MCT-243 : COMPUTER PROGRAMMING-II using Python 3 Object Oriented Programming

Prepared By:
Mr. Muhammad Ahsan Naeem

YouTube Playlist
https://youtube.com/playlist?list=PLWF9TXck7O_zuU2_BVUTrmGMCXYSYzjku

Lab 35: Object Oriented Programming

The class codes of last lab:

Point.py:

```
from math import sqrt
class Point:
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    @property
    def mag(self):
        return sqrt(self.x**2+self.y**2)
    def reset(self):
        self.move(0,0)
    def move(self,newx,newy):
        self.x=newx
        self.y=newy
    def movX(self,units):
        self.x+=units
    def movY(self,units):
        self.y+=units
    def __repr__(self):
        return f'({self.x},{self.y})'
    def __add__(self,other):
        return Point(self.x+other.x,self.y+other.y)
```

Student.py:

```
class MechaStudent:
    department='Mechatronics'
    offSubjects=['Mech','LA','ES','CP2','MOM','Isl/Pak','Proj']
    allStudents=[]
    def __init__(self,fName,lName,reg):
        self.fName=fName
        self.lName=lName
        self.reg=reg
        self.email=f'{reg.lower()}@uet.edu.pk'
        self.courses=['Proj']
        MechaStudent.allStudents.append(self)
    @property
    def fullName(self):
        return f'{self.fName} {self.lName}'
    def regCourse(self,*sub):
        for i in sub:
            if i in MechaStudent.offSubjects:
                if i not in self.courses:
                    self.courses.append(i)
            else:
                raise ValueError(f'{i} is not Offered!')
        self.courses.sort()
    def __repr__(self):
        return f'{self.fullName}-{self.reg}'
```

An object as input of instance method:

Instance methods are the methods which are bound to the instance and have the instance as first input argument named as self. There can be some case where we need another object (other than the **self**) as input argument of an instance method. One such possible case for **MechaStudent** class can be setting two students as group members for the course **Proj** meaning that those two will do the project together in a group. For that, we wish to create an instance method named as **setGroupMember** that will be called on some instance, of course, and should have another object which will be set as group member.
We can consider **groupMember** as one data attribute and set it to **None** inside **__init__()** method meaning that when a new student is created, it doesn't have any group member. Then we want to define the method **setGroupMember** that will set this attribute for the instance. The code is given here (only **__init__()** and **setGroupMember** for careful observation):

```
    def __init__(self,fName,lName,reg):
        self.fName=fName
        self.lName=lName
        self.reg=reg
        self.courses =['Proj']
        self.groupMember=None
        Student.allStudents.append(self)
    def setGroupMember(self, other):
        self.groupMember=other
```

Note that as convention, we should use keyword **other** to refer to the second object as we use **self** to refer to the actual object on which the method is being applied/called.

With above addition in the original class program, let's run the following Main program

```
from Student import MechaStudent
std1=MechaStudent('Anwar','Ali','MCT-UET-01')
std2=MechaStudent('Akbar','Khan','MCT-UET-02')
std3=MechaStudent('Asad','Shabir','MCT-UET-03')
std4=MechaStudent('Faisal','Iqbal','MCT-UET-04')
print(std1.groupMember) # Will print None
std1.setGroupMember(std2)
print(std1.groupMember) # Will print Akbar Khan-MCT-UET-02
print(std2.groupMember) # Will print None
```

If you try to print and see **groupMember** attribute of **std2**, it will still be **None** because the method **setGroupMember** sets this attribute for **self** only. We will have to apply this method on **std2** and provide **std1** to set **groupMember** for it. But the good approach is to handle it within the method **setGroupMember** such that **other** becomes the **groupMember** of **self** and **self** becomes the **groupMember** of **other**. Not only that, we must also check that the **groupMember** attribute of both **self** and **other** is set to **None** before setting them **groupMember** of each other to avoid setting a student as **groupMember** who is already **groupMember** of some other student. In such case we must generate an exception. Secondly, there must be a way to cancel out two group members. See the following code where these features are implemented:

```
    def setGroupMember(self, other):
        if(self.groupMember==None and other.groupMember==None):
            self.groupMember=other
            other.groupMember=self
        else:
            raise ValueError(
            'Both students should have no group member set earlier'
            )
    def dropGroupMember(self,other):
        if(self.groupMember==other.groupMember==None):
            return
        if(self.groupMember==other):
            self.groupMember=None
            other.groupMember=None
        else:
            raise ValueError('They are not group Members!')
```

