Q1. What is the meaning of multiple inheritance?

**When a class is derived from more than one base class** it is called multiple Inheritance.

Eg:

class Parent():

pass

class child1():

pass

class child2(Parent,child1):

pass

Q2. What is the concept of delegation?

Delegation provides a proxy object for any class thay you want on top of the main class. its like a wrapper to your class so that you can access limited resources of the main class.

it Wraps the object of main class into a smaller object with limited access

Simply Delegation means that you can include a instance of another class as an instance variable, and forward messages to the instance.

class Myclass:

def FSDS(self):

print('This is FSDS class')

def SQL(self):

print('It is the SQL class')

class NewClass:

def \_\_init\_\_(self,obj):

self.main = obj

def welcome(self):

self.main.FSDS()

m = Myclass()

n = NewClass(m)

m.FSDS()

n.main.FSDS()

n.welcome()

n.main.SQL()

Q3. What is the concept of composition?

 In the concept of Composition, a class refers to one or more other classes by using instances of those classes as a instance variable. irrespective of inheritence in this approach all the parent class members are not inherited into child class, but only required methods from a class are used by using class instances.

class Salary:

def \_\_init\_\_(self,pay):

self.pay = pay

def get\_total(self):

return self.pay\*12

class Employee:

def \_\_init\_\_(self,pay,bonus):

self.pay = pay

self.bonus = bonus

self.obj\_salary = Salary(self.pay)

def annual\_salary(self):

return f'Total Salary : {str(self.obj\_salary.get\_total())}'

obj\_emp = Employee(450,140)

print(obj\_emp.annual\_salary())

Q4. What are bound methods and how do we use them?

A bound method is **the one which is dependent on the instance of the class as the first argument**. It passes the instance as the first argument which is used to access the variables and functions

class Test:

def method\_one(self): # bound method

print("Called method\_one")

@classmethod

def method\_two(cls): # unbound method

print("Called method\_two")

@staticmethod

def method\_three(): # static method

print("Called method\_three")

test = Test()

test.method\_one()

test.method\_two()

Test.method\_two()

Test.method\_three()

Q5. What is the purpose of pseudoprivate attributes?

Pseudoprivate attributes are also useful in larger frameworks or tools, both to avoid introducing new method names that might accidentally hide definitions elsewhere in the class tree and to reduce the chance of internal methods being replaced by names defined lower in the tree. If a method is intended for use only within a class that may be mixed into other classes, the double underscore prefix ensures that the method won't interfere with other names in the tree, especially in multiple-inheritance scenarios