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

**Answer:** The \_\_getattr\_\_ and \_\_getattribute\_\_ methods in Python are both used for attribute access in classes, but they have some important differences:

Invocation:

\_\_getattr\_\_(self, name): This method is invoked when the requested attribute name is not found through the normal process of attribute lookup. It is only called if the attribute cannot be found in the object's dictionary, its class, or its parent classes.

\_\_getattribute\_\_(self, name): This method is invoked for every attribute access, regardless of whether the attribute is found or not. It is called before checking the object's dictionary, its class, or its parent classes.

Behavior:

\_\_getattr\_\_: This method is typically used for handling attribute access for non-existent attributes. It allows you to define custom behavior when an attribute is requested but not found. You can dynamically generate or compute the attribute value or raise an AttributeError to indicate that the attribute does not exist.

\_\_getattribute\_\_: This method is a more general-purpose attribute access method. It is called for every attribute access, including existing attributes. It allows you to intercept and modify attribute access behavior for all attributes. However, you need to be cautious when implementing \_\_getattribute\_\_ as it can potentially lead to infinite recursion if not handled correctly.

Error Handling:

\_\_getattr\_\_: This method is only called when an attribute is not found. If \_\_getattr\_\_ is defined, it is always called, even if the attribute is later found through other means.

\_\_getattribute\_\_: This method is called for every attribute access, regardless of whether the attribute exists or not. If the attribute is found, \_\_getattribute\_\_ is responsible for returning its value. If the attribute is not found, \_\_getattribute\_\_ can raise an AttributeError or return a default value.

Q2. What is the difference between properties and descriptors?

**Answer:** Properties and descriptors are both mechanisms in Python that allow you to define custom behavior for attribute access and modification, but they have some differences:

Usage:

Properties: Properties are a high-level way of defining attributes that automatically trigger getter, setter, and deleter methods when accessed or modified. They are defined using the @property, @<attribute>.setter, and @<attribute>.deleter decorators. Properties are typically used for managing the access and modification of individual attributes of an object.

Descriptors: Descriptors are a lower-level mechanism that allows you to define custom behavior for attribute access, modification, and deletion at the class level. Descriptors are implemented by defining a class with at least one of the \_\_get\_\_, \_\_set\_\_, or \_\_delete\_\_ methods. Descriptors provide more fine-grained control over attribute access and can be shared across multiple attributes or classes.

Scope of Application:

Properties: Properties are defined at the attribute level within a class. Each property is associated with a specific attribute and is accessed and modified using the attribute name.

Descriptors: Descriptors are defined at the class level and can be shared across multiple attributes or even multiple classes. They allow you to define a common behavior for multiple attributes that have similar access patterns.

Access Control:

Properties: Properties are often used to provide controlled access to attribute values. You can define custom getter and setter methods to perform additional validation, computations, or transformations when getting or setting the attribute value. Properties can enforce constraints and ensure data integrity.

Descriptors: Descriptors can provide even more control over attribute access. They can be used to define custom behavior not only for getting and setting attribute values but also for attribute deletion. Descriptors allow you to intercept and modify attribute access at a lower level, enabling you to implement more complex attribute management logic.

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

**Answer:** The key differences in functionality between \_\_getattr\_\_, \_\_getattribute\_\_, properties, and descriptors are as follows:

Invocation:

\_\_getattr\_\_(self, name): This method is invoked when the requested attribute name is not found through the normal attribute lookup process. It is only called if the attribute cannot be found in the object's dictionary, its class, or its parent classes.

\_\_getattribute\_\_(self, name): This method is invoked for every attribute access, regardless of whether the attribute is found or not. It is called before checking the object's dictionary, its class, or its parent classes.

Properties: Properties are accessed and modified using the attribute name, but their behavior is defined by the associated getter, setter, and deleter methods.

Descriptors: Descriptors define the behavior for attribute access, modification, and deletion at the class level. They are invoked when the attribute is accessed, modified, or deleted, regardless of whether the attribute is found in the object's dictionary or its class.

Error Handling:

\_\_getattr\_\_: This method is called only when an attribute is not found through other means. It allows you to define custom behavior for handling non-existent attributes, such as dynamically generating or computing attribute values or raising an AttributeError.

\_\_getattribute\_\_: This method is called for every attribute access, whether the attribute is found or not. If the attribute is not found, an AttributeError is raised unless it is handled within the \_\_getattribute\_\_ method itself.

Properties: Properties allow you to define custom behavior for attribute access and modification. You can perform additional validation, computation, or transformation on the attribute values, and raise appropriate exceptions if needed.

Descriptors: Descriptors provide even more control over attribute access and modification. They allow you to define custom behavior for getting, setting, and deleting attributes. You can enforce constraints, perform additional operations, or trigger side effects when attributes are accessed or modified.

Scope of Application:

\_\_getattr\_\_ and \_\_getattribute\_\_: These methods are typically defined within a specific class and are applied to all instances of that class. They provide attribute access customization at the instance level.

Properties: Properties are defined at the attribute level within a class. Each property is associated with a specific attribute and provides customized access and modification behavior for that attribute.

Descriptors: Descriptors are defined at the class level and can be shared across multiple attributes or even multiple classes. They allow you to define a common behavior for multiple attributes that have similar access patterns.