In [1]:
```
class Student:
    def __init__(mohit,name,rollno,marks):
        mohit.name=name
        mohit.rollno=rollno
        mohit.marks=marks
        print("Constructor is cALLED")
        print(mohit.name)
    def talks(mohit):
        print("Hello My name is ",mohit.name)
        print("My roll no is",mohit.rollno)
        print("MY marks is",mohit.marks)
s=Student("Pratyush",987,99)
s1=Student("Twarita",99,90)
s.talks()
s1.talks()
```

```
Constructor is cALLED
Pratyush
Constructor is cALLED
Twarita
Hello My name is  Pratyush
My roll no is 987
MY marks is 99
Hello My name is  Twarita
My roll no is 99
MY marks is 90
```

In [ ]:
```
#Types of Attributes/Variables
1.Instance Variable/Attributes --> Object level variables/attributes
2.Static Varible/Attributes -->Class level variables/attributes
3.Local variables--> inside any method
```

In [ ]:
```
Attributes and variable both are same
```

In [ ]:
```
#Types of Methods - Instance , Class and Static
```

In [ ]:
```
1.Instance Methods(Object Level Method)-->if you are using instance variable inside any method then
overall method will be an instance method.
2.The first argument of the instance method will be always be self.
3.Instance method are generally used to access instance variable
```

In [13]:
```
class Dog:
    def __init__(self,color,breed):
        self.color=color
        self.breed=breed

    def bark(self):
        print("This Dog color is",self.color)
        print("This dog breed is",self.breed)
    def sniffing(self):
        pass

x=Dog("Black","Labrador")
x.bark()


#You can call instance method with the help of object reference outside the class
#if You want to call instance method within the class then you need to use self variable
```

```
This Dog color is Black
This dog breed is Labrador
```

In [ ]:
```
#Class Method -->if you are using static variable(class level) inside any method then overall
#method will be a class method
If you want a define a class method then you need to use @classmethod decorator
if you are defining class method then you need to pass atleast one argument(cls) that is mandatory
you can directly access class method without creating an object
```

In [7]:
```
class Animal:
    legs=4
    @classmethod
    def walk(cls,name):
        print(str(name)+" walks with "+str(cls.legs)+" legs")
Animal.walk("Cat")
#If you want to access the static variable inside the class method then you need to use
#cls variable
#If you wnat to access the static variable inside the instance method then you need to use
#class name
```

```
Cat walks with 4 legs
```

In [ ]:
```
#Static Method -->General Utility methods
Inside these method we cannot use any instance or class variable
We can declare static method by using @staticmethod decorator
```

In [12]:
```
class Math:
    @staticmethod
    def add(x,y):
        print("The sum is ",x+y)
    @staticmethod
    def sub(x,y):
        print("The sub  is ",x-y)
    @staticmethod
    def product(x,y):
        print("The sub  is ",x*y)
#You can access static method with the help of class name or object reference
Math.add(10,20)
x=Math()
x.add(10,200)
```

```
The sum is  30
The sum is  210
```

In [ ]:
```
#important thing related to Methods
In general programming we are only using instance and class methods
Static method are not generally used
Instance method  --> 80%  --> object level methods
Class Method ---> 15% --> class level methods
Static method  --> 5%---> general utility method
```

In [ ]:
```
#Access Modifiers
Three types of access modifiers
1.Public Access modifiers
2.Private access modifiers
3.Protected Access modifiers
Facebook --> Profile Lock ---> private mode --> Only friends can see that profile
Linkedin --> 1st, 2nd connection ---> public
inheritance --> Protected
```

In [ ]:
```
#Public Access Modifier --> by default each attributes/variable is public in python
We can access public access modifier variables/attributes either within the class or outside the
class
Example:
    name="Anil"
```

In [ ]:
```
#Private access modifier --> can be accessed within the class.(from outside the class you cannot
access)
Example:
    __name="Sujita"
```

In [ ]:
```
#Protected Access modifiers --> can be accessed within the class and outisde the
#class only in child class. We can specify protected access modifier with the
help of single underscore.
Example:
    _name="Sagar"
```

In [19]:
```
class Test:
    x=10     #Public
    _y=20     #protected --> within the class or child class
    z=30     #private --> within the class

    def m1(self):
        print(Test.x)
        print(Test._y)
        print(Test.z)
a=Test()
a.x
a.z
```

Out[19]:
```
30
```

In [ ]:
```
#Getters and Setters --> these methods are generally used to access private variable
outisde the class by external user.(These methods are instance methods)
Data Validation
```

In [ ]:
```
Syntax of Getter and Setter Method
Getter Method: --> get the value
    def get_variable_name(self):
        return self.__variable
Setter Method --> set the value
    def set_variable_name(self,x):
        self.__variable_name=x
```

In [27]:
```
class Bank:
    def __init__(self,Account_number=0):
        self.__Account_number=Account_number
        print("Construtor Called")

    #Getter Method
    def get_Account_number(self):
        print("GetterMethod is Called")
        return self.__Account_number

    #Setter Method
    def set_Account_number(self,x):
        print("Setter method is called")
        self.__Account_number=x

Indian_bank=Bank()
Indian_bank.set_Account_number(10009)
print(Indian_bank.get_Account_number())
```

```
Construtor Called
Setter method is called
GetterMethod is Called
10009
```

In [ ]:
```
#Pillers of Object Oriented Programming
1.Encapsulation  --> Binding the data and functions together into one single enitity that is nothing
but a class
2.Polymorphism --> Poly means many , Morphism means forms
3.Inheritance  -->Code Reusability
4.Abstraction -->  Hiding the ireellavant data from the user
```