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

Answer: A class defines the structure and behaviors of all entities of a given type. An object is one particular "instance" of that type of entity. For example, if Dog is a class, then a particular dog named Lassie would be an object of type Dog.

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

Answer: Instance data is defined in the data division in the object paragraph of a class definition and is processed by procedural code in the instance methods of that class. Instance data is organized into logical records and independent data description entries in the same manner as program data

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

Answer: A knowledge class includes the following: A hierarchical structure of subject terms allowing multiple levels of knowledge organization, constructed on classification principles. Provision for open-ended search vocabulary derived from controlled vocabularies and/or free-text terms.

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

Answer: Unlike a function, methods are called on an object. Like in our example above we call our method . i.e. “my_method” on the object “cat” whereas the function “sum” is called without any object. Also, because the method is called on an object, it can access that data within it

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

Answer: An object-oriented programming language like Python, not only supports inheritance but multiple inheritance as well. The mechanism of inheritance allows programmers to create a new class from a pre-existing class, which supports code reusability.

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

Answer: In Python, Encapsulation can be achieved by declaring the data members of a class either as private or protected. In Python, 'Private' and 'Protected' are called Access Modifiers, as they modify the access of variables or methods defined in a class. Let us see how access modifiers help in achieving Encapsulation

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

Answer: Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.A class is a blueprint which you use to create objects. An object is an instance of a class - it's a concrete 'thing' that you made using a specific class. So, 'object' and 'instance' are the same thing, but the word 'instance' indicates the relationship of an object to its class.

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

Answer: 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.

The expression a+b is internally translated to the method call a.add(b). But if a and b are of different types, it is possible that a's implementation of addition cannot deal with objects of b's type (or maybe a does not have a add method, at all). So, if a.add(b) fails, Python tries b.radd(a) instead, to see if b's implementation can deal with objects of a's type.

Q10. When is it necessary to use a reflection method? When do you not need it, even though you support the operation in question?

Answer: 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. In other words, writing str() is the same thing as writing “”. Likewise, writing list() is the same thing as writing [].

Q11. What is the _ *iadd* _ method called?

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?

Answer: Yes, **init** method will be inherited by subclasses. If we want to customize its behaviour within a subclass, we can use super() method.

Double-click (or enter) to edit