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***:

the relationship between a class and its instances is a one-to-many relationship, where one class can have many instances created from it, each with its own unique set of instance variables.

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

***Ans***:

An instance refers to a single occurrence of a class. Instance data, also known as instance variables or attributes, are data that are specific to a particular instance of a class.

Instance data is typically initialized when an instance is created and can be accessed and modified through methods defined in the class. Instance data is also known as non-static data or member variables, and is distinguished from static data or class variables, which are shared among all instances of a class.

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

***Ans***:

The knowledge stored in a class can be divided into two categories: static and dynamic.

1. Static Knowledge: This refers to the properties of the class itself and is stored as class-level variables or static data members. Examples of static knowledge include the class name, the number of instances created, and any constants or default values associated with the class.

2. Dynamic Knowledge: This refers to the behaviour or functionality of the class and is stored as instance-level variables or instance data members. Examples of dynamic knowledge include the state of an instance, such as its current values for its attributes or properties, and the methods or functions that define its behaviour or actions.

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

***Ans***:

a method is a function that is associated with an object or a class. It defines the behaviour of an object or a class and operates on the data members or attributes of the object or class.

The main difference between a method and a regular function is that a method is defined within the context of a class, and it has access to the data members or attributes of that class. A regular function is independent of any class or object and operates on its input parameters only.

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

***Ans***:

Yes, inheritance is supported in Python,

Here is an example:

class Animal:

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

self.name = name

def speak(self):

print(f"{self.name} says: Hi there!")

class Dog(Animal):

def speak(self):

print(f"{self.name} barks: Woof woof!")

dog = Dog("Buddy")

dog.speak()

# Output: Buddy barks: Woof woof!

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

***Ans***:

Python supports a moderate level of encapsulation, but it does not have built-in support for making instance or class variables private in the same way as some other object-oriented programming languages.

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

***Ans***:

In Python, a class variable is a variable that is shared by all instances of a class, whereas an instance variable is a variable that is unique to each instance of the class.

A class variable is defined inside the class, but outside of any methods, and is usually initialized at the class level. It is accessed using the class name, not the instance name, and can be modified by any instance of the class or the class itself.

An instance variable, on the other hand, is defined inside the \_\_init\_\_ method of the class and is unique to each instance of the class. It is accessed using the instance name, and its value can be different for each instance.

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

***Ans***:

In Python, self is a conventionally used parameter name that is included in the definition of every instance method of a class. It is used to refer to the instance of the class that the method is being called on.

When you call an instance method of a class, you do not need to explicitly pass the instance as an argument, Python automatically passes the instance as the first argument to the method.

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

***Ans***:

the main difference between ‘\_\_add\_\_‘ and ‘\_\_radd\_\_‘ is that ‘\_\_add\_\_’ is called when the left-hand operand is an instance of the class, while ‘\_\_radd\_\_’ is called when the left-hand operand is not 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?

***Ans***:

In Python, reflection methods are used to provide information about a class and its instances at runtime. They are typically used to support introspection, which is the ability of a program to examine and manipulate its own code and data structures.

you should use reflection methods when you need to customize the behaviours of a class or its instances in a particular way and use the default behaviours provided by Python for simple operations that do not require customization.

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

***Ans***:

The \_\_iadd\_\_ method in Python is called the "in-place addition" method and is used to define the behaviour of the in-place addition operator (+=) for instances 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:

Yes, the \_\_init\_\_ method in Python is inherited by subclasses from their parent class, unless the subclass defines its own \_\_init\_\_ method.

If you need to customize the behaviour of \_\_init\_\_ within a subclass, you can simply define a new \_\_init\_\_ method in the subclass and provide the desired behaviour.