Wednesday, October 8

Announcements
- Project 2 due Thursday 10/9 @ 11:59pm
- Homework 5 due Wednesday 10/15 @ 11:59pm
- Special event on Tuesday 10/14 @ 7pm, John interviews Dropbox CEO/founder Drew Houston
  - No video, so come to Wheeler
  - Suggest questions and vote for your favorites at http://goo.gl/HtkXFf or on Piazza

Object-Oriented Programming
A method for organizing modular programs
- Data abstraction
- Bundling together information and related behavior
A metaphor for computation using distributed state
- Each object has its own local state
- Each object also knows how to manage its own local state, based on method calls
- Method calls are messages passed between objects
- Several objects may all be instances of a common type
- Different types may relate to each other
Specialized syntax & vocabulary to support this metaphor

Object-Oriented Programming

Classes
A class serves as a template for its instances.

Idea: All bank accounts have a balance and an account holder; the Account class should add those attributes to each newly created instance.

```
>>> a = Account('Jim')
>>> a.balance
0
>>> a.holder
'Jim'
```

Idea: All bank accounts should have "withdraw" and "deposit" behaviors that all work in the same way.

```
>>> a.deposit(15)
15
>>> a.withdraw(5)
5
```

Better idea: All bank accounts share a "withdraw" method and a "deposit" method.

```
>>> a.withdraw(10)
'Insufficient funds'
```

Object-Oriented Programming

Class Statements

The Class Statement
```
<name>:
<suite>
```
A class statement creates a new class and binds that class to <name> in the first frame of the current environment.

Assignment & def statements in <suite> create attributes of the class (not names in frames)

```
>>> class Clown:
...     nose = 'big and red'
...     def dance():
...         return 'No thanks'
...>>> Clown
<class '__main__.Clown'>
```

Object Construction

```
Ideas: All bank accounts have a balance and an account holder; the Account class should add those attributes to each of its instances

>>> a = Account('Jim')
>>> a.holder
'Jim'

>>> a.balance
0
```

When a class is called:

1. A new instance of that class is created:

```
>>> a = Account('Jim')
```

2. The __init__ method of the class is called with the new object as its first argument (named self), along with any additional arguments provided in the call expression.

```
>>> a = Account('Jim')
>>> a.holder
'Jim'
```
Object Identity

Every object that is an instance of a user-defined class has a unique identity:

```python
>>> a = Account('Jim')
>>> b = Account('Jack')
>>> a.balance
>>> a.holder
'Jim'
>>> b.holder
'Jack'
```

Identity operators "is" and "is not" test if two expressions evaluate to the same object:

```python
>>> a is a
True
>>> a is not b
True
```

Binding an object to a new name using assignment does not create a new object:

```python
>>> c = a
>>> c is a
True
```

Every call to Account creates a new Account instance. There is only one Account class.

```python
>>> a.balance
0
>>> b.holder
'Jack'
```

Methods

Methods are functions defined in the suite of a class statement:

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
    def deposit(self, amount):
        self.balance += amount
        return self.balance
    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance -= amount
        return self.balance
```

These def statements create function objects as always, but their names are bound as attributes of the class

```python
tom_account = Account('Tom')
tom_account.deposit(100)
```

Dot expressions

Objects receive messages via dot notation.

Dot notation accesses attributes of the instance or its class.

```
<expression> . <name>
```

The `<expression>` can be any valid Python expression.

The `<name>` must be a simple name.

Evaluates to the value of the attribute looked up by `<name>` in the object that is the value of the `<expression>`.

```python
tom_account.deposit[10]
```

Attributes

Accessing Attributes

Using getattr, we can look up an attribute using a string

```python
>>> getattr(tom_account, 'balance')
10
>>> hasattr(tom_account, 'deposit')
True
```

getattr and dot expressions look up a name in the same way:

```
Looking up an attribute name in an object may return:
• One of its instance attributes, or
• One of the attributes of its class
```

Methods and Functions

Python distinguishes between:

• Functions, which we have been creating since the beginning of the course, and
• Bound methods, which couple together a function and the object on which that method will be invoked.

```
Object + Function = Bound Method
```

```python
>>> type(Account.deposit)
<class 'function'>
>>> type(tom_account.deposit)
<class 'method'>
```

```python
>>> Account.deposit(tom_account, 1000)
Function: all arguments within parentheses
1010
>>> tom_account.deposit(1000)
Method: One object before the dot and other arguments within parentheses
2014
```
Looking Up Attributes by Name

<expression> . <name>

To evaluate a dot expression:

1. Evaluate the <expression> to the left of the dot, which yields the object of the dot expression.

2. <name> is matched against the instance attributes of that object; if an attribute with that name exists, its value is returned.

3. If not, <name> is looked up in the class, which yields a class attribute value.

4. That value is returned unless it is a function, in which case a bound method is returned instead.

Class Attributes

Class attributes are "shared" across all instances of a class because they are attributes of the class, not the instance.

class Account:
    interest = 0.02  # A class attribute
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
        # Additional methods would be defined here

>>> tom_account = Account('Tom')
>>> jim_account = Account('Jim')
>>> tom_account.interest
0.02
>>> jim_account.interest
0.02

Attribute Assignment

Assignment Statements and Attributes

Assignment statements with a dot expression on their left-hand side affect attributes for the object of that dot expression:

* If the object is an instance, then assignment sets an instance attribute
* If the object is a class, then assignment sets a class attribute