

Python Classes and Objects

A Basic Introduction

Coming up: Topics

Topics

- Objects and Classes
- Abstraction
- Encapsulation
- Messages

What are objects

- An object is a datatype that stores data, but ALSO has operations defined to act on the data. It knows stuff and can do stuff.
- Generally represent:
 - tangible entities (e.g., student, airline ticket, etc.)
 - intangible entities (e.g., data stream)
- Interactions between objects define the system operation (through message passing)

What are Objects

- A Circle drawn on the screen:
- Has **attributes** (knows stuff):
 - radius, center, color
- Has **methods** (can do stuff):
 - move
 - change color

Design of Circle object

- A `Circle` object:
 - `center`, which remembers the center point of the circle,
 - `radius`, which stores the length of the circle's radius.
 - `color`, which stores the color
- The `draw` method examines the `center` and `radius` to decide which pixels in a window should be colored.
- The `move` method sets the center to another location, and redraws the circle

Design of Circle

- All objects are said to be an *instance* of some *class*. The class of an object determines which attributes the object will have.
- A class is a description of what its instances will know and do.

Circle: classes and objects

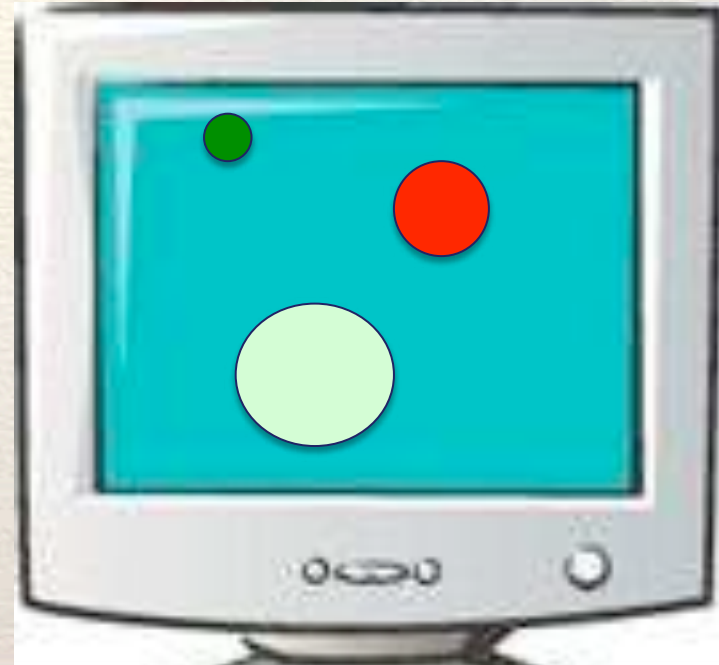
Classes are blueprints or directions on how to create an object

Circle Class

Attributes:
-location, radius,color

Methods:
- draw, move

Objects are instantiations of the class (attributes are set)



3 circle objects are shown (each has different attribute values)

Circle Class

```
class Circle(object):
```

Beginning of the class definition

```
    def __init__(self, center, radius):  
        self.center = center  
        self.radius = radius
```

The constructor. This is called when someone creates a new Circle, these assignments create attributes.

```
    def draw(self, canvas):  
        rad = self.radius  
        x1 = self.center[0]-rad  
        y1 = self.center[1]-rad  
        x2 = self.center[0]+rad  
        y2 = self.center[1]+rad  
        canvas.create_oval(x1, y1, x2, y2, fill='green')
```

A method that uses attributes to draw the circle

```
    def move(self, x, y):  
        self.center = [x, y]
```

A method that sets the center to a new location and then redraws it

objects/CircleModule.py

Constructors

- The object's constructor method is named `__init__`
- The primary duty of the constructor is to set the **state** of the object's attributes (instance variables)
- Constructors may have default parameters
- Calling an object's constructor (via the class name) is a signal to the interpreter to create (instantiate) a new object of the data type of the class
 - `myCircle = Circle([10,30], 20)` # Never pass "self", it's automatic

Creating a Circle

```
myCircle = Circle([10,30], 20)
```

- This statement creates a new `Circle` object and stores a reference to it in the variable `myCircle`.
- The parameters to the constructor are used to initialize some of the instance variables (`center` and `radius`) inside `myCircle`.

