CSC 110: Week 8, Lecture 2

Dr. Bodine

Last time…

- Objects as active data type
  - Information
  - How the information can be manipulated
- Object-oriented programming
  - Programmer defines not only the structure of data that is to be stored but also what operations can be done on the data
- Classes
  - Python class definitions
  - Objects as instances of a class
  - Relationship to functions
  - Relationship to what we already know/do

This time…

- Brief review of objects
- Continue with classes
  - Define our own classes
  - Work on understanding how classes are helpful
- Abstraction and Encapsulation

Objects, OOP, and Classes, oh my!

- Objects: active data types containing information and the methods you can use to manipulate the information
- OOP: programmer defines the structure of the information to be stored as well as the methods for manipulating that information
- Classes: a way to define the objects (information to be stored and how to manipulate the information)

Object attributes

- Instance variables
  - Information to be stored
  - Example, student names, student scores, length, height, position, …

- Methods
  - Ways of accessing the information
  - Ways of manipulating the information

Classes in python

```
class <classname>(<what it inherits from>):
        <statements, like function definitions>
```

- Remember out class should have an __init__ function/method to construct objects of the class
  - This is called during instantiation, creating a new object
- To access the data, functions/methods need a “self” parameter
  - Example: __init__(self)
  - This allows them to access/update the instance variables

Continuing with our student class example

- Want to store the student name (first and last) and score together
- Want to be able to access the information in a helpful way
- Want to be able to grade the students

Making our student class

- Need to know what information we want to store
  - Instance variables
  - First Name, Last Name, Score

- Need to know what we want to do with the information
  - Methods (functions)
  - Need methods to access the value(s)
  - Assign grades based on the value(s)

- Remember we need to define an __init__ function!

Our student class

```
class student():

    def __init__(self,Firstname,Lastname,Score):
        self.firstname=Firstname
        self.lastname=Lastname
        self.score=float(score)

    def getname(self):
        return self.firstname+" "+self.lastname

    def getscore(self):
        return self.score

    def grade(self):
        if self.score>=90: sgrade = 'A'
        elif self.score>=80: sgrade = 'B'
        elif self.score>=70: sgrade = 'C'
        elif self.score>=60: sgrade = 'D'
        else: sgrade = F
        return sgrade
```

Now what do we do with the student object?

- We might want to store the information so we want do things with them like compute averages

- Can we just throw these students into a list?
  - [student_1, student_2, … student_n]

Abstraction

- Before we stored everything in lists or a list of lists and now we are storing student object s in a list

- What does this all mean? Why does this "help" us?

- Store complex data in a simpler looking (easier to comprehend way)
  - Hides the fact that this student object is more complex than just a number
  - In future code I don't have to care how the grading is done or skip past the code. I can just ask for mystudent.grade() and get the grade back.

Let's write our own class!

- Write a class to represent the geometric solid sphere with the following methods:
  - `__init__(self, radius)` Creates a sphere having the given radius
  - `getRadius(self)` Returns the radius of the sphere
  - `surfaceArea(self)` Returns the surface area of the sphere
  - `volume(self)` Returns the volume of the sphere

```python
import math  #access  to math lib


# sphere class

class Sphere:

    def __init__(self,radius):
        self.Radius=radius

    def getRadius(self):
        return self.Radius

    def surfaceArea(self):
        return 4*math.pi*(self.Radius**2)

    def volume(self):
        return (4/3)*math.pi*(self.Radius**3)
```

```python
import math  #access  to math lib


# sphere class

class Sphere:

    def __init__(self,radius):
        self.Radius=radius

    def getRadius(self):
        return self.Radius

    def surfaceArea(self):
        return 4*math.pi*(self.Radius**2)

    def volume(self):
        return (4/3)*math.pi*(self.Radius**3)
```

```
>>> mysphere = Sphere(4)
>>> mysphere = Sphere(1)
>>> mysphere.getRadius()
1
>>> mysphere.surfaceArea()
12.566370614359172
>>> mysphere.volume()
4.1887902047863905
>>> mysphere.Radius
1
```

```python
import math  #access  to math lib


# sphere class

class Sphere:

    def __init__(self,radius):
        self.Radius=radius

    def getRadius(self):
        return self.Radius

    def surfaceArea(self):
        return 4*math.pi*(self.Radius**2)

    def volume(self):
        return (4/3)*math.pi*(self.Radius**3)
```

```
>>> mysphere = Sphere(4)
>>> mysphere = Sphere(1)
>>> mysphere.getRadius()
1
>>> mysphere.surfaceArea()
12.566370614359172
>>> mysphere.volume()
4.1887902047863905
>>> mysphere.Radius
1
>>> mysphere.radius
Traceback (most recent call last):
  File "<pyshell#344>", line 1, in <module>
    mysphere.radius
AttributeError: 'Sphere' object has no attribute 'radius'
>>> radius
Traceback (most recent call last):
  File "<pyshell#345>", line 1, in <module>
    radius
NameError: name 'radius' is not defined
>>>
```

