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: Relationship between a class and its instances is a one to many partnership.

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

Ans: An instance object contains instance variables specific to that particular instance object.

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

Ans: class creates a custom data structure with its own data members and member functions that can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

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

Ans: You can use the methods of the class to access the instance variables of that instance. Therefore, an object's state can be changed by its methods. This function cannot access attributes of instances of the class or change the state of the object.

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

Ans: Yes,Python supports inheritance. The Types of Inheritence Supported by Python are:

* Simple Inheritence
* Multiple Inheritence
* Multilevel lInheritence
* Hybrid Inheritence
* Hierracial Inheritence

class Car:

def \_\_init\_\_(self, company, model):

self.first\_name = company

self.last\_name = model

class Student(Car):

pass

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

Ans: Encapsulation describes the idea of ​​packaging data and how it can be manipulated within units. This restricts direct access to variables and methods and prevents accidental modification of data. To avoid accidental modification, object variables can only be modified by object methods.

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

Ans: Class attributes are available on all instance objects of this class. Instance attributes, on the other hand, are accessible only for objects or instances of that class.

A single copy of class attributes is maintained by pvm at the class level. Differential copies of instance attributes are managed by pvm at the object/instance level.

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

Ans: Yes, self can be included in class method definitions to access the instance variables inside class methods.

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

Ans: Entering \_\_radd\_\_ Python will first try \_\_add\_\_(), and if that returns Not Implemented Python will check if the right-hand operand implements \_\_radd\_\_, and if it does, it will call \_\_radd\_\_() rather than raising 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?

Ans: Reflection methods often encounter the requirement that they must assign a method of the executing object, a variable of the calling object, or a field of the object, but when encoding the code the method name or field name must be assigned to can't decide. Must be in the form of a string entered via a parameter.

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

Ans: \_\_iadd\_\_ method is called when we use implementation like a+=b which is a.\_\_iadd\_\_(b)

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

Ans: Yes, \_\_init\_\_ method will be inherited by subclasses. if we want to customize its behaviour within a subclass we can use super() method.