Creating a Circle

```
myCircle = Circle([10,30], 20)
```

- Once the object has been created, it can be manipulated by calling on its methods:

```
myCircle.draw(canvas)
```

```
myCircle.move(x, y)
```


Objects and Classes

- `myCircle = Circle([10,30], 20)`
- `myOtherCircle = Circle([4,60], 10)`
- `myCircle` and `myOtherCircle` are objects or instances of the Class `Circle`
- The circle class defines what a circle knows (attributes) and what it does (methods)... but to have a circle, you need to construct an object from that class definition
- Similar to a “list”. Python defines what a list is, and can do (slicing, indexing, `length(...)`, etc... but until you create one, you don’t really have one

Using the Circle

- from CircleModule import *

```
myCircle = Circle([10,30], 20)
print
    "CENTER :"+str(myCircle.center)
>>> CENTER : (10, 30)
```

To get an instance variable from an object, use: <<object>>.variable

What happens if the instance variable doesn't exist?

Using Instance Variables

```
myCircle = Circle([10,30], 20)
print "CENTER :"+str(circle.carl)
>>> AttributeError: Circle
      instance has no attribute
      'carl'
```


Using Instance Variables

```
myCircle.bob = 234
```

What happens if you set an instance variable that doesn't exist?

Think: What happens if you assign ANY variable in python that doesn't exist?

```
john = 234
```

Python automatically creates a new variable if it doesn't exist. For instance variables this works the same... if you assign an instance variable that doesn't exist, Python just creates it...

Bad practice though... create all instance variables in the constructor!

Summary: Using instance variables

- Creating new instance variables just means assigning them a value:
 - `self.bob = 234` # In constructor
- Using instance variables is done through dot notation:
 - `val = myCircle.bob` # Outside the class definition
 - `val = self.bob` # Inside class methods

Attributes / Instance Variables

- Attributes represent the characteristics of a class. When an object is instantiated and the values are assigned to attributes, they are then referred to as instance variables.
- The values of the instance variables define the **state** of the individual object
- They are referred to as instance variables because the values assigned to an individual object (instance of a class) are unique to that particular class
- Attributes may be public or private (although due to their specific implementation, they are not truly private in Python)
- If the attributes are private, they serve to enforce the concept of **information hiding**

Using methods in Objects

- Methods are created just like a function, but inside a class:

class Circle:

```
def myFunction(self, p1, p2):  
    << something >>>  
def function2(self, input1='55'):  
    <<something>>
```

- To use methods, call them using dot notation:

```
myCircle.myFunction(actualP1, actualP2)
```

Note: self is automatically passed in to all methods... you never pass it in directly!



Messages

- Process by which system components interact:
 - send data to another object
 - request data from another object
 - request object to perform some behavior
- Implemented as methods (not called functions).
 - Functions are processes that are object independent
 - Methods are dependent on the state of the object

Message Passing

- When calling a method in another class, OO uses the term “message passing” you are passing messages from one class to another
- Don't be confused... this is really just a new name for calling a method or a function

What is 'self'

- Self is a reference to the current instance. Self lets you access all the instance variables for the specific instance you're working with.
 - `myCircle.myFunction(actualP1, actualP2)` 
- is like calling:
 - `Circle.myFunction(myCircle, actualP1, actualP2)` 
- "self" really gets the value of "myCircle".. but it happens automatically!

Why use classes at all?

- Classes and objects are more like the real world. They minimize the semantic gap by modeling the real world more closely
- The semantic gap is the difference between the real world and the representation in a computer.

-

Do you care how your TV works?

