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 34: Object Oriented Programming

This Lab Session will start with the tasks related to OOP concepts covered in previous Lab Session. Save the Point class we created in last lab session with file name 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.x=units
    def showPoint(self):
        return f'({self.x},{self.y})'
```

Tasks:

[1] Create a new file with any name and import the `Point` class as:

```
from Point import Point
```

Do the following:

a. Create a user-defined function named as `addPoints` that will take two `Point` objects as input and will return a new point object as sum of the two input Points.
b. Create a list `L1` of five Point objects such that the `x` and `y` components of those points are random integers between `-10` and `10`.
c. Create another similar list `L2` of five points but this time the `x` and `y` components should be random integers between `-5` and `5`.
d. Create another list `L3` such that it has the five Points as the sum of corresponding Points in lists `L1` and `L2`. Use `map` function.
e. Display all Points in list L3 using the f`or` loop and `showPoint` function.
f. Sort the list `L3` based on the magnitude of the Point.
g. Display all Points in above sorted list using the `for` loop and `showPoint` function.

Similarly save the `MechaStudent` class code in the file `Student.py`. Here is the code we created in last lab session:

```
class MechaStudent:
    department='Mechatronics'
    offSubjects=['Mech','LA','ES','CP2','MOM','Isl/Pak']
    def __init__(self,fName,lName,reg):
        self.fName=fName
        self.lName=lName
        self.reg=reg
        self.email=f'{reg}@uet.edu.pk'
        self.courses=[]
    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()
```

h. There is one little change that now instead of name we have the properties **`fName`** and **`lName`** for first name and last name.

Tasks:

[2] Create a new file with any name and import the **`MechaStudent`** class as:

```
from Student import MechaStudent
```

Do the following:

a. The **`email`** should be in all lower letters. Update the class code such that it will be all lower-case letters.

b. Suppose that there is another course offered in this semester named as **`Proj`** and that is a compulsory course. Add the course in the **`offSubjects`** list and change the class code in a way that each time a **`MechaStudent`** object is created, the course **`Proj`** gets registered by itself.

c. Add another property for the **`MechaStudent`** object named as **`fullName`** such that its value is taken from the **`fName`** and **`lName`** properties. Moreover, if the **`fName`** or **`lName`** is changed, the **`fullName`** should get updated.

d. Create a class level list named as **`allStudents`**. It is required that whenever a **`MechaStudent`** object is created that should be appended to this class level list. So, we can access the class level list outside (or inside) the class and will get all student objects.

Magic Method __str__() and__repr__():

We have seen that if we print a class object it gets displayed as a little description about the object related to its memory location. For end user that is not a readable information. To print an object, we can create an instance method with required string being returned as we created **`showPoint()`** in case of **`Point`** class.

Let's do the same for the **`MechaStudent`** class. To print a student we can define an instance method named as **`showStudent()`** as:

```
    def showStudent(self):
        return f'{self.fullName}-{self.reg}'
```

In main program we can use it on any object of the class as:

```
print(std1.showStudent())
```

We can do it in a better way by using a special or the magical method named as `__str__()`. All we need to do is to change the `showStudent()` name to `__str__()` and the magic we get is that we don't need to call this function, rather we will pass the object in print statement and this magic method will be called automatically. Hence, very much like printing any other simple data type like int or string, we can print the object of our class. In above case, with `__str__()` defined, we can print the object simple as `print(std1)`. The class has the definition of `__str__()` as shown here:

```
    def __str__(self):
        return f'{self.fullName}-{self.reg}'
```

And in the main program we are simply using print statement as:

```
from Mechatronics import Student
std1=Student('Anwar','Ali','MCT-UET-01')
print(std1) # Will print Anwar Ali-MCT-UET-01
```

Remember that the return type of `__str__()` should be a string.

There is another magic method with similar use and is named as `__repr__()`. If we replace `__str__()` to `__repr__()` as:

```
    def __repr__(self):
        return f'{self.fullName}-{self.reg}'
```

And run the same Main program, we will get the same output. Before we discuss the difference between these two magic methods, let's see if we have both of these defined in the class, the print statement will pick which one of them. So, here is how the two methods are defined with different strings returned:

```
    def __repr__(self):
        return f'{self.fullName}-{self.reg}'
    def __str__(self):
        return f'{self.reg}-{self.fullName}'
```

Both methods differ in the order of two attributes (`fullName` and `reg`) in returned string. Now, if we run the following Main program:

```
from Mechatronics import Student
std1=Student('Anwar','Ali','MCT-UET-01')
print(std1)
```

The output will be:

```
MCT-UET-01-Anwar Ali
```

The output corresponds to `__str__()` method. Hence, we can conclude three points here:

- The `print` statement picks `__str__()` method to print the object.
- If `__str__()` is not defined, print statement will pick the `__repr__()` method.
- If both are not defined in class, print will display the object in default format with the memory location.

What is difference between the two?

Both `__str__()` and `__repr__()` return a string as representation of the object but there is a convention followed by the community to use `__str__()` for the end-users with the goal of better readability and use `__repr__()` for the developer using your class for the debugging purpose with a goal of complete and unambiguous representation of the object.

In the `MechaStudent` class, the purpose of `__str__()` will be to represent a student in readable format e.g. `Anwar Ali-MCT-UET-01` and the purpose of `__repr__()` will be to display the complete information of the student an again there is a convention for `__rep__()` method that it should return the string that can be directly copied and pasted to in the program to generate the same object. For example, a student is generated in main class as:

```
std1=Student('Anwar','Ali','MCT-UET-01')
```

Hence, the `__repr__()` method should return the string in above format. Using these conventions, the two methods for the Student class are defined as:

```
    def __str__(self):
        return f'{self.fullName}-{self.reg}'
    def __repr__(self):
        return f'Student("{self.fName}","{self.lName}","{self.reg}")'
```

And then in Main program print will pick the `__str__()` method as shown here:

```
std1=Student('Anwar','Ali','MCT-UET-01')
print(std1)      # Will print Anwar Ali-MCT-UET-01
```

The developer, using our class cannot directly use the print statement for `__repr__()`, hence he will have to use it directly by its name as shown here:

```
std1=Student('Anwar','Ali','MCT-UET-01')
print(std1.__repr__())
```

And it will have the following output:

```
Student("Anwar","Ali","MCT-UET-01")
```

You can see that the output is exactly in the format that is used to create that very object.

Disregarding the convention between __str__() and __repr__():

If we follow the convention as described for `MechaStudent` class, the two methods are defined as:

```
    def __str__(self):
        return f'{self.fullName}-{self.reg}'
    def __repr__(self):
        return f'Student("{self.fName}","{self.lName}","{self.reg}")'
```

In the Main program, if we define a few students and create a list of them and finally display that list as shown here:

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

The output is shown here:

```
[Student("Anwar","Ali","MCT-UET-01"), Student("Akbar","Khan","MCT-
UET-02"), Student("Asad","Shabir","MCT-UET-03"),
Student("Faisal","Iqbal","MCT-UET-04")]
```

You can see from the output that printing a list of objects picked `__repr__()` method instead of `__str__()` but we were looking for the end-user's representation we define in `__str__()` method. For a moment if we remove `__repr__()` method from the class with `__str__()` staying there and run the above Main program once again, the out will be:

```
[<Mechatronics.Student object at 0x000001FD9E71FFC8>,
<Mechatronics.Student object at 0x000001FD9E6CC8C8>,
<Mechatronics.Student object at 0x000001FD9E82C048>,
<Mechatronics.Student object at 0x000001FD9E82C308>]
```

The output is the default format not picking the `__str__()` method of the class. Now let's change `__str__()` to `__repr__()` which is violation of the convention being followed.

```
    def __repr__(self):
        return f'{self.fullName}-{self.reg}'
```

Now if run the same Main program once again, this will be the output:

```
[Anwar Ali-MCT-UET-01, Akbar Khan-MCT-UET-02, Asad Shabir-MCT-UET-03,
Faisal Iqbal-MCT-UET-04]
```

This is the output we were looking for. Hence, if we have to create and display the list of class objects, it will be better to disregard the convention. Usually, it is a common practice to have the list of objects.

Tasks:

[3] Create a new file with any name and import `MechaStudent` class. Create a few students and print the `allStudents` list. Sort the same list based on number of registered courses and print again.

[4] Define the `__repr__` for the Point class and update the part `e` and `g` of `Task-1`