Primer on Python Decorators

In this introductory tutorial, we'll look at what decorators are and how to create and use them.

First things first

How functions work

functions return a value based on the given arguments

```
def foo(bar):
  return bar + 1

print foo(2) == 3
```

First Class Objects

first-class (http://python-history.blogspot.com/2009/02/first-class-everything.html) objects

```
def foo(bar):
  return bar+1

print foo
print foo(2)
print type(foo)

def call_foo_with_arg(foo, arg):
  return foo(arg)

print call_foo_with_arg(foo, 3)
```

Nested Functions

define functions inside other functions

```
def parent():
  print "Printing from the parent() function."

  def first_child():
      return "Printing from the first_child() function."

  def second_child():
      return "Printing from the second_child() function."

  print first_child()
  print second_child()
```

`parent()`

```
Printing from the parent() function.
Printing from the first_child() function.
Printing from the second_child() function
```

`first_child()`

```
Traceback (most recent call last):
File "decorator3.py", line 15, in <module>
first_child()
NameError: name 'first_child' is not defined
```

What have we learned?

parent() first_child() second_child()

Returning Functions

return functions from other functions

```
def parent(num):

  def first_child():
      return "Printing from the first_child() function."

  def second_child():
      return "Printing from the second_child() function."

  try:
      assert num == 10
      return first_child
  except AssertionError:
      return second_child

foo = parent(10)
bar = parent(11)

print foo
print bar

print foo()
print bar()
```

```
<function first_child at 0x1004a8c08>
<function second_child at 0x1004a8cf8>
```

foo first_child() bar second_child()

```
Printing from the first_child() function.
Printing from the second_child() function.
```

```
            second_child()
              first_child
```

Now, my friend, you are ready to take on decorators!

Example 1:

```
def my_decorator(some_function):

  def wrapper():

      print "Something is happening before some_function() is called."

      some_function()

      print "Something is happening after some_function() is called."

  return wrapper

def just_some_function():
  print "Wheee!"


just_some_function = my_decorator(just_some_function)

just_some_function()
```

```
Something is happening before some_function() is called.
Wheee!
Something is happening after some_function() is called.
```

Put simply, decorators wrap a function, modifying its behavior.

Example 2:

```
def my_decorator(some_function):

  def wrapper():

      num = 10

      if num == 10:
          print "Yes!"
      else:
          print "No!"

      some_function()

      print "Something is happening after some_function() is called."

  return wrapper

def just_some_function():
 print "Wheee!"

just_some_function = my_decorator(just_some_function)

just_some_function()
```

```
Yes!
Wheee!
Something is happening after some_function() is called.
```

Time for some syntactic sugar!

`@`

Let’s create a module for our decorator:

```
def my_decorator(some_function):

  def wrapper():

      num = 10

      if num == 10:
          print "Yes!"
      else:
          print "No!"

      some_function()

      print "Something is happening after some_function() is called."

  return wrapper

if __name__ == "__main__":
  my_decorator()
```

```
from decorator7 import my_decorator

@my_decorator
def just_some_function():
  print "Wheee!"

just_some_function()
```

```
Yes!
Wheee!
Something is happening after some_function() is called.
```

`@my_decorator` `just_some_function = my_decorator(just_some_function)`

Real World

```
import time

def timing_function(some_function):

    """
    Outputs the time a function takes
    to execute.
    """

    def wrapper():
        t1 = time.time()
        some_function()
        t2 = time.time()
        return "Time it took to run the function: " + str((t2-t1)) + "\n"
    return wrapper

@timing_function
def my_function():
    num_list = []
    for x in (range(0,10000)):
        num_list.append(x)
    print "\nSum of all the numbers: " +str((sum(num_list)))


print my_function()
```

`my_function()`

```python
from time import sleep


def sleep_decorator(function):

    """
    Limits how fast the function is
    called.
    """

    def wrapper(*args, **kwargs):
        sleep(2)
        return function(*args, **kwargs)
    return wrapper


@sleep_decorator
def print_number(num):
    return num

print print_number(222)

for x in range(1,6):
    print print_number(x)
```

`login_required()`

`/secret`

```python
from functools import wraps
from flask import g, request, redirect, url_for

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if g.user is None:
            return redirect(url_for('login', next=request.url))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/secret')
@login_required
def secret():
    pass
```

Cheers!

Want to learn more? Download the Real Python course.

« Python Web Applications With Flask - Part III (/blog/python/python-web-applications-with-flask-part-iii/)

Primer on Jinja Templating » (/blog/python/primer-on-jinja-templating/)

Comments

3 Comments Real Python Login

Recommend Share Sort by Best

Join the discussion…

ddaypunk06 • 2 months ago

A good summation of decorators, time to use em!

3 • Reply • Share ›

michaelherman Mod ddaypunk06 • 2 months ago

Cheers!

• Reply • Share ›

Seth Williams • a year ago

A useful lesson on Python Decorators. Thanks a lot. You will find some more challenging stuff here http://www.fireboxtraining.com....

• Reply • Share ›

ALSO ON **REAL PYTHON** WHAT'S THIS?

Deploying a Django App to AWS Elastic Beanstalk

12 comments • 2 months ago

Eric — I am a long time Heroku user but since my personal app is finally reaching the 10k row limit, now I have to learn …

Django Rest Framework - class based views

9 comments • 5 months ago

Guess I'm Easily Annoyed — Sorry for such a nit-pick... that damn "Page Contents" area on the right of your page that follows me …

Docker in Action - fitter, happier, more productive

11 comments • a month ago

John McMahon — Hey Mike! Nice article. Glad to see you're doing so well with RealPython.I've been working on getting …

Setting up a Simple OCR Server

9 comments • 2 months ago

alexeiramone — Very nice post, thanks.

Subscribe Add Disqus to your site Privacy

Categories

-
-

-
-
-
-
-
-
-
-
-
-
-
-
-
-