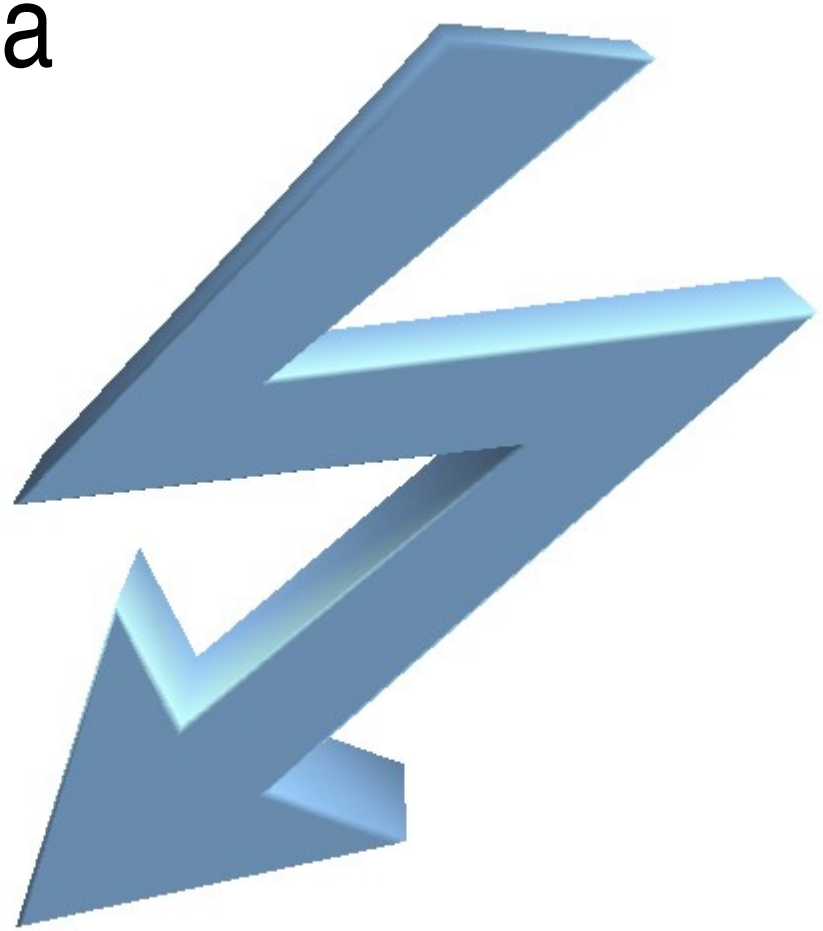
EECS 12: Lecture 8
Inheritance, `lambda`, and List Comprehensions

Mark E. Phair
mphair@gmail.com
UC Irvine EECS

July 26th, 2006

Agenda

- Magic number rant
- Inheritance
- `lambda`
- List Comprehensions
- Python factoid of the day



I am lambda, king of the functions!
Pull the other one!

Magic Number Rant

- Magic numbers: unexplained constants in the middle of the code
- SHOULD NEVER BE USED
  - At the VERY LEAST: comment it
  - Even better: Assign it to a variable at the beginning of the file... Conventionally, NAME_IT_LIKE_THIS
- Mostly imutable things like strings, ints, floats

Inheritance

- classes can be *derived* from other classes, this is called *inheritance*
- Think of it like a family tree of a class
- Plant
  - Fruit
    - Apple
    - Orange
  - Vegetable

Inheritance (continued)

Plant <---- Base class for Fruit, Vegetable

Fruit <----- child class of Plant, base of Apple

Apple <------- child class of Fruit

Orange <------- child class of Fruit

Vegetable <----- child class of Plant

Inheritance (continued)

```
class Plant: pass
class Fruit (Plant): pass
class Apple (Fruit): pass
class Orange (Fruit): pass
class Vegetable (Plant): pass
```

Inheritance (continued)

```
class Plant:
    def __init__(self):
        self.hasBeenWatered = False
        self.eaten = False
    def water(self):
        self.hasBeenWatered = True
    def eat(self):
        print 'Eaten.'
        self.eaten = True
```

Inheritance (continued)

```
class Fruit (Plant):
    def eat(self):
        if self.hasBeenWatered:
            print 'Yummy! Fructose!'
            self.eaten = True
        else:
            print "No way! It's all dry!"
```

Inheritance (continued)

```
p = Plant()
p.eat()

f = Fruit()
f.eat()
f.water()
f.eat()
```

```
Eaten.

No way! It's all dry!

Yummy! Fructose!
```

Inheritance (continued)

```
class Vegetable (Plant):
    def eat(self):
        if self.hasBeenWatered:
            print """I've eaten my
  vegetables... can I have dessert
  now?"""
            self.eaten = True
        else:
            print "No way! It's all dry!"
```

Inheritance (continued)

```
p = Plant()
p.eat()
```

```
Eaten.
```

```
v = Vegetable()
v.eat()
v.water()
v.eat()
```

```
No way! It's all dry!

I've eaten my
  vegetables... can I
  have dessert now?
```

Let's explore

- Create a base class called `Animal` that keeps track of whether or not it has been fed (you can feed it with the `feed()` method), and has a `makeSound()` method that just says “I'm an Animal!”
- Create two child classes, `Cow` and `Pig`, that will make the sounds “Moo!” and “Oink!” only if they have been fed.

Why Inheritance?

Major reason: polymorphism

If you assume that everything in the list will either be a `Plant` or some *subclass* (child class) of `Plant` (e.g., `Fruit`), then you can do things to all of them that you can do to all `Plants` without having to worry about which type of `Plant` it is.

Special Note About Children

- We've used the term *child* to refer to two different things:
  - *Child classes* are classes that have been derived from a *base* or *parent class*
  - *Child nodes* are nodes in a tree that are connected to a *parent node*
- Typically:
  - Classes = children in the inheritance sense
  - Objects = children in the tree sense

`lambda` functions

- A `lambda` function is a function object with no name
- You can pass it to anything that takes a function

```
>>> map(lambda x: x*2, range(5))
[0, 2, 4, 6, 8]
>>> reduce(lambda x,y: x+y, range(5))
10
```

Let's explore

Using `lambda` functions plus `map` and `reduce`, write a single line of code that sums the squares of the numbers in `range(10)`

Hint: the result should be `285`

List comprehensions

The functionality of `map` and `filter` can be expressed in another way with list comprehensions

```
mLst = map(f1, lst)
mLst = [f1(x) for x in lst]
fLst = filter(f2, lst)
fLst = [x for x in lst if f2(x)]
```

List comprehensions: combining functionality

```
mfLst = map(f1, filter(f2, lst))
mfLst = [f1(x) for x in lst if f2(x)]
```

List comprehensions: not just single functions, no need for `lambda`

```
mfLst = [x**2 for x in lst if x<5)]
```

Where you already could have used this...

```
return [self.value, \
        [child.dumpToList() \
           for child in \
                 self.children]]
```

Python Factoid of the day: `zip`

```
zip combines two lists together:
>>> zip([0,1,2], ['a','b','c'])
[(0, 'a'), (1, 'b'), (2, 'c')]
>>> lst = ['cow','dog','cat']
>>> for ii,animal in \
     zip(range(len(lst)), lst):
       print ii, animal
0 cow
1 dog
2 cat
```