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 :.class Thing:

pass

print(Thing)

<class '\_\_main\_\_.Thing'>

example = Thing()

print(example)

<\_\_main\_\_.Thing object at 0x000002B5D9C00D00>

2)create a new class called thing2 and add the value 'abc' to the letter class attributes .letters should be printed.

Ans :class Thing2:

letters = 'abc'

print(Thing2.letters)

abc

3)make yet another class called of course thing3.the 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 : class Thing3:

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

self.letters = 'xyz'

The variable letters belongs to any objects made from Thing3, not the Thing3 class itself:

print(Thing3.letters)

---------------------------------------------------------------------------

AttributeError Traceback (most recent call last)

<ipython-input-7-6f5d5916809a> in <module>

----> 1 print(Thing3.letters)

AttributeError: type object 'Thing3' has no attribute 'letters'

something = Thing3()

print(something.letters)

xyz

4)create an element class with the instance attributes name,symbol & 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

hydrogen = Element('Hydrogen')

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 : #Starting with the dictionary:

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

#Creating object called hydrogen from class Element using el\_dict

hydrogen = Element(el\_dict['name'], el\_dict['symbol'], el\_dict['number'])

hydrogen.name

'Hydrogen'

We can also initialize the object directly from the dictionary, because its key names match the arguments to "init"

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

hydrogen.name

'Hydrogen'

6)for the element class,define a method called dump()that prints the values of the objects attributes(name,symbol,number).create the hydrogen object from this new definition and use dump() to print its attributes.

Ans :class Element:

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

self.name = name

self.symbol = symbol

self.number = number

def dump(self):

print('name=%s, symbol=%s, number=%s' %(self.name, self.symbol, self.number))

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

hydrogen.dump()

name=Hydrogen, symbol=H, number=1

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 : print(hydrogen)

<\_\_main\_\_.Element object at 0x000002B5DB25E760>

class Element:

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

self.name = name

self.symbol = symbol

self.number = number

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

return ('name=%s, symbol=%s, number=%s' %(self.name, self.symbol, self.number))

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

print(hydrogen)

name=Hydrogen, symbol=H, number=1

8)modify elements to make the attributes name ,symbol,and number private.define a getter property for each to return its value.

Ans : 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('Hydrogen', 'H', 1)

hydrogen.name

'Hydrogen'

hydrogen.symbol

'H'

hydrogen.number

1

9)define the three classes .bear,rabbit,and octothorpe .for each define only one method .eats()this should return 'berries'(bear),'clover'(rabbit),or 'campers'(octothorpe).create a one object from each & print what its eat.

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 = Octothorpe()

print(b.eats())

berries

print(r.eats())

clover

print(o.eats())

campers

10)define a class :laser,claw and smartphone.each has only one method does().this return '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 components object do.

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.laser = Laser()

self.claw = Claw()

self.smartphone = SmartPhone()

def does(self):

return '''I have many attachments:

My laser, to %s.

My claw, to %s.

My smartphone, to %s.''' % (self.laser.does(),self.claw.does(),self.smartphone.does() )

robbie = Robot()

print( robbie.does())

I have many attachments:

My laser, to disintegrate.

My claw, to crush.

My smartphone, to ring.