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

In Python, both \_\_getattr\_\_ and \_\_getattribute\_\_ are methods that are called when an attribute is accessed on an object. However, there is a significant difference between the two:

1. \_\_getattr\_\_: This method is called when an attribute that does not exist is accessed on an object. It takes a single argument, the name of the attribute being accessed, and should return the value of the attribute or raise an AttributeError if the attribute is not found. This method is only called when the attribute is not found in the usual places (i.e., the object's instance dictionary, its class, or its superclasses).
2. \_\_getattribute\_\_: This method is called every time an attribute is accessed on an object, regardless of whether the attribute exists or not. It takes a single argument, the name of the attribute being accessed, and should return the value of the attribute or raise an AttributeError if the attribute is not found. This method is called before \_\_getattr\_\_, so if it is defined, \_\_getattr\_\_ will not be called for any attributes.

In general, \_\_getattribute\_\_ is more powerful but also more dangerous, as it can potentially lead to infinite recursion if not used carefully. \_\_getattr\_\_ is a more common way to define custom attribute access behavior in Python.

**Q2. What is the difference between properties and descriptors?**

Properties and descriptors are both ways of defining and controlling attribute access in Python, but they operate at different levels.

Properties are a way of defining getter, setter, and deleter methods for an attribute on a class, but they do not change how the attribute is stored on an instance. Properties are defined using the @property, @attribute\_name.setter, and @attribute\_name.deleter decorators. When a property is accessed on an instance, the getter method is called; when the property is assigned to, the setter method is called; and when the property is deleted, the deleter method is called. Properties are a convenient way to encapsulate attribute access logic without changing the interface of a class.

Descriptors, on the other hand, are a lower-level way of defining attribute access behavior. A descriptor is a class that defines one or more of the special methods \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_. When an instance attribute is accessed, Python looks for the descriptor in the class of the instance, and if it finds one, it calls the appropriate method on the descriptor object. Descriptors are more powerful than properties because they can change the way attributes are stored on instances, and they can be used to define custom attribute access behavior that goes beyond simple getter, setter, and deleter methods. However, they are also more complex to use and require a deeper understanding of the Python object model.

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

\_\_getattr\_\_ and \_\_getattribute\_\_ are both methods that are called when an attribute lookup is performed on an object. The difference between them is that \_\_getattr\_\_ is only called when the requested attribute is not found in the object's instance, class or superclass dictionary. On the other hand, \_\_getattribute\_\_ is called for all attribute lookups and is used to implement attribute access customization, which can be dangerous if not used with caution.

Properties and descriptors are both ways to customize attribute access in Python. Properties allow you to define methods that are accessed like attributes, and they can be used to implement getters, setters and deleters for an attribute. Descriptors are objects that define methods such as \_\_get\_\_, \_\_set\_\_ and \_\_delete\_\_, which allow customization of attribute access at the class level. They can be used to implement complex attribute access logic and validation, and they are often used in conjunction with properties to provide additional functionality.