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

Sol:-

\_\_getattr\_\_:

This method is called when the requested attribute is not found through the normal process of attribute lookup.

It is a fallback method that is only invoked if the attribute is not present in the object's instance dictionary, and it's not found in its class or any of its superclasses.

You can define \_\_getattr\_\_ to provide custom behavior for accessing non-existent attributes. It takes a single argument, the name of the attribute being accessed.

\_\_getattr\_\_ is not called for existing attributes, so it's a way to handle attribute access only when the attribute is not found.

If \_\_getattr\_\_ is not defined, attempting to access a non-existent attribute will raise an AttributeError.

\_\_getattribute\_\_:

This method is called for every attribute access, regardless of whether the attribute exists or not.

It is invoked before checking the instance dictionary or the class hierarchy for the attribute.

\_\_getattribute\_\_ allows you to intercept all attribute access and customize the behavior.

It takes a single argument, the name of the attribute being accessed, and should return the value of the attribute or raise an exception.

Caution should be exercised when using \_\_getattribute\_\_ since it affects all attribute access, including accessing existing attributes. Care must be taken to avoid infinite recursion by using the base class's \_\_getattribute\_\_ method or accessing attributes through the super() function.

Q2. What is the difference between properties and descriptors?

Sol:-

Properties:

Properties are a high-level mechanism for managing attribute access in Python classes.

They allow you to define methods that are used to access, set, or delete attribute values, but they are bound to a specific attribute name.

Properties are defined using the @property decorator for the getter method, and additional decorators (@<attribute\_name>.setter and @<attribute\_name>.deleter) can be used to define setter and deleter methods.

Properties provide a convenient way to define computed or derived attributes, encapsulate attribute access logic, or enforce certain constraints on attribute values.

Properties are accessed as if they were regular attributes, and the getter, setter, and deleter methods are automatically called when accessing, setting, or deleting the property.

Descriptors:

Descriptors are a lower-level mechanism for managing attribute access and can be more granular and flexible than properties.

A descriptor is an object that implements one or more of the special methods (\_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_) that control attribute access.

Descriptors can be defined as individual classes or as methods within a class.

Descriptors can be assigned to class attributes, allowing you to control how attribute access is handled for instances of that class.

Descriptors provide fine-grained control over attribute access, allowing you to customize behavior at the attribute level.

Descriptors can be shared across multiple attributes or even multiple classes, providing a way to reuse attribute access logic.

Unlike properties, which are bound to a specific attribute name, descriptors can be used to manage access for multiple attributes in a class or across multiple classes.

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

Sol:-

\_\_getattr\_\_ vs. \_\_getattribute\_\_:

\_\_getattr\_\_ is called only when an attribute is not found through normal attribute lookup, while \_\_getattribute\_\_ is called for every attribute access, regardless of existence.

\_\_getattr\_\_ is a fallback method that is invoked only when the attribute is not present in the object's instance dictionary or in its class hierarchy. In contrast, \_\_getattribute\_\_ is invoked before checking the instance dictionary or class hierarchy, making it more powerful but also more dangerous due to the potential for infinite recursion.

\_\_getattr\_\_ is used for handling non-existent attributes, while \_\_getattribute\_\_ allows customization of attribute access behavior for both existing and non-existent attributes.

Properties vs. Descriptors:

Properties provide a high-level mechanism for managing attribute access by defining getter, setter, and deleter methods for a specific attribute name. They allow you to encapsulate attribute access logic and enforce constraints on attribute values.

Descriptors are a lower-level mechanism for controlling attribute access. They are objects that define the \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_ methods to control attribute access for one or more attributes. Descriptors can be shared across multiple attributes or even multiple classes, providing more fine-grained control and reusability.

Properties are bound to a specific attribute name and are accessed like regular attributes, automatically invoking the getter, setter, and deleter methods. Descriptors, on the other hand, can be assigned to class attributes and control how attribute access is handled for instances of that class, allowing more flexibility in customizing behavior.

Properties are more convenient for simple attribute access control and computed attributes, while descriptors offer greater flexibility and granularity in managing attribute access, but they require a deeper understanding of the descriptor protocol.