OOP in Python

To map with real world scenarios, we started using objects in code.

This is called object oriented programming.

Class & Object in Python

Class is a blueprint for creating objects.

```
 #creating class

class Student:
     name = “karan kumar”


 #creating object (instance)

s1 =  Student( )
print( s1.name )
```

Class & Instance Attributes

```
Class.attr
obj.attr
```

_ _init_ _ Function

Constructor

All classes have a function called _init_(), which is always executed when the object is being initiated.

```
#creating class

class Student:
    def _ _init_ _( self, fullname ):
        self.name =  fullname
```

```
#creating object

s1 =  Student( “karan” )
print( s1.name )
```

*The **self** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Methods

Methods are functions that belong to objects.

```
#creating class

class Student:
    def __init__( self, fullname ):
        self.name =  fullname

    def hello( self ):
        print( "hello", self.name)
```

```
#creating object

s1 =  Student( "karan" )
s1.hello( )
```

Let‘s Practice

Create student class that takes name & marks of 3 subjects as arguments in constructor.
Then create a method to print the average.

Static Methods

Methods that don't use the self parameter (work at class level)

```
class Student:
    @staticmethod    #decorator
    def college( ):
        print( "ABC College" )
```

*Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it

Important

Abstraction

Hiding the implementation details of a class and only showing the essential features to the user.

Encapsulation

Wrapping data and functions into a single unit (object).

Let‘s Practice

Create Account class with 2 attributes - balance & account no.

Create methods for debit, credit & printing the balance.