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

Ans1- The relationship between a class and its instances is a one-to-many partnership. A class is a blueprint or template for creating objects (instances), so you can create multiple instances of the same class. Each instance has its own set of attributes and can perform actions defined by the class.

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

Ans2- Instances hold data specific to themselves. These are instance variables that store unique values for each object. Instance variables represent the object's state and are different for each instance.

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

Ans3- A class stores knowledge in the form of attributes (variables) and methods (functions). Attributes define the properties or characteristics of objects, while methods define the behaviors or actions that objects can perform.

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

Ans4- A method is a function defined within a class. It operates on the data and attributes of instances of that class. The key difference between a method and a regular function is that a method is bound to a specific class and can access and modify the class's attributes.

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

Ans5- yes inheritance is supported in Python

class ChildClass(ParentClass):

# ChildClass inherits attributes and methods from ParentClass

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

Ans6- Python supports encapsulation to a certain extent through name mangling and conventions. You can make instance variables private by prefixing them with a double underscore (e.g., \_\_private\_var).

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

Ans7- A class variable is shared among all instances of a class and is defined within the class but outside any instance methods. An instance variable, on the other hand, is specific to each instance and is defined within the constructor method (\_\_init\_\_) using the self keyword.

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

Ans8- self is included in a class's method definitions as the first parameter to refer to the instance itself. It is required in all instance methods to access instance-specific attributes and methods.

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

Ans9- The \_\_add\_\_ method is called when an object is on the left side of the + operator, while \_\_radd\_\_ is called when an object is on the right side of the + operator. These methods allow objects to define their behavior when participating in addition operations.

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

Ans10- Reflection methods are necessary when you want to customize attribute access or manipulation behavior for instances. You don't need them when you have straightforward attribute access or manipulation requirements that Python's default behavior handles correctly

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

Ans11- The \_\_iadd\_\_ method is called in Python when an object participates in the += operation. It allows objects to define their behavior when augmented assignment (in-place addition) is used.

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

Ans12- The \_\_init\_\_ method is not automatically inherited by subclasses, but you can call the superclass's \_\_init\_\_ method explicitly within the subclass using super().\_\_init\_\_(). If you need to customize its behavior in a subclass, you can override it in the subclass by providing your own implementation while still calling the superclass's \_\_init\_\_ method to ensure proper initialization.