Defining Classes

Announcements

- Lab 10 written exercise due Friday in lecture
- Lab 10 programs due Saturday at midnight
- Ninja sessions tonight and Friday night

Plan

- Review how to use objects
- Look at how to define our own classes of objects

Review

- An **object** consists of data and methods.

- Objects are **compound** values that let us associate multiple pieces of data within a single entity.

- Each object is an **instance** of a **class**. We create an object by calling the **constructor** for its class, which has the same name as the class.

- We access, modify, or otherwise use an object's data through its **methods**. Calling a method is like sending a message to the object.

Object Syntax

```
# Calling the Point constructor for the Point class
p1 = Point(50, 70)

# Evaluates to 50
p1.getX()

# Mutates p1
p1.move(10, 10)

# Evalutes to 60
p1.getX()
```

Class definition syntax

```
class Point(object):

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

  def __str__(self):
    return "Point at (%d, %d)" % (self.x, self.y)

def getX(self):
  return self.x

def getY(self):
  return self.y

def move(self, dx, dy):
  self.x = self.x + dx
  self.y += dy
```

Defining classes

- A template for a group of custom objects
- The class definition specifies how instances of this class will be constructed and printed, what data is stored in each object, and what methods can be applied to these objects.
- You can create many instances of the same class—each will have its own data.

Instance Variables

- **Instance variables** store the data that's private to a particular instance of a class.

- They are typically defined in `__init__`. But regardless of where they are defined, they are available in every method definition.

- For an instance variable called `x`, we would set and access `x` by referring to `self.x`:

```
self.x = 10

return self.x
```

- Because `self` is automatically a parameter to every method, we always have access to the instance variables when we're defining methods.

The self parameter

- Each method has `self` as its first parameter.
- When you construct an object or call a method on an object, `self` is automatically set to the object itself.
- This means we call a method/constructor with a number of arguments that is **one fewer** than the number of parameters in the definition of the method/constructor.

Methods

- Each class needs an `__init__` method, which acts as its constructor and a `__str__` method, which dictates how instances will be printed by Python's `print` function.

- There are also getters, or methods which access and return an instance variable and setters, or methods which modify one or more instance variables.

- There may be other methods that don't fall into either category.

Examples