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:** Instance is something which can be used to access properties, methods and attributes of class. Yes, a class have one to many relationships with instances, a class can have multiple instances.

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

**Ans:** We can say that instance is not holding any kind of physical data, it just having reference of class attributes and methods, and once they called via instance the actual code run comes into picture.

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

**Ans:** Class stored methods and attributes which it want to expose have knowledge is stored in a class.

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

**Ans:** Pythonmethod is similar to function in generic functionality wise, but it has some distinct features like passing variable number of arguments in method and returning multiple values from method.

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

**Ans:** Yes inheritance is supported in python, syntax is – Child class(Parent class).

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

**Ans:** Instance of a class can be made by syntax class = Class(), and values can be assigned either at the time of object creation or later by accessing properties with . notation and assign values. Private variables in a class can be created by prefixing \_ to variable name.

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

**Ans:** Class variable is declared inside a class, while instance variable is declared outside a class. Values to the class variables assigned with the help of instance variable.

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

**Ans:** When we require to initialise outside parameters to class variables, in that case we use “self” keyword in class methods.

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

**Ans:** \_\_radd\_\_ are only called if the left operand does not support the corresponding add operation and the operands are of different types, whereas \_\_add\_\_ method called when both of the operand supports the corresponding add operation.

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 refers to the ability for code to be able to examine attributes about objects that might be passed as parameters to a function. For example, if we write type(obj) then Python will return an object which represents the type of obj.

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. Further, if we write str() we get a string which is the empty string.

Reflection-enabling functions include type(), isinstance(), callable(), dir() and getattr().

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

**Ans:** \_iadd\_\_ is actually storing that value its adding into the self.value.

def \_\_iadd\_\_(self, other):

self.value = self + other

return self.value

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 inherited by subclass, to customize the behaviour in a subclass we have to override \_\_init\_\_method in subclass.