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

\_\_getattr\_\_ and \_\_getattribute\_ are both methods in Python that allow you to retrieve an attribute of an object, but they work in slightly different ways.

**getattr(object, name[, default])**

retrieves the value of the named attribute from the given object. If the attribute does not exist, it returns the specified default value (if provided) or raises an AttributeError in simple word the attribute does exist, **getattr** won’t be invoked

class Ccp:  
 def \_\_init\_\_(self, address):  
 self.address = address  
  
 def \_\_getattr\_\_(self, name):  
 return name.lower()  
s = Ccp('rampur')  
print(s.address)  
print(s.chinmay)

rampur

chinmay

On the other hand, getattribute is a built-in method that is called for every attribute access on an object.

class Ccp:  
 def \_\_getattribute\_\_(self, attr):  
 return 'New Value'  
d = Ccp  
d.value = "Old value"  
print(d.value)

Old value

Q2. What is the difference between properties and descriptors?

Using the built-in property function in Properties, we can associate getter and setter methods with an attribute name. We may create a distinct class for getter, setter, and deleter methods in descriptor. The attribute name is then assigned to an object of this class.

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

To access the attribute of class which are not defined in the class, we use **getattr**() method. But if the attribute does exist, **getattr** won’t be invoked. **getattribute** will look for every attribute, doesn’t matter if the attribute exists or not.

In Properties, We can bind getter, setter functions with an attribute name, using the built-in property function. In descriptor, We can bind getter, setter (and deleter) functions into a separate class. We then assign an object of this class to the attribute name.