Name: Keith Thomson

Activity 3: Classes

In this activity, we'll take a first look at object-oriented programming.
Classes provide a means of bundling data and functionality together.

Content Learning Objectives

After completing this activity, students should be able to:

- Write a class definition that has several attributes and methods.
- Explain what a constructor is, when it is called, and what it does.
- Discuss what ``object-oriented'' means using concrete examples.

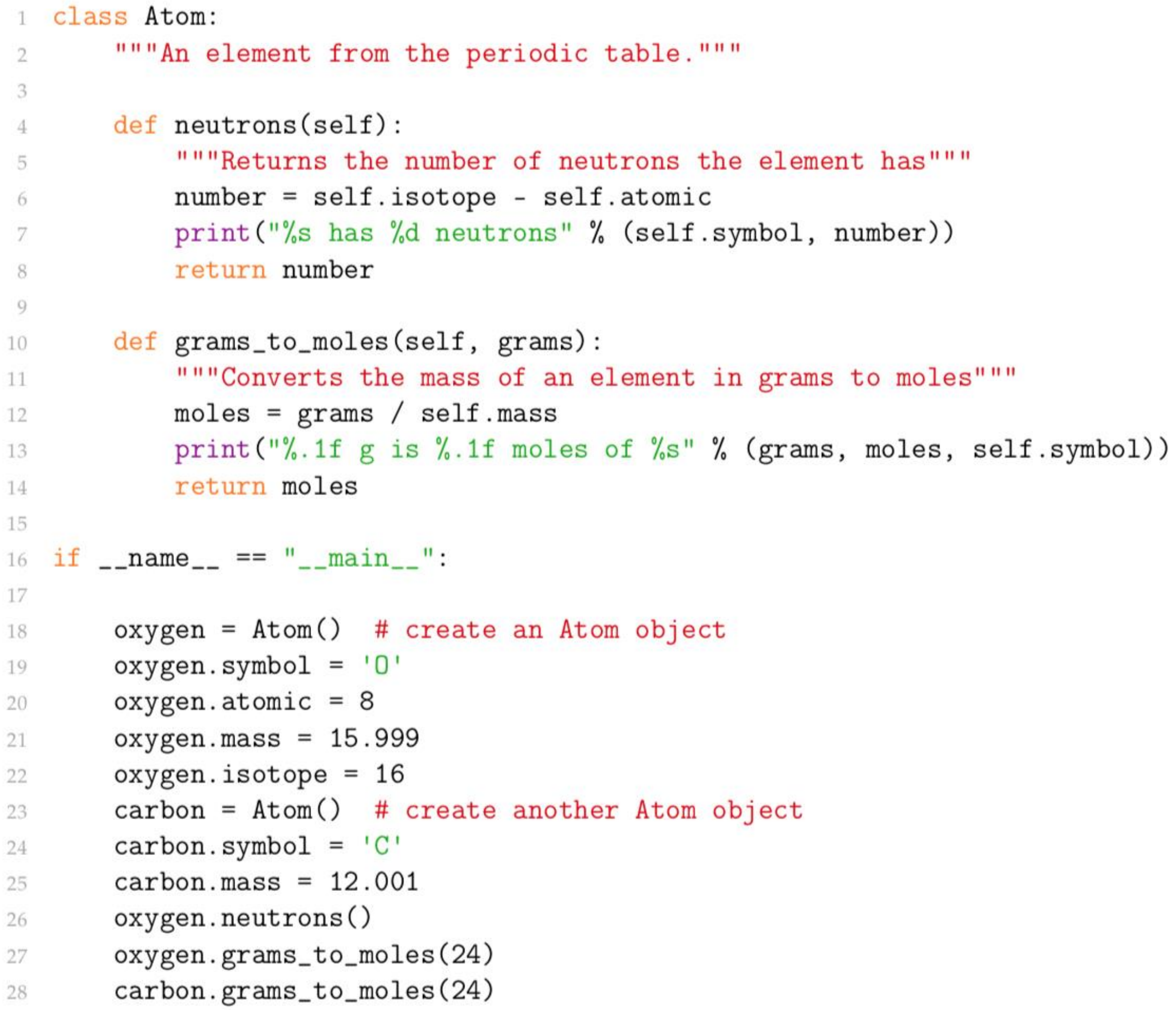
Process Learning Objectives

After completing this activity, students should make progress toward:

