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

Ans: - A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods). The user-defined objects are created using the class and are called instances of the class. So, the relationship between a class and its instances is a one-to-many relationship.

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

Ans: - An instance variable is a variable whose value is instance-specific and is not shared among instances. These variables can have different values for different objects (instances) of the class.

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

Ans: -A class stores attributes (data members) and methods (functions). It encapsulates all the data including member functions, variables, etc.

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

Ans: - A function is a piece of code that is called by name. It can be passed data to operate on (i.e., the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed. A method, on the other hand, is a piece of code that is called by a name that is associated with an object. In most respects, it is identical to a function except for two key differences: a method is implicitly passed the object on which it was called, and a method can operate on data that is contained within the class.

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

Ans: - Yes, inheritance is supported in Python. The syntax for inheritance in Python is as follows: -

```
class BaseClass:
    # Body of base class

class DerivedClass(BaseClass):
    # Body of derived class
```

Here, DerivedClass is the class that will inherit from the BaseClass.

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

Ans: - Python supports a degree of encapsulation through the use of protected (single underscore, _) and private (double underscore, __) members. However, it's more of a convention than a strict enforcement. Python does not have strong encapsulation (at least, not in the sense that languages like Java do).

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

Ans: - Class variables are variables that are shared by all instances of a class. They're defined within a class but outside any of the class's methods. Instance variables, on the other hand, are variables whose value is instance-specific and not shared among instances.

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

Ans: - In Python, self is included in the method definitions of a class to refer to the instance of the class. It's the first parameter of every class method, including __init__. By convention, this argument is always named self.

Q9. What is the difference between the _ _add_ _ and the _ _radd_ _ methods?

Ans: - The __add__ method in Python is used to override the "+" operator for the class instances. It defines the behavior for the addition operator. On the other hand, __radd__ is a special method that is used for implementing the functionality of "+" when the left operand doesn't have an __add__ method, or that method does not support the addition with the right operand.

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

Ans: - Reflection methods are used when you want to manipulate or access the features of a class dynamically at runtime. They're necessary when you don't know everything about a class at compile time. However, you don't need reflection if you know about the class at compile time and you don't need to change the behavior at runtime.

Q11. What is the _ _iadd_ _ method called?

Ans: - The __iadd__ method in Python is a special method that is used to override the "+=" operator for the class instances. It's called when the "+=" operator is used on the object 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?

Ans: - The __init__ method is not inherited by subclasses in Python. If you need to customize its behavior within a subclass, you would need to define a new __init__ method in the subclass. If you still want to keep the initialization of the parent class, you can call the __init__ method of the parent class in the __init__ method of the subclass.