Python Decorators

Chris Calloway

Triangle Zope and Python Users Group: TriZPUG

What is a Decorator?

An object.

Triangle Zope and Python Users Group: TriZPUG

What is a Decorator?

An object.

A callable object which is passed a function reference as its sole argument.

Triangle Zope and Python Users Group: TriZPUG

What is a Decorator?

An object.

A callable object which is passed a function reference as its sole argument.

The return value of the callable object rebinds the function's identifier.

Triangle Zope and Python Users Group: TriZPUG

What is a Decorator?

An object.

A callable object which is passed a function reference as its sole argument.

The return value of the callable object rebinds the function's identifier.

The return value of the callable object is usually another function.

Triangle Zope and Python Users Group: TriZPUG

Why Decorators?

A way to add or replace define-time or run-time attributes and behavior in functions and methods.

Triangle Zope and Python Users Group: TriZPUG

Why Decorators?

A way to add or replace define-time or run-time attributes and behavior in functions and methods.

A way to register, annotate, wrap... decorate... a function.

Triangle Zope and Python Users Group: TriZPUG

Why Decorators?

Kind of like superclassing a function in a weird kind of way.

Kind of like mixins are for classes.

But not really.

Triangle Zope and Python Users Group: TriZPUG

What is a Decorator?

The result of PEP 318.

Created June 2003

Last modified January 2005

Realized in Python 2.4 November 2004

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
funcdef ::= [decorators] "def" funcname "(" [parameter_list]
                                        ")" ":" suite

decorators ::= decorator+

decorator ::= "@" dotted_name ["(" [argument_list [","]] ")"]
              NEWLINE

dotted_name ::= identifier ("." identifier)*

parameter_list ::=  (defparameter ",")*
                    (~~"*" identifier [, "**" identifier]
                     | "**" identifier
                     | defparameter [","] )

defparameter ::= parameter ["=" expression]
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

The expression is evaluated to return a callable

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

The expression is evaluated to return a callable

Python remembers the callable object in an internal reference

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

The expression is evaluated to return a callable

Then a function is defined

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

The expression is evaluated to return a callable

Then after that the rememberd callable is invoked with the function as the sole argument

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

is the form of a *decorator expression*

The expression is evaluated to return a callable

Then after that the return value of the callable is rebound to the function's identifier

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference
```

If the decorator expression has no arguments, then the decorator expression is a reference to the decorator itself

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

If the decorator expression has arguments, then the decorator expression evaluates a callable factory for decorators, passing the arguments to the decorator factory

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@someReference[(*args)]
```

Calling the decorator factory returns a decorator

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
@dec1
def func(arg1, arg2, ...):
    pass
```

is equivalent to:

```
def func(arg1, arg2, ...):
    pass
func = dec1(func)
```

Triangle Zope and Python Users Group: TriZPUG

What do decorators look like?

```
@dec2
@dec1
def func(arg1, arg2, ...):
    pass
```

is equivalent to:

```
def func(arg1, arg2, ...):
    pass
func = dec2(dec1(func))
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
def dec1(func):
    print "decorating", func
    return func
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
def dec1(func):
    print "decorating", func
    return func

print "before definition"
@dec1
def f(x):
    print "f of", x
print "after definition"
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
print "before definition"
@dec1
def f(x):
    print "f of", x
print "after definition"
```

outputs:

```
before definition
decorating <function f at 0x54fe70>
after definition
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator look like?

```
before definition
decorating <function f at 0x54fe70>
after definition

>>> f(1)
f of 1
>>>
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator factory look like?

```
>>> def factory(name):
...     def dec(func):
...         func.attribute = name
...         return func
...     return dec
...
>>>
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator factory look like?

```
>>> @factory("what's in a name")
... def f(x):
...     print "f of", x
...
>>> f(1)
f of 1
>>> f.attribute
"what's in a name"
>>>
```

Triangle Zope and Python Users Group: TriZPUG

What does a decorator factory look like?

A decorator factory can have arguments.

Triangle Zope and Python Users Group: TriZPUG

What does a decorator factory look like?

A decorator factory can have arguments.

A decorator factory's arguments are separate and distinct from the decorator's sole (function) argument.

