**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 is a one-to-many partnership. That is, a single class can have many instances created from it, each with its own unique set of values for the properties defined in the class.

For example, if we have a class called "Person," we can create multiple instances of this class, each representing a different person. Each instance will have its own set of properties such as name, age, and address, which are defined in the class. The class acts as a template for creating these instances, but each instance is distinct and can be modified independently of the others.

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

**ANSWER**: In object-oriented programming, instances hold data that is specific to that particular instance, which is not shared with other instances of the same class. This data is often referred to as instance variables or instance fields. instance variables are data that is unique to an instance of a class, and is not shared with other instances of the same class. They are defined within the class, but their values are assigned and accessed through the instances.

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

**ANSWER:** In object-oriented programming, a class serves as a blueprint for creating objects or instances that share common properties and behaviours. The class contains the definition of these properties and behaviours, which are inherited by its instances. class stores knowledge about the properties and behaviours of objects that are created from it, including data, methods, inheritance, and access modifiers. This knowledge serves as a template for creating instances of the class with the same characteristics and behaviours.

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

**ANSWER**: A method is a function that is associated with an object or a class. It is a piece of code that can be called to perform a specific task or operation on an object, or to manipulate the data or behaviour of an object.

The key difference between a method and a regular function is that a method is associated with an object or a class, while a function is not. A method is called on an instance of a class and operates on that specific instance's data. A regular function, on the other hand, operates independently of any specific object or class.

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

**ANSWER:** Yes, inheritance is supported in Python. It is a key feature of object-oriented programming in Python and allows for code reuse and polymorphism.

The syntax for creating a subclass in Python and inheriting from a superclass is as follows:

class SuperClass:

# code for superclass goes here

class SubClass(SuperClass):

# code for subclass goes here

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

**ANSWER**: In Python, encapsulation can be achieved using naming conventions to indicate that certain methods or variables are intended to be private, but there is no strict enforcement of access control like other programming languages such as Java. By convention, variables and methods that are intended to be private are prefixed with a single underscore (\_).

However, it is important to note that this is just a convention, and these variables and methods can still be accessed from outside the class. In Python, there is no strict enforcement of access control, and any variable or method can be accessed from outside the class if the caller knows the variable or method name.

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

**ANSWER**: In Python, a class variable is a variable that is shared by all instances of a class, while an instance variable is a variable that is specific to each instance of a class.

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

**ANSWER:** In Python, the self parameter is used to refer to the current instance of the class. It is a convention to include self as the first parameter of a class method definition in order to reference the instance variables and methods of that particular instance. Therefore, self is typically included in almost all of a class's method definitions.

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

**ANSWER**: In Python, the \_\_add\_\_ and \_\_radd\_\_ methods are used to define the behaviour of the + operator when applied to objects of a class. The \_\_add\_\_ method is called when the + operator is applied to an instance of the class and another object, while the \_\_radd\_\_ method is called when the + operator is applied to an object of a different type and an instance of the class.

**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 can be useful in cases where you want to provide more dynamic behavior for your objects or when you want to customize the way that attribute access works. However, they are not always necessary and should be used judiciously to avoid unnecessarily complicating your code.

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

**ANSWER:** The \_\_iadd\_\_ method is called when the += operator is applied to an instance of a class. It is used to define in-place addition behaviour for mutable objects.

**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 automatically inherits all the methods and attributes of its parent class, including the \_\_init\_\_ method.

If you need to customize the behavior of the \_\_init\_\_ method in a subclass, you can override it by defining a new \_\_init\_\_ method in the subclass. The new method should have the same name as the parent class's \_\_init\_\_ method, and can take additional arguments or implement new functionality as needed**.**