**ADITYA AMIN**

**ASSIGN : 02**

Q1. What is the relationship between classes and modules?

A module in Python is a file that contains Python code. It serves as a container for related code, such as classes, functions, and variables. Modules provide a way to organize and package code into separate files, allowing for better code organization and reusability.

On the other hand, a class is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class can have. Classes are used to create instances (objects) that encapsulate data and functionality.

Q2. How do you make instances and classes?

class MyClass:

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

self.attribute = attribute

def method(self):

print("Hello, I'm a method!")

my\_object = MyClass("example")

print(my\_object.attribute) # Output: "example"

my\_object.method() # Output: "Hello, I'm a method!"

Q3. Where and how should be class attributes created?

In Python, class attributes can be created inside the class definition and are shared among all instances of that class. Class attributes are defined directly within the class body but outside any method. They are accessed using the class name or an instance of the class.

Q4. Where and how are instance attributes created?

In Python, instance attributes are created within the methods of a class, typically within the \_\_init\_\_ method, which serves as the constructor for the class. Instance attributes belong to individual instances (objects) of the class and store specific data unique to each instance.

Q5. What does the term "self" in a Python class mean?

In Python, the term "self" is a convention used to refer to the instance of a class within the class methods. It acts as a reference to the current object or instance that the method is being called on. "Self" is not a reserved keyword in Python, but it is widely used and recommended as a naming convention.

Q6. How does a Python class handle operator overloading?

In Python, operator overloading allows you to define how operators behave for objects of a custom class. By implementing special methods, also known as magic methods or dunder methods (double underscore methods), you can specify the behavior of operators such as addition, subtraction, comparison, and more.

Q7. When do you consider allowing operator overloading of your classes?

Allowing operator overloading in your classes can be considered when it enhances the usability and intuitiveness of your code.

Q8. What is the most popular form of operator overloading?

In Python, one of the most popular forms of operator overloading is the overloading of arithmetic operators, such as +, -, \*, /, and %. This is because arithmetic operations are fundamental and widely used in various domains.

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

To comprehend Python Object-Oriented Programming (OOP) code effectively, there are two key concepts that are crucial to understand:

Classes and Objects: Classes are the blueprints or templates that define the structure and behavior of objects. They encapsulate data (attributes) and functions (methods) that operate on that data. Objects, on the other hand, are instances of classes that are created based on those blueprints. They hold specific values for the attributes defined in the class and can perform actions using the methods defined in the class.

Understanding how classes and objects work and how they relate to each other is fundamental to grasping OOP in Python. This includes understanding concepts such as constructors (\_\_init\_\_ method), instance attributes, class attributes, and methods.

Inheritance and Polymorphism: Inheritance allows you to create new classes (derived or child classes) based on existing classes (base or parent classes), inheriting their attributes and methods. This concept promotes code reuse and supports the creation of specialized classes that inherit and extend the functionality of their parent classes.