- Developing and testing the design of a program incrementally. (Problem Solving)

Model 1 Attributes and Methods

Previously you have used built-in types like `int`, `str`, and `list`. Each of these types comes with its own **methods**, such as `isdigit` and `append`. You can create new types of data, and methods to go with them, by defining a `class`. Classes specify the **attributes** (instance variables) and methods (functions) that each object belonging to the class will have.



```python
class Atom:
    """An element from the periodic table."""

    def neutrons(self):
        """Returns the number of neutrons the element has"""
        number = self.isotope - self.atomic
        print("%s has %d neutrons" % (self.symbol, number))
        return number

    def grams_to_moles(self, grams):
        """Converts the mass of an element in grams to moles"""
        moles = grams / self.mass
        print("%.1f g is %.1f moles of %s" % (grams, moles, self.symbol))
        return moles

if __name__ == "__main__":

    oxygen = Atom()  # create an Atom object
    oxygen.symbol = 'O'
    oxygen.atomic = 8
    oxygen.mass = 15.999
    oxygen.isotope = 16
    carbon = Atom()  # create another Atom object
    carbon.symbol = 'C'
    carbon.mass = 12.001
    oxygen.neutrons()
    oxygen.grams_to_moles(24)
    carbon.grams_to_moles(24)
```

Questions (15 min)

start time:

1. Examine the **class definition** (the top half of the code):

a. What is the name of the class? Atom

b. What are the names of the two methods?
Neutrons, grams _to_moles

c. What is the name of the first parameter for all methods?

self

2. Now examine the `"__main__"` block of code:
   a. How many different `Atom` objects were created? two

   b. Identify the variable name of each object. Oxygen, carbon

   c. How many attributes were assigned in the `oxygen` object? List the names.

      Symbol, atomic, mass, isotope

   d. How does the number of arguments for each method call differ from the number of parameters specified in the method definition?

grams_to_moles takes a parameter

3. How does the syntax referencing an attribute differ inside vs. outside the class definition?

you can utilize _var or __var

4. When the `grams_to_moles` method is called (in the last two lines), what is the value of the self parameter? symbol, O

Enter the expression `type(oxygen)` in a Python Shell. Explain the meaning and significance of the output.

It outputs the type (class) and “main” because we created it in main, I am guessing.

<class '__main__.Atom'>

5. Write code to create a new `Atom` object called `hydrogen`, and assign one of the attributes listed in Question #2c.

```
gold = Atom(24, 'Au', 'Gold', 96.96)
print(gold.symbol)
```

6. Call the `neutrons` method on carbon in a Python Shell. What is the reason for the error?

Carbon wasn’t assigned an atomic number

Model 2 Constructors

For each class defined, you can provide a **constructor** that initializes attributes of a new object. In Python, the constructor is always named `__init__` (with two underscores before and after `init`). The constructor is called automatically when you create a new object.

Add the following constructor to the top of your `Atom` class. By convention, the constructor is typically the first method in a class definition. Also edit the `"__main__"` block of code as shown.

```
class Atom:
    """An element from the periodic table."""

    def __init__(self, symbol, atomic, mass, isotope=12):
        """Constructs an Atom with the given values."""
        self.symbol = symbol
        self.atomic = atomic
        self.mass = mass
        self.isotope = isotope

    ... previous methods from Model 1 ...

if __name__ == "__main__":

    oxygen = Atom('O', 8, 15.999, 16)
    carbon = Atom('C', 6, 12.001)
    oxygen.neutrons()
    carbon.neutrons()
    oxygen.grams_to_moles(24)
    carbon.grams_to_moles(24)
```

Questions (15 min)

start time:

7. What is always the name of the constructor?

__init__

8. Although there is no direct call to the constructor, explain how you know this method is executed when an object is created.

