Decorators Demystified

Anand Chitipotu

Python India 2014 - Sept 26, 2014

In [2]:
```
def square(x):
    return x*x

print square(4)
```
```
16
```

In [3]:
```
print square
```
```
<function square at 0x10b3ccaa0>
```

In [4]:
```
f = square
```

In [5]:
```
print f
```
```
<function square at 0x10b3ccaa0>
```

In [6]:
```
print f(4)
```
```
16
```

In [7]:
```
#square = <new-function-here>
```

In [8]:
```
def sum_of_squares(x, y):
    return square(x) + square(y)
```

In [9]:
```
sum_of_squares(3, 4)
```

Out[9]:
```
25
```

In [10]:
```
def cube(x):
    return x*x*x

def sum_of_cubes(x, y):
    return cube(x) + cube(y)

print sum_of_cubes(3, 4)
```

```
91
```

In [11]:
```
def sum_of(f, x, y):
    return f(x) + f(y)

print sum_of(square, 3, 4)
print sum_of(cube, 3, 4)
print sum_of(abs, 3, -4)
```

```
25
91
7
```

```
In [12]: def mod3(x):
             return x % 3

         print sum_of(mod3, 4, 8)
```

```
3
```

```
In [13]: print sum_of(lambda x: x%3, 4, 8)
```

```
3
```

```
In [14]: print sum_of(lambda x: x*x*x, 4, 8)
```

```
576
```

```
In [15]: max(3, 4)
```

```
Out[15]: 4
```

```
In [16]: max(["Python", "Java"])
```

```
Out[16]: 'Python'
```

```
In [18]: max(["Python", "Haskell"])
```

```
Out[18]: 'Python'
```

```
In [19]: max(["Python", "Haskell"], key=len)
```

```
Out[19]: 'Haskell'
```

```
In [22]: names = ["C", "Java", "C++", "Perl", "Python", "Ruby", "Haskell"]
         sorted(names)

Out[22]: ['C', 'C++', 'Haskell', 'Java', 'Perl', 'Python', 'Ruby']
```

```
In [23]: sorted(names, key=len)

Out[23]: ['C', 'C++', 'Java', 'Perl', 'Ruby', 'Python', 'Haskell']
```

```
In [24]: len("hello")

Out[24]: 5
```

Problem Implement a function `maximum` that takes 2 values `x` and `y` and a `key` function as argument and finds the maximum by comparing `key(x)` and `key(y)`.

```
>>> maximum(3, -4, abs)
-4
>>> maximum("Python", "Haskell", len)
'Haskell'
>>> maximum("java", "Python", lambda s: s.lower())
'Python'
```

```
In [27]: max("java", "Python",)

Out[27]: 'java'
```

```
In [28]: max("java", "Python", key=lambda s: s.lower())

Out[28]: 'Python'
```

```
In [29]: def maximum(x, y, key):
             if key(x) > key(y):
                 return x
             else:
                 return y
```

```
In [30]: print maximum(3, -4, abs)
```

```
-4
```

```
In [31]: maximum("java", "Python", lambda s: s.lower())
```

```
Out[31]: 'Python'
```

Default Arguments

```
In [32]: def incr(x, amount=1):
             return x+amount

         print incr(4)
```

```
5
```

```
In [33]: print incr(4, amount=2)
         print incr(4, 2)
```

```
6
6
```

```
In [34]: def sub(x, y):
             return x-y

         print sub(3, 2)

         1
```

```
In [36]: print sub(x=3, y=2)

         1
```

```
In [37]: print sub(y=2, x=3)

         1
```

```
In [39]: print sub(3, y=2)

         1
```

Variable number of arguments and keyword arguments

```
In [40]: max(1, 2, 3)

Out[40]: 3
```

```
In [41]: max(1, 2, 3, 4)

Out[41]: 4
```

In [43]:
```
def f(*a):
    print a

f()
f(1)
f(1, 2)
f(1, 2, 3)
```

```
()
(1,)
(1, 2)
(1, 2, 3)
```

In [44]:
```
def xprint(label, *args):
    for a in args:
        print label, a

xprint("INFO", 1, 2, 3)
```

```
INFO 1
INFO 2
INFO 3
```

Problem Implement a function add that takes variable number of arguments and returns their sum.

Hint: You can use built-in function `sum` for computing sum of a list of numbers.

```
>>> add(1, 2, 3)
6
>>> add(1, 2, 3, 4)
```

```
10
```

Problem Write a function `strjoin` that takes a separator as first argument followed by variable number of strings to join with that separator.

```
>>> strjoin("-", "a", "b", "c")
"a-b-c"
```

Just like variable arguments, we can write functions that can take arbitrary keyword arguments.

