Introduction to Python
More OO – Decorators and Generators

Christopher Barker

UW Continuing Education / Isilon

August 08, 2012

Table of Contents

1 Review/Questions

2 More OO

3 Special Methods

4 Iterators / Generators

Review of Previous Class

- Really Quick OO overview
- Built an html generator, using:
    - A base class with a couple methods
    - Subclasses overriding class attributes
    - Subclasses overriding a method
    - Subclasses overriding the `__init__`

Homework review

Homework notes

multiple inheritance

Multiple inheritance:
Pulling from more than one class

```
class Combined(Super1, Super2, Super3):
    def __init__(self, something, something else):
        Super1.__init__(self, ......)
        Super2.__init__(self, ......)
        Super3.__init__(self, ......)
```

(calls to the super class `__init__` are optional – case dependent)

Attribute resolution – left to right

(Why would you want to do this?)

mix-ins

Hierarchies are not always simple

- Animal
    - Mammal
        - GiveBirth()
    - Bird
        - LayEggs()

Where do you put a Platypus or an Armadillo?

Real World Example: `FloatCanvas`

properties

Simple attributes:

```
In [5]: class C(object):
            def __init__(self):
                    self.x = 5
In [6]: c = C()
In [7]: c.x
Out[7]: 5
In [8]: c.x = 8
In [9]: c.x
Out[9]: 8
```

Iterators

Iterators are one of the main reasons Python code is so readable:

```
for x in just_about_anything:
    do_stuff(x)
```

you can loop through anything that satisfies the iterator protocol

http://docs.python.org/library/stdtypes.html#iterator-types

Iterator Protocol

An iterator must have the following methods:

`iterator.__iter__()`

Return the iterator object itself. This is required to allow both containers and iterators to be used with the for and in statements.

`iterator.next()`

Return the next item from the container. If there are no further items, raise the StopIteration exception.

Example Iterator

```
class IterateMe_1(object):
    def __init__(self, stop=5):
        self.current = 0
        self.stop = 5
    def __iter__(self):
        return self
    def next(self):
        if self.current < self.stop:
            self.current += 1
            return self.current
        else:
            raise StopIteration
```

This is a simple version of `xrange()`

LAB

- Extend (iterator_1.py) to be more like xrange() – add three input parameters:
  `iterator_2(start, stop, step=1)`
- See what happens if you break out in the middle of the loop:

```
it = IterateMe_2(2, 20, 2)
for i in it:
    if i > 10:  break
    print i
```

  And then pick up again:

```
for i in it:
    print i
```

- Does xrange() behave the same?
  – make yours match xrange().