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

ANSWER.

The relationship between a class and its instances in object-oriented programming can be described as a one-to-many relationship, where one class represents a blueprint or template for creating multiple instances or objects.

In object-oriented programming, a class serves as a blueprint or template that defines the properties (attributes) and behaviors (methods) common to all instances of that class. Instances, also known as objects, are individual entities created based on the class definition. Each instance encapsulates its own state (attribute values) and behavior (methods) but shares the structure and behavior defined by the class.

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

ANSWER.

Data held only in an instance includes instance-specific attributes or instance variables, which define the state of each individual object and are unique to that instance.

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

ANSWER.

While both methods and functions are blocks of code that perform tasks, methods are associated with objects and defined within classes, while functions are standalone and can be called independently. Methods encapsulate object-specific behavior and have access to the object's attributes, while functions are more general-purpose and operate independently of objects.

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

ANSWER.

Yes, inheritance is supported in Python, allowing classes to inherit attributes and methods from other classes. The syntax for inheritance in Python is as follows:

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

ANSWER.

While Python supports varying degrees of encapsulation through naming conventions and name mangling, it relies heavily on developer discipline and convention rather than language-enforced access control. Developers are encouraged to follow conventions to indicate the intended visibility of attributes and methods and to use name mangling cautiously for truly private members.

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

ANSWER.

In Python, class variables and instance variables are both types of attributes associated with a class, but they are used and accessed in different ways and serve different purposes.

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

ANSWER.

`self` is included in a class's method definitions when defining instance methods and the initializer method (`\_\_init\_\_`). It allows methods to operate on instance-specific data and interact with other methods and attributes associated with the instance.

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

ANSWER.

The `\_\_add\_\_` and `\_\_radd\_\_` methods in Python are special methods used to define the behavior of addition operations for objects of a class. The main difference between them lies in their invocation and the order of operands during addition.

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

ANSWER.

Reflection methods are necessary when you need to customize attribute access behavior dynamically or perform additional actions during attribute access, assignment, or deletion. However, they are not always needed if attribute behavior is static or can be managed explicitly without reflection methods.

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

ANSWER.

The `\_\_iadd\_\_` method in Python is called the "in-place addition" method. It is a special method used to define the behavior of the `+=` operator when used with objects of a class.

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

ANSWER.

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