

# Class Decorators Radically Simple

Jack Diederich  
PyCon UK 2008

# Updates, Post PyCon UK

- change 3<sup>rd</sup> party decorator module example to use standard functools instead.

# How to get a new passport, *FAST*

(U.S. Edition)

- Make an appointment at passport office
- Bring airline ticket
- Bring identification docs
- Bring \$160
- Wait 2 hours
- Done!

# Decorators

- Functions that accept one argument
- Return *Anything*

```
def decorator(arg):  
    return 'Hello World!'
```

```
@decorator  
def my_func(): pass
```

```
def my_func(): pass  
my_func = decorator(my_func)
```

# Decorator History

- Function decorators in Python 2.4
- Grammar changed in 2.6 and 3.0 to

decorated: decorators (classdef | funcdef)

# Decorators and Metaclasses

- Decorator is once, metaclass is always
- Interface

```
def decorator(ob):  
    return ob
```

```
class MyMeta(type):  
    def __new__(meta, name, bases, dict):  
        return type.__new__(meta, name, bases, dict)  
    def __init__(cls, name, bases, dict): pass
```

# Identity Decorator

```
def identity(ob):  
    return ob
```

```
@identity  
def my_func(a, b, c): pass
```

```
@identity  
class my_class(object): pass
```

# Nested Decorator

```
def ignore_args(func):  
    def replacement_func(*args, **opts):  
        return func()  
    return replacement_func
```

```
@ignore_args  
def hello_world():  
    print("Hello World")
```

# Nested Decorator (w/ args)

```
def ignore_n_args(n):  
    def wrapper_func(func):  
        def replacement_func(*args):  
            return func(*[n:])  
        return wrapper_func
```

```
@ignore_n_args(3)  
def hello_world(*args):  
    print("Hello World", len(args))
```

# Tidy Decorators

[NB, this slide was edited after PyCon UK]

```
import functools
```

```
# use update_wrapper to wrap our decorator
```

```
@functools.update_wrapper
```

```
def identity(ob):
```

```
    return ob
```

```
@identity # now preserves signature and docstring
```

```
def func(a, b):
```

```
    """ return a + b """
```

```
    return a + b
```

# Popular Patterns

- Verify
- Register
- Munge/Transform

# Verify Pattern

```
def assert_candy(cls):  
    assert cls.sweet == True  
    assert cls.calories > 100  
    return cls
```

```
@assert_candy  
class ChocolateBar(object):  
    sweet = True  
    calories = 200
```

# Registration Pattern

```
import cron
```

```
@cron.schedule(cron.O_DARK_THIRTY)
```

```
class SalesReport(Report):
```

```
    def run(self):
```

```
        # do stuff
```

```
class SalesReport(Report, metaclass=cron.meta):
```

```
    cron_when = O_DARK_THIRTY
```

# Registration Pattern

```
class Factory():  
    def __init__(self):  
        self.all = []  
    def register(self, cls):  
        self.all.append(cls)  
        return cls
```

```
animals = Factory()
```

```
@animals.register  
class Horse(): pass
```

```
@animals.register  
class Cow(): pass
```

```
assert animals.all == [Horse, Cow]
```

# Registration Metaclass

```
def new_factory_type():
    class FactoryMeta(type):
        all = []
        def __init__(cls, name, bases, dict):
            FactoryMeta.all.append(cls)
            return type.__init__(cls, name, bases, dict)
    return FactoryMeta
```

```
animals = new_factory_type()
```

```
class Horse(metaclass=animals): pass
class Sheep(metaclass=animals): pass
```

```
assert animals.all == [Horse, Sheep]
```

# Munge Pattern

```
def stop_writing_java(cls):
    """ de-privatize access to self.__bar attributes """
    private = '_' + cls.__name__ + '_'

    def new_getattr(self, key):
        if key.startswith(private):
            key = key[len(private):]
        return self.__dict__[key]

    def new_setattr(self, key, value):
        if key.startswith(private):
            key = key[len(private):]
        self.__dict__[key] = value

    cls.__getattr__ = new_getattr
    cls.__setattr__ = new_setattr
    return cls
```

# Munge Pattern

```
@stop_writing_java  
class Java():  
    def __init__(self):  
        self.__sekret = True
```

```
>>> ob = Java  
>>> print(ob.__sekret)  
True
```

# Munge Pattern

cherrypy/\_\_\_init\_\_\_.py

```
class AttributeDocstrings(type):
    def ___init__(cls, name, bases, dict):
        for (k, v) in dict.items():
            if k.endswith('___doc'):
                delattr(cls, k)
                cls. ___doc__ += '\n\n' + v
        return cls
```

# Best Practices

- Return the original class
- Don't assume you are the only decorator
- Maybe you want a metaclass
- Don't be a dick
  - `class Foo:`
    - `class __metaclass__(type): pass`
  - don't define `__slots__` in your metaclass

# Links

jackdied.blogspot.com # slides, python blog

<http://www.ibm.com/developerworks/linux/library/l-cpdecor.html>

“Decorators Make Magic Easy” by David Mertz

<http://www.phyast.pitt.edu/~micheles/python/documentation.html>

decorator module by Michele Simionato

Jack Diederich  
jackdied@gmail.com  
PyCon UK 2008

(I do python stuff for money)