**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?**

Here's an example of how you could create a class Thing and an object example from it, and print both:

class Thing:

pass

example = Thing()

print(Thing)

print(example)

The printed values will be different. When you print Thing, you'll see the class definition, which looks like this:

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

And when you print example, you'll see an instance of the class Thing, which looks like this:

<\_\_main\_\_.Thing object at 0x7f68981479b0>

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

Here's an example of how you could create a class Thing2 with a class attribute letters and print its value:

class Thing2:

letters = 'abc'

print(Thing2.letters)

output: abc

**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?**

Yes, you need to create an object from the class in order to access and print the value of an instance attribute. Here's an example of how you could create a class Thing3 with an instance attribute letters and print its value:

class Thing3:

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

self.letters = 'xyz'

example = Thing3()

print(example.letters)

When you run this code, you'll see the following output: 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.**

Here's an example of how you could create an Element class with instance attributes name, symbol, and number, and a class object hydrogen with the values 'Hydrogen', 'H', and 1:

class Element:

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

self.name = name

self.symbol = symbol

self.number = number

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

Now you have an object hydrogen with the instance attributes name, symbol, and number set to 'Hydrogen', 'H', and 1, respectively. You can access and print these values like this:

print(hydrogen.name)

print(hydrogen.symbol)

print(hydrogen.number)

The output will be:

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.**

Here's an example of how you could create a dictionary with the keys and values 'name', 'symbol', and 'number', and use it to create an object hydrogen from the Element class:

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

class Element:

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

self.name = name

self.symbol = symbol

self.number = number

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

Here, the \*\* operator is used to unpack the values of the dictionary element\_data as arguments to the Element constructor. This creates an object hydrogen with the instance attributes name, symbol, and number set to 'Hydrogen', 'H', and 1, respectively.

You can access and print these values in the same way as in the previous example:

print(hydrogen.name)

print(hydrogen.symbol)

print(hydrogen.number)

The output will be:

Hydrogen

H

1

**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.**

Here's an example of how you could define a method dump in the Element class that prints the values of the object's attributes, create the hydrogen object, and use 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}')

print(f'symbol: {self.symbol}')

print(f'number: {self.number}')

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

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

hydrogen.dump()

When you run this code, you'll see the following output:

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.**

Here's an example of how you could call print(hydrogen) before and after changing the name of the method dump to \_\_str\_\_ in the Element class:

class Element:

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

self.name = name

self.symbol = symbol

self.number = number

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

return f'Element: name={self.name}, symbol={self.symbol}, number={self.number}'

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

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

print(hydrogen)

When you run this code, you'll see the following output:

Element: name=Hydrogen, symbol=H, number=1

In this example, the \_\_str\_\_ method is used to provide a string representation of the Element object. When you call print(hydrogen), the \_\_str\_\_ method is automatically called, and its return value is used as the string representation of the object.

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

Here's an example of how you could modify the Element class to make the attributes name, symbol, and number private, and define a getter property for each to return its value:

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

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

return f'Element: name={self.name}, symbol={self.symbol}, number={self.number}'

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

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

print(hydrogen)

In this example, the attributes \_name, \_symbol, and \_number are used to store the values of the name, symbol, and number attributes, respectively. The getter properties name, symbol, and number are defined using the property decorator, and return the values of the corresponding private attributes.

When you run this code, you'll see the same output as in the previous example:

Element: name=Hydrogen, symbol=H, number=1

Note that the use of the underscore prefix in the private attributes is a convention in Python to indicate that they are intended to be private and should not be directly accessed from outside the class. However, this is not enforced by the language, and the attributes can still be accessed if you know their names. The getter properties provide a way to access the values of these attributes in a controlled manner.

**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.**

Here's an example of how you could define the three classes Bear, Rabbit, and Octothorpe, and define the eats method for each:

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(f'The bear eats {bear.eats()}')

print(f'The rabbit eats {rabbit.eats()}')

print(f'The octothorpe eats {octothorpe.eats()}')

When you run this code, you'll see the following output:

The bear eats berries

The rabbit eats clover

The octothorpe 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.**

Here's an example of how you could define the classes Laser, Claw, SmartPhone, and Robot, and define the does method for each:

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'The laser does: {self.laser.does()}')

print(f'The claw does: {self.claw.does()}')

print(f'The smartphone does: {self.smartphone.does()}')

robot = Robot()

robot.does()

When you run this code, you'll see the following output:

The laser does: disintegrate

The claw does: crush

The smartphone does: ring