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

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

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

Instance objects contains the Instance variables which are specific to that specific Instance object.

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

 Class creates a user-defined data structure, which holds its own data members and member functions, which 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?**

A method, like a function, is a set of instructions that perform a task. The difference is that **a method is associated with an object, while a function is not**.

Methods in python are very similar to functions except for two major differences.

* The method is implicitly used for an object for which it is called.
* The method is accessible to data that is contained within the class.

Let’s understand the method through one simple code –

**class Pet(object):**

**def my\_method(self):**

**print("I am a Cat")**

**cat = Pet()**

**cat.my\_method()**

## Output

I am a Cat

In the above code, we first defined class “Pet”. Then we created the object “cat” from this blueprint. Next, we call our custom method called my\_method with the object(.i.e. cat).

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

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

1. Simple Inheritence
2. Multiple Inheritence
3. Multilevel lInheritence
4. Hybrid Inheritence
5. Hierracial Inheritence

In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.

1. **class** derive-**class**(<base **class** 1>, <base **class** 2>, ..... <base **class** n>):
2. <**class** - suite>

**class** Animal:

**def** speak(self):

**print**("Animal Speaking")

#child class Dog inherits the base class Animal

**class** Dog(Animal):

**def** bark(self):

**print**("dog barking")

d = Dog()

d.bark()

d.speak()

**Output:**

dog barking

Animal Speaking

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

Encapsulation can be achieved by declaring the data members and methods of a class either as private or protected. But **In Python, we don't have direct access modifiers like public, private, and protected**.

Private members are the same as protected members. The difference is that class members who have been declared private should not be accessed by anyone outside the class or any base classes. Python does not have Private instance variable variables that can be accessed outside of a class.

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

A class variable is a variable that defines a particular property or attribute for a class. We can share these variables between class and its subclasses. It is usually defined whenever we begin the execution of the program. We have to declare these variables with the help of the static keyboard.

An instance variable is a variable whose value is specified to the Instance and shared among different instances. We cannot share these variables between classes. However, they only fit in a particular class. It is usually defined whenever we create an instance of the class. We have to declare these variables without utilizing the static keyword.

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

Yes, self can included in class method definations to access the instance variables inside class methods.

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

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?**

Reflection method we often encounter the requirement that a method in the executing 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.

**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?**

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