Defining Classes

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

Manager:		Recorder:	
Presenter:		Reflector:	

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 Skill Goals

During the activity, students should make progress toward:

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

Facilitation Notes

NOTE: This activity was designed with an "objects late" approach in mind. It might not be suitable for an "objects early" course, given the order in which the example class is presented.

The source code for Model 1 should be provided to students (see *atom.py*). When reporting out, focus the discussion on Questions #5 and #7. It may be helpful to step through the code with a debugger. Students with prior experience might wonder why Line 19 doesn't raise an error (since the attribute wasn't "defined" in a constructor).

Model 2 solves the problem introduced in Question #7, namely that different objects might have different attributes. Don't let students get bogged down with the chemistry examples; it's not essential that they understand the details. (An isotope is one of two or more atoms with the same atomic number but with different numbers of neutrons.)

By Model 3, students should have a complete version of the `Atom` class implemented with all four methods. You may need to guide the discussion and report out both during the middle and the end of these questions. Give multiple teams the opportunity to explain object-oriented programming in their own words.

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.

```
1  class Atom:
2      """An element from the periodic table."""
3
4      def neutrons(self):
5          """Returns the number of neutrons the element has"""
6          number = self.isotope - self.atomic
7          print("{} has {} neutrons".format(self.symbol, number))
8          return number
9
10     def grams_to_moles(self, grams):
11         """Converts the mass of an element in grams to moles"""
12         moles = grams / self.mass
13         print("{.1f} g is {.1f} moles of {}".format(grams, moles, self.symbol))
14         return moles
15
16 if __name__ == "__main__":
17
18     oxygen = Atom()  # create an Atom object
19     oxygen.symbol = 'O'
20     oxygen.atomic = 8
21     oxygen.mass = 15.999
22     oxygen.isotope = 16
23     carbon = Atom()  # create another Atom object
24     carbon.symbol = 'C'
25     carbon.mass = 12.001
26     oxygen.neutrons()
27     oxygen.grams_to_moles(24)
28     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 and 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 and carbon

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

4: symbol, atomic, mass, isotope

d) How do the number of arguments for each method call differ from the number of parameters specified in the method definition? one less

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

inside class: self.identifier vs. outside class: object.identifier

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

In this example, oxygen or carbon (depending on which object was before the method call).

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

The output is `<class '__main__.Atom'>`, meaning that `oxygen` belongs to the `Atom` class from the `__main__` module. It's significant because we are the ones who defined this data type.

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

```
hydrogen = Atom()
hydrogen.symbol = 'H'
```

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

The attributes `self.symbol`, `self.isotope`, and `self.atomic` have not yet been assigned.

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 the word `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:

8. What is always the name of the constructor?

`__init__`

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

The instance variables have a value when used in the methods `neutrons` and `grams_to_moles`. There is no error message this time.

10. Consider your answer to Question #7. What is one advantage of defining a constructor for a class?

It automatically initializes the attributes so that methods don't error if they aren't called in the correct order.

11. 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?

`hydrogen = Atom('H', 1)` Based on the definition of `__init__`, at least three parameters are required to define an `Atom` object.

12. When creating an object of the `Atom` class, what is the value of `isotope` if:

a) three arguments are given? 12

b) four arguments are given? the value of the last argument

13. Print the value of `self.isotope` in a Python shell.

a) What is the reason for the error? NameError: 'self' is not defined

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

14. For each line below, what is the value of `self`?

a) oxygen = Atom('O', 8, 15.999, 16) The object that will be assigned to `oxygen`

b) carbon = Atom('C', 6, 12.001) The object that will be assigned to `carbon`

c) oxygen.neutrons() The object currently referenced by `oxygen`

d) carbon.neutrons() The object currently referenced by `carbon`

15. 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 defined in the `__init__` function (not `__main__`)

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

it's used outside the constructor in other methods

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

Through the `self` parameter, which is a local variable in each method.

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 {}".format(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:**

16. Examine the `grams_to_moles` method (from Model 1):

a) Identify the three main variables used in `grams_to_moles`:

self, grams, moles

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

local, local, local

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

The `self` parameter defines (stores) all the attributes. Local variables, like `moles`, are defined in methods directly.

18. Now examine the `grams_to_atoms` method (from Model 3).

a) What variable was initialized in the `Atom` class outside the constructor and methods?

Atom.avogadros

b) How does the syntax of a class variable differ from an attribute (instance variable)?

It uses the class name in the identifier rather than `self` or the object name.

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

Yes; simply define the function outside of the class. Rename the parameter `self` to `atom` to avoid confusion.

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

```
grams_to_atoms(oxygen, 24)
```

21. Consider the built-in `str` class:

a) Given the statement `s = "Hello"`, what data is stored in the `str` object?

The word Hello.

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

```
print(s.upper())
```

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

```
print(upper(s))
```

22. Based on the previous two questions, explain what the term "object-oriented" means.

It means that methods are built into objects, as opposed to being standalone functions. We call those methods through the objects.

23. Summarize the advantages you perceive for writing code as methods in classes instead of global functions.

Either way, we have to pass the object to the function. It's easier to think about the object first, and then call methods that take the remaining parameters.