**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- In object-oriented programming, a class is a blueprint for creating objects (also known as instances) that share common attributes and behaviours. Each instance is an individual object created from the class, with its own unique set of attribute values.The relationship between a class and its instances is a one-to-many partnership. One class can be used to create multiple instances, each with its own independent state. Each instance is created using the same class, but can have different attribute values and methods.

For example, consider a ‘Car’ class that has attributes such as make, model, and year. Each instance of the ‘Car’ class represents a unique car with its own set of attribute values. We can create multiple instances of the ‘Car’ class, each representing a different car, but all using the same blueprint or class definition.

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

Ans- In object-oriented programming, the data held only in an instance (also known as object data) is specific to that instance and not shared by other instances of the same class. This data is stored as instance variables (also known as attributes or properties) and is defined within the class constructor or other methods of the class. Instance data can include any type of data that is specific to that instance.

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

Ans- In object-oriented programming, a class is a blueprint for creating objects (also known as instances) that share common attributes and behaviours. The knowledge stored in a class is the definition of these attributes and behaviours.

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

Ans- In Python, a method is a function that is defined inside a class and is intended to be called on an instance of that class. Methods are a way to define the behaviour of an object and allow objects to interact with each other.

The main difference between a method and a regular function is that a method is called on an instance of a class, while a regular function is called independently of any instance. When a method is called on an instance, the instance itself is passed as the first argument to the method . This allows the method to access the attributes and methods of the instance.

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

Ans- Yes, inheritance is supported in Python. Inheritance is a way to create a new class that is a modified version of an existing class, by reusing the attributes and behaviours of the existing class.

In Python, you can create a new class that inherits from an existing class by specifying the existing class as the base class in the new class definition. The syntax for creating a subclass in Python is as follows:

class Subclass(BaseClass):

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

Ans- In Python, encapsulation is not enforced by the language itself, unlike in some other object-oriented programming languages such as Java. However, Python does provide some support for encapsulation through the use of naming conventions and certain language features.

In Python, variables and methods can be made private by prefixing their names with a double underscore (‘\_\_’). This causes the interpreter to "mangle" the name, effectively making it harder to access the variable or method from outside the class. However, it is still possible to access these members by using their mangled names, which are formed by adding a single underscore and the class name before the original name.

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

Ans- In Python, a class variable is a variable that is defined in the class definition and is shared by all instances of that class. An instance variable, on the other hand, is a variable that is defined in the instance and is unique to that instance.

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

Ans-n Python, the ‘self’ parameter is typically included as the first parameter in a class's method definitions. The ‘self’ parameter is used to refer to the instance of the class on which the method is being called. Generally, the ‘self’ parameter is included in a class's method definitions whenever the method needs to access or modify the instance's attributes.

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

Ans­- In Python, the \_\_add\_\_ and \_\_radd\_\_ methods are used to define how objects of a class should behave when they are added to other objects.

The \_\_add\_\_ method is called when the + operator is used to add an object of the class to another object. The \_\_radd\_\_ method, on the other hand, is called when the object of the class is on the right-hand side of the + operator, and the object on the left-hand side does not have an \_\_add\_\_ method that can handle the operation.

**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 in Python (also known as magic methods or dunder methods) are special methods that are used to implement operator overloading and other language features. Reflection methods allow you to define how instances of your class should behave in response to certain operations, such as addition, subtraction, and comparison.

It is necessary to use a reflection method whenever you want to support an operation on instances of your class. For example, if you want instances of your class to be able to be added together using the + operator, you would need to define the \_\_add\_\_ method.

You should define reflection methods whenever you want to customize the behaviour of your class in response to a specific operation. However, you don't need to define a reflection method if the default behaviour is sufficient for your class, or if the operation can be implemented using other methods.

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

Ans-The \_\_iadd\_\_ method is called the "in-place addition" method in Python. It is a special method that is used to define the behaviour of the ‘+=’ operator when applied to objects of a class.

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

Ans-Yes, the \_\_init\_\_ method is inherited by subclasses in Python. When you create a subclass, it inherits all the methods (including \_\_init\_\_) of its parent class.

If you need to customize the behaviour of the \_\_init\_\_ method in a subclass, you can override the method by defining a new \_\_init\_\_ method in the subclass. Your new method will replace the inherited method in the subclass, but instances of the subclass will still inherit the original \_\_init\_\_ method from the parent class.