1. Define the relationship between a class and its instances. Is it a one-to-one or a one-to-many partnership, for example?

**Ans**: one to many partnership

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

**Ans**: For each object or instance of a class, the instance variables are different, hence allow for each object or instance to have different values assigned to instance variables.

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

**Ans**: Class holds its own data members and member functions, which can be accessed and used by creating an instance of that class. It is a blueprint for an object.

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

**Ans**: The methods with a class can be used to access the instance variables of its instance. A function can't access the attributes of an instance of a class or can't modify the state of the object.

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

**Ans**: Yes,Python supports inheritance.
Example:

```
class Employee:

  def __init__(self, name):
    self.name = name

  def printname(self):
    print(self.name)
```

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

**Ans**: Python does not have the private keyword, like in other object oriented languages, but supports encapsulation by convention that a class variable should not directly be accessed and should be prefixed with an underscore.

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

**Ans**: The class attribute is available to all the instance objects of that class. whereas Instance Attributes are accessible only to the object or Instance of that class.A single copy of slass attributes is maintained by pvm at the class level. Whereas different copies of instance attributes are maintained by pvm at objects/instance level.

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

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

9. What is the difference between the _ _add_ _ and the _ _radd_ _ methods?

**Ans**: Entering __radd__ in Python will first try __add__(), and if that returns Not Implemented Then checks if the right-hand operand implements __radd__, and if it does, it will call __radd__() rather than raising a TypeError.

10. 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 method is used if a method is required to execute an object, or a variable in the calling object, or a field of the object should be assigned, while the method name or field name can not be determined when encoding the code, and need to be input in the form of passing strings through parameters.

11. What is the _ _iadd_ _ method called?

**Ans**: __iadd__ method is called when we use implementation like a+=b which is a.__iadd__(b)

12. 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.super() method can be used if we want to customize its behaviour within a subclass.