   Because if not I would’ve had to assign all the variables manually instead of giving them as arguments.

9. Consider your answer to Question #7. What is one advantage of defining a constructor for a class?
   It automatically instantiates the class variables; code readability; encapsulation.

10. In a Python Shell, try to create a new `Atom` object called `hydrogen` with only two arguments. Write your statement in the space below. What is the reason for the error you see?
    The error is because I didn’t include all the arguments.

11. When creating an object of the `Atom` class, what is the value of `isotope` if:
    a. four arguments are given?

14

    b. three arguments are given?
    And error

12. Print the value of `self.isotope` in a Python shell.
    a. What is the reason for the error?

Self is not defined

    b. In order to eliminate this error, what should be printed instead?

Oxygen.isotope will output the isotope provided as an argument.

13. Recall that a variable may be ``local'' (defined within a function), ``global'' (defined in the non-indented or `"__main__"` block of code), or ``built-in'' (part of Python itself).
    a. Explain why the `isotope` attribute is not a global variable.

it’s not defined in main

    b. Explain why the `isotope` attribute is not a local variable.

it’s not within a specific function. It's defined in the class so it can be used to make other classes.

    c. How is each method of the class able to access the `isotope` attribute?

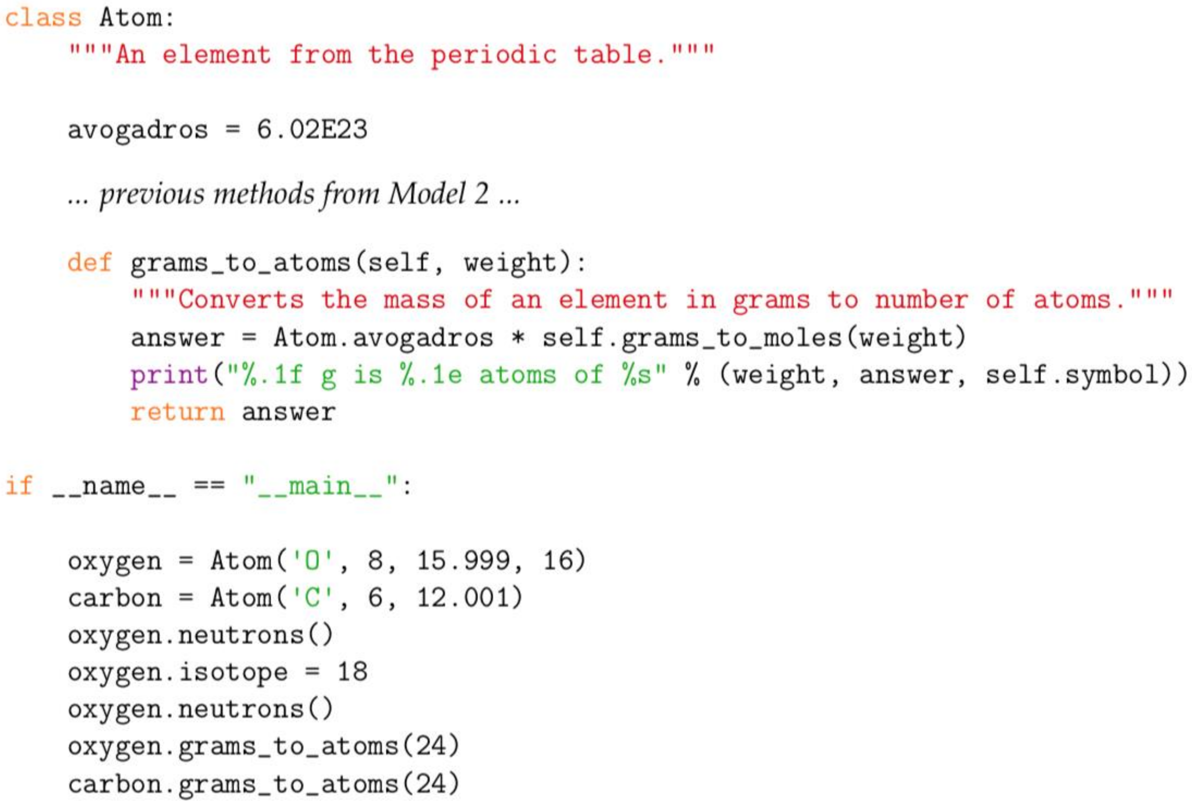
By explicitly setting it like gold = Atom(13, ‘Au’, ‘Gold’, 34, 10) 10 being the isotope.

