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

Q2. What is the difference between properties and descriptors?

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

Ans: Q1. The main difference between `\_\_getattr\_\_` and `\_\_getattribute\_\_` lies in their functionality and behavior.

- `\_\_getattr\_\_` is called when an attribute is not found through the usual process of attribute lookup. It is only invoked when the requested attribute is not present in the object's dictionary, nor in its class hierarchy. It allows you to dynamically compute or fetch an attribute when it is accessed.

- `\_\_getattribute\_\_` is called for every attribute access, regardless of whether the attribute exists or not. It is invoked before `\_\_getattr\_\_` and allows you to intercept and customize all attribute access in the object, including existing attributes.

In summary, `\_\_getattr\_\_` is used when an attribute is not found, while `\_\_getattribute\_\_` is used for every attribute access, regardless of its existence.

Q2. Properties and descriptors are both used to add behavior to attributes, but they have different levels of flexibility and control.

- Properties: Properties are a high-level way of defining attribute accessors (getters, setters, and deleters) in Python classes. They provide a simple way to encapsulate attribute access and perform additional actions when getting, setting, or deleting an attribute. Properties are defined using the `@property` decorator.

- Descriptors: Descriptors provide a lower-level protocol for defining attribute access. They allow you to define how attribute access is handled at the class level, rather than at the instance level. Descriptors can be used to customize attribute access, implement computed attributes, and enforce constraints on attribute values. Descriptors are defined by creating classes that implement the descriptor protocol, which involves defining `\_\_get\_\_`, `\_\_set\_\_`, and/or `\_\_delete\_\_` methods.

In summary, properties provide a simpler and more convenient way to add attribute accessors, while descriptors offer more control and customization options for attribute access.

Q3. The key differences in functionality between `\_\_getattr\_\_`, `\_\_getattribute\_\_`, properties, and descriptors are as follows:

- `\_\_getattr\_\_`: Called when an attribute is not found via the usual attribute lookup. It allows you to dynamically compute or fetch attributes that are not present.

- `\_\_getattribute\_\_`: Called for every attribute access, whether the attribute exists or not. It provides a way to intercept and customize attribute access at a low level.

- Properties: Provide a high-level way to define attribute accessors using the `@property` decorator. They allow you to customize the behavior of getting, setting, and deleting attributes, while still accessing them as normal attributes.

- Descriptors: Provide a lower-level protocol for defining attribute access. They allow you to define how attribute access is handled at the class level and provide more control over the behavior of getting, setting, and deleting attributes.

In summary, `\_\_getattr\_\_` and `\_\_getattribute\_\_` are methods used for attribute access customization, while properties and descriptors offer different levels of abstraction and control for defining attribute accessors and behaviors.