Python Decorator Cheatsheet

Friday, February 13, 2015 (permalink)

I can never remember the syntax for writing decorators. I always have to look it up. Worse, I always have to remember where to look to find references. Hence the reason for this article. I'll never lose this reference: It's on my laptop and the internet.

Each type will include a basic version, a `functools.wraps` version, and a wrapt version.

Decorators Without Arguments

These are decorators that do not accept arguments.

```
import functools  # Part of Python standard library

def decorator(wrapped_function):
    def _wrapper(*args, **kwargs):
        # do something before the function call
        result = wrapped_function(*args, **kwargs)
        # do something after the function call
        return result
    return _wrapper

# decorator with functools.wraps added
def decorator_with_wraps(wrapped_function):
    @functools.wraps(wrapped_function)
    def _wrapper(*args, **kwargs):
        # do something before the function call
        result = wrapped_function(*args, **kwargs)
        # do something after the function call
        return result
    return _wrapper

import wrapt  # Requires installing the 'wrapt' library

# decorator powered by wrapt
@wrapt.decorator
def decorator_with_wrapt(wrapped_function, instance, args, kwar
gs):
    # do something before the function call
    result = wrapped_function(*args, **kwargs)
    # do something after the function call
    return result
```

```
def test_decorators():

    @decorator
    def func1():
        return 'I'

    @decorator_with_wraps
    def func2():
        return 'code'

    @decorator_with_wrapt
    def func3():
        return 'python'

    assert func1() == 'I'
    assert func2() == 'code'
    assert func3() == 'python'
```

Decorators With Arguments

These are decorators that accept arguments.

```
def arguments_decorator(arg1, arg2):
    def _outer_wrapper(wrapped_function):
        def _wrapper(*args, **kwargs):
            # do something before the function call
            result = wrapped_function(*args, **kwargs)
            # do something after the function call

            # Demonstrating what you can do with decorator argu
ments
            result = result * arg1 * arg2

            return result
        return _wrapper
    return _outer_wrapper

def arguments_decorator_with_wraps(arg1, arg2):
    def _outer_wrapper(wrapped_function):
        @functools.wraps(wrapped_function)
        def _wrapper(*args, **kwargs):
            # do something before the function call
            result = wrapped_function(*args, **kwargs)
            # do something after the function call

            # Demonstrating what you can do with decorator argu
ments
            result = result * arg1 * arg2

            return result
        return _wrapper
    return _outer_wrapper

def arguments_decorator_with_wrapt(arg1, arg2):
    @wrapt.decorator
    def _wrapper(wrapped_function, instance, args, kwargs):
        # do something before the function call
        result = wrapped_function(*args, **kwargs)
        # do something after the function call
```

At the end of a so-called 'age of peace', two great nations of immortals march against each other, with humanity caught in the middle. In this world of conflict, three very different individuals are thrust into the face of danger.

Follow Me

Twitter

Github

Facebook

Subscribe To My Feed

While you're here, why not add my atom feed to your RSS reader?

Tags

pypi usability eventbrite capoeira Los Angeles i18n clojure ppoftw jinja2 meteor europe djangocon family howto pycon-2013-guide vs dsf pyladies perl postgresql cookiecutter eskrima markdown paypal pyramid review flask testing consumernotebook pydiversity colombia packaging Consumer-Notebook pumpkin philippines sprint tools lahackthons binstar gondor cryptocurrency dotcloud holidays surgery functions LaTeX travel recipe hackathon php django-rest-framework friends casestudy blog pypy los-angeles la joke twoscoops coreapi WhartonWC class-based-views training halloween brambles ingredients resolutions lahackathons setup Cartwheel Web rails rant ruby wsgi OAuth mongodb book bitcoin nasa australia nodejs javascript science api travel tips for geeks RestructuredText cheatsheet pycon djangodash django python python3 forms meme conda heroku audrey unicode LA argentina

```
        # Demonstrating what you can do with decorator argument
s
        result = result * arg1 * arg2

        return result
    return _wrapper


def test_arguments_decorators():

    @arguments_decorator(2, 3)
    def func4():
        return 'We'

    @arguments_decorator_with_wraps(2, 2)
    def func5():
        return 'code'

    @arguments_decorator_with_wrapt(3, 2)
    def func6():
        return 'python'

    assert func4() == 'WeWeWeWeWeWe'
    assert func5() == 'codecodecodecode'
    assert func6() == 'pythonpythonpythonpythonpythonpython'
```

