**PYTHON ADVANCE ASSIGNMENT\_10**

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

Both \_\_getattr\_\_ and \_\_getattribute\_\_ are special methods in Python that are used to handle attribute access in objects, but they have different behaviors.

\_\_getattr\_\_ is called when an attribute is not found via the usual lookup process, i.e., when the attribute does not exist in the instance or its class. This method can be used to dynamically generate or compute the value of an attribute when it is first accessed. If \_\_getattr\_\_ is not defined, then AttributeError is raised.

On the other hand, \_\_getattribute\_\_ is called for every attribute access, whether the attribute exists or not. This method can be used to implement customized behavior for attribute access. If \_\_getattribute\_\_ is not defined, then the default attribute access behavior is used.

Here's an example to illustrate the difference:

python

class MyClass:

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

print(f"\_\_getattr\_\_ called for {name}")

return 42

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

print(f"\_\_getattribute\_\_ called for {name}")

return object.\_\_getattribute\_\_(self, name)

obj = MyClass()

print(obj.foo)

print(obj.bar)

In this example, when obj.foo is accessed, \_\_getattr\_\_ is called and returns 42. When obj.bar is accessed, \_\_getattribute\_\_ is called, which in turn calls the default attribute access behavior, since bar exists neither in the instance nor its class. The output of the code would be:

sql

\_\_getattr\_\_ called for foo

42

\_\_getattribute\_\_ called for bar

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'MyClass' object has no attribute 'bar'

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

In Python, properties and descriptors are both mechanisms for controlling access to class attributes, but they work in different ways and have different use cases.

Properties are a type of descriptor that allows you to define methods that act like attributes. When you access a property, the method associated with that property is called and its return value is returned instead of the method itself. This can be useful for computing attributes on the fly or for enforcing some kind of validation or transformation on an attribute before it is returned.

Descriptors, on the other hand, are a more general mechanism for defining how attributes are accessed and manipulated. A descriptor is an object that defines at least one of the following methods: \_\_get\_\_, \_\_set\_\_, or \_\_delete\_\_. When an instance of a class that uses a descriptor is accessed, Python looks up the descriptor object and calls its appropriate method to get, set, or delete the attribute.

The key difference between properties and descriptors is that properties are a simple way to define a specific kind of descriptor that is mainly used for defining read-only or read-write attributes that require some additional processing, while descriptors are a more general and flexible mechanism that can be used to define a wide range of attribute access patterns, including custom validation, type coercion, and more complex behavior.

In summary, properties are a special kind of descriptor that allows you to define methods that act like attributes, while descriptors are a more general mechanism for controlling attribute access and manipulation.

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

\_\_getattr\_\_ and \_\_getattribute\_\_ are both methods in Python that are used to handle attribute access for objects. However, they differ in how they handle attribute access and when they are called.

\_\_getattribute\_\_ is called whenever an attribute is accessed on an object, regardless of whether the attribute actually exists or not. This means that if you define \_\_getattribute\_\_ for a class, all attribute access will go through it, even if the attribute exists in the instance dictionary or is defined by a parent class.

On the other hand, \_\_getattr\_\_ is only called when an attribute is not found in the instance dictionary or in any of its parent classes. It is essentially a fallback method for attribute access. This means that if you define \_\_getattr\_\_ for a class, it will only be called when the attribute doesn't already exist on the instance.

Properties and descriptors are two ways to define custom behavior for accessing and modifying attributes on an object.

A property is a special kind of attribute that is accessed and modified like a regular attribute, but its value is computed dynamically based on some underlying data. Properties are defined using the @property decorator in Python, and they allow you to define custom getter and/or setter methods that are called when the property is accessed or modified.

A descriptor, on the other hand, is a more general mechanism for defining custom behavior for accessing and modifying attributes. A descriptor is an object that defines one or more of the methods \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_. When an attribute is accessed, modified, or deleted, these methods are called to perform the appropriate action. Descriptors are often used to define custom data types that have more complex behavior than simple properties.