# 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 32 : Object Oriented Programming

**Object-oriented programing** is a paradigm where we create objects. Objects are created from abstract data types that encapsulate data and functions together.

There are primarily two methods of programming in use today: procedural and object-oriented. The earliest programming languages were procedural, meaning a program was made of one or more procedures. You can think of a procedure simply as a function that performs a specific task such as gathering input from the user, performing calculations, reading or writing files, displaying output, and so on. The programs that you have written so far were procedural.

Whereas procedural programming is centered on creating procedures (functions), object-oriented programming (OOP) is centered on creating objects. An object is a software entity that contains both data and procedures. The data contained in an object is known as the object's data attributes. An object's data attributes are simply variables that reference data. The procedures that an object performs are known as methods. An object's methods are functions that perform operations on the object's data attributes. The object is, conceptually, a self-contained unit that consists of data attributes and methods that operate on the data attributes.

Let's define a class for students of Mechatronics department and name it as **MechaStudent**. An empty class with this name is defined as shown here:

```
class MechaStudent:
    pass
std1=MechaStudent()
print(f'std1= {std1}')
print(f'Type of std1= {type(std1)}')
print(f'Is std1 instance of MechaStudent: {isinstance(std1,MechaStude
nt)}')
```

No class attribute (data or function) is defined inside the class at the moment. When the class is used in the main program by creating an object (or instance) and we can see the output to verify that the created object is an instance of class **MechaStudent**. The output is shown here:

```
std1= <__main__.MechaStudent object at 0x000002B54D6A2AC8>
Type of std1= <class '__main__.MechaStudent'>
Is std1 instance of MechaStudent: True
```

In the first line of the main program of the above code an object is created, we say that an object or instance is **instantiated**.
You can see **__main__** before class name in the first two outputs, that is because Python creates a module named as **__module__** in which the whole class is defined. There is nothing to worry about it as this is done and managed automatically.

Once an object is created, we can add different attributes to it as shown here:

```
std1=MechaStudent()
std2=MechaStudent()
```

```
std1.name='MUHAMMAD USMAN'
std1.reg='2018-MC-01'
std1.sec='A'
std2.name='DANISH'
std2.reg='2018-MC-51'
std2.sec='B'
```

And later, we can access these attributes in anyway e.g.:

```
print(std1.reg) #Will print 2018-MC-01
```

There is a lot of code we need to write every time we want to set data attributes after instantiating an object. A better approach will be to use a function that should do this for us. So, let's create our first function/method in our class that will set the data attributes to the object of **`MechaStudent`** class. This process is known as initializing the object. So, let's define the function within the class as:

```
class MechaStudent:
    def initialize(obj,n,r,s):
        obj.name=n
        obj.reg=r
        obj.sec=s
```

Note: You might get some warnings for above code depending upon the IDE, for this moment just ignore those.
Once defined, we can use this function on Instantiated objects as shown here:

```
#Objects Instantiation

std1=MechaStudent()
std2=MechaStudent()

#Objects Initialization

MechaStudent.initialize(std1,'MUHAMMAD USMAN','2018-MC-01','A')
MechaStudent.initialize(std2,'DANISH','2018-MC-51','B')

#Using the Objects
print(std1.name)
print(std2.sec)
```

See carefully the way, **`initialize()`** function is being used. It is used by calling it on class and passing in the object as first input argument as defined within the function.

There is a better way to use the function/method. For that, consider the following section first:

## Method Bindings:

Methods defined within the class have the bindings i.e. they are bound(connected) to something. Based on these bindings, methods are of three types:

### Instance Method:

A method bound (connected) with the instance/object of a class is known as Instance Method. By default, a method defined in class is Instance Method.

### Class Method:

A method bound with the class is known as Class Method. We will see later, how to define a class method.

### Static Method:

A method defined in a class but is neither bound to class or the instance is known as Static Method. We will explore that later.

## Instance Method:

As mentioned earlier, a method defined in a class are Instance Method i.e. they are bound to the instance of the class. Earlier we defined a method and used it with class as:

```
MechaStudent.initialize(std1,'MUHAMMAD USMAN','2018-MC-01','A')
```

We can call this method directly on the instance. **When we call it on an instance, that instance is passed as the first input argument automatically**. So, we should not pass that object by ourselves. See the code here:

```
class MechaStudent:
    def initialize(obj,n,r,s):
        obj.name=n
        obj.reg=r
        obj.sec=s

#Objects Instantiated
std1=MechaStudent()
std2=MechaStudent()

#Objects Initialized
std1.initialize('MUHAMMAD USMAN','2018-MC-01','A')
std2.initialize('DANISH','2018-MC-51','B')

print(std1.name)
print(std2.sec)
```

See carefully that the function is defined with four input arguments but when used, we are passing only three because the instance gets passed as first input argument automatically because it is an Instance Method.

If you try to pass the instance by yourself as did while calling it with class as:

```
std1.initialize(std1,'MUHAMMAD USMAN','2018-MC-01','A')
```

You will see the following error:

```
TypeError: initialize() takes 4 positional arguments but 5 were given
```

You can see that the interpreter gives the error that 5 arguments were given while actually 4 were given. As mentioned earlier, the instance is passed as first input argument by default so even apparently 4 arguments are passed in above statement but actually, they are 5.

