Object-Oriented Programming (OOP) in Python

Object-Oriented Programming (OOP) is a programming paradigm that is centered around the concept of objects. In Python, OOP allows you to structure your code using classes and objects, which make it easier to organize, reuse, and scale your code. OOP helps to group data (attributes) and behavior (methods) together, which can be more intuitive when dealing with real-world scenarios like modeling a bank account, a student, or a car.

Class & Object

What is a class?

A class in Python is like a blueprint for creating objects. It defines the attributes (data) and methods (functions) that the objects created from the class will have. In python a class is created by the keyword class.

What is an object?

An object is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances in the following manner

Instance = class(arguments) -->

```
class Student():
    # Class attribute (shared by all instances of the class)
    collage_name = "Bahria Collage"
    name = "Zubair"  # class attribute (this is not used inside
__init__)



    # Constructor method to initialize object attributes
    def __init__(self, name, age, grade):
```

```
        self.name = name  # Object attribute (unique to each instance)
        self.age = age    # Object attribute (unique to each instance)
        self.grade = grade  # Object attribute (unique to each
instance)



# Creating an instance (object) of the Student class with object-
specific attributes
student1 = Student("Laila", 19, "A+")


# Accessing the object attributes (specific to student1) and printing
them
print(f"Name: {student1.name}, Age: {student1.age}, Grade:
{student1.grade}")


# Accessing class attributes (shared by all instances) using the class
name
print(f"Collage Name: {Student.collage_name}, Student Name:
{student1.name}")



Name: Laila, Age: 19, Grade: A+
Collage Name: Bahria Collage, Student Name: Laila
```

if will print student name (laila) in both cases because

Object attribute > class attribute

Constructor in Python:

In Python, the constructor method is named **init**(). It is called automatically when you create a new object from a class. You can pass arguments to the constructor to initialize object-specific values.

Example:

```
class Person:
    def __init__(self, name, age):
        # This is the constructor
        self.name = name  # Initializing object attribute 'name'
```

```
        self.age = age    # Initializing object attribute 'age'


# Creating an instance (object) of the Person class
person1 = Person("Alice", 25)


# Accessing the initialized attributes
print(f"Name: {person1.name}, Age: {person1.age}")
```

```
Name: Alice, Age: 25
```

Adding methods in class

```
class Student():
    collage_name = "Bahria Collage"



    def __init__(self, name, age, grade):
        self.name = name    # object attribute
        self.age = age
        self.grade = grade


# Method 1: A function that welcomes the student
    def welcome(self):
        # This method returns the grade of the student
        print(f"Welcome student {self.name} to {self.collage_name}")


# Method 2: A function to return the student's grade
    def get_grade(self):
        return self.grade



# Creating an instance (object) of the Student class
s1 = Student("Aayan", 19, "A+")

s1.welcome()  # Output: Welcome student Aayan to Bahria Collage

print(s1.get_grade())  # Output: A+



Welcome student Aayan to Bahria Collage
A+
```

Let's Practice

Create a student class that takes name & marks of 3 subjects as arguments in the constructor. Then create a method to print the average. The class should also have a method to check if the student is passing or not. The student is passing if the average of marks is greater than or equal to 40.

```
class Student():
    def __init__(self, name, math: int, physics: int, english: int):
        self.name = name
        self.math = math
        self.physics = physics
        self.english = english

    def average(self):
            return (self.math + self.physics + self.english) / 3


    def pass_fail(self):
        if self.average() >= 70:
            return "Pass"
        else:
            return "Fail"


std1 = Student("Sohail", 60,79,86)

print(f"Student name::{std1.name}, Math::{std1.math},Physics::
{std1.physics},English::{std1.english}")

print(f"\nAverage::{std1.average()},Pass/Fail::{std1.pass_fail()}")
```

```
Student name::Sohail, Math::60,Physics::79,English::86

Average::75.0,Pass/Fail::Pass
```

Static Method

A static method in Python is a method defined within a class that doesn't require access to the instance (object) or class itself. It behaves like a regular function but belongs to the class's namespace. You can call it on the class itself or on instances of the class, and it doesn't take self or cls as its first parameter.

Incorrect use of static method in your case:

```
class Student():
    def __init__(self, name, math: int, physics: int, english: int):
        self.name = name
        self.math = math
        self.physics = physics
        self.english = english

    @staticmethod
    def average():
        # This will raise an error because static methods can't access
self.math, self.physics, etc.
        return (self.math + self.physics + self.english) / 3

    def pass_fail(self):
        if self.average() >= 70:
            return "Pass"
        else:
            return "Fail"


std1 = Student("Sohail", 60, 79, 86)
print(f"Average::{std1.average()}")
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[30], line 21
     17             return "Fail"
     20 std1 = Student("Sohail", 60, 79, 86)
---> 21 print(f"Average::{std1.average()}")

Cell In[30], line 11, in Student.average()
      8 @staticmethod
      9 def average():
```

```
     10     # This will raise an error because static methods can't
access self.math, self.physics, etc.
---> 11     return (self.math + self.physics + self.english) / 3

NameError: name 'self' is not defined
```

Correct use of Static Method

```python
class Student():
    def __init__(self, name, math: int, physics: int, english: int):
        self.name = name
        self.math = math
        self.physics = physics
        self.english = english

    @staticmethod
    def average(math, physics, english):
        return (math + physics + english) / 3

    def pass_fail(self):
        if self.average(self.math, self.physics, self.english) >= 70:
            return "Pass"
        else:
            return "Fail"


std1 = Student("Sohail", 60, 79, 86)
print(f"Student name::{std1.name}, Average::{std1.average(std1.math,
std1.physics, std1.english)}, Pass/Fail::{std1.pass_fail()}")
```

```
Student name::Sohail, Average::75.0, Pass/Fail::Pass
```

```
False
```