- No... you are a user of the TV, the TV has operations and they work. You don't care how.

Why use classes at all?

- Classes and objects allow you to define an interface to some object (it's operations) and then use them without know the internals.
- Defining classes helps modularize your program into multiple objects that work together, that each have a defined purpose

Encapsulation

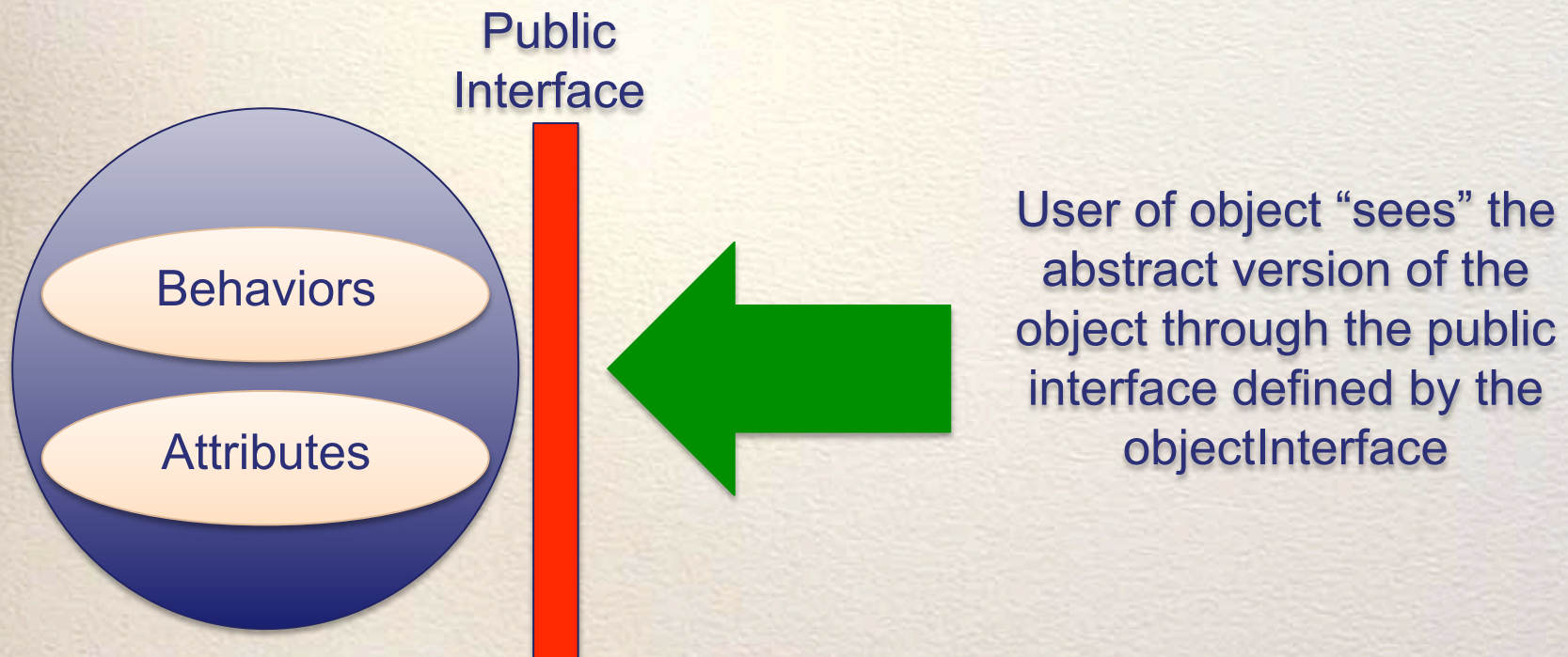
- **Attributes** and **behaviors** are enclosed (encapsulated) within the logical boundary of the object entity
 - In structured or procedural systems, data and code are typically maintained as separate entities (e.g., modules and data files)
 - In Object Technology systems, each object contains the data (attributes) and the code (behaviors) that operates upon those attributes

Abstraction

- Encapsulation implements the concept of **abstraction**:
 - details associated with object sub-components are enclosed within the logical boundary of the object
 - user of object only “sees” the public interface of the object, all the internal details are hidden

Note - In Python, encapsulation is merely a programming convention. Other languages (e.g., Java) enforce the concept more rigorously.

