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

Answer:

|  |  |
| --- | --- |
| \_\_getattr\_\_ | \_\_getattribute\_\_ |
| It is called when the attribute is not found in the instance in the usual way. | It is called every time an attribute is accessed on the object, regardless of whether the attribute exists or not. |
| It’s invoked last, if Python can’t find the attributre. | It’s invoked first, if Python find or don’t find the attribute. |
| If no attribute is found, \_\_getattr\_\_ returns a default value. | It is used to find an attribute of a class. It raises an AttributeError if it fails to find an attribute of a class. |

Q2. What is the difference between properties and descriptors?

Answer:

|  |  |
| --- | --- |
| Properties | Descriptors |
| It is help to bind getter, setter and delete functions together with an attribute name, using the built-in property function or @property decorator | It is help to bind getter, setter and delete functions into a separate class and then assign an object of this class to the attribute name in the main class. |
| It is a special case of descriptors that are used to provide access to a single attribute with a getter, setter, and delete methods. | It is a more general mechanism that can be used to define how attribute access is handled in general. |
| It is a high-level application of descriptors i.e., it is implemented using descriptors. | It is a low-level mechanism that helps us to access of an object’s attributes. |

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

Answer:

|  |  |
| --- | --- |
| \_\_getattr\_\_ | \_\_getattribute\_\_ |
| This is called only when the attribute not found. | This is called unconditionally when an attribute is being retrieved from an instance. |
| This can only be used to fetch attributes | This can be used to implement attribute access control and modify the values of exiting attributes. |
| This is usually used for lazily computing attributes | This is more suitable for implementing low-level attribute access control. |

|  |  |
| --- | --- |
| properties | descriptors |
| This is implemented using the property() built-in function or decorator while binding getter, setter, and delete functions together with an attribute name. | This is implemented by defining a separate class with one or more of the getter, setter, and delete functions or methods. After that it is needed to assign an object of this class to the attribute name in the main class. |
| It is provide a high-level interface to an object’s attributes. | It is used for low-level access control and behaviour modification. |