

# Python Advanced Assignment 5

## 1. What is the meaning of multiple inheritance?

Ans: Inheritance is nothing but reusing the code of Parent class by the child class. Similarly when a child class inherits its properties from multiple Parent classes this scenario is called **Multiple Inheritance**

```
In [1]: class Parent_one:
        pass
        class Parent_two:
        pass
        class child(Parent_one,Parent_two):
        pass
```

## 2. What is the concept of delegation?

Ans: Delegation provides a proxy object for any class that you want on top of the main class. It's 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 an instance of another class as an instance variable, and forward messages to the instance.

```
In [2]: class MyClass:
        def sayHi(self):
            print('Hey iam back')
        def whoAmI(self):
            print('Iam the main class')
        class NewClass:
            def __init__(self,obj):
                self.main = obj
            def welcome(self):
                self.main.sayHi()

m = MyClass()
n = NewClass(m)
m.sayHi()
n.main.sayHi()
n.welcome()
n.main.whoAmI()
```

```
Hey iam back
Hey iam back
Hey iam back
Iam the main class
```

## 3. What is the concept of composition?

Ans: In the concept of Composition, a class refers to one or more other classes by using instances of those classes as an instance variable. Irrespective of inheritance 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.

```
In [3]: 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(800,500)
print(obj_emp.annual_salary())
```

```
Total Salary : 9600
```

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

Ans: If a function is an attribute of class and it is accessed via the instances, they are called **bound methods**. A bound method is one

that has **self** as its first argument. Since these are dependent on the instance of classes, these are also known as **instance methods**.

```
In [4]: 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() # accessing through instance object
test.method_two() # accessing through instance object
Test.method_two() # accessing directly through class
Test.method_three() # accessing directly through class

Called method_one
Called method_two
Called method_two
Called method_three
```

## 5. What is the purpose of pseudoprivate attributes?

Ans: 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

Pseudoprivate names also prevent subclasses from accidentally redefining the internal method's names,

```
In [6]: class Super:
        def method(self): # A real application method
            pass
        class Tool:
            def _method(self): # becomes _Tool_method
                pass
            def other(self): # uses internal method
                self._method()
        class Sub1(Tool, Super):
            def actions(self):
                self.method()
        class Sub2(Tool):
            def __init__(self):
                self.method = 99
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js