Abstraction



Encapsulation makes abstraction possible

Abstraction in your life



You know the public interface. Do you know implementation details?
Do you care?

As long as the public interface stays the same, you don't care about implementation changes

Implementing Public/Private Interfaces

Can we ENFORCE use of getters and setters? If I design a class I would like to make sure no one can access my instance variables directly, they **MUST** use my getters and setters

- CS211 Preview: In Java you will be able to enforce access restrictions on your instance variables... you can (and should) make them *private* so Java itself enforces data encapsulation.
- So... does Python support “*private*” instance variables? Yes (and no)

Implementing Public/Private Interfaces

- Python attributes and methods are public by default.
 - **public attributes**: any other class or function can see and change the attribute `myCircle.radius = 20`
 - **public method**: any other class or function can call the method `myCircle.method1()`
- Make things private by adding `__` (two underscores) to the beginning of the name:
 - `self.__radius = 20` **# Private attribute**
 - `def __method1():` **# Private method**

Implementing Public/Private Interfaces

- Private attributes can (almost) only be accessed by methods defined in the class
- Private methods can (almost) only be called by other methods defined in the class
- Idea: Everything defined in the class has access to private parts.

Hiding your private parts (in Python)

- You can create somewhat private parts in Python. Naming an instance variable with an `__` (two underscores) makes it private.

```
class Student(object):
    def __init__(self, first, last, age):
        self.__name = first
        self.__lastName = last
        self.__age = age

    def getName(self):
        return self.__name

def main():
    aStudent = Student("Karl", "Johnson", 18)
    nm = aStudent.name
    #nm = circ.getName()
    print nm
```

```
Traceback (most recent call last):
  File "Private.py", line 23, in <module>
    main()
  File "Private.py", line 17, in main
    nm = aStudent.name
AttributeError: 'Student' object has no attribute 'name'
...s/cs112/spring09/samplecode/objects >
```


Hiding your private parts (in Python)

- Be a little sneakier then.. use `__name`:

```
class Student(object):
    def __init__(self, first, last, age):
        self.__name = first
        self.__lastName = last
        self.__age = age

    def getName(self):
        return self.__name

def main():
    aStudent = Student("Karl", "Johnson", 18)
    nm = aStudent.__name
    #nm = circ.getName()
    print nm
```

```
Traceback (most recent call last):
  File "Private.py", line 23, in <module>
    main()
  File "Private.py", line 17, in main
    nm = aStudent.__name
AttributeError: 'Student' object has no attribute '__name'
...s/cs112/spring89/samplecode/objects >
```

Nice try, but that won't work!

Hiding your private parts (in Python)

- Be super sneaky then.. use `__Student__name`:

```
class Student(object):
    def __init__(self, first, last, age):
        self.__name = first
        self.__lastName = last
        self.__age = age

    def getName(self):
        return self.__name

def main():
    aStudent = Student("Karl", "Johnson", 18)
    nm = aStudent.__Student__name
    #nm = aStudent.getName()
    print nm
```

```
...s/cs112/spring09/samplecode/objects > python Private.py
Karl
...s/cs112/spring09/samplecode/objects > █
```

Ahh... you saw my private parts... that was rude!

So, it is possible to interact with private data in Python, but it is difficult and good programmers know not to do it. Using the defined interface methods (getters and setters) will make code more maintainable and safer to use

Getters and Setters (or) Accessors and Mutators

- These methods are a coding convention
- Getters/Accessors are methods that return an attribute
 - `def get_name(self):`
- Setters/Mutators are methods that set an attribute
 - `def set_name(self,newName):`

Why use getters?

