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

Ans:- getattribute: Is **used to retrieve an attribute from an instance**. getattr: Is executed as the last resource when attribute is not found in an object.

\_\_getattribute\_\_

This method **should return the (computed) attribute value or raise an 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

Python getattr() function is used **to access the attribute value of an object** and also gives an option of executing the default value in case of unavailability of the key.

The main difference between \_\_getattr\_\_ and \_\_getattribute\_\_ is that if the attribute was not found by the usual way then \_\_getattr\_\_ is used.

Whereas the \_\_getattribute\_\_ is used before looking at the actual attributes on the object. You will have to use it more consciously otherwise very easily you can end up in infinite recursions.

Q2. What is the difference between properties and descriptors?

Ans:- The Cliff's Notes version: descriptors are a low-level mechanism that lets you hook into an object's attributes being accessed. Properties are **a high-level application** of this; that is, properties are implemented using descriptors.

Descriptors are **a powerful, general purpose protocol**. They are the mechanism behind properties, methods, static methods, class methods, and super() . They are used throughout Python itself.Descriptor methods \_\_set\_\_ , \_\_ get\_\_ , and \_\_delete\_\_ .

Descriptor example:

class Celsius( object ):

def \_\_init\_\_( self, value=0.0 ):

self.value= float(value)

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

return self.value

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

self.value= float(value)

class Farenheit( object ):

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

return instance.celsius \* 9 / 5 + 32

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

instance.celsius= (float(value)-32) \* 5 / 9

class Temperature( object ):

celsius= Celsius()

farenheit= Farenheit()

>>>

oven= Temperature()

>>>

oven.farenheit= 450

>>>

oven.celsius

232.22222222222223

>>>

oven.celsius= 175

>>>

oven.farenheit

347.0

Property example:

class Temperature( object ):

def fget( self ):

return self.celsius \* 9 / 5 + 32

def fset( self, value ):

self.celsius= (float(value)-32) \* 5 / 9

farenheit= property( fget, fset )

def cset( self, value ):

self.cTemp= float(value)

def cget( self ):

return self.cTemp

celsius= property( cget, cset, doc="Celsius temperature" )

>>>

oven= Temperature()

>>>

oven.farenheit= 450

>>>

oven.celsius

232.22222222222223

>>>

oven.celsius= 175

>>>

oven.farenheit

347.0

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

Ans:- \_\_getattr\_\_ method is run for fetches of undefined attribute only ie: those not present on an instance not inherited from any of its classes . code inside it freely fetch other attribute if they are defined.

\_\_getattribute\_\_, method is called for every attribute fetch whether the attribute is defined or not.It uses special code for all such attribute fetches to avoid looping

Properties serves a specific role while Descriptor are more general Properties define a get, set ,delete function for a specific attribute; Descriptor provide a class with methods for these actions, too, but they provide extra flexibility to support more arbitrary actions. In fact, properties are really a simple way to create a specific type of descriptor – one that run functions on attribute access. Property is created with a built-in-function and descriptor is coded with a class. moreover in addition to the instance’s state information, descriptor has local state of their own, so they can avoid name collisions in the instance