Python Advanced Assignment-10

Q1. What is the difference between `__getattr__` and `__getattribute__`?

Ans.

- `__getattr__`: This method is called only when an attribute that is **not** found in an object is accessed. It acts as a fallback mechanism when the attribute being accessed does not exist. It is useful for defining behavior when accessing nonexistent attributes.

```
class Sample:
    def __getattr__(self, name):
        return f"'{name}' attribute not found"

obj = Sample()
print(obj.nonexistent)  # Output: "'nonexistent' attribute not found"
```

- `__getattribute__`: This method is called every time any attribute of an object is accessed, regardless of whether it exists. It has broader control but should be used carefully to avoid infinite recursion. To avoid recursion, access attributes using `super().__getattribute__(name)`.

```
class Sample:
    def __getattribute__(self, name):
        print(f"Accessing attribute '{name}'")
        return super().__getattribute__(name)

obj = Sample()
# Accessing any attribute will trigger __getattribute__
```

Q2. What is the difference between properties and descriptors?

Ans.

- Properties:

- A convenient way to manage attribute access using `@property` decorators.

- Properties define methods that are accessed like attributes (getters, setters, and deleters).

- They are easy to use for simple use cases where you want to define behavior for accessing or modifying a single attribute.

```
class Person:
    def __init__(self, name):
        self._name = name

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value
```

- Descriptors:

- Descriptors are objects that define methods (`__get__`, `__set__`, and `__delete__`) to control attribute access and modification.

- They provide more flexibility and power, allowing you to manage attribute access at a more granular level.

- Descriptors are useful for more complex cases, such as when multiple attributes share the same access pattern or when creating reusable behavior across classes.

```
class Descriptor:
    def __get__(self, instance, owner):
        return instance._value

    def __set__(self, instance, value):
        instance._value = value
```

Q3. What are the key differences in functionality between `__getattr__` and `__getattribute__`, as well as properties and descriptors?

Ans.

- `__getattr__` vs. `__getattribute__`:

- `__getattr__` is called **only** when an attribute is **not found** in an object, making it useful for handling missing attributes dynamically.

- `__getattribute__` is called **every time** any attribute is accessed, giving more control but requiring caution to avoid recursion.

- Use `__getattr__` for handling missing attributes, and use `__getattribute__` when you need to control or log all attribute accesses.

- Properties vs. Descriptors:

- Properties are simpler and are suited for managing a single attribute. They use the `@property` decorator and are easy to implement for basic access control or modification.

- Descriptors provide greater flexibility and can be used across multiple attributes or classes. They allow you to create reusable behavior and manage attribute access at a more complex level.

- Properties internally use descriptors but are a higher-level and more straightforward way to manage attributes in simple cases. Descriptors are more suitable when building custom attribute access patterns that need to be reused.