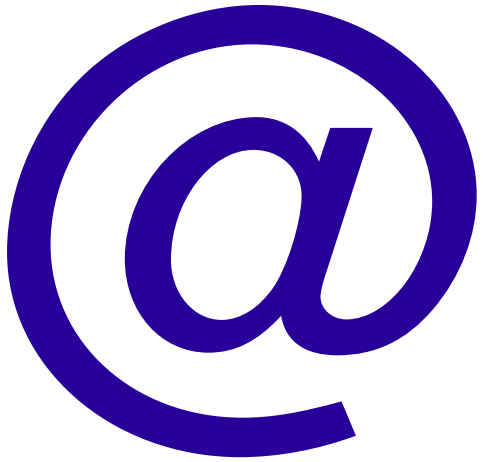

Decorators From Basics to Decorator Libraries to Class Decorators

Charles Merriam
charles.merriam@gmail.com

What's A Decorator?

A design choice to separate two concepts, when one concept modifies when or why the other concept will execute

Good Decorators

"simple to understand"

```
@classmethod

@register_url

@require_admin

@log
```

```
def update_price(item, new_price):
    if require_manager():
        item.price = new_price
    else:
        log("Attempt to update_price")
```

```
@require_manager
def update_price(Item, new_price):
    - item.price = new_price
```

```
def update_price(item, new_price):
    if require_manager():
       item.price = new_price
   else:
       log("Attempt to update_price")

def find_tax(price):
    return price * TAX_RATE

def mark_for_clearance(Item)
    if require_manager():
        item.price *= 0.25
   else:
        log("Attempt to clearance")

def discount_price(item, discount):
    if require_manager():
        item.price *= (1-discount)
   else:
      log("Attempt to update_price")
```

```
@require_manager
def update_price(item, new_price):
   item.price = new_price

def find_tax(price):
    return price * TAX_RATE

@require_manager
def mark_for_clearance(Item)
    item.price *= 0.25

@require_manager
def discount_price(item, discount):
    item.price *= (1-discount)
```

How it works

"Concrete Decorator": A function. It takes a function as input and returns a function as output.

```
@concrete
def func(arg1):
    ....
```

```
def func(arg1):
    ....
func =
concrete(func)
```

Simple “Call Once” Decorator

```
def register(function):
  print “LOOK! “, function.__name__
  return function

@register
def print_hello():
  print “Hello”

print_hello()
print_hello()
```

More Complicated

```
@log_usage
@require("manager")
def markdown():  ...
```

```
def markdown(): …
req_dec = require("manager")
req_markdown = req_dec(markdown)
markdown = log_usage(req_markdown)
```

Hard Way, nested

```
def require(level):
    def take_params(function):
        def concrete(*args, **kwargs):
            if check_me(level):
                return function(*args,
                    **kwargs)
            return None
        return concrete
    return take_params
```

Hard Way, with Class

```
class require(object):

    def __init__(level):
        self.level = level
    def __call__(function):
        self.function = function
        return self.dec
    def dec(self, *args, **kwargs):
        if check_me(self.level):
            -return
              self.function(*args,
              **kwargs)
        return None
```

Or use dectools

```
import dectools

@dectools.make_call_if

def require(function, args, kwargs,
            level):

    return check_me(level)
```

Standard Decorator Stuff

- Security
- Install Check
- Pre/Post Conditions
- Login Required
- Framework & Callback Registration
- Lazy Setup
- Locks
- Memoize/Cache
- Trace/Log/Stats
- Contract Programs
- Lock Management
- Type Checking

If/Before/After/Once/Instead

- Security
- Install Check

- Pre/Post Conditions
- Login Required

- Framework Registration
- Lazy Setup

- Locking and Atomic
- Memoize/Cache
- Trace/Log/Stats
- Contract Programming
- Type Checking

DecTools

- Clean API
- Signatures
- Lab-ware

easy_install dectools.py

```
@make_call_if/before/after/instead
def my_picture(function, args, kwargs, arg1):
    ...

@make_call_once
def just_once(function, args, kwargs, arg1):
```

Makes decorators:

```
@my_picture
@just_once
```

```
@invariant
class Item(object):
    @post("DBItem.find(name)", globals(), locals())
    def __init__(self, name, price):
        self.name, self.price = name, price
       DBAdd(name, price)

    def _invariant(self):
       assert self.price >=  0 and len(self.name)

    @log
    @pre("adjustment < 0")
    def adjust_price(self, adjustment):
        self.price += adjustment
```

Thank you!

- Hope this answers questions!
- Time for Questions
- charles.merriam@gmail.com
- http://charlesmerriam.com

Class Decorators

- A function that takes a class as an argument and returns a class as a return value.

How it works

Class Decorator: A function. It takes a class as input and returns a class as output.

```
@classdec

class C():....
```

```
Class C():....

C = classdec(C)
```

Class Decorator Stuff

- Framework & Callback Registration
- Contract Programming
- Apply Decorator To Each Item
- Dictionary Transmogrify
- Non-inheritance Mix-in Madness
- Non-class Descriptors