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 relationship. A class serves as a blueprint or template for creating multiple instances or objects. Each instance represents a distinct occurrence or realization of the class

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

In object-oriented programming, instances (objects) hold instance data, which refers to the specific data unique to each individual instance of a class. Instance data is distinct for each object and is not shared among different instances of the same class

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

In object-oriented programming, a class is a blueprint or template that defines the structure and behavior of objects. It encapsulates knowledge in the form of class-level data and class-level methods.

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

In object-oriented programming, a method is a function that is defined within a class and is associated with instances of that class. Methods represent the behavior or actions that objects of a class can perform. Here are the key characteristics and differences between methods and regular functions.

Defined within a Class

Access to Instance Data

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

Yes, inheritance is supported in Python. Inheritance allows a class to inherit attributes and methods from a parent or base class, enabling code reuse and hierarchical organization of classes.

class ChildClass(ParentClass):

# Additional attributes and methods specific to the child class

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

In Python, encapsulation, which involves making instance or class variables private, is supported to a certain extent. Python follows a convention of using naming conventions to indicate the intended visibility of variables and methods, but it does not enforce strict access control like some other programming languages do

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

In Python, class variables and instance variables are distinguished by their scope and where they are defined

Class Variable: A class variable is defined within the class definition but outside any instance methods. It is shared among all instances of the class. Class variables are typically declared at the class level and have the same value for all instances.

Instance Variable: An instance variable is defined within an instance method or the \_\_init\_\_ method of the class. It is specific to each instance of the class. Each instance maintains its own copy of the instance variable with its own assigned value

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

In Python, the self parameter is typically included in a class's method definitions to refer to the instance of the class itself. It is a convention to name the first parameter of instance methods as self, although you can choose any valid name for it.

The self parameter allows you to access instance variables, other instance methods, and perform operations specific to the instance within the method. It serves as a reference to the current instance on which the method is called

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

The main difference between \_\_add\_\_ and \_\_radd\_\_ is the order in which the operands are processed. The \_\_add\_\_ method is called on the left operand, while the \_\_radd\_\_ method is called on the right operand when the left operand does not support the addition operation.

It's important to note that if both \_\_add\_\_ and \_\_radd\_\_ methods are defined in a class, \_\_add\_\_ takes precedence over \_\_radd\_\_. However, if the left operand does not support addition with the right operand, and the right operand does not implement \_\_radd\_\_, a TypeError will be raised

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

In Python, reflection methods, also known as "magic" or "dunder" methods, are special methods that provide introspection and control over how objects behave in various operations. These methods are not always necessary, but they can be used to customize or enhance the behavior of objects in certain situations

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

The \_\_iadd\_\_ method in Python is called when the in-place addition operator (+=) is used to add an object to another object. It is a reflection method that allows you to define the behavior of the in-place addition operation for objects 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?

The \_\_init\_\_ method is not automatically inherited by subclasses in Python. Each class, including subclasses, needs to define its own \_\_init\_\_ method if customization is required. the \_\_init\_\_ method is not inherited by subclasses automatically. If you need to customize its behavior within a subclass while preserving the initialization behavior of the parent class, you can use super() to invoke the \_\_init\_\_ method of the parent class and then extend it with additional initialization steps specific to the subclass