Decorators**¶**

Agenda**¶**

* The concept
* The need
* The syntax
* Commonly available decorators
  + classmethod
  + staticmethod
  + ...

The Concept**¶**

Adding functionality to a function

*Without* modifying the function

func(func) -> func

Function that takes a function (as arg) and returns another function (usually with functionality added)

Function arg**¶**

In [ ]:

def add(x,y):  
 return x+y  
  
def sub(x,y):  
 return x-y  
  
def apply(f,x,y):  
 return f(x,y)

In [ ]:

apply(add,2,3)

In [ ]:

apply(sub,5,2)

Other common functions that take functions?

* map
* reduce
* filter

In [ ]:

Function return**¶**

or, function in a function

In [ ]:

def outer():  
 x = 1  
 def inner():  
 print x  
 return inner  
  
foo = outer()  
foo()

In [ ]:

def outer(x):  
 def inner():  
 print x  
 return inner  
  
print1 = outer(1)  
print2 = outer(2)

In [ ]:

print1()

In [ ]:

print2()

In [ ]:

In [ ]:

def outer(func):  
 def inner():  
 print "Before"  
 func()  
 print "After"  
 return inner  
  
def foo():  
 print "Hi"  
   
foo = outer(foo)  
foo()

Decorator!**¶**

In [ ]:

def italics(fn):  
 def fn2():  
 return '<i>' + fn() + '</i>'  
 return fn2  
  
@italics  
def msg():  
 return "Hello"  
  
  
msg = italics(msg)  
  
msg()

In [ ]:

def bold(fn):  
 def fn2():  
 return '<b>' + fn() + '</b>'  
 return fn2  
  
@italics  
@bold  
def msg():  
 return "Hello"  
  
msg()

In [ ]:

\*args and \*\*kwargs**¶**

In [ ]:

def fn(x,y,\*args):  
 print x,y,args  
   
fn(1,2)

In [ ]:

fn(1,2,3,4,5)

In [ ]:

def f():  
 print "Hello"  
   
def f(x):  
 print "Hello"+x

In [ ]:

f()

In [ ]:

In [ ]:

def fn2(\*\*kwargs):  
 print kwargs  
   
fn2(x=1,y=2)

In [ ]:

In [ ]:

def logger(fn):  
 def inner(\*args, \*\*kwargs):  
 print "Agrs were " + str(args) + str(kwargs)  
 return fn(\*args, \*\*kwargs)  
 return inner  
  
@logger  
def msg(x):  
 return "Hello"+x  
  
@logger  
def msg2(x,y):  
 return x,':',y  
  
msg('MrX')

In [ ]:

msg2("hi",5)

In [ ]:

In [ ]:

Class Decorators**¶**

In [ ]:

registry = { }  
  
def register(cls):  
 registry[cls.\_\_clsid\_\_] = cls  
 return cls

In [ ]:

@register  
class Foo(object):  
 \_\_clsid\_\_ = "123-456"   
 def bar(self):  
 pass

In [ ]:

class Foo(object):  
 \_\_clsid\_\_ = "123-456"   
 def bar(self):  
 pass  
  
register(Foo)

Both are exactly the same

Decorator is just syntactic sugar

In [ ]:

pwd

In [ ]:

In [ ]: