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:** Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. Some points on Python class: Classes are created by keyword class.

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

**Ans:** Python has a built-in function called instance () that compares the value with the type given. It the value and type given matches it will return true otherwise false. Using isinstance(), you can test for string, float, int, list, tuple, dict, set, class, etc.

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

**Ans:** Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. Some points on Python class: Classes are created by keyword class.

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

**Ans:** A method is a function that “belongs to” an object. (In Python, the term method is not unique to class instances: other object types can have methods as well. For example, list objects have methods called append, insert, remove, sort, and so on. A method, like a function, is a set of instructions that perform a task. The difference is that a method is associated with an object, while a function is not.

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

**Ans:** In Python, there are two types of Inheritance: Multiple Inheritance. Multilevel Inheritance. Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class.

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

**Ans:** Python does not have the private keyword, unlike some other object-oriented languages, but encapsulation can be done. Instead, it relies on the convention: a class variable that should not directly be accessed should be prefixed with an underscore.

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

**Ans:** Python instance variables can have different values across multiple instances of a class. Class variables share the same value among all instances of the class. The value of instance variables can differ across each instance of a class. Class variables can only be assigned when a class has been defined. A class describes a data type. An instance of a class is an object of the data type that exists in memory. ... Unlike structures, the members of a class are private by default.

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

**Ans:** Self is a convention and not a Python keyword. Self is parameter in Instance Method and user can use another parameter name in place of it. But it is advisable to use self because it increases the readability of code, and it is also a good programming practice. Self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes.

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

**Ans:** Modifying the \_\_add\_\_ method of a Python Class which means that we can control the result of a sum of two objects by modifying or defying the \_\_add\_\_ method. We can define the \_\_add\_\_ method to return a Day instance with the total number of visits and contacts: class Day(object). Loading when this answer was accepted… \_\_radd\_\_ is only called if the left object does not have an \_\_add\_\_ method, or that method does not know how to add the two objects (which it flags by returning Not Implemented).

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:** Using reflection, we can write one recursive reverse function that will work for strings, lists, and any other sequence that supports slicing and concatenation. If an obj is a reference to a string, then Python will return the str type object.

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

**Ans:** Magic methods in Python are the special methods that start and end with the double underscores. ... For example, the \_\_add\_\_ method is a magic method which gets called when we add two numbers using the + operator. iadd() :- This function is used to assign and add the current value. This operation does “a+=b” operation. Assigning is not performed in case of immutable containers, such as strings, numbers, and tuples.

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

**Ans:** A subclass inherits everything from its superclass, which is referred to as inheritance in the object-orientation methodology and object-oriented programming. By inheritance, the superclass's attributes will not repeat in any of its subclasses.