OBJECT ORIENTED PROGRAMMING IN PYTHON

Object-Oriented Programming (OOP) is a powerful paradigm that allows developers to structure their code in a more organized and modular way.

 In Python, OOP is implemented through classes and objects, which encapsulate data and behavior into reusable components.

Python is an object-oriented language, and it supports all the features typically associated with OOP, such as encapsulation, inheritance, and polymorphism.

Fundamental concepts of Object-Oriented Programming (OOP).

1. Class
2. Object

Classes:

A class is a blueprint for creating objects.

It defines the attributes (data) and methods (functions) that all objects of that class will have.

Classes provide a way to create user-defined data structures with their own properties and behaviors.

 In Python, a class is defined using the class keyword.

Ex: To create a class named student

**code**

class student:

 Pass

Objects:

An object is an instance of a class.

It is a concrete realization of the class blueprint, with its own unique data and behavior.

Objects are created based on the structure defined by the class. Multiple objects can be created from the same class, each with its own distinct set of attributes and methods.

Multiple objects can be created from the same class, each with its own distinct set of attributes and methods.

 In Python, objects are created by calling the class as if it were a function.

Ex: To create one object of the class named student
Code:

obj=student ( )

Example

==========================================================

class Student:

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

 self.name = name

 self.grades = grades

 def cal\_avg\_grade(self):

 if self.grades:

 return sum(self.grades) / len(self.grades)

 else:

 return "No grades available for this student."

 def is\_passed(self):

 average\_grade = self.cal\_avg\_grade()

 if average\_grade != "No grades available for this student.":

 return average\_grade >= 60

 else:

 return False

# Creating instances of the Student class

student1 = Student("Vamsi", [75, 80, 90, 65, 70])

student2 = Student("Akash", [50, 55, 60, 45])

# Calculating avg grades and checking if students passed or not

print(f"{student1.name}'s average grade: {student1.cal\_avg\_grade()}")

print(f"{student1.name} passed: {student1.is\_passed()}")

print(f"{student2.name}'s average grade: {student2.cal\_avg\_grade()}")

print(f"{student2.name} passed: {student2.is\_passed()}")

====================================================

**Explanation of the above example**

Created a class named student with attributes names and grades.

This class student has two methods
1. cal\_avg\_grade ( ) to calculate the average of the student

2. is\_passed ( ) to check whether the student is passed or not based on the average of the grade. In our class it is 60 %.

In this class we created two objects named student1 and student2 with names vamsi and akash.

In Python, self and \_\_init\_\_() are special features used within classes, primarily to handle object instances and initialization.

**About self in Class**

It is a reference to the current instance of the class. It is used to access the attributes and methods of the object within the class definition.

In Python, all instance methods within a class must have self as their first parameter. When a method is called on an object, Python automatically passes the object itself as the self argument.

Within the class definition, self is used to access the object's attributes and methods. It allows methods to operate on the object's data.

**About \_\_init\_\_**

It is a special method in Python classes that serves as the constructor. It is automatically called when an object is created from the class.

This method initializes the object's attributes with specified values. It allows you to set up the initial state of the object when it is instantiated.

It can accept parameters to customize the initialization process. These parameters are passed when creating an object and used to initialize the object's attributes.

Any initialization logic, such as setting default attribute values or performing setup operations, can be placed within the \_\_init\_\_() method.