

Python Decorators

Justin Findlay
<http://jfindlay.us/>
jfindlay@gmail.com

2016 September 7

Confusion

¹<https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-decorators/>

Confusion

- ▶ Many confusions and confusing explanations on decorators¹

¹<https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-decorators/>

Confusion

- ▶ Many confusions and confusing explanations on decorators¹
- ▶ We need to start with Python's object model and follow the implications it has for metaprogramming with functions

¹<https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-decorators/>

Confusion

- ▶ Many confusions and confusing explanations on decorators¹
- ▶ We need to start with Python's object model and follow the implications it has for metaprogramming with functions
- ▶ Then we will see that decorators are merely a corollary result of this design

¹<https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-decorators/>

Python Functions are Objects

- ▶ In Python, everything is an object

Python Functions are Objects

- ▶ In Python, everything is an object
- ▶ In particular functions are objects

```
>>> def f():  
...     print('called')  
...  
>>> g = f  
>>> g()  
called
```

Metaprogramming

Metaprogramming

- ▶ **Programming:** apply logic to data

Metaprogramming

- ▶ **Programming:** apply logic to data
- ▶ **Metaprogramming:** apply logic to logic, or work with logic (code) as if it were data

Metaprogramming

- ▶ **Programming:** apply logic to data
- ▶ **Metaprogramming:** apply logic to logic, or work with logic (code) as if it were data
- ▶ Simple Examples of metaprogramming
 - ▶ file templating
 - ▶ macros
 - ▶ makefiles

Metaprogramming

- ▶ **Programming:** apply logic to data
- ▶ **Metaprogramming:** apply logic to logic, or work with logic (code) as if it were data
- ▶ Simple Examples of metaprogramming
 - ▶ file templating
 - ▶ macros
 - ▶ makefiles
- ▶ Nonsimple Examples of metaprogramming
 - ▶ C++ templates
 - ▶ Python decorators

Python Object Model

- ▶ What metaprogramming features do we need for decorators to work? \forall functions f, g, h :

Python Object Model

- ▶ What metaprogramming features do we need for decorators to work? \forall functions f, g, h :
 1. Assignment: $g = f$

Python Object Model

- ▶ What metaprogramming features do we need for decorators to work? \forall functions f, g, h :
 1. Assignment: $g = f$
 2. Argument: $g(f)$

Python Object Model

- ▶ What metaprogramming features do we need for decorators to work? \forall functions f, g, h :
 1. Assignment: $g = f$
 2. Argument: $g(f)$
 3. Map: `return h`

Python Object Model

- ▶ What metaprogramming features do we need for decorators to work? \forall functions f, g, h :
 1. Assignment: $g = f$
 2. Argument: $g(f)$
 3. Map: `return h`
- ▶ All of these operations work for any Python object, including callable objects (functions), so decorators can be used in Python

Decorator

Still wondering what a decorator is?

²See wikipedia references at the end of the presentation.

Decorator

Still wondering what a decorator is?

- ▶ What the `@decorator` is a decorator?

Decorator

Still wondering what a decorator is?


- ▶ What the `@decorator` is a decorator?
- ▶ There is plenty of theory on decorator and associated models²

²See wikipedia references at the end of the presentation.

Decorator

Still wondering what a decorator is?


- ▶ What the `@decorator` is a decorator?
- ▶ There is plenty of theory on decorator and associated models²
- ▶ We only care about an informal, working definition for the Python construct

²See wikipedia references at the end of the presentation. 

Decorator

Still wondering what a decorator is?


- ▶ What the `@decorator` is a decorator?
- ▶ There is plenty of theory on decorator and associated models²
- ▶ We only care about an informal, working definition for the Python construct
- ▶ **Decorator:** A callable (function) that accepts a function as an argument and returns a function

²See wikipedia references at the end of the presentation. 

Decorator

Still wondering what a decorator is?

- ▶ What the `@decorator` is a decorator?
- ▶ There is plenty of theory on decorator and associated models²
- ▶ We only care about an informal, working definition for the Python construct
- ▶ **Decorator:** A callable (function) that accepts a function as an argument and returns a function
- ▶ Usually the returned function is a convenient synonym or simple generalization of the function argument

²See wikipedia references at the end of the presentation. 