In [45]:
```
def f(**kwargs):
    print kwargs
```

In [46]:
```
f(x=1, y=2)
```

```
{'y': 2, 'x': 1}
```

In [54]:
```
def render_tag(tagname, **attrs):
    pairs = ['%s="%s"' % (k, v) for k, v in attrs.items()]
    pairs_str = " ".join(pairs)
    return "<%s %s>" % (tagname, pairs_str)

print render_tag("a",
                 href="http://in.pycon.org/",
                 title="PyCon India 2014")
```

```
<a href="http://in.pycon.org/" title="PyCon India 2014">
```

```
In [55]: def f(x, y):
             return x+y
```

```
In [56]: def call_func(f, args):
             return f(*args)

         print call_func(square, [3])
         print call_func(sum_of_squares, [3, 4])
```

```
9
25
```

```
In [60]: def call_func1(f, *args):
             return f(*args)

         print call_func1(square, 3)
         print call_func1(sum_of_squares, 3, 4)
```

```
9
25
```

```
In [61]: def call_func1(f, *args, **kwargs):
             return f(*args, **kwargs)

         print call_func1(square, 3)
         print call_func1(sum_of_squares, 3, 4)
         print call_func1(square, x=3)
         print call_func1(sum_of_squares, x=3, y=4)
```

```
9
25
9
25
```

In [62]:
```
print call_func1(sum_of_squares, 3, y=4)
```

```
25
```

Functions as return value

In [63]:
```
def make_adder(x):
    def add(y):
        return x+y
    return add

add5 = make_adder(5)
print add5(2)
```

```
7
```

In [64]:
```
data = [["A", 10], ["B", 34], ["C", 5]]
print max(data)
```

```
['C', 5]
```

In [69]:
```
def column(n):
    def f(row):
        return row[n]
    return f

print max(data, key=column(1))
```

```
['B', 34]
```

Decorators

In [76]:
```
%%file sum.py

def square(x):
    print "square", x
    return x*x

def sum_of_squares(x, y):
    print "sum_of_squares", x, y
    return square(x) + square(y)

if __name__ == "__main__":
    print sum_of_squares(3, 4)
```

```
Overwriting sum.py
```

In [77]:
```
!python sum.py
```

```
sum_of_squares 3 4
square 3
square 4
25
```

In [109]:

```
%%file trace0.py

def trace(f):
    def g(*args):
        print f.__name__, args
        return f(*args)
    return g
```

```
Overwriting trace0.py
```

In [105]:
```
%%file sum1.py

from trace0 import trace

@trace
def square(x):
    return x*x

# @trace is same as:
# square = trace(square)

@trace
def sum_of_squares(x, y):
    return square(x) + square(y)

if __name__ == "__main__":
    print sum_of_squares(3, 4)
    print square
```

```
Overwriting sum1.py
```

In [106]:
```
!python sum1.py
```

```
sum_of_squares (3, 4)
square (3,)
square (4,)
25
<function g at 0x102aecd70>
```

In [101]:
```
%%file blackhole.py

def blackhole(f):
    return 0

@blackhole
def square(x):
    return x*x

print square
```

```
Writing blackhole.py
```

In [102]:
```
!python blackhole.py
```

```
0
```

In [122]:
```
%%file trace1.py
import functools
import os

level = 0
def trace(f):
    if os.getenv("DEBUG") != "true":
        return f

    @functools.wraps(f)
    def g(*args):
        global level
        print "|  " * level + "|--", f.__name__, args

        level += 1
        result = f(*args)
        level -= 1
        return result

    #functools.update_wrapper(f, g)
    return g
```

```
Overwriting trace1.py
```

In [125]:
```
%%file sum2.py

from trace1 import trace

@trace
def square(x):
    return x*x

# @trace is same as:
# square = trace(square)

@trace
def sum_of_squares(x, y):
    return square(x) + square(y)

if __name__ == "__main__":
    print sum_of_squares(3, 4)
```

```
Overwriting sum2.py
```

In [126]:
```
!python sum2.py
```

```
25
```

In [127]:
```
!DEBUG=true python sum2.py
```

```
|-- sum_of_squares (3, 4)
|   |-- square (3,)
|   |-- square (4,)
25
```

In [133]:
```
%%file fib0.py
from trace1 import trace

@trace
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

if __name__ == "__main__":
    import sys
    n = int(sys.argv[1])
    print fib(n)
```

```
Overwriting fib0.py
```

In [137]:
```
!DEBUG=true python fib0.py 4
```

```
|-- fib (4,)
|  |-- fib (3,)
|  |  |-- fib (2,)
|  |  |  |-- fib (1,)
|  |  |  |-- fib (0,)
|  |  |-- fib (1,)
|  |-- fib (2,)
|  |  |-- fib (1,)
|  |  |-- fib (0,)
5
```

Problem Write a function `with_retries` that continue to retry for 5 times if there is any exception raised in the function.

```
@with_retries
def wget(url):
    return urllib2.urlopen(url).read()

wget("http://google.com/no-such-page")
```

Should print:

```
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Giving up!
```

