13. Define inheritance. Explain the types of inheritance with programming examples. [L2]
14. If the specified attribute is not present in the current class, then where will be the search be made? Explain the concept in terms of multiple inheritance? [L2]
15. What will happen when a class inherits from another class with the same attributes or methods. Will it override them? [L3]
16. Write a program that has a class *Student* to store the details of students in a class. Derive another class *Toppers* from the Student that stores record of only top three students of the class. [L3]
17. We Care insurance company wants to calculate premium of vehicles. Vehicles are of two types – “Two-Wheeler” and “Four-Wheeler”. Each vehicle is identified by vehicle id, type, cost and premium amount. Premium amount is 2% of the vehicle cost for two wheelers and 6% of the vehicle cost for four wheelers. Calculate the premium amount and display the vehicle details. Write a Python program to implement the class chosen with its attributes and methods. [L4]

    Note:

    1. Consider all instance variables to be private and methods to be public
    2. Include getter and setter methods for all instance variables.