**1. Make a class called Thing with no contents and print it. Then, create an object called example from this class and also print it. Are the printed values the same or different?**

Ans-Here's the implementation of the Thing class and creating an instance of it named 'example':

class Thing:

pass

print(Thing())

example = Thing()

print(example)

Here, both the printed values are different, but they represent the same object of class Thing.

**2. Create a new class called Thing2 and add the value 'abc' to the letters class attribute. Letters should be printed.**

Ans-Creating a class called Thing2 and adding the value 'abc' to the letters class attribute:

class Thing2:

letters = 'abc’

print(Thing2.letters)

**3. Make yet another class called, of course, Thing3. This time, assign the value 'xyz' to an instance (object) attribute called letters. Print letters. Do you need to make an object from the class to do this?**

Ans-Here's the implementation of Thing3 class with an instance attribute 'letters' assigned the value 'xyz' and printing it without creating an object:

class Thing3:

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

self.letters = 'xyz'

instance = Thing3()

print(instance.letters) ##output: xyz

To print the letters attribute, we need to create an object (instance) of the Thing3 class. Then we can access the attribute using instance.letters.

**4. Create an Element class with the instance attributes name, symbol, and number. Create a class object with the values 'Hydrogen,' 'H,' and 1.**

Ans-Creating an Element class with the instance attributes name, symbol, and number, and creating a class object with the values 'Hydrogen,' 'H,' and 1:

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

element = Element('Hydrogen', 'H', 1)

print(element.name)

print(element.symbol)

print(element.number)

**5. Make a dictionary with these keys and values: 'name': 'Hydrogen', 'symbol': 'H', 'number': 1. Then, create an object called hydrogen from class Element using this dictionary.**

Ans-Creating a dictionary with keys and values, then creating an object called hydrogen from the Element class using this dictionary:

element\_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}

hydrogen = Element(\*\*element\_dict)

print(hydrogen.name)

print(hydrogen.symbol)

print(hydrogen.number)

**6. For the Element class, define a method called dump() that prints the values of the object’s attributes (name, symbol, and number). Create the hydrogen object from this new definition and use dump() to print its attributes.**

Ans-Defining a method called dump() in the Element class to print the values of its attributes, then creating the hydrogen object and using dump() to print its attributes:

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

def dump(self):

print(f"Name: {self.name}, Symbol: {self.symbol}, Number: {self.number}")

hydrogen = Element('Hydrogen', 'H', 1)

hydrogen.dump()

**7. Call print(hydrogen). In the definition of Element, change the name of method dump to \_\_str\_\_, create a new hydrogen object, and call print(hydrogen) again.**

Ans-When you call print(hydrogen), it internally calls the \_\_str\_\_() method of the object. Let's change the dump() method to \_\_str\_\_() and create a new hydrogen object to demonstrate this:

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

def \_\_str\_\_(self):

return f"Name: {self.name}, Symbol: {self.symbol}, Number:{self.number}"

hydrogen = Element(\*\*element\_dict)

print(hydrogen)

**8. Modify Element to make the attributes name, symbol, and number private. Define a getter property for each to return its value.**

Ans-To make the attributes name, symbol, and number private, we can prefix them with double underscores (\_\_). Then, we can define getter properties for each attribute to retrieve their values:

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.\_\_name = name

self.\_\_symbol = symbol

self.\_\_number = number

@property

def name(self):

return self.\_\_name

@property

def symbol(self):

return self.\_\_symbol

@property

def number(self):

return self.\_\_number

hydrogen = Element(\*\*element\_dict)

print(hydrogen.name) # Output: Hydrogen

print(hydrogen.symbol) # Output: H

print(hydrogen.number) # Output: 1

**9. Define three classes: Bear, Rabbit, and Octothorpe. For each, define only one method: eats(). This should return 'berries' (Bear), 'clover' (Rabbit), or 'campers' (Octothorpe). Create one object from each and print what it eats.**

Ans-Here are the definitions for the classes Bear, Rabbit, and Octothorpe, with the respective eats() method:

class Bear:

def eats(self):

return 'berries'

class Rabbit:

def eats(self):

return 'clover'

class Octothorpe:

def eats(self):

return 'campers'

bear = Bear()

rabbit = Rabbit()

octothorpe = Octothorpe()

print(bear.eats()) # Output: berries

print(rabbit.eats()) # Output: clover

print(octothorpe.eats()) # Output: campers

**10. Define these classes: Laser, Claw, and SmartPhone. Each has only one method: does(). This returns 'disintegrate' (Laser), 'crush' (Claw), or 'ring' (SmartPhone). Then, define the class Robot that has one instance (object) of each of these. Define a does() method for the Robot that prints what its component objects do.**

Ans-Here's an implementation of the classes Laser, Claw, SmartPhone, and Robot:

class Laser:

def does(self):

return 'disintegrate'

class Claw:

def does(self):

return 'crush'

class SmartPhone:

def does(self):

return 'ring'

class Robot:

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

self.laser = Laser()

self.claw = Claw()

self.smartphone = SmartPhone()

def does(self):

print(f"Laser does {self.laser.does()}, Claw does {self.claw.does()}, Smartphone does {self.smartphone.does()}")

robot = Robot()

robot.does()