Python

Algis Dumbris

Version 1.0

Table of Contents

Functions

Decorators

```
import functools

def p_decor(func):
    func.__doc__ += "Wrapped"
    @functools.wraps(func)
    def inner(name):
        print("Before fun")
        return func(name)
    return inner

@p_decor
def t_fun(name):
    """
    T fun doc string
    """
    print("Hello %s" % name)


if __name__ == '__main__':
    help(t_fun)
    t_fun("Vitas")
```

Class method decorator

```
import functools

def m_decor(func):
    @functools.wraps(func)
    def inner(*args, **kvargs):
        return "<p>{}</p>".format(func(*args, **kvargs))
    return inner

def param_decor(p1, p2):
    def m_decor(func):
        @functools.wraps(func)
        def inner(*args, **kvargs):
            return "<{0}><{1}>{2}</{1}></{0}>".format(p1, p2, func(*args, **kvargs))
        return inner
    return m_decor

class Person():
    def __init__(self, fname, sname):
        self.fname = fname
        self.sname = sname

    @m_decor
    def get_fullname(self):
        return "{} {}".format(self.fname, self.sname)

    @param_decor("p","span")
    def get_fname(self):
        return "{}".format(self.fname)

    def __str__(self):
        return self.get_fullname()

if __name__ == '__main__':
    p = Person("Vitas", "Doo")
    print(p)
    print(p.get_fname())
```

- Output

```
<p>Vitas Doo</p>
<p><span>Vitas</span></p>
```

Contex manager

```
import contextlib


class LikeFile():
    def __init__(self):
        self.string = """
        first string
        second string
        other string
        <END>
        """

    def __enter__(self):
        print("begin")
        return self

    def __exit__(self, *args):
        print("exit")
        print("{}".format(" ".join([str(i) for i in args])))
        return False


if __name__ == '__main__':
    with LikeFile() as lf:
        print(type(lf))
        print(lf.string)

    print("well done")
```

- Output

```
begin
<class '__main__.LikeFile'>

        first string
        second string
        other string
        <END>

exit
None None None
well done
```

Classes

Decorators

```
VAL_ORIG = 1
VAL_OVER = 2

class Decor():
    def __init__(self, arg):
        self.arg = arg

    def __call__(self, cls):
        class OC(cls):
            """Overrided class"""
            classattr = self.arg

            def meth(self):
                return VAL_OVER

            def meth2(self):
                return self.classattr

        return OC


@Decor("some text")
class MyClass():
    def __init__(self):
        pass

    def meth(self):
        return VAL_ORIG


def test_classdecor():
    mc = MyClass()
    assert mc.meth() != VAL_ORIG
    assert mc.meth() == VAL_OVER
    assert mc.meth2() == "some text"
```

Delegation

```
class upcase:
    def __init__(self, out):
        self._out = out

    def write(self, s):
        self._outfile.write(s.upper())

    def __getattr__(self, name):
        return getattr(self._out, name)


if __name__ == '__main__':
    up = upcase(int)
    print(str(`1`))
```

Diamond

```
class A(object):
    def __init__(self):
        print('Running A.__init__')
        super(A,self).__init__()
    def t(self):
        return "A"
class B(A):
    def __init__(self):
        print('Running B.__init__')
        super(B,self).__init__()
        #A.__init__(self)
    def t(self):
        return "B"

class C(A):
    def __init__(self):
        print('Running C.__init__')
        super(C,self).__init__()
    def t(self):
        return "C"
#class D(C,B):
class D(B,C):
    def __init__(self):
        print('Running D.__init__')
        super(D,self).__init__()
    def t2(self):
        return "D"

foo=D()
print(foo.t())
```

- Output

```
Running D.__init__
Running B.__init__
Running C.__init__
Running A.__init__
B
```

Super

```
class Person:
    def __init__(self, name):
        self.name = name

    def showname(self):
        return self.name


class Programmer(Person):
    def __init__(self, title):
        super(Programmer, self).__init__('unk')
        self.title = title

    def showname(self):
        return self.name


def test_name():
    prog = Programmer('senior')
    assert 'unk' == prog.showname()

def test_update_name():
    prog = Programmer('senior')
    prog.name = "Me"
    assert 'Me' == prog.showname()
```