Announcements

- Homework 5 is due Tuesday 10/15 @ 11:59pm
- Project 3 is due Thursday 10/24 @ 11:59pm
- Midterm 2 is on Monday 10/28 7pm-9pm

61A Lecture 16

Friday, October 11

Attributes

Terminology: Attributes, Functions, and Methods

All objects have attributes, which are name-value pairs
Classes are objects too, so they have attributes
Instance attribute: attribute of an instance
Class attribute: attribute of the class of an instance

Methods

Functions are objects.

Bound methods are also objects: a function that has its first parameter "self" already bound to an instance.

Dot expressions evaluate to bound methods for class attributes that are functions.

<instance>.<method_name>

Looking Up Attributes of an Object

<expression>.<name>

To evaluate a dot expression:

1. Evaluate the <expression>.
2. <name> is matched against the instance attributes.
3. If not found, <name> is looked up in the class.
4. That class attribute value is returned unless it is a function, in which case a bound method is returned.
Assignment to 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.

```
Instance Attribute Assignment:  
  tom_account.interest = 0.08

But the name ("interest") is not looked up.

Attribute assignment statement adds or modifies the attribute named "interest" of tom_account.
```

```
Class Attribute Assignment:  
  Account.interest = 0.04

This expression evaluates to an object.
```

Inheritance

Inheritance is a method for relating classes together. A common use: Two similar classes differ in their degree of specialization. The specialized class may have the same attributes as the general class, along with some special-case behavior.

```
class <name> (<base class>):
   <suite>
```

Conceptually, the new subclass "shares" attributes with its base class. The subclass may override certain inherited attributes. Using inheritance, we implement a subclass by specifying its differences from the the base class.

```
Inheritance Example

A CheckingAccount is a specialized type of Account.

>>> ch = CheckingAccount('Tom')
>>> ch.interest  # Lower interest rate for checking accounts
0.01
>>> ch.deposit(20)  # Deposits are the same
20
>>> ch.withdraw(5)  # Withdrawals incur a $1 fee
14

Most behavior is shared with the base class Account.
```

```
Looking Up Attribute Names on Classes

Base class attributes aren't copied into subclasses!

To look up a name in a class:
1. If it names an attribute in the class, return the attribute value.
2. Otherwise, look up the name in the base class, if there is one.
```

```
>>> ch = CheckingAccount('Tom')  # Calls Account.__init__
>>> ch.interest  # Found in CheckingAccount
0.01
>>> ch.deposit(20)  # Found in Account
20
>>> ch.withdraw(5)  # Found in CheckingAccount
14
```
Object-Oriented Design

Designing for Inheritance

Don’t repeat yourself; use existing implementations. Attributes that have been overridden are still accessible via class objects. Look up attributes on instances whenever possible.

```python
class CheckingAccount(Account):
    # A bank account that charges for withdrawals.
    interest = 0.01
    withdraw_fee = 1
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

Attributes look-up on base class Preferred to CheckingAccount.withdraw_fee to allow for specialized accounts.

Inheritance and Composition

Object-oriented programming shines when we adopt the metaphor. Inheritance is best for representing is-a relationships.

E.g., a checking account is a specific type of account. So, CheckingAccount inherits from Account.

Composition is best for representing has-a relationships.

E.g., a bank has a collection of bank accounts it manages. So, A bank has a list of accounts as an attribute.

(Demo)

Multiple Inheritance

A class may inherit from multiple base classes in Python.

```python
class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(self, amount - self.deposit_fee)
```

A class may inherit from multiple base classes in Python.

```python
class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1 # A free dollar!
```

CleverBank marketing executive wants:

• Low interest rate of 1%
• A $1 fee for withdrawals
• A $2 fee for deposits
• A free dollar when you open your account

```python
such_a_deal = AsSeenOnTVAccount("John")
```

```python
such_a_deal.balance # 1
such_a_deal.deposit(20) # 19
such_a_deal.withdraw(5) # 13
```
Resolving Ambiguous Class Attribute Names

```python
>>> such_a_deal = AsSeenOnTVAccount("John")
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
>>> such_a_deal.withdraw(5)
13
```

Complicated Inheritance

Biological Inheritance

Moral of the story: Inheritance can be complicated, so don’t overuse it!