- Definition of my getter:

```
def getName(self):  
    return self.name
```

What if I want to store the name instead as first and last name in the class?
Well, with the getter I only have to do this:

```
def getName(self):  
    return self.firstname + self.lastname
```

If I had used dot notation outside the class, then all the code **OUTSIDE** the class would need to be changed because the internal structure **INSIDE** the class changed. Think about libraries of code... If the Python-authors change how the Button class works, do you want to have to change YOUR code? No! Encapsulation helps make that happen. They can change anything inside they want, and as long as they don't change the method signatures, your code will work fine.

Getters help you hide the internal structure of your class!

Setters

- Another common method type are “setters”

```
def setAge(self, age):  
    self.age = age
```

Why? Same reason + one more. I want to hide the internal structure of my Class, so I want people to go through my methods to get and set instance variables. What if I wanted to start storing people's ages in dog-years? Easy with setters:

```
def setAge(self, age):  
    self.age = age / 7
```

More commonly, what if I want to add validation... for example, no age can be over 200 or below 0? If people use dot notation, I cannot do it. With setters:

```
def setAge(self, age):  
    if age > 200 or age < 0:  
        # show error  
    else:  
        self.age = age / 7
```


Getters and Setters

- Getters and setters are useful to provide data encapsulation. They hide the internal structure of your class and they should be used!

Printing objects

```
>>> aStudent = Student("Karl","Johnson", 18)
>>> print aStudent
<__main__.Student object at 0x8bd70>
```

Doesn't look so good! Define a special function in the class
“__str__” that is used to convert your object to a string whenever
needed

```
def __str__(self):
    return "Name is:" + self.__name
```

```
Name is:KarlJohnson
```


-
- See BouncingBall Slides.

Data Processing with Class

- A class is useful for modeling a real-world object with complex behavior.
- Another common use for objects is to group together a set of information that describes a person or thing.
 - Eg., a company needs to keep track of information about employees (an `Employee` class with information such as employee's name, social security number, address, salary, etc.)

Data Processing with Class

- Grouping information like this is often called a *record*.
- Let's try a simple data processing example!
- A typical university measures courses in terms of credit hours, and grade point averages are calculated on a 4 point scale where an "A" is 4 points, a "B" is three, etc.

Data Processing with Class

- Grade point averages are generally computed using quality points. If a class is worth 3 credit hours and the student gets an “A”, then he or she earns $3(4) = 12$ quality points. To calculate the GPA, we divide the total quality points by the number of credit hours completed.

Data Processing with Class

- Suppose we have a data file that contains student grade information.
- Each line of the file consists of a student's name, credit-hours, and quality points.

Adams, Henry	127	228
Comptewell, Susan	100	400
DibbleBit, Denny	18	41.5
Jones, Jim	48.5	155
Smith, Frank	37	125.33

Data Processing with Class

- Our job is to write a program that reads this file to find the student with the best GPA and print out their name, credit-hours, and GPA.
- The place to start? Creating a `Student` class!
- We can use a `Student` object to store this information as instance variables.

Data Processing with Class

- ```
class Student:
 def __init__(self, name, hours, qpoints):
 self.name = name
 self.hours = float(hours)
 self.qpoints = float(qpoints)
```
- The values for `hours` are converted to `float` to handle parameters that may be floats, ints, or strings.
- To create a student record:  

```
aStudent = Student("Adams, Henry", 127, 228)
```
- The coolest thing is that we can store all the information about a student in a single variable!



# Data Processing with Class

- We need to be able to access this information, so we need to define a set of accessor methods.

- ```
def getName(self):  
    return self.name  
  
def getHours(self):  
    return self.hours  
  
def getQPoints(self):  
    return self.qpoints  
  
def gpa(self):  
    return self.qpoints/self.hours
```

These are commonly
called “getters”

- For example, to print a student's name you could write:

```
print aStudent.getName()
```
- `aStudent.name`

Data Processing with Class

- How can we use these tools to find the student with the best GPA?
- We can use an algorithm similar to finding the max of n numbers! We could look through the list one by one, keeping track of the best student seen so far!

