

Introduction to Python Decorators – Debugging – Packages and Packaging

Christopher Barker

UW Continuing Education / Isilon

August 22, 2012

Table of Contents

- 1 Review/Questions
- 2 Decorators
- 3 Debugging
- 4 Packages and Packaging

Review of Previous Class

Lightning talk today: Peter

- Some more OO
 - Multiple inheritance / mix-ins
 - Properties
 - `staticmethod` and `classmethod`
 - Special methods (“dunder”)
- Iterators
- Generators

Homework review

Who added some classes to some “real” code?

- Multiple inheritance / mix-ins ?
- Property ?
- `staticmethod` or `classmethod` ?
- Special methods ?
- Iterator or Generators ?

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: `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:

`memoize.py`

Decorator examples

Examples from the stdlib:

Does this structure:

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

look familiar from last class?

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):
        """I'm the 'x' property."""
        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

LAB



Lightning Talk

Lightning Talk:

Peter

First Topic

A topic

some code example

LAB



First Topic

A topic

some code example

LAB



Wrap up



Homework

