Inheritance

Announcements

Lab 6 Review

Lab 6: Email

Sending an email:

```
>>> s = Server()
>>> a = Client(s, 'John')
>>> b = Client(s, 'Jack')
>>> a.compose('Hi!', 'Jack')
>>> b.inbox[0].msg
'Hi!'
```

```
class Email:
    def __init__(self, msg, sender, recipient_name):
        self.msg = msg
        self.sender = sender
        self.recipient_name = recipient_name
class Server:
    """Each Server has a dictionary from client names to client objects."""
    def __init__(self):
        self.clients = {}

    def send(self, email):
        """Append the email to the inbox of the client it is addressed to."""
        self.clients[email.recipient_name].inbox.append(email)

    def register_client(self, client):
        """Add a client to the dictionary of clients."""
        self.clients[client.name] = client
class Client:
    """A client has a server, a name (str), and an inbox (list)."""
    def __init__(self, server, name):
        self.inbox = []
        self.server = server
        self.name = name
        server.register_client(self)

    def compose(self, message, recipient_name):
        """Send an email with the given message to the recipient."""
        email = Email(message, self, recipient_name)
        self.server.send(email)
```

Lab 6: Make Change

```
                    25    {2: 2, 3: 2, 4: 3, 5: 1}
def make_change(amount, coins):
    """Return a list of coins that sum to amount, preferring the smallest coins
    available and placing the smallest coins first in the returned list."""
    if not coins:
        return None
    smallest = min(coins)    -> 2
    rest = remove_one(coins, smallest)  -> {2: 1, 3: 2, 4: 3, 5: 1}
    if amount < smallest:
        return None

    elif amount == smallest:

        return [smallest]

    else:
                                    23
        result = make_change(amount-smallest, rest)  -> [3, 3, 4, 4, 4, 5]
        if result:
            return [smallest] + result  [2] + [3, 3, 4, 4, 4, 5] -> [2, 3, 3, 4, 4, 4, 5]
        else:
            return make_change(amount, rest)
```

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

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.

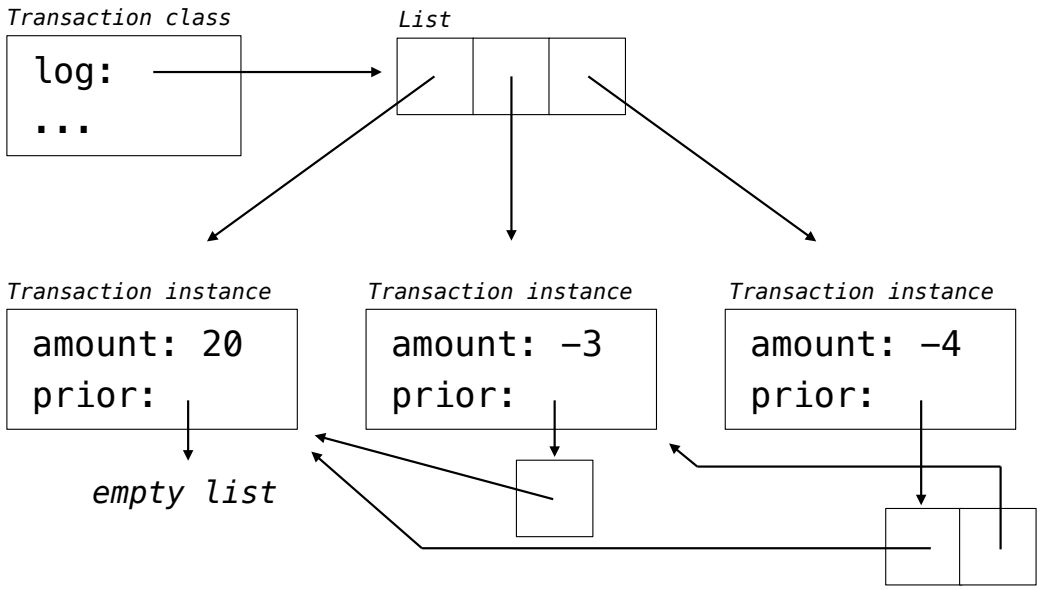
```
class Transaction:
    """A logged transaction.

    >>> 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)

Accessing Attributes

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)
```

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'