

Week 3: Advanced OO and Special Topics

Christopher Barker

UW Continuing Education

March 26, 2013

Table of Contents

- 1 lambda
- 2 Decorators
- 3 Advanced-OO

A diversion...

A number of you are already using iPython

It's a very useful tool

And the iPython notebook is even cooler ..
particularly for in-class demos.

So I'll use it some today:

[http://ipython.org/ipython-doc/dev/
interactive/htmlnotebook.html](http://ipython.org/ipython-doc/dev/interactive/htmlnotebook.html)

String formatting...

A handy note about something that came up in last week's debugging exercise:

```
>>> print "%f, %f"%(fp, complex)
```

```
-----  
NameError
```

```
----> 1 print "%f, %f"%(fp, complex)
```

```
NameError: name 'fp' is not defined
```

(Demo in the iPython notebook...)

lambda

We didn't get to it last class, so let's do it now:

```
https://docs.google.com/presentation/d/  
1GMMrDXzYFMFRn9ufrVUGb0vSBG07VkV6GLAdu46CVzA/  
pub?start=false&loop=false&delayms=3000  
(that should be clickable...)
```

If not, open:

```
code\link_to_lambda_slides.html
```

Decorators

Decorators are wrappers around functions

They let you add code before and after the execution of a function

Creating a custom version of that function

Decorators

Syntax:

```
@logged
def add(a, b):
    """add() adds things"""
    return a + b
```

Demo and Motivation:

code\decorators\basicmath.py

PEP: <http://www.python.org/dev/peps/pep-0318/>

Decorators

@ decorator operator is an abbreviation:

```
@f  
def g:  
    pass
```

same as

```
def g:  
    pass  
g = f(g)
```

“Syntactic Sugar” – but really quite nice

Decorators

demo:

`code\decorators\DecoratorDemo.py`

Decorator examples

Examples from the stdlib:

Does this structure:

```
def g:  
    pass  
g = f(g)
```

look familiar ?

Decorator examples

staticmethod()

```
class C(object):  
    def add(a, b):  
        return a + b  
    add = staticmethod(add)
```

Decorator examples

`staticmethod()`

Decorator form:

```
class C(object):  
    @staticmethod  
    def add(a, b):  
        return a + b
```

(and `classmethod`)

examples

property()

```
class C(object):
    def __init__(self):
        self._x = None
    def getx(self):
        return self._x
    def setx(self, value):
        self._x = value
    def delx(self):
        del self._x
    x = property(getx, setx, delx,
                  "I'm the 'x' property.")
```

becomes...

Decorator examples

```
class C(object):  
    def __init__(self):  
        self._x = None  
    @property  
    def x(self):  
        return self._x  
    @x.setter  
    def x(self, value):  
        self._x = value  
    @x.deleter  
    def x(self):  
        del self._x
```

Puts the info close to where it is used

examples

CherryPy

```
import cherrypy
class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello World!"
cherrypy.quickstart(HelloWorld())
```

examples

Pyramid

```
@template
def A_view_function(request)
    .....

@json
def A_view_function(request)
    .....
```

so you don't need to think about what your view is returning...

decorators...

For this class:

Mostly want you to know how to use decorators
that someone else has written

Have a basic idea what they do when you do use
them

A nice intro (and talks about closures...):

[http://simeonfranklin.com/blog/2012/jul/1/
python-decorators-in-12-steps/](http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/)

LAB

- Write a decorator that can be used to wrap any function that returns a string in a `<p>` element – auto-generation of simple html.

(p_wrapper.py)

- Try using a class to make a decorator that will wrap a specified tag around a function that returns a string:

```
@tag_wrapper('h1')
def func2(x, y=4, z=2):
    return "the sum of %s and %s and %s is %s"%(x, y, z)

>>> print func2(3,4)
<h1>the sum of 3 and 4 and 2 is 9</h1>
```

Accessing Attributes

One of the strengths of Python is lack of clutter

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
```

Getter and Setters?

What if you need to add behavior later?

- do some calculation
- check data validity
- keep things in sync

Getter and Setters?

