Welcome to Data C88C!

Lecture 14: Inheritance
Thursday, July 17th, 2025
Week 3
Summer 2025
Instructor: Eric Kim (ekim555@berkeley.edu)

Announcements

- Lab07, HW07 due Friday July 18th
- Maps due: Thursday July 24th
- Reminder: no lab today!

Lecture Overview

- OOP review
- Inheritance

Lab 6 Review

Lab 6 Question 2: Email

```
class Email:
    def __init__(self, msg, sender, recipient_name):
        self.msg = msg
        self.sender = sender
        self.recipient_name = recipient_name

class Server:
    def __init__(self):
        self.clients = {}

    def send(self, email):
        # Append the email to the inbox of the client it is addressed to.
```

Server dict Client list

```
        self.clients[email.recipient_name].inbox.append(email)
        ------------------------------------------------------------------
    ...
```

Email

```
class Client:
    def __init__(self, server, name):
        self.inbox = []
        self.server = server
        self.name = name
        ...
    ...
```

A **Client** can **send** an **Email** to its **Server**.

The Server then delivers it to the inbox of another Client.

To achieve this, a Server has a dictionary called clients that can look up each **Client instance** by the **name** of the Client.

Question: fill out the `Server.send()` method

Attribute Lookup Practice

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.

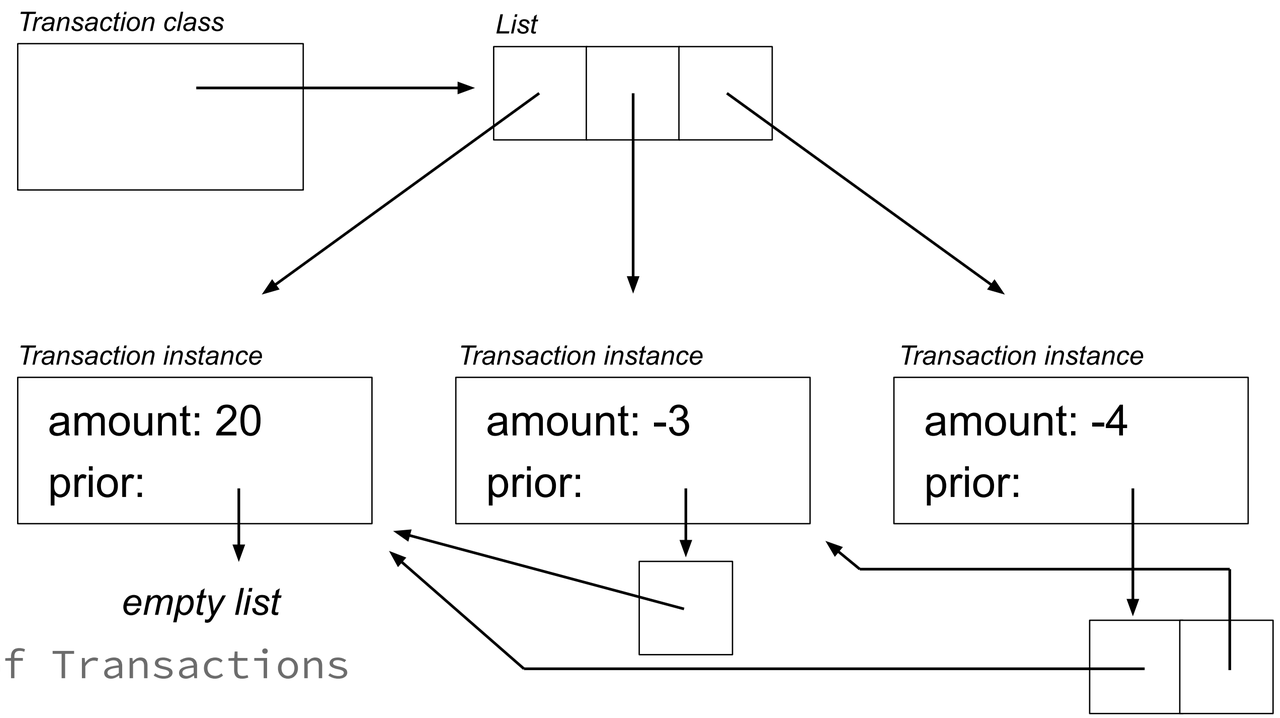
```
class Transaction:
    """A logged transaction.

    >>> 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 = ________________ # a list of Transactions
        self.log.append(self)

    def balance(self):
        """The sum of amounts for this transaction and all prior transactions"""
        return self.amount + sum(__________________________ )
```

Always bound to some Transaction instance



list(self.log)

[t.amount for t in self.prior]

Equivalently: list(type(self).log) **or** list(Transaction.log)

Accessing Attributes

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

```
>>> tom_account.balance
10
```

```
>>> 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

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.get(self, 0) + 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

Inheritance

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

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

or

```
        return super().withdraw(      amount + self.withdraw_fee)
```

Aside: which of these is best? Turns out the answer is complicated. Personally, in 2025: I prefer `super()`: [link]

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
```

Example: Three Attributes

```
class A:
    x, y, z = 0, 1, 2

    def f(self):
        return [self.x, self.y, self.z]

class B(A):
    """What would Python Do?

    >>> A().f()

    [0, 1, 2]

    >>> B().f()
     [6, 1, 'A']
    -------------

    """
    x = 6
    def __init__(self):
        self.z = 'A'
```

A class

x: 0
y: 1
z: 2

B class

x: 6

A instance

B instance

z: 'A'

Aside: Multiple inheritance

- Most OOP languages (including Python) support inheriting from multiple classes ("Multiple inheritance")
- **In this class, we will not be covering multiple inheritance**

```
class A:
    ...

class B:
    ...

class C(A, B):
    ...
```

C inherits from both `A` and `B`