LECTURE 8

DECORATORS

MCS 275 Spring 2021
Emily Dumas

LECTURE 8: DECORATORS

Course bulletins:

- Project 1 posted. Deadline 6pm CST on Fri Feb 5.
- Project 1 autograder opens on Monday.
- Quiz 2 solutions and grades posted.
- Quiz 3 will be posted Monday at Noon, due Tuesday at Noon (all times CST).

PLAN

Discuss a Python language feature that allows us to attach "modifiers" to functions, called **decorators**.

This feature is never required, but it sometimes leads to code that is easier to read and understand.

(Some Python modules, e.g. `Flask`, are meant to be used primarily through decorators.)

FUNCTION ARGUMENTS

Functions in Python can accept functions as arguments.

```python
def dotwice(f):
    """Call function f twice"""
    f()
    f()
```

A better version works with functions that accept arguments:

```python
def dotwice(f,*args,**kwargs):
    """Call function f twice"""
    f(*args,**kwargs)
    f(*args,**kwargs)
```

RETURNING FUNCTIONS

Functions in Python can return functions. Often this is used with a return value that is a defined inside the function body, making a "function factory".

```
def return_power(n):
    def inner(x): # function inside a function!
        """Raise x to a power"""
        return x**n
    return inner
```

MODIFYING FUNCTIONS

```python
def return_twice_doer(f):
    """Return a new function which calls f twice"""
    def inner(*args,**kwargs):
        """Call a certain function twice"""
        f(*args,**kwargs)
        f(*args,**kwargs)
    return inner
```

REPLACING FUNCTIONS

In some cases we might want to replace an existing function with a modified version of it (e.g. as returned by some other function).

```
def g(x):
    """Print the argument with a message"""
    print("Function got value",x)

# actually, I wanted to always print that message twice!
g = return_twice_doer(g)
```

DECORATOR SYNTAX

There is a shorter syntax to replace a function with a modified version.

```
@modifier
def fn(x,y):
    """Function body goes here"""
```

is equivalent to

```
def fn(x,y):
    """Function body goes here"""
fn = modifier(fn)
```

The symbol `@modifier` (or any `@name`) before a function definition is called a **decorator**.

RETURNING VALUES

Usually, the inner function of a decorator should return the value of the (last) call to the argument function.

```python
def return_twice_doer(f):
    """Return a new function which calls f twice"""
    def inner(*args,**kwargs):
        """Call a certain function twice"""
        f(*args,**kwargs)
        return f(*args,**kwargs)
    return inner
```

DECORATOR ARGUMENTS

Python allows `@decorator(arg1,arg2,...)`.

```
@dec(2)
def printsq(x):
    print(x*x)
```

is equivalent to

```
thisdec = dec(2)

@thisdec
def printsq(x):
    print(x*x)
```

In other words, if a decorator is given arguments, then the name after `@` is expected to be a **decorator factory**.

A FEW BUILT-IN DECORATORS

- **`@functools.lru_cache(100)`** -- Save arguments and return values for up to 100 recent calls to a function; reuse stored return values when possible. Good for expensive operations.*
- **`@classmethod`** -- Make a method a class method (callable from the class itself, gets class as first argument). E.g. for alternate constructors.
- **`@atexit.register`** -- Ask that this function be called just before the program exits.

* In Python 3.9+ there is also the simpler `functools.cache` decorator which stores an unlimited number of past function calls..

MULTIPLE DECORATORS

Each must be on its own line.

```python
@dec1
@dec2
@dec3
def f(x):
    """Function body goes here"""
```

replaces `f` with `dec1(dec2(dec3(f)))`.

So the decorator closest to the function name acts first.

REFERENCES

- See *Lutz*, Chapter 39 for a detailed discussion of Python decorators.
- See *Beazley & Jones*, Chapter 9 for several examples of decorators.

ACKNOWLEDGMENT

- I reviewed course materials created by Danko Adrovic (UIC MSCS faculty member) while preparing this lecture.

REVISION HISTORY

- 2021-01-30 Fix accidental use of Python 3.9 feature (`functools.cache`)
- 2021-01-28 Initial publication