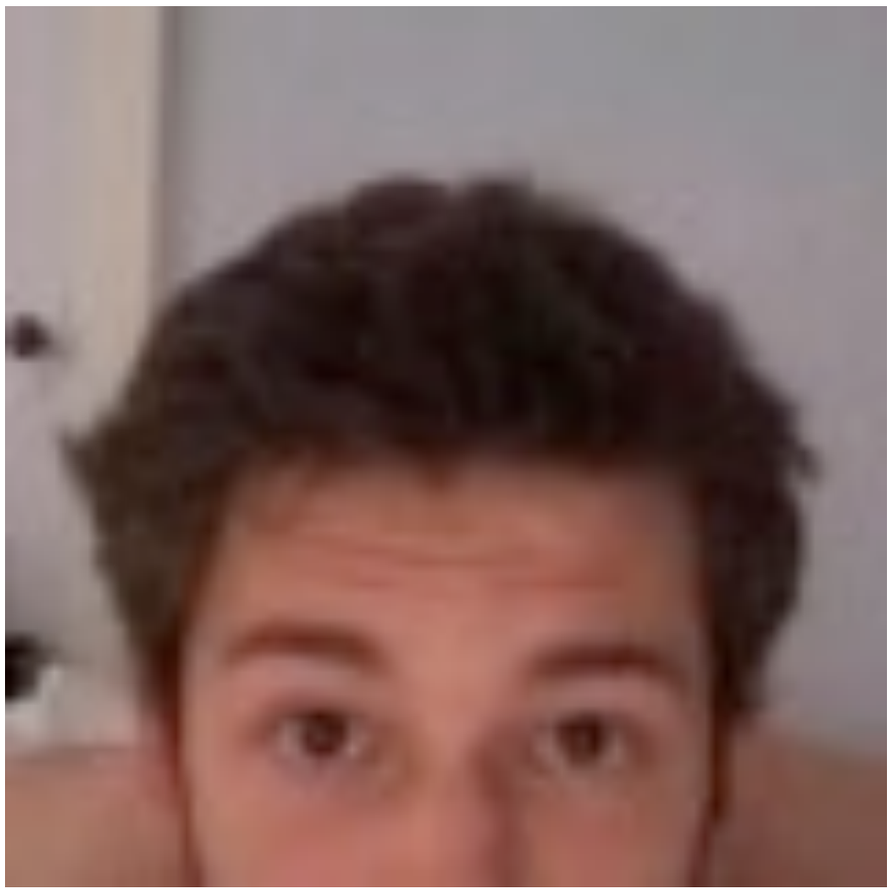
decorations.py

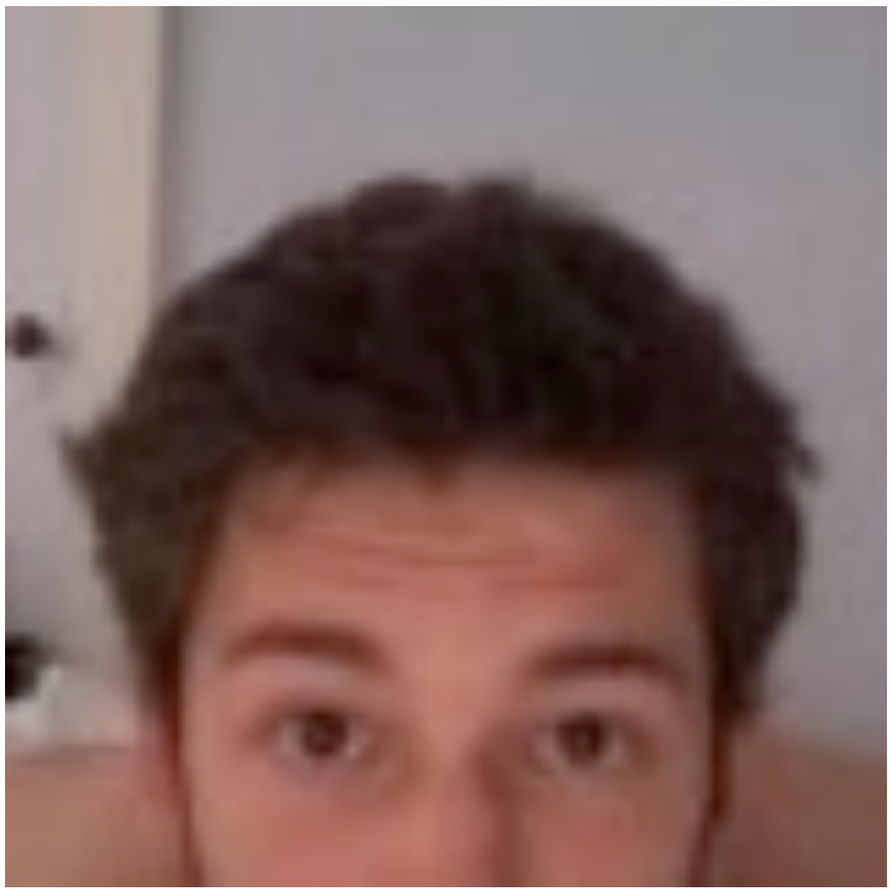
decorations.py

yay!

