Homework can now be completed in pairs, if you wish.
- Every individual should still submit his/her own homework
- Please write your partner’s name at the top of your file
- I strongly recommend that you try problems on your own first

Some questions will be deferred to office hours & after class
- Deferred: Questions about related topics, extensions, etc.
- Answered: Clarifications, examples, confusions, etc.
- Your job: Keep asking all your questions; I'll answer fewer

Code examples distributed in lecture
- Code examples are always on the course website
- Homework solutions are also online
- I’ll print out the code on code-intensive days

Implementing an Object System

Today’s topics:
- What is a class?
- What is an instance?
- How do we create inheritance relationships?
- How do we write code for attribute look-up procedures?

Tools we’ll use:
- Dispatch dictionaries
- Higher-order functions

The OOP Abstraction Barrier (a.k.a. the Line)

Above the Line:
- Objects with local state & interact via message passing
- Objects are instantiated by classes, which are also objects
- Classes may inherit from other classes to share behavior
- Mechanics of objects are governed by “evaluation procedures”

Below the Line:
- Objects have mutable dictionaries of attributes
- Attribute look-up for instances is a function
- Attribute look-up for classes is another function
- Object instantiation is another function

Implementing the Object Abstraction

Fundamental OOP concepts:
- Object instantiation and initialization
- Attribute look-up and assignment
- Method invocation
- Inheritance

Not-so-fundamental issues (that we’ll skip):
- Dot expression syntax
- Multiple inheritance
- Introspection (e.g., what class does this object have?)

Instances

Dispatch dictionary with messages ‘get’ and ‘set’

Attributes stored in a local dictionary “attributes”

def make_instance(cls):
    """Return a new Object instance."
    return cls()
def get_value(name):
    if name in attributes:
        return attributes[name]
    else:
        value = clsrouch[attribute](name)
        return bind_method(value, instance)
def set_value(name, value):
    attributes[name] = value
    attributes = {}
    instance = ('get': get_value, 'set': set_value)
    return instance
Bound Methods

If looking up a name returns a class attribute value that is a function, getattr returns a bound method

```python
def make_instance(cls):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            value = cls['get'](name)
            return bind_method(value, instance)

    return

def bind_method(value, instance):
    if callable(value):
        method(*args):
            return method(instance, *args)
    else:
        return value
```
Example: Using Inheritance

CheckingAccount is a special case of Account

```python
def make_checking_account_class():
    """Return the CheckingAccount class."""
    def withdraw(self, amount):
        return Account['get']('withdraw')(self, amount + 1)
    return make_class({'withdraw': withdraw, 'interest': 0.01}, Account)
```

Relationship to the Python Object System

Object attributes are stored as dictionaries

Some "magic" names, __<name>__, require special handling

An object has an "attribute" called __dict__ that is a dictionary of its instance attributes

Demo

In Python, classes have classes too

The equivalent of init_instance can be customized (metaclass)