In [145]:
```
%%file with_retries.py

import urllib2
import functools

def with_retries(f):
    @functools.wraps(f)
    def g(*args):
        for i in range(5):
            try:
                return f(*args)
            except:
                print "Failed to download, retrying..."
        print "Giving up!"
    return g

@with_retries
def wget(url):
    return urllib2.urlopen(url)

x = wget("http://google.com/no-such-page")
```

```
Overwriting with_retries.py
```

In [146]:
```
!python with_retries.py
```

```
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Giving up!
```

In [156]:
```
%%file memoize.py

def memoize(f):
    # create a cache for remembering return values
    cache = {}

    def g(*args):
        # if the function is not called before with those arguments
        if args not in cache:
            # call it now and remember the result.
            cache[args] = f(*args)
        # return the remembered result
        return cache[args]
    return g
```

```
Overwriting memoize.py
```

In [157]:

```
%%file fib1.py
from trace1 import trace
from memoize import memoize

@memoize
@trace
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

if __name__ == "__main__":
    import sys
    n = int(sys.argv[1])
    print fib(n)
```

```
Overwriting fib1.py
```

In [150]:

```
!DEBUG=true python fib1.py 10
```

```
|-- fib (10,)
|  |-- fib (9,)
|  |  |-- fib (8,)
|  |  |  |-- fib (7,)
|  |  |  |  |-- fib (6,)
|  |  |  |  |  |-- fib (5,)
|  |  |  |  |  |  |-- fib (4,)
|  |  |  |  |  |  |  |-- fib (3,)
|  |  |  |  |  |  |  |  |-- fib (2,)
|  |  |  |  |  |  |  |  |  |-- fib (1,)
|  |  |  |  |  |  |  |  |  |-- fib (0,)
89
```

In [155]:
```
!DEBUG=true python fib0.py 6
```

```
|-- fib (6,)
|  |-- fib (5,)
|  |  |-- fib (4,)
|  |  |  |-- fib (3,)
|  |  |  |  |-- fib (2,)
|  |  |  |  |  |-- fib (1,)
|  |  |  |  |  |-- fib (0,)
|  |  |  |  |-- fib (1,)
|  |  |  |-- fib (2,)
|  |  |  |  |-- fib (1,)
|  |  |  |  |-- fib (0,)
|  |  |-- fib (3,)
|  |  |  |-- fib (2,)
|  |  |  |  |-- fib (1,)
|  |  |  |  |-- fib (0,)
|  |  |  |-- fib (1,)
|  |-- fib (4,)
|  |  |-- fib (3,)
|  |  |  |-- fib (2,)
|  |  |  |  |-- fib (1,)
|  |  |  |  |-- fib (0,)
|  |  |  |-- fib (1,)
|  |  |-- fib (2,)
|  |  |  |-- fib (1,)
|  |  |  |-- fib (0,)
13
```

Now lets try to make the `with_retries` function take the number of retries as argument.

In [164]:
```
%%file with_retries1.py

import urllib2
import functools

def with_retries(num_retries):
    # with_retries is not a decorator.
    # the return value of with_retries is a decorator.
    def decor(f):
        @functools.wraps(f)
        def g(*args):
            for i in range(num_retries):
                try:
                    return f(*args)
                except:
                    print "Failed to download, retrying..."
            print "Giving up!"
        return g
    return decor

@with_retries(3)
def wget(url):
    return urllib2.urlopen(url)

#decor = with_retries(3)
#wget = decor(wget)

x = wget("http://google.com/no-such-page")
```

```
Overwriting with_retries1.py
```

In [165]:
```
!python with_retries1.py
```

```
Failed to download, retrying...
Failed to download, retrying...
Failed to download, retrying...
Giving up!
```

Example: A web framework

In [189]:
```
%%file fakeweb.py

mapping = []

def route(path):
    def decor(f):
        mapping.append((path, f))
    return decor

def request(path):
    for p, func in mapping:
        if p == path:
            return func()
    return "not found"

def wsgifunc(env, start_response):
    path = env['PATH_INFO']
    start_response('200 OK', [("Content-type", "text/plain")])
    return request(path)

def run(port=8080):
    from wsgiref.simple_server import make_server
    server = make_server("localhost", port, wsgifunc)
    server.serve_forever()
```

```
Overwriting fakeweb.py
```

In [191]:
```
%%file hello.py
from fakeweb import route

@route("/hello")
def hello():
    return "Hello, world!"

@route("/bye")
def bye():
    return "Good bye!"

# @after_request
# def layout(response):
#     line = "\n" + "=" * 10 + "\n"
#     return line + response + line
```

```
Overwriting hello.py
```

In [187]:
```
%%file client.py
import hello
from fakeweb import request, run
import sys

if __name__ == "__main__":
    if "--web" in sys.argv:
        run()
    else:
        print request("/hello")
        print request("/bye")
```

```
Overwriting client.py
```

In [190]:
```
!python client.py
```

```
Hello, world!
Good bye!
```

In [ ]:
```

```