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 is a one-to-many partnership. A class defines a blueprint or template for creating objects, and instances are individual objects created from that class.

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

Instance-specific data, also known as instance variables or attributes, are held only in an instance. These data attributes vary from instance to instance and store specific values unique to each object.

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

A class stores knowledge in the form of class variables (shared among all instances) and methods (functions defined within the class). Class variables represent shared data, while methods define behaviors associated with the class.

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

A method is a function defined within a class and operates on its instances. It takes the instance itself (usually named self) as its first argument and can access and modify instance variables. A regular function is not bound to a specific class and operates independently.

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

class Subclass(BaseClass):

# Subclass definition

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

Python supports encapsulation to some extent. Instance variables are not truly private but are conventionally indicated as private using a single leading underscore (e.g., \_variable). Strong encapsulation (making variables truly private) is not enforced, relying on naming conventions.

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

A class variable is shared among all instances of a class and is defined within the class body. An instance variable is specific to each instance and is assigned within methods using the self keyword.

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

self is always included as the first parameter in a class's method definitions to reference the instance itself. It is required for methods to access instance attributes and perform operations on the instance.

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

The \_\_add\_\_ and \_\_radd\_\_ methods are used for addition operator overloading. \_\_add\_\_ is called when the left operand is the instance, while \_\_radd\_\_ is called when the left operand is not the instance and the right operand supports 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?

A reflection method is used when you need to implement reverse operations for operators (e.g., \_\_radd\_\_). You do not need it when the operation is commutative, and the order of operands does not matter.

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

The \_\_iadd\_\_ method is called for the in-place addition operator (+=). It is used to customize the behavior of the in-place addition operation for objects.

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

The \_\_init\_\_ method is inherited by subclasses, but if you need to customize its behavior within a subclass, you can override it in the subclass by redefining it. This allows you to add or modify initialization steps specific to the subclass.