```
class C(object):  
    def get_x(self):  
        return self.x  
    def set_x(self, x):  
        self.x = x  
  
>>> c = C()  
>>> c.get_x()  
>>> 5  
>>> c.set_x(8)  
>>> c.get_x()  
>>> 8
```

Ugly and verbose – Java?

<http://dirtsimple.org/2004/12/python-is-not-java.html>

properties

When (and if) you need them:

```
class C(object):  
    def getx(self):  
        return self._x  
    def setx(self, value):  
        self._x = value  
    def delx(self):  
        del self._x  
    x = property(getx, setx, delx, "docstring")
```

Interface is still like simple attribute access
(properties_sample.py)

properties

When (and if) you need them:

```
class C(object):  
    def getx(self):  
        return self._x  
    def setx(self, value):  
        self._x = value  
    def delx(self):  
        del self._x  
    x = property(getx, setx, delx, "docstring")
```

Interface is still like simple attribute access
(properties_sample.py)

staticmethod

A method that doesn't get self!

```
class C(object):  
    def add(a, b):  
        return a + b  
    add = staticmethod(add)
```

```
>>> C.add(3,4)  
7  
>>> c = C()  
>>> c.add(2, 2)  
4
```

When you don't need self – can be used from either an instance or the class itself

see: `static_method.py`

classmethod

Method gets the class object, rather than an instance the first argument

```
class C(object):  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
    def a_class_method(klass, y):  
        print "in a_class_method", klass  
        return klass( y, y**2 )  
    a_class_method = classmethod(a_class_method)
```

When you need the class object rather than an instance – plays well with subclassing

see: `class_method.py`

dict.fromkeys()

classmethod often used for alternate constructors:

```
>>> d = dict([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot convert dictionary update
sequence element #0 to a sequence
>>> d = dict.fromkeys([1,2,3])
>>> d
{1: None, 2: None, 3: None}
```

dict.fromkeys()

```
class Dict: ...
    def fromkeys(klass, iterable, value=None):
        "Emulate dict_fromkeys() in dictobject.c"
        d = klass()
        for key in iterable:
            d[key] = value
        return d
    fromkeys = classmethod(fromkeys)
```

See also `datetime.datetime.now()`, etc....

For a low-level look:

<http://docs.python.org/howto/descriptor.html>

super

getting the superclass:

```
class SafeVehicle(Vehicle):  
    """  
    Safe Vehicle subclass of Vehicle base class...  
    """  
    def __init__(self, position=0, velocity=0, icon='S'):  
        Vehicle.__init__(self, position, velocity, icon)
```

not DRY

also, what if we had a bunch of references to superclass?

super

getting the superclass:

```
class SafeVehicle(Vehicle):  
    """  
    Safe Vehicle subclass of Vehicle base class  
    """  
    def __init__(self, position=0, velocity=0, icon='S'):  
        super(SafeVehicle, self).__init__(position, velocity)
```

“super() considered super!” by Raymond Hettinger

[http://rhettinger.wordpress.com/2011/05/26/
super-considered-super/](http://rhettinger.wordpress.com/2011/05/26/super-considered-super/)

maybe use super() for your html subclassing...

LAB

- Write a `Circle` class with decorator syntax for properties:
 - instantiate with a radius: `c = Circle(4)`
 - use a property for the diameter: get and settable:
`d = c.diameter`
`c.diameter = 5`
 - use a property for the area: only gettable
`a = c.area`
`a.area = 5 => AttributeError`
- (`circle_properties.py` and
`test_circle_properties.py`)

Wrap up

A better understanding of the underpinnings of OO
in Python?

Do you see a use for any of this in your projects?

Next Week:

Relational databases, SQL

– Jeff

And of course, your projects...

Project Time!

- Have you got your structure in place?
- Are your goals clear?
- Anyone want a public code review?
- Let's get to work!