Announcements
- Project 2 due Thursday 2/26 @ 11:59pm
- Extra office hours on Wednesday 2/25 4pm-6pm in Bechtel (Garbarini Lounge)
- Bonus point for early submission by Wednesday 2/25 @ 11:59pm!
- Relocated office hours on Thursday 2/26: 380 Soda (11am-3pm) & 606 Soda (3pm-6pm)

Object-Oriented Programming

A method for organizing 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.

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

**Better idea**: All bank accounts share a "withdraw" and "deposit" behaviors that all work in the same way.

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

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

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

Class Statements

**Class Statements**

When a class is called:
1. A new instance of that class is created:
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.

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder
```

Object Construction

An account instance

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

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 = self.balance + amount
        return self.balance
    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient funds'
        self.balance = 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
>>> Account.balance
1000
>>> Account.holder
'Jim'
```

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.

```python
>>> Account.deposit(100)
101
>>> Account.deposit(200)
```

Invoking Methods

All invoked methods have access to the object via the self parameter, and so they can all access and manipulate the object's state.

```python
class Account:
    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance
```

```python
>>> Account.deposit(100)
101
```

Dot Expressions

Objects receive messages via dot notation. Dot notation accesses attributes of the instance or its class.

```python
<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)
```

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

```python
>>> getattr(tom_account, 'balance')
10
```

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.

```python
>>> Account.deposit(100)
```

```python
>>> Account.deposit(200)
```

```python
>>> Account.deposit(tom_account, 100)
```

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.

```python
>>> Account.deposit(100)
```

```python
>>> Account.deposit(tom_account, 200)
```

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.

```python
>>> Account.deposit(100)
```

```python
>>> Account.deposit(tom_account, 200)
```
Looking Up Attributes by 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')
>>> tom_account = Account('Jim')
0.02
>>> tom_account = Account('Jim')
0.02

The interest attribute is not part of the instance, it's part of the class.

Attribute Assignment

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

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

>>> tom_account = Account('Tom')
>>> tom_account = Account('Jim')
>>> tom_account = Account('Jim')
0.02
>>> tom_account = Account('Jim')
0.02
```

Instance attributes of jim_account

```
>>> jim_account = Account('Jim')
>>> tom_account = Account('Tom')
>>> tom_account.balance 0.02
>>> Account.interest = 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
```

Instance attributes of tom_account

```
>>> jim_account = Account('Jim')
>>> tom_account = Account('Tom')
>>> tom_account.balance 0.02
>>> Account.interest = 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
>>> tom_account.interest 0.04
```

Attribute Assignment Statements

```
Account class attributes
  interest: 0.02
  withdrawal, deposit, __init__

Instance attributes of jim_account
  balance: 0
  holder: 'Jim'
  interest: 0.02

Instance attributes of tom_account
  balance: 0
  holder: 'Tom'
  interest: 0.02
```

Assignment to Attributes

```
Account.interest = 0.04
```

Instance : Account
  interest: 0.04

Attribute Assignment statement adds or modifies the attribute named "interest" of Account

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

Account class attributes
  interest: 0.04

Instance attributes of jim_account
  balance: 0
  holder: 'Jim'
  interest: 0.02

Instance attributes of tom_account
  balance: 0
  holder: 'Tom'
  interest: 0.02
```