

61A Lecture 17

Friday, October 10

Announcements

- Homework 5 is due Wednesday 10/15 @ 11:59pm
- Project 3 is due Thursday 10/23 @ 11:59pm
- Midterm 2 is on Monday 10/27 @ 7pm

Attributes

Terminology: Attributes, Functions, and Methods

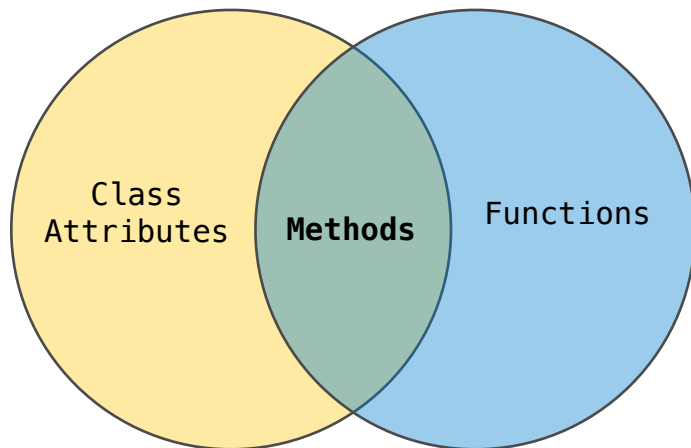
All objects have attributes, which are name-value pairs

Classes are objects too, so they have attributes

Instance attribute: attribute of an instance

Class attribute: attribute of the class of an instance

Terminology:



Python object system:

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>`

Inheritance

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 of 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`

```
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(self, 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
```

(Demo)

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.

```
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)
```

Attribute 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

Multiple Inheritance

```
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.

CleverBank marketing executive has an idea:

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

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

Multiple Inheritance

A class may inherit from multiple base classes in Python.

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

Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount('John')
```

```
>>> such_a_deal.balance
```

```
1
```

SavingsAccount method

```
>>> such_a_deal.deposit(20)
```

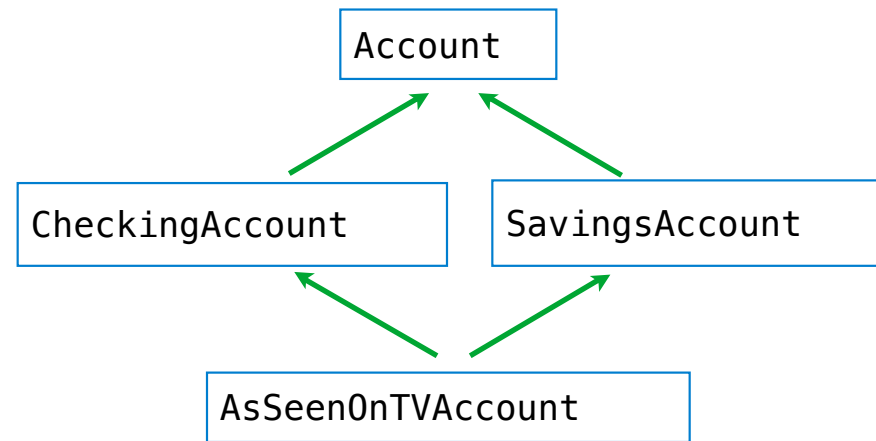
```
19
```

CheckingAccount method

```
>>> such_a_deal.withdraw(5)
```

```
13
```

Resolving Ambiguous Class Attribute Names



Instance attribute

```
>>> such_a_deal = AsSeenOnTVAccount('John')
```

```
>>> such_a_deal.balance
```

```
1
```

SavingsAccount method

```
>>> such_a_deal.deposit(20)
```

```
19
```

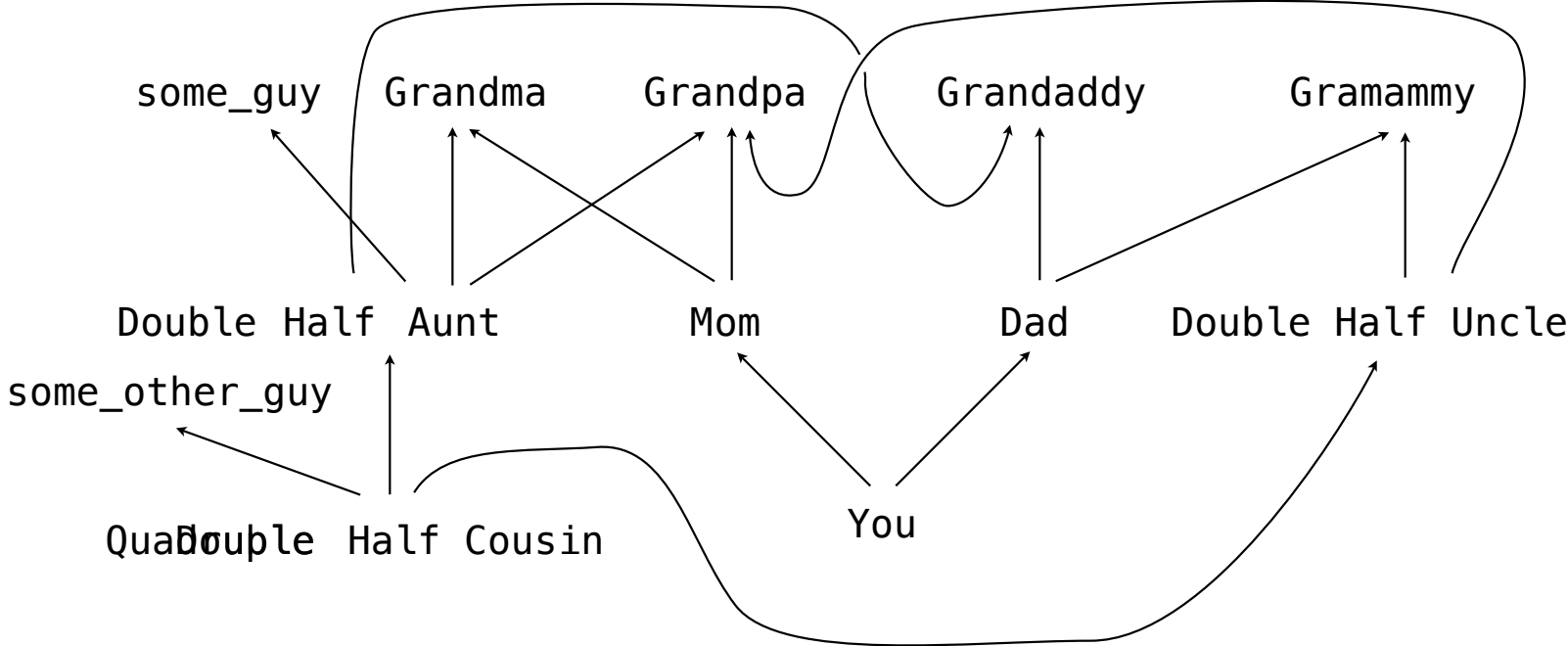
CheckingAccount method

```
>>> such_a_deal.withdraw(5)
```

```
13
```

Complicated Inheritance

Biological Inheritance



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