Content 11

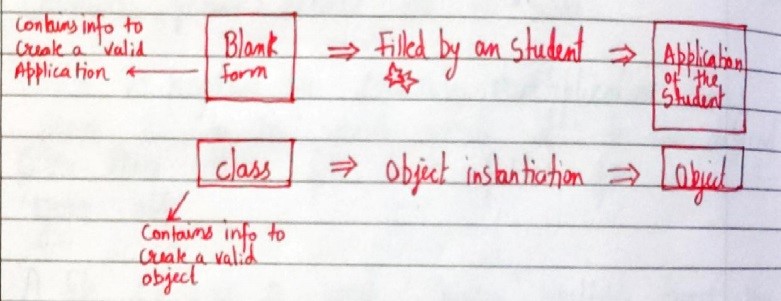
**Object Oriented Programming**

Solving a problem by creating objects is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code. (Implements DRY principle)

##### **Class**

A class is a blueprint for creating objects.



The syntax of a class looks like this:

Class Employee: [classname is written in PascalCase]

#methods & variables

##### **Object**

An object is an instantiation of a class. When class is defined, a template(info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user.     #Abstraction & Encapsulation!

**Modelling a problem in OOPs**

We identify the following in our problem

Noun -> Class -> Employee

Adjective -> Attributes -> name,age,salary

Verbs -> Methods -> getSalary(), increment()

**Class Attributes**

An attribute that belongs to the class rather than a particular object.

Example:

Class Employee:

company = “Google” #Specific to each class

harry = Employee() #Object instantiation

harry.company

Employee.company = “YouTube” #changing class attribute

**Instance Attributes**

An attribute that belongs to the Instance (object)

Assuming the class from the previous example:

harry.name = “Harry”

harry.salary = “30K” #Adding instance attributes

Note: Instance attributes take preference over class attributes during assignment and retrieval.

harry.attribute1  :

1. Is attribute1 present in object?
2. Is attribute1 present in class?

**Program for Class and Also decribes class and instance attributes:**

class Employee:

    name="Aman"     #This is class attributes

    employ\_Id= 10    #This is class attributes

    salary= "10k"     #This is class attributes

e = Employee()

print("1st Employee")

print("Name is: ",e.name)

print("Id is: ",e.employ\_Id)

print("The salary is: ",e.salary)

print("\n\n2nd Employee")

e2 = Employee()

e2.name="Shameer"   #This is instance attribute, we can also declare here also

print(e2.name)

print(e2.salary)

**Output:**

**1st Employee...**

**Name is: Aman**

**Id is: 10**

**The salary is: 10k**

**2nd Employee...**

**Shameer**

**10k**

**‘self’ parameter**

self refers to the instance of the class.

It is automatically passed with a function call from an object.

harry.getSalary()

here, self is harry and above line of code is equivalent to Employee.getSalary(harry)

This function getsalary is defined as:

class Employee:

company = “Google”

def getSalary(self):

print(“Salary is not there”)

**Code for Self parameter:**

class Employee:

    company= "Google"

    def getsalary(self):    #self used for takling both class attribute as well as instance attribute

        print(f"The Company: {self.company} \tand Salary is: {self.salary}")

e = Employee()

print(e.company)

e.salary=5333400

e.getsalary()

#for 2nd Employee

e2 = Employee()

e2.company= "You Tube"

print(e2.company)

e2.salary="54769k"

e2.getsalary()

**Output:**

**Google**

**The Company: Google and Salary is: 5333400**

**You Tube**

**The Company: You Tube and Salary is: 54769k**

**Static method**

Sometimes we need a function that doesn’t use the self-parameter. We can define a static method like this:

@staticmethod #decorator to mark greet as a static method

def greet():

print(“Hello user”)

**Code For Static Methos:**

class Employee:

    company="Google"

    salary=19030

    def getsalary(self, signature):

        print(f"The company is {self.company} and salary {self.salary} \n {signature}")

    @staticmethod

    def greet():

        print("Good Morning Sir...")

e = Employee()

e.getsalary("Thanks!!")     #I added here by declaring signature above

e2 = Employee()

e2.company = "You Tube"

e2.salary = "10k"

e2.getsalary("Have a nice Day!!")   #I added here by declaring signature above

e2.greet()

**Output:**

**The company is Google and salary 19030**

**Thanks!!**

**The company is You Tube and salary 10k**

**Have a nice Day!!**

**Good Morning Sir...**

**\_\_init\_\_() constructor**

\_\_init\_\_() is a special method which runs as soon as the object is created.

\_\_init\_\_() method is also known as constructor.

It takes self-argument and can also take further arguments.

For Example:

class Employee:

def \_\_init\_\_(self,name):

self.name = name

def getSalary(self):

#Some code…

harry = Employee(“Harry”) #Object can be instantiated using constructor like this!

**Simple Code for Constructor:**

class student:

    def \_\_init\_\_(self, name, marks):

        self.name = name

        self.marks = marks

    def getdetails(self):

        print(f"Name is: {self.name} And marks is: {self.marks}")

s1 = student("Arman", 48)

s1.getdetails()

s2 = student("Aamaan", 21)

s2.getdetails()

**Output:**

Name is: Arman And marks is: 48

Name is: Aamaan And marks is: 21

**Code for Constructor:**

class Employee:

    def \_\_init\_\_(self,company,salary,age):

        self.company= company

        self.salary= salary

        self.age= age

    def getdetails(self):

        print(f"The name of Company is: {self.company}")

        print(f"The salary is: {self.salary}")

        print(f"The age is: {self.age}")

    @staticmethod

    def getdata():

        print("In the name of god, who is merciful")

print("Using the constructor....")

e = Employee("Microsoft",53800,29)

e.getdetails()

**Output:**

Using the constructor....

The name of Company is: Microsoft

The salary is: 53800

The age is: 29