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

**Answer:-**

The \_\_getattr\_\_ and \_\_getattribute\_\_ are two special methods in Python that are used for attribute access in classes. However, there is a fundamental difference between them:

1. \_\_getattr\_\_ is invoked only when the requested attribute is not found through the normal attribute lookup process. It is called when an attribute is accessed, and if the attribute is not found as an instance attribute or through inheritance, \_\_getattr\_\_ is called. This method allows you to define a fallback behavior when an attribute is not present. Example:

**class MyClass:**

**def \_\_getattr\_\_(self, name):**

**return f"Attribute {name} does not exist."**

**obj = MyClass()**

**print(obj.some\_attribute) # Output: Attribute some\_attribute does not exist.**

\_\_getattribute\_\_ is called for every attribute access, whether the attribute exists or not. It is invoked before checking if the attribute exists as an instance attribute or through inheritance. This method gives you the opportunity to intercept and customize attribute access behavior. However, it's important to use super().\_\_getattribute\_\_() within \_\_getattribute\_\_ to avoid infinite recursion. Example:

**class MyClass:**

**def \_\_getattribute\_\_(self, name):**

**print(f"Accessing attribute: {name}")**

**return super().\_\_getattribute\_\_(name)**

**obj = MyClass()**

**print(obj.some\_attribute) # Output: Accessing attribute: some\_attribute**

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

**Answer:-**

Properties and descriptors are both mechanisms in Python for managing attribute access, but they differ in their implementation and usage:

Properties:

* Properties are a high-level way to define attribute access behavior in a class.
* They use the @property decorator to define a method that is accessed like an attribute but executed like a method.
* Properties allow you to customize the retrieval (getter) and assignment (setter) of an attribute value.
* They provide a clean and intuitive way to encapsulate attribute access and add additional logic, such as validation or computation, while maintaining a similar syntax as accessing regular attributes.
* Properties are defined on a per-attribute basis within a class.

**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\_\_ and \_\_getattribute\_\_, as well as properties and descriptors, are as follows:

1. \_\_getattr\_\_ vs. \_\_getattribute\_\_:
   * \_\_getattr\_\_ is called only when an attribute is not found through the normal attribute lookup process, while \_\_getattribute\_\_ is called for every attribute access, regardless of whether the attribute exists or not.
   * \_\_getattr\_\_ is a fallback mechanism for handling missing attributes, while \_\_getattribute\_\_ allows interception of all attribute accesses, including existing attributes.
   * \_\_getattr\_\_ is called after the normal attribute lookup has failed, while \_\_getattribute\_\_ is called before attempting the normal attribute lookup.
   * \_\_getattr\_\_ is commonly used for implementing dynamic attributes or handling undefined attributes, while \_\_getattribute\_\_ is typically used for attribute interception and customization.
2. Properties vs. Descriptors:
   * Properties are a high-level and convenient way to define attribute access behavior using decorators like @property, @<attribute>.setter, and @<attribute>.deleter. Descriptors, on the other hand, are a lower-level protocol implemented through special methods like \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_.
   * Properties allow customization of attribute retrieval and assignment on a per-attribute basis, while descriptors provide more fine-grained control over attribute access behavior, enabling advanced features like lazy loading, attribute delegation, or validation.
   * Properties encapsulate attribute access within methods that are accessed like attributes, providing a clean syntax. Descriptors are typically defined at the class level rather than the instance level and require explicitly accessing them via the class or instance.
   * Properties are commonly used for managing attribute access within a class, while descriptors offer a more flexible mechanism for customizing attribute access behavior.