## Naming Conventions:

We defined the function as:

```
def initialize(obj,n,r,s):
        obj.name=n
        obj.reg=r
        obj.sec=s
```

This worked but there are a few naming conventions followed by the Python community and we must follow these. The first naming convention is to use name **`self`** as first input argument for the instance. Secondly, the names of the input arguments should be same as the names of data attributes of the instance. With these conventions, the above code should be written as:

```
def initialize(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
```

## Magic Method __init__():

Python has a special method or the magic method **`__init__()`** known as constructor in other languages that we should define inside the class. So, if we just replace the above function name to this magic function, it is shown here:

```
class MechaStudent:
    def __init__(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
```

**What magic this function does?**

With this function defined, now we can instantiate and initialize the object at the same time. Now, we can pass in the input arguments at the time of instantiation and the magic function gets called automatically. Hence, a new object will be created as:

```
std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
```

## Adding data attributes latterly:

If a class is defined with a `__init__()` method, we must provide all data attributes defined inside `__init__()` method. Latterly, we can change or add more data attributes as shown here:

```
class MechaStudent:
    def __init__(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
#Objects Initialization
std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
std2=MechaStudent('DANISH','2018-MC-51','B')

#Update data attributes
std1.sec='B'
print(std1.sec)

#Adding more attributes
std2.email='2018-MC-51@mct.uet.edu.pk'
print(std2.email)
print(std1.email) #This will generate the error
```

## Setting a data attribute from another attribute:

We can also set a data attribute from one or more other attributes. For example, if we want to set email for each student, we can do it like:

```
class MechaStudent:
    def __init__(self,name,reg,sec,email):
        self.name=name
        self.reg=reg
        self.sec=sec
        self.email=email
```

If each student has to be assigned the email in format as reg@mct.uet.edu.pk then it will be better to define it as:

```
class MechaStudent:
    def __init__(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
        self.email=reg+'@mct.uet.edu.pk'
```

Now we don't need to pass email while initializing the instance. It will automatically be created and assigned to the instance as shown here:

```
std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
```

```
print(std1.email)
```

## Data Attributes of Class:

We have seen data attributes of instance. We can also define data attribute(s) for a class. It will assign that attribute to each instance of the class without manually doing it. For example, if we want to set **department** as data attribute of the instance and we want to set it to **"Mechatronics"** for each student, then we should set is as Class attribute as shown here:

```
class MechaStudent:
    department='Mechatronics'
    def __init__(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
        self.email=reg+'@mct.uet.edu.pk'
#Objects Initialization
std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
print(std1.department)
```

Later, we can change this data attribute of any instance and it will be changed for that instance without effecting the other instances. See this example:

```
std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
std2=MechaStudent('DANISH','2018-MC-51','B')
std1.department='Civil'
print(std1.department)
print(std2.department)
```

## Adding more Instance Methods:

We can add a list of offered subjects with few subjects in there as a Class attribute and then can have a list of registered subjects as Instance data attribute and have its initial value set to empty list. Further we want to create a new instance method names as **registerSubj()** that must take one or more subjects as input argument (other than the instance itself) and should add those to the list of registered subjects. The complete code is shown here:

```
class MechaStudent:
    department='Mechatronics'
    offSubjects=['Mech','LA','ES','CP2','MOM','Isl/Pak']
    def __init__(self,name,reg,sec):
        self.name=name
        self.reg=reg
        self.sec=sec
        self.email=reg+'@mct.uet.edu.pk'
        self.subjects=[]
    def registerSubj(self,*subj):
        for i in subj:
            if i in MechaStudent.offSubjects:
                self.subjects.append(i)

std1=MechaStudent('MUHAMMAD USMAN','2018-MC-01','A')
std2=MechaStudent('DANISH','2018-MC-51','B')
print(std1.subjects)
std1.registerSubj('LA','CP1','CP2')
print(std1.subjects)
```

In above code we have checked if the subject name is in offered subject list or not. If not in that list, we are not adding the subject. A better approach will be to raise an exception for such case (`ValueError` is suitable here) as shown here:

```
def registerSubj(self,*subj):
        for i in subj:
            if i in MechaStudent.offSubjects:
                self.subjects.append(i)
            else:
                raise ValueError('Subject should be from offered subj
ect list')
```

## ***Tasks:***

**[1]** Upgrade the student class so that after registering the subject(s) the list of registered subjects should get sorted. Create an instance of class, add a few subjects and display them to verify if they are sorted.

**[2]** For the same student class, create a list of few students with different registered subject. Sort the list based on number of subjects registered

**[3]** A **Point** on a cartesian plane has two co-ordinates as x-coordinate and y-coordinate. Do the followings:

a. Define a class named as `Point`.
b. A Point class instance should have two data attributes. Define those attributes as `x` and `y` with default value of `0` for both.
c. Define an instance method named as `reset()`. It will not have any input argument (except `self`) and it should reset the point to origin i.e. `0` for both `x` and `y` coordinates.
d. Define an instance method named as `move()`. It will have two input arguments as `x` and `y` coordinates and will set the `x-y` coordinates of the point to these inputs.
e. Update the `reset()` method created at in (c) so that it uses `move()` method to reset the coordinates.
f. Define two instance methods named as `xTranslate()` and `yTranslate()`. These methods will take one input argument and will move the point instance from its current position to a new position which is input units towards `x` direction in case of `xTraslate()` and towards `y` direction in case of `yTranslate().`
g. Define an instance method named as `showPoint()` that will display the points as `(x,y)` on the screen.
h. Define an instance method named as `distOrigin()` that will calculate the distant of the point from origin.

Use all above methods in main program to test their correct working.