Лекция 6
Декораторы

23 марта 2017 г.

Декораторы (создание)

Пример создания декоратора

```
def notify_about_calls(func):
    def decorated(*args, **kwargs):
        print "Called", func.__name__
        return func(*args, **kwargs)
    return decorated
```

Пример создания декоратора

```
def notify_about_calls(func):
    def decorated(*args, **kwargs):
        print "Called", func.__name__
        return func(*args, **kwargs)
    return decorated

def f(a, b):
    return a + b
g = notify_about_calls(f)
print g(1, 2)
```

Пример создания декоратора

```
def notify_about_calls(func):
    def decorated(*args, **kwargs):
        print "Called", func.__name__
        return func(*args, **kwargs)
    return decorated

def f(a, b):
    return a + b
g = notify_about_calls(f)
print g(1, 2)
```

```
Called f
3
```

Упрощенная запись

```
@decorate
def f(a, b):
    return a + b
```

Эквивалентно

```
def f(a, b):
    return a + b
f = decorate(f)
```

Несколько декораторов

```
@decorator1
@decorator2
def f(a, b):
    return a + b
```

Эквивалентно

```
def f(a, b):
    return a + b
f = decorator1(decorator2(f))
```

Классы-декораторы

```
class Logger(object):
    def __init__(self, func):
        self.func = func
        self.log = []
    def __call__(self, *args, **kwargs):
        self.log.append((args, kwargs))
        return self.func(*args, **kwargs)

logged = Logger
```

Классы-декораторы

```
@logged
def f(x, y=0):
    pass

f(1)
f(1, y=2)
print f.log
```

```
[((1,), {}), ((1,), {'y': 2})]
```

Метаданные функций

```
def notify_about_calls(func):
    def decorated(*args, **kwargs):
        print "Called", func.__name__
        return func(*args, **kwargs)
    return decorated
```

Метаданные функций

```
@notify_about_calls
def some_function(x, y):
    """Docstring."""
    return x + 2 * y
```

Метаданные функций

```
@notify_about_calls
def some_function(x, y):
    """Docstring."""
    return x + 2 * y

some_function(1, 2)
print some_function.__name__
print some_function.__doc__
```

```
Called some_function
decorated
None
```

functools.wraps

```
import functools

def notify_about_calls(func):
    @functools.wraps(func)
    def decorated(*args, **kwargs):
        print "Called", func.__name__
        return func(*args, **kwargs)
    return decorated
```

functools.wraps

```
@notify_about_calls
def some_function(x, y):
    """Docstring."""
    return x + 2 * y
```

functools.wraps

```
@notify_about_calls
def some_function(x, y):
    """Docstring."""
    return x + 2 * y

some_function(1, 2)
print some_function.__name__
print some_function.__doc__
```

```
Called some_function
some_function
Docstring.
```

Декоратор `staticmethod`

`staticmethod`

- Применяется к методу класса
- Делает метод статическим
- Позволяет игнорировать экземпляр (`self`)

Декоратор staticmethod

```
class A(object):
    @staticmethod
    def f(a, b):
        return a + b

a = A()
print a.f(1, 2)
print A.f(1, 2)
```

```
3
3
```

Декораторы с аргументами

Как сделать такой декоратор?

```
@check_return_type(float)
def calculate_something(a, b, c):
    ...
    return x
```

Декораторы с аргументами

Как сделать такой декоратор?

```
@check_return_type(float)
def calculate_something(a, b, c):
    ...
    return x


def check_return_type(type_):
    def decorator(func):
        def decorated(*args, **kwargs):
            val = func(*args, **kwargs)
            assert isinstance(val, type_)
            return val
        return decorated
    return decorator
```