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

object.**\_\_getattr\_\_**(*self*, *name*)

Called when the default attribute access fails with an [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError) (either [\_\_getattribute\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattribute__) raises an [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError) because *name* is not an instance attribute or an attribute in the class tree for self; or [\_\_get\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__get__) of a *name* property raises [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError)). This method should either return the (computed) attribute value or raise an [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError) exception.

Object.**\_\_getattribute\_\_**(*self*, *name*)

Called unconditionally to implement attribute accesses for instances of the class. If the class also defines [\_\_getattr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattr__), the latter will not be called unless [\_\_getattribute\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattribute__) either calls it explicitly or raises an [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError). This method should return the (computed) attribute value or raise an [AttributeError](https://docs.python.org/3/library/exceptions.html" \l "AttributeError" \o "AttributeError) exception. In order to avoid infinite recursion in this method, its implementation should always call the base class method with the same name to access any attributes it needs, for example, object.\_\_getattribute\_\_(self, name).

Q2. What is the difference between properties and descriptors?

Properties :

every object can be either mutable or immutable based on the type of data they hold. Some of these objects like lists and dictionaries are mutable , meaning you can change their content without changing their identity. Other objects like integers, floats, strings and tuples are objects that can not be changed.

The properties of an object are :

* An object has identity (each object is a distinct individual).
* An object has state (it has various properties, which might change).
* An object has behavior (it can do things and can have things done to it).

Descriptors :

Python descriptors are created to manage the attributes of different classes which use the object as reference. A descriptor is a mechanism behind properties, methods, static methods, class methods, and super() .

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

**\_\_getattr\_\_**: Python will call this method whenever you request an attribute that hasn't already been defined

**\_\_getattribute\_\_** : This method will invoked before looking at the actual attributes on the object. Means,if we have \_\_getattribute\_\_ method in our class, python invokes this method for every attribute regardless whether it exists or not.

**Properties:** With Properties we can bind getter, setter and delete functions together with an attribute name, using the built-in property function or @property decorator. When we do this, each reference to an attribute looks like simple, direct access, but involes the appropriate function of the object.

**Descriptor:** With Descriptor we can bind getter, setter and delete functions into a seperate class. we then assign an object of this class to the attribute name in our main class. When we do this, each reference to an attribute looks like simple, direct access but invokes an appropriate function of descriptor object.