Q1. What is the difference between \_\_getattr\_\_ and \_\_getattribute\_\_?

A1. In Python, **\_\_getattr\_\_** and **\_\_getattribute\_\_** are both methods of a class that get called when an attribute is accessed. The key difference between them is that **\_\_getattribute\_\_** is called every time an attribute is accessed, while **\_\_getattr\_\_** is only called when an attribute is not found through normal lookup. Here's a more detailed explanation:

**\_\_getattribute\_\_(self, name)**: This method is called every time an attribute is accessed, regardless of whether the attribute exists or not. It takes the name of the attribute as an argument, and must return the value of the attribute or raise an AttributeError if the attribute does not exist. **\_\_getattribute\_\_** is a low-level method that can be used to intercept all attribute access in a class.

**\_\_getattr\_\_(self, name)**: This method is called only when an attribute is not found through normal lookup (i.e. not in the object's **\_\_dict\_\_**, class hierarchy, or instance variables). It takes the name of the attribute as an argument, and must return the value of the attribute or raise an AttributeError if the attribute cannot be found. **\_\_getattr\_\_** is typically used to implement dynamic attribute lookup or delegation.

In summary, **\_\_getattribute\_\_** is called every time an attribute is accessed, while **\_\_getattr\_\_** is called only when an attribute cannot be found through normal lookup.

Q2. What is the difference between properties and descriptors?

A2. Both properties and descriptors provide a way to manage attribute access in Python. The key difference between them is the level of control they provide.

Properties are a simpler mechanism that allows us to add behavior to an existing attribute of an object. They define methods that can be used to get, set or delete the value of an attribute, and these methods are called automatically when we access the attribute. Properties are defined using the **@property**, **@<attr>.setter**, and **@<attr>.deleter** decorators.

Descriptors, on the other hand, provide a more flexible and powerful way to control attribute access. A descriptor is an object that defines one or more of the **\_\_get\_\_**, **\_\_set\_\_**, and **\_\_delete\_\_** methods, which are called by Python to handle attribute access. Descriptors are usually implemented as a separate class, and they can be shared across multiple objects. Examples of descriptors include the **property**, **classmethod**, and **staticmethod** built-in functions. Descriptors can also be used to implement custom attribute access logic.

Q3. What are the key differences in functionality between \_\_getattr\_\_ and \_\_getattribute\_\_, as well as properties and descriptors?

A3. The **\_\_getattr\_\_** method and **\_\_getattribute\_\_** method are used in Python classes to intercept attribute access. The main difference between them is that **\_\_getattr\_\_** is called only when the attribute being accessed is not found in the usual places, such as the instance's dictionary or the class's dictionary, while **\_\_getattribute\_\_** is called for every attribute access, regardless of whether or not the attribute exists in the instance or class.

On the other hand, properties and descriptors are two different ways of customizing attribute access in Python. A property is essentially a method that is accessed like an attribute, allowing us to run custom code when an attribute is accessed or set. A descriptor is a more general mechanism for customizing attribute access, allowing us to define custom behaviors for attribute access, assignment, and deletion at both the class and instance level. In summary, properties are a specific use case of descriptors, which can be more general and powerful.