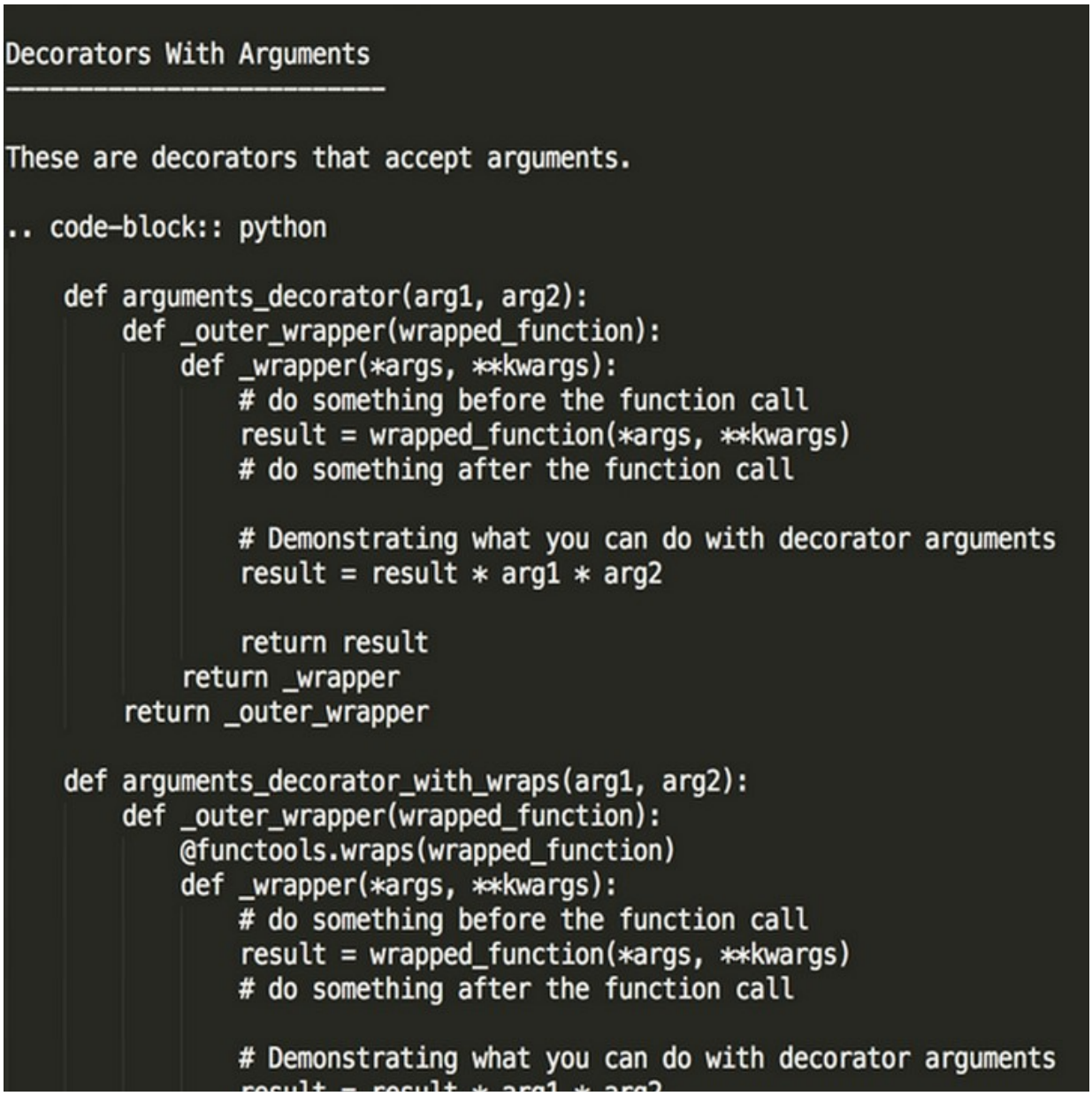
Summary

This article is a cheatsheet, not a tutorial.

Instead of explaining why Python has decorators, how to use them, how they work, or why to use them, this article is a reference. Nothing more.

References:

- Graham Dumpleton's voluminious series on decorators
- Graham Dumpleton's Introspecting a function article on decorators for concerns about `functools.wraps`)
- https://wiki.python.org/moin/PythonDecoratorLibrary



```
Decorators With Arguments
-------------------------

These are decorators that accept arguments.

.. code-block:: python

    def arguments_decorator(arg1, arg2):
        def _outer_wrapper(wrapped_function):
            def _wrapper(*args, **kwargs):
                # do something before the function call
                result = wrapped_function(*args, **kwargs)
                # do something after the function call

                # Demonstrating what you can do with decorator arguments
                result = result * arg1 * arg2

                return result
            return _wrapper
        return _outer_wrapper

    def arguments_decorator_with_wraps(arg1, arg2):
        def _outer_wrapper(wrapped_function):
            @functools.wraps(wrapped_function)
            def _wrapper(*args, **kwargs):
                # do something before the function call
                result = wrapped_function(*args, **kwargs)
                # do something after the function call

                # Demonstrating what you can do with decorator arguments
                result = result * arg1 * arg2
```

Tags: *python* *python3* *cheatsheet* *ppoftw*

Subscribe!

If you read this far, you might want to follow me on twitter or github and subscribe via email below (I'll email you new articles when I publish them).

Email Address Subscribe

Comments