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

Ans:

a. \_\_getattribute\_\_ gets called “first”(the highest priority), whether or not there’s the attribute.

b. \_\_getattr\_\_ gets called “last”(the lowest priority), if Python cannot find the attribute.

Q2. What is the difference between properties and descriptors?

Ans: Properties and descriptors are both ways to define attributes that have custom behavior in Python, but they differ in how they are implemented and used.

**a. Properties:**

Properties are a built-in feature of Python that allow us to define methods that can be accessed like attributes. They are defined using the @property decorator and provide a way to customize the behavior of attribute access and assignment. When we access a property, a method is called that computes the value of the attribute on the fly. When we assign to a property, another method is called that handles the assignment. Properties are generally used for simple attribute-like behavior that requires some computation or validation.

Here is an example of a property:

class Circle:

def \_\_init\_\_(self, radius):

self.\_radius = radius

@property

def radius(self):

return self.\_radius

@radius.setter

def radius(self, value):

if value < 0:

raise ValueError("Radius cannot be negative")

self.\_radius = value

c = Circle(5)

print(c.radius) # prints 5

c.radius = 10

print(c.radius) # prints 10

c.radius = -5 # raises a ValueError

In this example, radius is defined as a property with a getter and a setter method. The getter method returns the value of the \_radius attribute, and the setter method checks that the new value is non-negative before setting the \_radius attribute.

**b. Descriptors:**

Descriptors are a more powerful and flexible mechanism for defining custom attribute behavior in Python. They are defined by implementing one or more of the descriptor protocol methods: \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_. These methods are called when the descriptor is accessed or modified, and they allow us to customize the behavior of attribute access and assignment in a variety of ways. Descriptors are generally used for more complex attribute behavior, such as lazy evaluation, caching, or data validation.

Here is an example of a descriptor:

class PositiveNumber:

def \_\_get\_\_(self, instance, owner):

return instance.\_value

def \_\_set\_\_(self, instance, value):

if value < 0:

raise ValueError("Value cannot be negative")

instance.\_value = value

class Rectangle:

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

width = PositiveNumber()

height = PositiveNumber()

r = Rectangle(5, 10)

print(r.width) # prints 5

print(r.height) # prints 10

print(r.area()) # prints 50

r.width = 20

r.height = 30

print(r.width) # prints 20

print(r.height) # prints 30

print(r.area()) # prints 600

r.width = -5 # raises a ValueError

In this example, PositiveNumber is defined as a descriptor that enforces the constraint that the attribute value must be non-negative. The \_\_get\_\_ method returns the value of the \_value attribute, and the \_\_set\_\_ method checks that the new value is non-negative before setting the \_value attribute. The width and height attributes of Rectangle are defined as instances of the PositiveNumber descriptor, which enforces the non-negative constraint on these attributes.

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

Ans: \_\_getattr\_\_ and \_\_getattribute\_\_ are both special methods in Python that are used for customizing attribute access in classes. However, there are some important differences in functionality between the two:

\_\_getattr\_\_ is only called when an attribute cannot be found through the usual lookup process. This method can be used to provide default values for attributes or to handle attribute names that are not valid Python identifiers.

On the other hand, \_\_getattribute\_\_ is called every time an attribute is accessed, regardless of whether it exists or not. This method can be used to implement more fine-grained control over attribute access and to intercept attribute access for specific attributes.

Properties and descriptors are both mechanisms for customizing attribute access, but they differ in how they are implemented and used:

Properties are a built-in feature of Python that allow you to define methods that can be accessed like attributes. They are used to customize the behavior of attribute access and assignment for specific attributes. Properties are generally used for simple attribute-like behavior that requires some computation or validation.

Descriptors are a more flexible and powerful mechanism for customizing attribute access. They are implemented by defining one or more of the descriptor protocol methods: \_\_get\_\_, \_\_set\_\_, and \_\_delete\_\_. These methods are called when the descriptor is accessed or modified, and they allow you to customize the behavior of attribute access and assignment in a variety of ways. Descriptors are generally used for more complex attribute behavior, such as lazy evaluation, caching, or data validation.

In summary, \_\_getattr\_\_ and \_\_getattribute\_\_ are used to customize attribute access at the class level, while properties and descriptors are used to customize attribute access at the attribute level. \_\_getattr\_\_ is only called when an attribute is not found through the usual lookup process, while \_\_getattribute\_\_ is called every time an attribute is accessed. Properties are a simpler and more limited mechanism for customizing attribute access, while descriptors are a more powerful and flexible mechanism.