Inheritance

Announcements

Attributes & Methods

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

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

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

```
>>> Account.deposit(tom_account, 1001)
1011
>>> tom_account.deposit(1007)
2018
```

Function: all arguments within parentheses

Method: One object before the dot and other arguments within parentheses

(Demo)

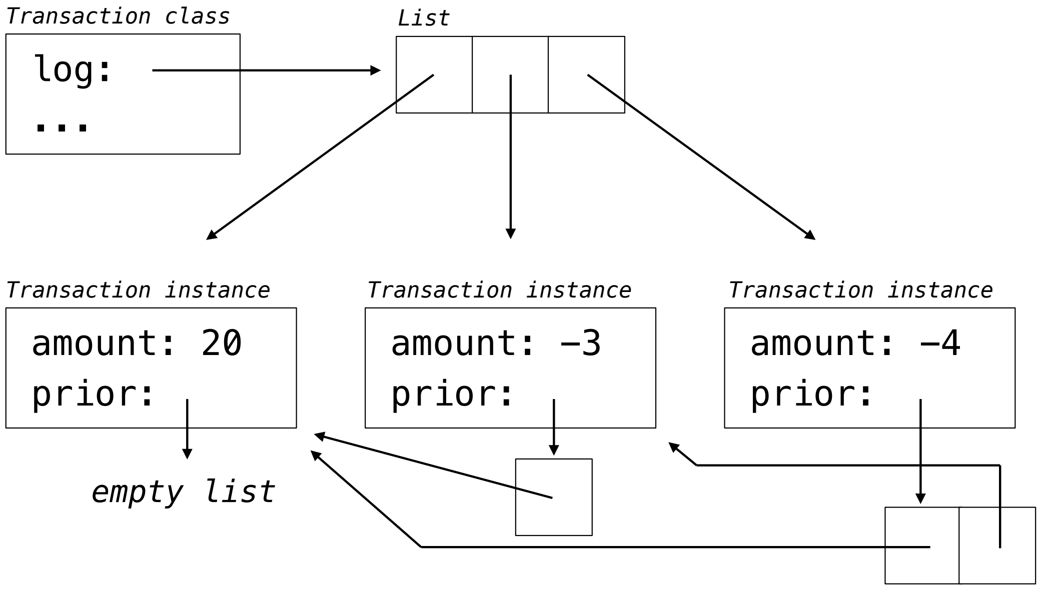
Class Attributes

A class attribute can be accessed from either an instance or its class. There is only one value for a class attribute, regardless of how many instances.

```
class Transaction:
    """A logged transaction.

    >>> s = [20, -3, -4]
    >>> ts = [Transaction(x) for x in s]
    >>> ts[1].balance()
    17
    >>> ts[2].balance()
    13
    """
    log = []

    def __init__(self, amount):
        self.amount = amount
        self.prior = list(self.log)
        self.log.append(self)

    def balance(self):
        return self.amount + sum([t.amount for t in self.prior])
```

Always bound to some Transaction instance

Equivalently: list(type(self).log)



(Demo)

Example: Close Friends

```
class Friend:
    def __init__(self, name):
        self.name = name
        self.heard_from = {}

    def hear_from(self, friend):
        if friend not in self.heard_from:
            self.heard_from[friend] = 0
        self.heard_from[friend] += 1
        friend.just_messaged = self

    def how_close(self, friend):
        bonus = 0

        if hasattr(self, 'just_messaged') and self.just_messaged == friend:
            bonus = 3

        return friend.heard_from[self] + bonus

    def closest(self, friends):

        return max(friends, key=self.how_close)
```

A **Friend** instance tracks the number of times they **hear_from** each other friend.

A **Friend just_messaged** the friend that most recently heard from them.

how_close is one Friend (**self**) to another (**friend**)?

- The number of times **friend** has heard from **self**
- Plus a bonus of 3 if they are the one that most recently heard from **self**

self's closest friend among a list of **friends** is the one with the largest **self.how_close(friend)** value

(Demo)

Attribute Assignment

Attribute Assignment Statements

Account class attributes

interest: ~~0.02~~ ~~0.04~~ 0.05
(withdraw, deposit, __init__)

Instance attributes of jim_account

```
balance:   0
holder:    'Jim'
interest: 0.08
```

Instance attributes of tom_account

```
balance:   0
holder:    'Tom'
```

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

```
>>> jim_account.interest = 0.08
>>> jim_account.interest
0.08
>>> tom_account.interest
0.04
>>> Account.interest = 0.05
>>> tom_account.interest
0.05
>>> jim_account.interest
0.08
```