**ADITYA AMIN**

**ASSIGN : 19**

Q1. Define the relationship between a class and its instances. Is it a one-to-one or a one-to-many partnership, for example?

The relationship between a class and its instances in object-oriented programming is typically a one-to-many relationship. A class can be seen as a blueprint or template that defines the common properties and behaviors of a group of objects. An instance, also known as an object, is a specific occurrence or realization of that class.

In this one-to-many relationship, the class serves as a general representation or abstraction, while the instances represent specific occurrences or individual entities based on that class. Each instance can have its own distinct state, and changes made to one instance generally do not affect other instances of the same class.

Q2. What kind of data is held only in an instance?

In object-oriented programming, an instance holds data that is specific to that particular instance. This data is often referred to as instance variables or member variables. Instance variables store the state or characteristics of an object, and their values can vary between different instances of the same class.

Q3. What kind of knowledge is stored in a class?

a class acts as a blueprint or template that combines both data and behavior, defining the knowledge and capabilities of objects instantiated from it. It provides a way to organize and encapsulate related information and functionality, promoting modularity, reusability, and maintainability in software development.

Q4. What exactly is a method, and how is it different from a regular function?

In the context of programming, a method is a function that is associated with an object or a class. It represents the behavior or actions that objects of a particular class can perform. Methods are an essential part of object-oriented programming and are defined within the class definition

On the other hand, a regular function is not associated with an object or a class. It is a self-contained block of code that performs a specific task or computation. Functions can be defined and called independently, and they typically operate on their own input parameters. Functions may or may not have access to any object state unless explicitly passed as arguments.

Q5. Is inheritance supported in Python, and if so, what is the syntax?

Yes, inheritance is supported in Python. Inheritance allows a class to inherit attributes and methods from another class, called the parent class or superclass. The class that inherits from the parent class is called the child class or subclass. In Python, the syntax for defining and using inheritance is as follows:

class ParentClass:

# Parent class definition

class ChildClass(ParentClass):

# Child class definition

Q6. How much encapsulation (making instance or class variables private) does Python support?

In Python, encapsulation is supported to a certain extent through naming conventions and access modifiers, although the language does not enforce strict visibility control like some other programming languages. Python follows a principle called "we are all consenting adults here," which encourages developers to use naming conventions to indicate the intended visibility and access level of variables and methods.

Q7. How do you distinguish between a class variable and an instance variable?

Class Variable:

Scope: A class variable is defined within the class but outside any methods. It is shared among all instances of the class.

Declaration: A class variable is typically declared directly beneath the class declaration, at the class level.

Instance Variable:

Scope: An instance variable is defined within a class's methods or the class's constructor (\_\_init\_\_() method). It is specific to each instance of the class.

Declaration: Instance variables are usually declared within the \_\_init\_\_() method or any other method in the class.

Q8. When, if ever, can self be included in a class's method definitions?

In Python, the self parameter is typically included as the first parameter in a class's method definitions. It is used to refer to the instance of the class on which the method is called. However, it's important to note that self is just a convention and can be replaced with any other valid parameter name, although it's highly recommended to stick with the convention of using self.

Q9. What is the difference between the \_ \_add\_ \_ and the \_ \_radd\_ \_ methods?

The distinction between \_\_add\_\_ and \_\_radd\_\_ becomes important when performing addition operations between objects of different types. If both objects have defined their respective \_\_add\_\_ methods, those methods are used. However, if the right-hand side object does not have an \_\_add\_\_ method, or it returns NotImplemented, Python tries to use the \_\_radd\_\_ method of the left-hand side object.

Q10. When is it necessary to use a reflection method? When do you not need it, even though you support the operation in question?

Reflection methods, such as \_\_getattr\_\_, \_\_setattr\_\_, \_\_getattribute\_\_, and \_\_setattr\_\_, provide a way to intercept attribute access and modification operations in Python classes. They allow you to customize the behavior of attribute access and assignment. However, their usage is not always necessary or recommended.

Q11. What is the \_ \_iadd\_ \_ method called?

The \_\_iadd\_\_ method in Python is called the "in-place addition" method. It is a special method that allows objects to support the += operator for in-place addition operations. The \_\_iadd\_\_ method modifies the object itself rather than creating a new object.

Q12. Is the \_ \_init\_ \_ method inherited by subclasses? What do you do if you need to customize its behavior within a subclass?

Yes, the \_\_init\_\_ method is inherited by subclasses in Python. When a subclass is created, it inherits all the methods, including the \_\_init\_\_ method, from its parent class.

If you need to customize the behavior of the \_\_init\_\_ method within a subclass, you can override the method by defining a new \_\_init\_\_ method in the subclass. This allows you to provide a specialized implementation specific to the subclass while still retaining the inheritance of other behaviors from the parent class.