With these methods added in class, run and see the output of following Main code:

```
from Student import MechaStudent
std1=MechaStudent('Anwar','Ali','MCT-UET-01')
std2=MechaStudent('Akbar','Khan','MCT-UET-02')
std3=MechaStudent('Asad','Shabir','MCT-UET-03')
std4=MechaStudent('Faisal','Iqbal','MCT-UET-04')

std1.setGroupMember(std2)
print(std1.groupMember) # Will print Akbar Khan-MCT-UET-02
print(std2.groupMember) # Will print Anwar Ali-MCT-UET-01
std1.dropGroupMember(std2)
print(std1.groupMember) # Will print None
print(std2.groupMember) # Will print None
std1.setGroupMember(std2)
print(std1.groupMember) # Will print Akbar Khan-MCT-UET-02
print(std2.groupMember) # Will print Anwar Ali-MCT-UET-01
# std1.setGroupMember(std3) # Will generate Error (ValueError: Both
students should have no group member set earlier)
```

Tasks:

[1] In main program create two lists of same number of students (at least 4). Then you have to make the students of both lists on corresponding indices as group members of each other. Verify the results by printing group members of first list.

Class Attributes (Data and Function):

We have already seen the class level data attribute. One such example is `department='Mechatronics'` in `MechaStudent` class. We know that the class data attribute can be accessed via the class and via the instance of the class. Another good example of a class data attribute in `MechaStudent` class is a list `allStudents` that contains each instance created from the class.

Now let's discuss the class methods. As mentioned earlier, the methods in a class have bindings. The instance methods are bound to the instances and likewise the class methods are bound to the class. Just like the instance method where the first input argument is the instance denoted by `self`, in class methods the first input argument is the class and is denoted by `cls`. By default, methods in a class are instance methods. To make a method as class method, that method must be decorated by the decorator `@classmethod`.

So, let's create a class method inside the `MechaStudent` class that will return a list of all students which have not been assigned a group member. First, think that why it should be a class method and not the instance method. Because it is not specific to one instance and we will not be applying it to a specific instance of the class. Rather we need to process the class attribute `allStudents` and hence should be a class method. This can be done as:

```
@classmethod
def withoutGroupMember(cls):
   return list(filter(lambda s: s.groupMember==None,cls.allStudents))
```

The method is tested in main program as:

```
from Student import MechaStudent
std1= MechaStudent('Anwar','Ali','MCT-UET-01')
std2= MechaStudent('Akbar','Khan','MCT-UET-02')
std3= MechaStudent('Asad','Shabir','MCT-UET-03')
std4= MechaStudent('Faisal','Iqbal','MCT-UET-04')

std1.setGroupMember(std2)
a=Student.withoutGroupMember()
print(a) #Will print [Asad Shabir-MCT-UET-03, Faisal Iqbal-MCT-UET-
04]
```

Static Method:

Unlike instance and class methods which are bound to instance and class respectively, the static methods are bound neither to instance nor the class. These methods do not need an instance or the class as first input argument. This means that these methods are not meant for accessing and processing instance or class attributes. Then what is the use of such methods?

These methods in general are not meant to be called outside the class though we can. These methods usually serve as helping or utility function to simplify a complex logic e.g., some calculation or checking the validity parameter for which we do not need any instance or class attribute. Let's see this with an example but first note that a method will become a static method by the decorator **`@staticmethod`** used on it.

We will create meaningful static methods later but here just for understanding purpose, let’s create a sample static method. We will name it as test. It will not have the instance or class as first input argument and can have some other input arguments. Let’s say, it will have a number as input argument and will return the square of the that number. This is how we will define it inside the class:

```
    @staticmethod
    def test(a):
        return a**2
```

We can use this method inside the class (usually in some other method) or outside the class. An example use outside the class is here:

```
print(MechaStudent.test(5))
```

How to choose between Instance Method, Class Method and Static Method:

Most of the methods in any class are meant for the instances of the class and hence are the instance method. A few methods are not for the instance and need the class data, and hence should be the class methods. There is very rare chance that a method neither needs the instance nor the class and hence we rarely see static methods in a class. So, generally we start with writing a method as Instance method and later if it is revealed that it does not need the instance but the class data only, it is changed to class method. And finally, if we come to know that it doesn’t even need a class level data, we convert that to a static method.

Tasks:

[2] We want to find the list of subjects in which no student is registered. Create a method in class with name **`notRegSub`** (Decide whether it should be instance method, a class method or static method). In main program, create a few students, register few subjects for those students and the call the method **`notRegSub`** to find and display the list of subjects in which no student is registered.

References:

[1] https://youtu.be/WZbgr14jyTk
[2] https://youtu.be/ml7pl2a-wK0