Me



Me



yay!

I <3 decorators

I <3 decorators

You should too.

I <3 decorators

You should too.

- DRY

I <3 decorators

You should too.

- DRY
- Abstraction

I <3 decorators

You should too.

- DRY
- Abstraction
- Simple, clean code

Decorator

From PEP-318:

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns
```

Decorator

From PEP-318:

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns

@returns((int,float))
def func(arg1, arg2):
    return arg1 * arg2
```

Decoratee

- From PEP-318:

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns

@returns((int,float))
def func(arg1, arg2):
    return arg1 * arg2
```

Decoration?

Decoration ⊂ Decorator

From PEP-318:

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns
```

Decoration ⊂ Decorator

From PEP-318:

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns
```

Decoration ⊂ Decorator

From PEP-318:

CRUFT

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns
```

Decoration ⊂ Decorator

From PEP-318:

CRUFT

```
def returns(rtype):
    def check_returns(f):
        def new_f(*args, **kwds):
            result = f(*args, **kwds)
            assert isinstance(result, rtype), \
                   "return value %r does not match %s" % (result,rtype)
            return result
        new_f.func_name = f.func_name
        return new_f
    return check_returns
```

DECORATION

Decoration!

```
def returns(rtype):
```

Decoration!!

```
@postdecoration
def returns(result, rtype):
```

Decoration!!!

```
@postdecoration
def returns(result, rtype):
    assert isinstance(result, rtype), \
            "return value %r does not match %s" % (result,rtype)
```

yay! :)

Two problems

Two problems

- Two common kinds of decorators

Two problems

- Two common kinds of decorators
- Evolution of decorators

- Two common kinds of decorators

@pre

- Do something before decoratee executes

@pre @post

- Do something before decoratee executes
- Do something after decoratee executes

- Evolution of decorators

Evolution of decorators

- @my_decorator

Evolution of decorators

- @my_decorator => @my_decorator(...)

Evolution of decorators

TWO PROBLEMS

Evolution of decorators

TWO PROBLEMS

- First, extending the decoration naively breaks all current uses of the decoration.

Evolution of decorators

TWO PROBLEMS

- First, extending the decoration naively breaks all current uses of the decoration.
- Second, it requires intricate changes to the decorator - an extra closure must be added to capture the decorator arguments.

</problems>

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...
```

```
@foo
def bar(c, d, e, f):
    # ...
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...

@foo(1, 2, a=3, b=4)
def baz(c, d, e, f):
    # ...
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...

@foo
def bar(c, d, e, f):
    # ...

@foo(1, 2, a=3, b=4)
def baz(c, d, e, f):
    # ...
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...

@foo
def bar(c, d, e, f):
    # ...

@foo(1, 2, a=3, b=4)
def baz(c, d, e, f):
    # ...

bar(5, 6, e=7, f=8)
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...

@foo
def bar(c, d, e, f):
    # ...
```

```
@foo(1, 2, a=3, b=4)
def baz(c, d, e, f):
    # ...

bar(5, 6, e=7, f=8)
baz(9, 0, e=10, f=11)
```

```
@predecoration
def foo(f, args, kwargs, dec_params, dec_kwargs):
    # ...

@foo
def bar(c, d, e, f):
    # ...
```

```
@foo(1, 2, a=3, b=4)
def baz(c, d, e, f):
    # ...

bar(5, 6, e=7, f=8)
baz(9, 0, e=10, f=11)
```

Real world?

GET arguments

```
def my_django_view(request):
    some_get_var = request.GET['some_get_var']
    other_get_var = request.GET['other_get_var']
    # ...
```

I
acceptsGET/0

```
@predecoration
def acceptsGET(f, args):
    fargs = inspect.getargspec(f).args
    get = args[0].GET
    return util.filter_dict(get, fargs[1:])

@acceptsGET
def my_django_view(request, some_get_var, other_get_var):
    # ...
```

II
redirects_to_referer

```
@decoration
def redirects_to_referer(f, args, kwargs):
    f(*args, **kwargs)
    return HttpResponseRedirect(args[0].META.get('HTTP_REFERER'))
```

II
redirects_to_referer

```
@decoration
def redirects_to_referer(f, args, kwargs):
    f(*args, **kwargs)
    return HttpResponseRedirect(args[0].META.get('HTTP_REFERER'))

@redirects_to_referer
@postargs
def updatefoos(req, foo_1, foo_2):
    # ...
```

III

postargs/n | n>0

```
@redirects_to_referer
@postargs(foos="foo_(\d+)")
def updatefoos(req, foos):
    # ...
```

THE END

THE END!

The end!

http://github.com/andreyf/decorations

anfedorov@gmail.com

@anfedorov