1. Create a Thing class with no contents and print it. Then, from this class, construct an object called example and print it. Are the written values the same as those on the screen, or are they different?

Ans:- class Thing:  
   pass

Print(Thing)

Output:-<class’\_\_main\_\_.Thing’>

example = Thing()  
print(example)

output:-<\_\_main\_\_Thing object at 0x2b2fd5e06100>

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

Ans:- class Thing2:  
  def \_\_init\_\_(self, letters):  
    self.letters = letters  
      
  
p1 = Thing2(‘abc’)  
  
print(p1.letters)  
output:-abc

1. Create a new class, obviously called Thing3, and assign the value 'xyz' to the letters instance (object) attribute. Letters should be printed. Is it necessary to create an object from the class to accomplish this?

Ans:- class Thing3:  
  def \_\_init\_\_(self, letter):  
    self.letter = letter  
      
  
p1 = Thing3(‘xyz’)  
  
print(p1.letter)

Output:-xyz

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

Ans: class Element:  
   def \_\_init\_\_(self, name, symbol, number):  
     self.name = name

self.symbol = symbol  
   self.number = number  
  
p1 = Element(‘Hydrogen’, ’H’, 1)  
  
print(p1.name)  
print(p1.symbol)

print(p1.number)

1. Build a dictionary using the following keys and values: 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'Hydrogen,' 'H Then, using this dictionary, construct a hydrogen object from the class Element.

Ans:- class Element:

pass

Hydrogen = Element({‘H’:[‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’, ‘Hydrogen’],})  
print(Element)

1. Create a dump() method for the Element type, which prints the values of the object's attributes (name, symbol, and number). Create a hydrogen object from this new description and print its attributes with dump().

Ans:- class Element:  
  def \_\_init\_\_(self, name, symbol, number):  
    self.name = name

self.symbol = symbol  
    self.number = number

 def dump(self):  
    print("Hello my name is " + self.name)  
 print(“Hello my symbol is “ +self.symbol)

Hydrogen = Element(‘Hydrogen’, ’H’, 1)  
Hydrogen.dump()

Output:- Hello my name is Hydrogen  
 Hello my symbol is H

7. Make a print call (hydrogen). Change the name of method dump to \_\_str\_\_ in the Element specification, generate a new hydrogen entity, and call print(hydrogen) again.

class Element:

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

self.name = name

self.symbol = symbol

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

return f'name is {self.name} and symbol is {self.symbol}'

Hydrogen = Element('Hydrogen', H)

print(p.\_\_str\_\_())

Output:

name is Hydrogen and symbol is H

1. Change the name, symbol, and number attributes of Element to private. To return the value of each, define a getter property.

* Ans:- **\_\_init\_\_**:- It is used to initialize the attributes or properties of a **class**.
  + **\_\_a**:- It is a private attribute.
* **get\_a**:- It is used to get the values of private attribute **a**.
* **set\_a**:- It is used to set the value of **a** using an **object** of a class

class Element:  
   def \_\_init\_\_(self, name, symbol, number):  
     self.\_\_name = name

self.\_\_symbol = symbol  
    self.\_\_number = number

def name(self):

return self.\_\_name

def symbol(self):

return self.\_\_symbol

def number(self):

return self.\_\_number

p1 = Element(‘Hydrogen’, ’H’, 1)  
  
p1.name

p1.symbol

p1.number

1. Make three different classes: Bear, Rabbit, and Octothorpe. Define only one form for each: eats (). 'Berries' (Bear), 'clover' (Rabbit), or 'campers' (Rabbit) should be returned (Octothorpe). Make a single item out of each and print out what it eats.

Ans:- class Bear:

def eats(self):

return 'Berries'

class Rabbit:

def eats(self):

return 'clover'

class Octothorpe:

def eats(self):

return 'campers'

b.\_Bear()

r.\_Rabbit()

o.\_Octothrope()

print(b.eats())

berries

print(r.eats())

clover

print(o.eats())

campers

1. Create the Laser, Claw, and SmartPhone classes. There is only one method for each: does (). This gives you the options of 'disintegrate' (Laser), 'crush' (Claw), or 'ring' (Ring) (SmartPhone). Then create a class called Robot with one instance (object) for each of these. Create a does() method for the Robot that prints the actions of its components.

Ans:- 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.lasser =Lasser()

self.claw =Claw()

self.smartphone =SmartPhone()

def does(self):

return ‘’’ I have many attachments:

my lasser, to %s

my claw, to %s

my smartphone,to %s’’’ %(

self.lasser.does()

self.claw.does()

self.smartphone.does()

Robbie = Robot()

Print(Robbie.does())

output: - I have many attachments:

my lasser, to disintegrate,

my claw to crush

my smartphone,to ring