Q1. Define the relationship between a class and its instances. Is it a one-to-one or a one-to-many partnership, for example?

Class can have many instances / entities. A class is formed from the instances (Like meta class).

Its relation is one-to-many.

Like for a class any number of instances can be create like objects

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

Instance variables, Instance methods will be held by instances.

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

It will be stored in the form of class variables or class methods.

An implementation of a user defined steps which are required to accomplish a certain task that will be stored in a class.

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

Methods definitions are always present inside a class whereas no need a class to define a function.

Methods are associated with the objects of the class they belong to, as functions are not associated with any object.

A method is called ‘on’ an object. We cannot invoke it just by its name, we can invoke a function just by its name.

Methods can operate on the data of the object they associate with. functions operate on the data you pass to them as arguments.

Methods are dependent on the class they belong to, functions are independent entities in a program.

A method requires to have ‘self’ as its first argument. Functions do not require any ‘self’ argument. They can have zero or more arguments.

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

Yes its supported in python.

Syntax:

Class A:

Pass

Class B(A):

Pass:

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

3 encapsulation are being supported by python like public, private, protected.

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

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. Instance variables, on the other hand, can be assigned or changed at any time.

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

The reason you need to use self. is because Python use to refer to instance attributes.

self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python

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

if myobj + 4 was valid but 4 + myobj was invalid. That's arbitrary and restrictive — addition is supposed to be commutative. Enter \_\_radd\_\_. Python will first try (4).\_\_add\_\_(myobj), and if that returns NotImplemented Python will check if the right-hand operand implements \_\_radd\_\_, and if it does, it will call myobj.\_\_radd\_\_(4) rather than raising a TypeError. And now everything can proceed as usual, as your class can handle the case and implement your behavior, rather than the built-in type's \_\_add\_\_ which is fixed and doesn't know about your 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?

Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object. Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

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

With this \_\_iadd\_\_ method will perform the addition inplace even when we are adding the num with objects. Which will not works with the \_\_add\_\_ method.

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

\_ \_init\_ \_ method will be inherited in the subclasses

If need to customize the behaviours then need to add the super in it as below:

Super()\_\_init\_\_(): and in that we can call the super class method as well as we can add new functionality as well.