OOP [Basics | Part-01]

... ...ST_Sagor_T04

#Create a class and initiate object...

In [1]:

```
# Design or Blueprint
class Usis:
    pass

# Driver or Tester
obj=Usis()
```

#Create a class, write the signature of default constructor(abstract) and initiate object...

In [2]:

```
# Design or Blueprint
class Usis:
    def __init__(self):
        pass

# Driver or Tester
obj=Usis()
```

```
#Yet, no difference has been observed, right?
```

#Create a class, implement default constructor and initiate object...

In [3]:

```
# Design or Blueprint
class Usis:
    def __init__(self):
        print("Hello from constructor!!!")

# Driver or Tester
obj=Usis()
```

```
Hello from constructor!!!
```

#Create a class, a class variable, implement default constructor and initiate object...

In [4]:

```
# Design or Blueprint
class Usis:
    user_dashboard_count=0
    def __init__(self):
        print("Hello from constructor!!!")

# Driver or Tester
obj=Usis()
```

Hello from constructor!!!

#Note: Now we can observe a fact, that is whenever we create an instance to map with a class, we are actually calling a function at the backend, and it is the _ *init* _(self)>>>default constructor of that class.

#Create a class, a class variable, implement default constructor, initiate object and call the class variable using the reference object...

In [5]:

```
# Design or Blueprint
class Usis:
    user_dashboard_count=0#class variable or static variable
    def __init__(self):
        print("Hello from constructor!!!")

# Driver or Tester
obj=Usis()
print(obj.user_dashboard_count)
```

Hello from constructor!!!
0

#Create a class, a class variable, implement default constructor, initiate object, update the class variable for each instance inside the constructor and call the class variable using the reference object...

In [6]:

```
# Design or Blueprint
class Usis:
    user_dashboard_count=0#class variable or static variable
    def __init__(self):
        Usis.user_dashboard_count+=1#update the class variable using class-name

# Driver or Tester
obj=Usis()
print("obj>class variable call>>>",obj.user_dashboard_count)
```

obj>class variable call>>> 1

#Check the previous example for two objects and many more...

In [7]:

```
# Design or Blueprint
class Usis:
    user_dashboard_count=0#class variable or static variable
    def __init__(self):
        Usis.user_dashboard_count+=1#update the class variable using class-name

# Driver or Tester
obj1=Usis()
print("obj1>class variable call>>>",obj1.user_dashboard_count)
obj2=Usis()
print("obj2>class variable call>>>",obj2.user_dashboard_count)
```

obj1>class variable call>>> 1
obj2>class variable call>>> 2

#Note: A class variable is a shared variable. So if we change the value for one instance, it will affect all other instance as a whole...

#Check the id or memory location of each object...

In [8]:

```
# Design or Blueprint
class Usis:
    def __init__(self):
        pass

# Driver or Tester
obj1=Usis()
print("id(obj1)>>>",id(obj1))
obj2=Usis()
print("id(obj2)>>>",id(obj2))
```

id(obj1)>>> 2770045143880
id(obj2)>>> 2770045142408

Are they same?

In [9]:

```
obj1==obj2
```

Out[9]:

False

#Note: This is why the OOP concept is so powerful and we can keep track of data fed into a system using different memory ids and their referenced objects...

#Have you seen a keyword "self" that was passed in the **init**() ?

#Let's print the id(self) and find out the definition of self...

In [10]:

```
# Design or Blueprint
class Usis:
    def __init__(self):
        print("id(self)>>>",id(self))

# Driver or Tester
obj=Usis()
print("id(obj)>>>",id(obj1))
```

```
id(self)>>> 2770045210888
id(obj)>>> 2770045143880
```

#Note: We can notice from the above scenario that the self is actually the inner reference of the initiated object and the object is the outer reference to the class to access the (public) class variables and (class, instance,static) methods of that class. As a result, though we are not passing any arguments while initiating the object, the object itself is being passed as the first parameter...

#Let's move to our regular functions and local variables

In [11]:

```
# Design or Blueprint
class Usis:
    user_dashboard_count=0#class variable or static variable
    def __init__(self):
        Usis.user_dashboard_count+=1#class variable or static variable
    #instance method
    def registration(self,name,email,password):
        self.name=name
#         self.name>>>instance variable...!
#         name>>>passed as an argument>>>local variable...!!!
        self.email=email
        self.password=password

# Driver or Tester
obj=Usis()
obj.registration("Hamim","hamim.cse.2020@gmail.com","1C9B5y$P")
```

In [12]:

```
1 print(obj)
```

<__main__.Usis object at 0x00000284F38F9CC8>

In [13]:

```
1 print(id(obj))
```

2770045213896

In [14]:

```
1 print(obj.name)
```

Hamim

In [15]:

```
1 print(obj.email)
```

hamim.cse.2020@gmail.com

In [16]:

```
1 print(obj.password)
```

1C9B5y$P

#Note: Now you see? We can actually, use the object to set and fetch those values from the class...
Here,

def registration(self,name,email,password)	value
self	1641401138184
name	Hamim
email	hamim.cse.2020@gmail.com
password	1C9B5y$P

#Note: In OOP we say methods, instead of functions(actions or behavior of an instance of class)...