Triangle Zope and Python Users Group: TriZPUG

What does a decorator factory look like?

A decorator factory can have arguments.

A decorator factory's arguments are separate and distinct from the decorator's sole (function) argument.

"Baking in" a decorator factory's arguments into a decorator definition is a form of a programming pattern called a CLOSURE

Triangle Zope and Python Users Group: TriZPUG

Annotating a function with attributes

```
def author(name):
    def decorator(func):
        func.author = name
        return func
    return decorator

@author("Mark Pilgrim")
def diveIntoPython():
    print "Great book!"
```

Triangle Zope and Python Users Group: TriZPUG

Does a decorator have to return a function?

```
>>> def stringThing(func):
...     return "thing"
...
>>> @stringThing
... def printSomeThing():
...     print "something"
...
>>>
```

Triangle Zope and Python Users Group: TriZPUG

Does a decorator have to return a function?

```
>>> printSomeThing()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object is not callable
>>>
```

Triangle Zope and Python Users Group: TriZPUG

Does a decorator have to return a function?

```
>>> printSomeThing()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object is not callable
>>> printSomeThing
'thing'
>>>
```

Triangle Zope and Python Users Group: TriZPUG

Does a decorator have to return a function?

```
>>> printSomeThing()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object is not callable
>>> printSomeThing
'thing'
>>> type(printSomeThing)
<type 'str'>
>>>
```

Triangle Zope and Python Users Group: TriZPUG

Is it useful for a decorator to return something besides a function?

```
>>> class traced:
...     def __init__(self, func):
...         self.func = func
...     def __call__(self, *args, **kwargs):
...         print "entering", self.func
...         try:
...             return self.func(*args, **kwargs)
...         finally:
...             print "exiting", self.func
...
>>>
```

Triangle Zope and Python Users Group: TriZPUG

Is it useful for a decorator to return something besides a function?

```
>>> @traced
... def hello():
...     print "my pretty"
...
>>> hello()
entering <function hello at 0x558770>
my pretty
exiting <function hello at 0x558770>
>>>
```

Triangle Zope and Python Users Group: TriZPUG

What if an outer decorator expression expects a function?

```
def traced(func):
    def wrapper(*args, **kwargs):
        print "entering", func
        try:
            return func(*args, **kwargs)
        finally:
            print "exiting", func
    return wrapper
```

Triangle Zope and Python Users Group: TriZPUG

What if an outer decorator expression expects a function?

"Baking in" an outer function's free variables into an inner function definition is a form of a programming pattern called a LEXICAL CLOSURE

Triangle Zope and Python Users Group: TriZPUG

Can a decorator factory return a decorator which defines a wrapper?

```
def factory(expr):
    def ensure(func):
        def wrapper(*args, **kwargs):
            assert eval(expr), "Uh-oh"
            return func(*args, **kwargs)
        return wrapper
    return ensure
```

Triangle Zope and Python Users Group: TriZPUG

Can a decorator factory return a decorator which defines a wrapper?

```
@factory("len(args) == 1")
def printOne(*args):
    print args[0]
```

Triangle Zope and Python Users Group: TriZPUG

Can a decorator factory return a decorator which defines a wrapper?

```
>>> printOne("TriZPUG")
TriZPUG
>>> printOne("TriZPUG", "HZPUG")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in wrapper
AssertionError: Uh-oh
>>>
```

Triangle Zope and Python Users Group: TriZPUG

How do I practice "safe decorating?"

```
def factory(expr):
    def ensure(func):
        def wrapper(*args, **kwargs):
            assert eval(expr), "Uh-oh"
            return func(*args, **kwargs)
        wrapper.__name__ = func.__name__
        wrapper.__dict__ = func.__dict__
        wrapper.__doc__ = func.__doc__
        return wrapper
    return ensure
```

Triangle Zope and Python Users Group: TriZPUG

How do I practice "safe decorating?"

Always return a function from a decorator

Always copy the old functions attributes to the new function

Triangle Zope and Python Users Group: TriZPUG

Acknowledgments

Dive Into Python (closures)

Python in a Nutshell (function attributes)

Phillip Eby in Dr. Dobbs (decorators)

Python Reference Manual (functions, PEP)

Triangle Zope and Python Users Group: TriZPUG