Then calling it like gold.isotope

Default is isotope=14

Model 3 Object-Oriented

Edit the `Atom` class further to include the variable `avogadros`, the method `grams_to_atoms`, and the modified `"__main__"` block of code. Note that **class variables** (like `avogadros`) are typically defined before the `__init__` method.



```
class Atom:
    """An element from the periodic table."""

    avogadros = 6.02E23

    ... previous methods from Model 2 ...

    def grams_to_atoms(self, weight):
        """Converts the mass of an element in grams to number of atoms."""
        answer = Atom.avogadros * self.grams_to_moles(weight)
        print("%.1f g is %.1e atoms of %s" % (weight, answer, self.symbol))
        return answer

if __name__ == "__main__":

    oxygen = Atom('O', 8, 15.999, 16)
    carbon = Atom('C', 6, 12.001)
    oxygen.neutrons()
    oxygen.isotope = 18
    oxygen.neutrons()
    oxygen.grams_to_atoms(24)
    carbon.grams_to_atoms(24)
```

Questions (15 min)

start time:

14. Examine the `grams_to_moles` method (from Model 1):
    a. Identify the three main variables used in `grams_to_moles`:

Avogadros, weight, answer

    b. For each variable, what is its scope? (local or global)

Avogadros is global, weight is local, answer is local.

15. What determines whether a variable is defined as an attribute or a local variable?

Whether you need it to be persistent and how you want them to be accessed. Attributes can be set with self.attr = attr

16. Now examine the `grams_to_atoms` method (from Model 3).
    a. What variable was initialized in the `Atom` class outside the constructor and methods? avogrados

    b. How does the syntax of a class variable differ from an instance variable?
    Class variables don't need to be instantiated with self. Definition and accessibility

17. Would it be possible to rewrite the `grams_to_atoms` method as a function instead? If so, explain how the function would differ.

The difference is that outside the class grams_to_moles needs an atom instance as an argument def grams_to_m(atom, grams): is how i did it.

18. How would you rewrite the line `oxygen.grams_to_atoms(24)` to call the function defined in the previous question?

Def grams_to_m(atom, grams) So I don't need 'self' in there.

19. Consider the built-in `str` class:
    a. Given the statement `s = "Hello"`, what data is stored in the `str` object?

A string

    b. Show an example line of code that calls the `upper` method on the object `s`.

s.upper()

    c. If the `upper` method were defined as a global function instead, how would you call it?

upper()

20. Based on the previous two questions, explain what the term ``object-oriented'' means.

It means creating efficient, reusable data structures that incorporate OOP principles like encapsulation, inheritance, polymorphism, composition. You can manipulate data as objects with methods or use variables and functions without classes and have less readable, reusable and efficient code.

21. Summarize the advantages you perceive for writing code as methods in classes in

comparison to functions.

It can be much more complex and therefore it can be extensive and increases readability as well. I look at it like grouping data and behavior together. I think OOP has many use cases that wouldn’t work as well with standard functions with parameters and no classes.