Administrivia

Checkoff 6 due today
Office Hours closing at 4 because snow
Class will likely be cancelled tomorrow due to blizzard, get some work done

Review Classes

Attributes and Methods

Special Methods
- __init__ in particular (Show optional argument syntax)

“self” - demonstrate two objects and “self” keyword in each case (inefficient_dict.py)

Inheritance

A way for a class to “inherit” (start with) the methods and attributes of another, and then overwrite or add functionality

Parent class / Super class vs. Child class / Sub class

Child class is a “specific type” of Parent class

Technically don’t need inheritance (show inheritance implemented with attributes), but it is a natural way of expressing things that helps

Different Methods

Sub classes could use some common methods from the Parent class and then implement different methods for the different things they can do

Same Methods

Sub classes could use some common methods from the Parent class, and then define different implementations for the “same” method. This suggests that different subclasses can do the “same thing”, but might do them differently.

Often the parent class is “Abstract”, does not define one or more vital methods and isn’t meant to be used directly. Instead it is subclassed, and the subclass defines the missing method(s)

Attribute vs. Inheritance

If most things that are done with inheritance could be easily done by storing an instance of the parent as an attribute, when is it appropriate to do one or the other?

Attributes are used when one class "**has**" another class, as a tool or otherwise. e.g. a School **has** Students, in a video game a Level **has** players, a Car **has** an Engine

Inheritance is used when one class "**is**" another class. Typically, the sub class is a special case of the parent class. e.g. a High School Student **is** a Student (a College Student might be another), in a video game, a character **is** a player or a computer controlled character, a Ferrari **is** a Sports Car, which **is** a Car

Note that the hiearchy can stretch back indefinitely. How far it needs to be stretched depends on what we're modelling

```
class A:
    def __init__(self):
        pass
    def my_method(self):
        return "Method Called"

class B(A):
    def my_other_method(self):
        return "Other Method Called"

class C:
    def __init__(self):
        self.a = A()
    def my_method(self):
        return self.a.my_method()
    def my_other_method(self):
        return "Other Method Called"
```

```
import math

class Shape(object):
    def __init__(self, x, y, z):
        self.center = (x, y, z)

class Shape2D(Shape):
    def __init__(self, x, y):
        Shape.__init__(self, x, y, 0)
    def perimeter(self):
        raise NotImplementedError
    def area(self):
        raise NotImplementedError

class Shape3D(Shape):
    def surface_area(self):
        raise NotImplementedError
    def volume(self):
        raise NotImplementedError

class Rectangle(Shape2D):
    def __init__(self, x, y, height, width):
        Shape2D.__init__(self, x, y)
        self.height = height
        self.width = width
    def perimeter(self):
        return 2*self.width + 2*self.height
    def area(self):
        return self.width*self.height

class Square(Rectangle):
    def __init__(self, x, y, length):
        Rectangle.__init__(self, x, y, length, length)

class Circle(Shape2D):
    def __init__(self, x, y, radius):
        Shape2D.__init__(self, x, y)
        self.radius = radius
    def perimeter(self):
        return 2*math.pi*self.radius
    def area(self):
        return math.pi*(self.radius)**2
```

```
class RectangularPrism(Shape3D):
    def __init__(self, x, y, z, length_x, width_y, height_z):
        Shape3D.__init__(self, x, y, z)
        self.length = length_x
        self.width = width_y
        self.height = height_z
    def surface_area(self):
        return 2*self.length*self.width + 2*self.length*self.height + 2*self.width*self.height
    def volume(self):
        return self.length*self.width*self.height

class Cube(RectangularPrism):
    def __init__(self, x, y, z, length):
        RectangularPrism.__init__(self, x, y, z, length, length, length)

class Sphere(Shape3D):
    def __init__(self, x, y, z, radius):
        Shape3D.__init__(self, x, y, z)
        self.radius = radius
    def surface_area(self):
        return (4.0/3.0)*math.pi*self.radius**2
    def volume(self):
        return 4*math.pi*self.radius**3
```