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 can be described as a one-to-many relationship. A class defines a blueprint or template for creating objects, while instances are specific objects created from that class.

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

Answer : In object-oriented programming, instance data refers to the data that is specific to each individual instance of a class. It is unique to each instance and separate from the class-level data or static data.

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

Answer : In object-oriented programming, a class represents a blueprint or template for creating objects. It encapsulates both data and behavior, providing a way to define the structure, attributes, and methods that objects of that class will possess.

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

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

Answer : Yes, inheritance is supported in Python. It is a fundamental concept in object-oriented programming that allows classes to inherit attributes and methods from other classes. In Python, the syntax for inheritance is as follows:

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def eat(self):

print(f"{self.name} is eating.")

def sleep(self):

print(f"{self.name} is sleeping.")

class Dog(Animal):

def bark(self):

print("Woof!")

# Creating instances of the classes

animal = Animal("Generic Animal")

animal.eat() # Output: Generic Animal is eating.

animal.sleep() # Output: Generic Animal is sleeping.

dog = Dog("Bobby")

dog.eat() # Output: Bobby is eating.

dog.sleep() # Output: Bobby is sleeping.

dog.bark() # Output: Woof!

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

Answer : In Python, encapsulation is supported to some extent through the use of naming conventions and access modifiers, although it is not enforced as strictly as in some other programming languages.

Python follows a naming convention to indicate the visibility of variables and methods:

Public: By default, all instance variables and methods are considered public and can be accessed from outside the class. They are typically named in lowercase or lowercase with underscores (snake\_case) to indicate their intended use.

Protected: Variables and methods intended for internal use within a class or its subclasses are prefixed with a single underscore \_. While they can still be accessed from outside the class, it is a convention to treat them as protected and discourage direct access. However, this naming convention does not enforce actual access restrictions.

Private: Variables and methods intended to be strictly private to a class are prefixed with a double underscore \_\_. This triggers name mangling, where the variable or method name is modified to include the class name as a prefix, making it harder to access from outside the class. However, it is still possible to access them using the mangled name, so it is more of a naming convention than a strict access control mechanism.

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

Answer : In Python, class variables and instance variables are two types of variables with distinct characteristics and usage within a class. The key differences between them are as follows:

Scope:

Class variables: They are defined within the class but outside of any methods. They have class-level scope, meaning they are shared among all instances of the class as well as the class itself.

Instance variables: They are defined within the methods or the \_\_init\_\_ method of a class and are specific to each instance of the class. They have instance-level scope and are unique to each object created from the class.

Memory Allocation:

Class variables: They are allocated memory only once when the class is defined. All instances of the class share the same memory space for class variables.

Instance variables: Each instance of a class has its own memory allocation for instance variables. When an object is created, memory is allocated separately for its instance variables.

Access and Modification:

Class variables: They can be accessed and modified using the class name itself or any instance of the class. Changes made to class variables are reflected across all instances of the class.

Instance variables: They are accessed and modified using the instance name. Each instance has its own set of instance variables that are independent of other instances.

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

Answer :

In Python, the self parameter is included in a class's method definitions to refer to the instance of the class itself. It is a convention to use the name self, although any valid variable name can be used.

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

Answer : The \_\_add\_\_ and \_\_radd\_\_ methods in Python are used to define the behavior of the addition operator (+) when applied to objects of a class. The main difference between them lies in the order of operand evaluation.

\_\_add\_\_ method:

The \_\_add\_\_ method is invoked when the + operator is used and the left operand is an instance of the class.

It defines the behavior for adding the object of the class with another object.

The \_\_add\_\_ method is called on the left operand, and the right operand is passed as an argument.

If the \_\_add\_\_ method is not defined or unsupported for the class, a TypeError is raised.

\_\_radd\_\_ method:

The \_\_radd\_\_ method is invoked when the + operator is used and the left operand does not support addition with the right operand.

It is called on the right operand when the left operand is an object that does not have the \_\_add\_\_ method or does not support addition with the right operand.

The \_\_radd\_\_ method allows the right operand to define its behavior when added to an object of the class.

If the \_\_radd\_\_ method is not defined or unsupported for the class, the addition operation results in a TypeError.

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 : Yes, the \_\_init\_\_ method in Python is inherited by subclasses. When a subclass is created, it inherits the \_\_init\_\_ method from its superclass (or parent class). This means that the subclass will have its own constructor that can be used to initialize its specific attributes.

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

Answer : The \_\_iadd\_\_ method in Python is called the "in-place addition" method. It is a special method that corresponds to the += operator for modifying an object in place. It allows an object to define custom behavior when the += operator is used on instances of that object.

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