A closer look…

```
import math  #access  to math lib

# sphere class

class Sphere:

    def __init__(self,radius):
        self.Radius=radius

    def getRadius(self):
        return self.Radius

    def surfaceArea(self):
        return 4*math.pi*(self.Radius**2)

    def volume(self):
        return (4/3)*math.pi*(self.Radius**3)
```

- __init__ function
  - parameters
- Instance variables
- Methods

Let's write our own class!

- Write an Account class
  - At instantiation the object should be created with just a name associated with the account (i.e. no balance variable set)
  - Should have an instance variable associated with the balance that can be set with setBalance method
  - Should have methods to getBalance, withdraw, and deposit

```
class Accounts(object):

    def __init__(self,name):
        self.name=name

    def setBalance(self,amount):
        self.balance=amount

    def deposit(self,amount):
        self.balance = self.balance + amount

    def withdraw(self,amount):
        self.balance = self.balance - amount

    def getBalance(self):
        return self.balance
```

```
>>> test = Accounts("checking")
>>> test.setBalance(1000)
>>> test.getBalance()
1000
>>>
>>>
>>> test.withdraw(500)
>>> test.getBalance()
500
>>>
```

Encapsulation

- Our main program only worries about what objects are and what they can do not how it is done *

- Implementation details are *encapsulated* in the class definition
  - They are not directly accessible to code outside the class defintion
  - That is, outside code should not interact with class variables

- Complete separation not enforced in all languages
  - For example, it is not enforced in python! But it is a convention!

*That being said, more often than not it is important to have a good understanding of how things are done in order to to use them correctly

Encapsulation, continued…

- References to instance variables should only be used inside the class definition
- Outside code should use methods to access the object data
- Is this really fundamentally different?
  - Yes and no…
  - Abstraction, coding standards, conventions

Encapsulation example

- Start with a student class that contains a variable called score
- Say we have a program that instantiates this class by creating an object mystudent (thereby assigning a score)
- If we want to access mystudent's score in our larger program how should we do that?
  - mystudent.score vs mystudent.getscore()
  - Best practice is to have a method getscore that returns the score

Classes that are related to each other…

- Remember we don’t want to duplicate code
- Sometimes we have an overarching type of object with subtypes
  - Example: circles and rectangles are two-dimensional shapes
- Sub-classing and inheritance allow us to avoid duplicating code.

Start with two completely separate classes…

```
class Circle:

    def __init__(self, radius):
        self.Radius = radius

    def getRadius(self):
        return self.Radius

    def area(self):
        return math.pi*(self.Radius**2)

    def circumference(self):
        return 2*math.pi*self.Radius
```

```
class Rectangle:

    def __init__(self, length, width):
        self.Length = length
        self.Width =  width

    def getLength(self):
        return self.Length

    def getWidth(self):
        return self.Width

    def area(self):
        return self.Length*self.Width

    def perimeter(self):
        return 2*self.Length+2*self.Width
```

Now, add a position to them…

```
class Circle:

    def __init__(self, xi, yi, radius):
        self.Radius = radius
        self.x = xi
        self.y = yi

    def getRadius(self):
        return self.Radius

    def area(self):
        return math.pi*(self.Radius**2)

    def circumference(self):
        return 2*math.pi*self.Radius
```

Now, add a position to them…

```
class Circle:

    def __init__(self, xi, yi, radius):
        self.Radius = radius
        self.x = xi
        self.y = yi

    def getRadius(self):
        return self.Radius

    def area(self):
        return math.pi*(self.Radius**2)

    def circumference(self):
        return 2*math.pi*self.Radius
```

```
class Rectangle:

    def __init__(self, xi, yi, length, width):
        self.Length = length
        self.Width =  width
        self.x = xi
        self.y = yi

    def getLength(self):
        return self.Length

    def getWidth(self):
        return self.Width

    def area(self):
        return self.Length*self.Width

    def perimeter(self):
        return 2*self.Length+2*self.Width
```

Make 2 new classes that have overlap

```
class Circle:

    def __init__(self, xi, yi, radius):
        self.Radius = radius
        self.x = xi
        self.y = yi

    def getRadius(self):
        return self.Radius

    def area(self):
        return math.pi*(self.Radius**2)

    def circumference(self):
        return 2*math.pi*self.Radius
```

REPEATED CODE!

```
class Rectangle:

    def __init__(self, xi, yi, length, width):
        self.Length = length
        self.Width =  width
        self.x = xi
        self.y = yi

    def getLength(self):
        return self.Length

    def getWidth(self):
        return self.Width

    def area(self):
        return self.Length*self.Width

    def perimeter(self):
        return 2*self.Length+2*self.Width
```

What if I wanted code to getx and gety?
That would be repeated as well!

Let’s take the overlap and put it in a new class

```
class Shape:

    def __init__(self, xi, yi):
        self.x = xi
        self.y = yi

    def moveto(self,xnew,ynew):
        self.x = xnew
        self.y = ynew

    def getx(self):
        return self.x

    def gety(self):
        return self.y

    def getcenter(self):
        return (self.x,self.y)
```

Let’s take the overlap and put it in a new class

```
class Shape:

    def __init__(self, xi, yi):
        self.x = xi
        self.y = yi

    def moveto(self,xnew,ynew):
        self.x = xnew
        self.y = ynew

    def getx(self):
        return self.x

    def gety(self):
        return self.y

    def getcenter(self):
        return (self.x,self.y)
```

```
class Circle(Shape):

    def __init__(self, xi, yi, radius):
        self.Radius = radius
        Shaped2d.__init__(self,xi,yi)

    def getRadius(self):
        return self.Radius

    def area(self):
        return math.pi*(self.Radius**2)

    def circumference(self):
        return 2*math.pi*self.Radius
```

Inheritance…

```
class Shape:

    def __init__(self, xi, yi):
        self.x = xi
        self.y = yi

    def moveto(self,xnew,ynew):
        self.x = xnew
        self.y = ynew

    def getx(self):
        return self.x

    def gety(self):
        return self.y

    def getcenter(self):
        return (self.x,self.y)
```

```
>>> mycircle = Circle(0,0,2)
>>> mycircle.getcenter()
(0, 0)
>>>
>>> mycircle.moveto(1,1)
>>> mycircle.getcenter()
(1, 1)
```

Inheritance…

```
class Shape:

    def __init__(self, xi, yi):
        self.x = xi
        self.y = yi

    def moveto(self,xnew,ynew):
        self.x = xnew
        self.y = ynew

    def getx(self):
        return self.x

    def gety(self):
        return self.y

    def getcenter(self):
        return (self.x,self.y)
```

- Circle inherits from Shaped
- Can access the moveto method
- Can access getx, gety, etc…
- Has all the same…and more!

Object-oriented programming

- Objects are active data types
  - Information AND how to handle the information
  - Everything is an object in python

- Object-Oriented Programming
  - Programmer defines the structure of information and how it will be handled
  - Abstraction and encapsulation

- Classes
  - Way of defining objects
  - Inheritance