Data Processing with Class

Pseudocode:

```
Get the file name from the user
Open the file for reading
Set best to be the first student
For each student s in the file
    if s.gpa() > best.gpa
        set best to s
Print out information about best
```


Data Processing with Class

```
# gpa.py
# Program to find student with highest GPA
import string

class Student:

    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)

    def getName(self):
        return self.name

    def getHours(self):
        return self.hours

    def getQPoints(self):
        return self.qpoints

    def gpa(self):
        return self.qpoints/self.hours

    def makeStudent(infoStr):
        name, hours, qpoints = string.split(infoStr, "\t")
        return Student(name, hours, qpoints)
```

```
def main():
    filename = raw_input("Enter name the grade file: ")
    infile = open(filename, 'r')
    best = makeStudent(infile.readline())
    for line in infile:
        s = makeStudent(line)
        if s.gpa() > best.gpa():
            best = s
    infile.close()
    print "The best student is:", best.getName()
    print "hours:", best.getHours()
    print "GPA:", best.gpa()

if __name__ == '__main__':
    main()
```


Helping other people use your classes

- Frequently, you will need to write classes other people will use
- Or classes you will want to use later, but have forgotten how

Answer: Document your class usage!

Putting Classes in Modules

- Sometimes we may program a class that could be useful in many other programs.
- If you might be reusing the code again, put it into its own module file with documentation to describe how the class can be used so that you won't have to try to figure it out in the future from looking at the code!

Module Documentation

- You are already familiar with “#” to indicate comments explaining what’s going on in a Python file.
- Python also has a special kind of commenting convention called the *docstring*. You can insert a plain string literal as the first line of a module, class, or function to document that component.

Module Documentation

- Why use a docstring?
 - Ordinary comments are ignored by Python
 - Docstrings are accessible in a special attribute called `__doc__`.
- Most Python library modules have extensive docstrings. For example, if you can't remember how to use `random`:

```
>>> import random
>>> print random.random.__doc__
random() -> x in the interval [0, 1).
```


Module Documentation

- Docstrings are also used by the Python online help system and by a utility called PyDoc that automatically builds documentation for Python modules. You could get the same information like this:

```
>>> import random
>>> help(random.random)
Help on built-in function random:

random(...)
    random() -> x in the interval [0, 1).
```


Module Documentation

- To see the documentation for an entire module, try typing `help(module_name)!`
- The following code for the projectile class has docstrings.

Module Documentation

```
# projectile.py

"""projectile.py
Provides a simple class for modeling the flight of projectiles."""

from math import pi, sin, cos

class Projectile:

    """Simulates the flight of simple projectiles near the earth's
    surface, ignoring wind resistance. Tracking is done in two
    dimensions, height (y) and distance (x)."""

    def __init__(self, angle, velocity, height):
        """Create a projectile with given launch angle, initial
        velocity and height."""
        self.xpos = 0.0
        self.ypos = height
        theta = pi * angle / 180.0
        self.xvel = velocity * cos(theta)
        self.yvel = velocity * sin(theta)
```


Module Documentation

```
def update(self, time):
    """Update the state of this projectile to move it time seconds
    farther into its flight"""
    self.xpos = self.xpos + time * self.xvel
    yvell = self.yvel - 9.8 * time
    self.ypos = self.ypos + time * (self.yvel + yvell) / 2.0
    self.yvel = yvell

def getY(self):
    "Returns the y position (height) of this projectile."
    return self.ypos

def getX(self):
    "Returns the x position (distance) of this projectile."
    return self.xpos
```


PyDoc

- PyDoc The pydoc module automatically generates documentation from Python modules. The documentation can be presented as pages of text on the console, served to a Web browser, or saved to HTML files.
- `pydoc -g` # Launch the GUI