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)

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

Ans:

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:

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)

\*above statement rises an exception

\*so,

something = Thing3()

print(something.letters)

o/p: xyz

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:

class Element:

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

self.name = name

self.symbol = symbol

self.number = number

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

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

o/p: '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

o/p : 'Hydrogen'

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: 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()

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)

o/p:

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

In [17]:

**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)

o/p:

name=Hydrogen, symbol=H, number=1

8. Modify Element 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

Out[30]:

'Hydrogen'

In [31]:

hydrogen**.**symbol

Out[31]:

'H'

In [32]:

hydrogen**.**number

Out[32]:

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:**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()

In [34]:

print(b**.**eats())

berries

In [36]:

print(r**.**eats())

clover

In [35]:

print(o**.**eats())

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: **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() )

In [39]:

robbie **=** Robot()

In [41]:

print( robbie**.**does())

I have many attachments:

My laser, to disintegrate.

My claw, to crush.

My smartphone, to ring.