Week 3: Advanced OO and Special Topics

Christopher Barker

UW Continuing Education

March 26, 2013

Table of Contents

A diversion...

A number of you are already using `iPython`

It's a very useful tool

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

So I'll use it some today:

`http://ipython.org/ipython-doc/dev/interactive/htmlnotebook.html`

String formating...

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

```
>>> 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/`
`1GMMrDXzYFMFRn9ufrVUGb0vSBGO7VkV6GLAdu46CVzA/`
`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 to you to know how to use decorators that someone else has written

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

LAB

- Re-write the properties from the `Circle` class to use the decorator syntax (see a couple slides back for an example)
  (`circle_properties.py` and
  `test_circle_properties.py`)

- 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`)

Wrap up

A better underdatnidng of the underpining 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!