Inheritance

Announcements

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

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    # Additional methods would be defined here
```

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

The **interest** attribute is ***not*** part of the instance; it's part of the class!

(Demo)

Some History of Electronic Banking

“I think the computer has from the beginning been a fundamentally conservative force. It has made possible the saving of institutions pretty much as they were, which otherwise might have had to be changed…
banks were faced with the fact that the population was growing at a very rapid rate…
if the computer had not been invented, what would the banks have had to do? They might have had to decentralize, or they might have had to regionalize in some way. In other words, it might have been necessary to introduce a social invention, as opposed to the technical invention.

What the coming of the computer did, "just in time," was to make it unnecessary to create social inventions, to change the system in any way. So in that sense, the computer has acted as fundamentally a conservative force, a force which kept power or even solidified power where it already existed.”

–Joseph Weizenbaum

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
```

Looking Up Attributes by Name

Both instances and classes have attributes that can be looked up by dot expressions

<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

The Account Class

```
class Account:
```

__init__ is a special method name for the function that constructs an Account instance

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

self is the instance of the Account class on which deposit was invoked: a.deposit(10)

```
    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
```

Methods are functions defined in a class statement

(Demo)

```
>>> a = Account('John')
>>> a.holder
'John'
>>> a.balance
0
>>> a.deposit(15)
15
>>> a.withdraw(10)
5
>>> a.balance
5
>>> a.withdraw(10)
'Insufficient funds'
```

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)
```

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'

More From Joseph Weizenbaum

Q: Did you have these concerns when you were designing the banking system?

A: Not in the slightest. It was a very technical job…there were a number of very, very difficult problems…It was a whale of a lot of fun attacking those hard problems, and it never occurred to me at the time that I was cooperating in a technological venture which had certain social side effects which I might come to regret. That never occured to me; I was totally wrapped up in my identity as a professional, and besides, it was just too much fun.

–Joseph Weizenbaum