1a. Describe the difference between class and object. Give an example of a class and an object (in the object-oriented programming sense.

Everything in python is an object. Objects have attributes and methods.

Classes are logical collections of attributes describing a kind of object, and also describe how to make an object of that particular kind.

1b. Define the three key features of object-oriented programming.

-Encapsulation, which is the property of owning data.

-Inheritance, which establishes a hierarchy of relationships between models.

-Polymorphism, which allows models to customize their own behavior.

1c. T/F: All variables, data types and functions in python are classes.

TRUE

1d. T/F It is good coding practice to call dunder methods explicitly.

FALSE

1e. What python built-in is available to show the attributes of an object?

Dir() function.

2. Create a class definition to describe an element. Give it attributes of density, atomic number, and atomic mass. Give it a constructor that requires the atomic number to instantiate the class and a method to print out the state of the object.

class Element:

def \_\_init\_\_(self, density, atomic\_num, atomic\_mass):

self.density = density

self.atomic\_num = atomic\_num

self.atomic\_mass = atomic\_mass

uranium235 = Element(19.1, 92, 235.04)

print(uranium235.density)

print(uranium235.atomic\_num)

print(uranium235.atomic\_mass)

3. How do you overrule duck typing? Give your own example of why you might think you need to do so?

The built-in isinstance() function can overrule duck typing via explicit typing. Isinstance() takes the object that you want to test and the type you want to test against. For instance, having an object that has the same name as a common variable could lead to confusion, and so one can use isinstance() to ensure that they have not mistakenly applied the constant instead of the object.

4. Create a class definition to describe an isotope. Have the class inherit from the element class. Give it attributes of number of neutrons and half-life. Give it a constructor that requires the number of neutrons and half-life to instantiate the class. Give it a method that can print the decay constant (note ensure the method is functional).

import math

class isotope(Element):

def \_\_init\_\_(self,neutrons,half\_life):

self.neutrons = neutrons

self.half\_life = half\_life

def calc\_decay(self):

self.decay = math.log(2)/self.half\_life

uranium235=isotope(235-92,703.8\*10\*\*6)

uranium235.calc\_decay()

print(uranium235.neutrons)

print(uranium235.half\_life)

print(uranium235.decay)

5. What are class decorators? Why would you want to use them?

Class decorators work similar to function decorators. Inside of a class decorator one can add attributes and methods to an existing class, or even return a brand-new class. Note that the decorator is a function.

6. What is one concept you found difficult in the reading?

The reading itself was not very difficult, but I did struggle a bit implementing the class inheritance in my syntax.

COLLABORATION – I checked my work with CPT Drake and Maj Freeman