Implementing a Decorator

- ▶ Now that we've proven decorators can be implemented in Python, let's try to find the simplest possible implementation

```
>>> def italic(text_func):  
...     def italicize(*a, **kw):  
...         return '<i>{}</i>'.format(text_func(*a, **kw))  
...     return italicize  
...  
>>> def format_text(text):  
...     return text  
...  
>>> format_text = italic(format_text)  
>>> format_text('Learning Decorators')  
'<i>Learning Decorators</i>'
```


Decoration

- ▶ Having defined the *italic* decorator on the previous slide, the code

```
>>> def format_text(text):  
...     return text  
...  
>>> format_text = italic(format_text)
```

is equivalent to

```
>>> @italic  
>>> def format_text(text):  
...     return text  
...
```

and thus we see that using the *@italic* syntax is merely a convenience or shortcut for a simple function metaprogramming construct

Decorator Properties/Examples

- ▶ Stacking/nesting decorators

³<https://stackoverflow.com/a/1594484>

Decorator Properties/Examples

- ▶ Stacking/nesting decorators
- ▶ decorator sandwich³

```
>>> def bread(func):  
...     def wrapper():  
...         print('<=====>')  
...         func()  
...         print('<=====>')  
...     return wrapper  
...  
>>> def vegetables(func):  
...     def wrapper():  
...         print('#tomatoes#')  
...         func()  
...         print('~leaves~')  
...     return wrapper  
...  
>>> @vegetables  
... @bread  
... def sandwich(food='bean-product'):  
...     print(food)  
...  
>>> sandwich()  
#tomatoes#  
<=====>  
bean-product  
<=====>  
~leaves~
```

³<https://stackoverflow.com/a/1594484>

Decorator Properties/Examples

- ▶ Decorated arguments: We have seen that arguments pass naturally from the inner, constructed function to the decorated function, the decorator works only with the function object itself and so does not care about its arguments, because it does not call it

Decorator Properties/Examples

- ▶ Decorated arguments: We have seen that arguments pass naturally from the inner, constructed function to the decorated function, the decorator works only with the function object itself and so does not care about its arguments, because it does not call it
- ▶ Decorator arguments: Nothing prevents you from passing additional arguments to the decorator, including additional functions

Decorator Properties/Examples

- ▶ Decorated arguments: We have seen that arguments pass naturally from the inner, constructed function to the decorated function, the decorator works only with the function object itself and so does not care about its arguments, because it does not call it
- ▶ Decorator arguments: Nothing prevents you from passing additional arguments to the decorator, including additional functions
- ▶ Decorating classes: Decorators can be applied to class methods and object methods as well as classes themselves in any case using the `@decorator` convenience syntax

Decorator Properties/Examples

- ▶ Decorating a decorator: Again, due to the generality of the function metaprogramming provided by Python's object model, nothing prevents you from decorating a decorator: a decorator which returns a decorator

Decorator Properties/Examples

- ▶ Decorating a decorator: Again, due to the generality of the function metaprogramming provided by Python's object model, nothing prevents you from decorating a decorator: a decorator which returns a decorator
- ▶ Can a decorator decorate itself: decorator recursion (an exercise left for the meetup participant)

Further Examples

- ▶ creating class methods or static methods
- ▶ adding function attributes
- ▶ tracing
- ▶ setting pre- and postconditions
- ▶ synchronization
- ▶ tail recursion elimination
- ▶ memoization
- ▶ improving the writing of decorators
- ▶ properties
- ▶ contextmanager
- ▶ functools
- ▶ python wiki on decorators

References

- ▶ <https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-decorators/>
- ▶ <https://stackoverflow.com/a/1594484>
- ▶ https://en.wikipedia.org/wiki/Python_syntax_and_semantics#Decorators
- ▶ https://en.wikipedia.org/wiki/Decorator_pattern
- ▶ https://en.wikipedia.org/wiki/Advice_%28programming%29
- ▶ <https://wiki.python.org/moin/PythonDecoratorLibrary>