这是一篇关于 python 装饰器的博文

在学习 python 的过程中处处受阻，之前的学习中 Python 的装饰器学习了好几遍也没能真正的弄懂。这一次抓住视频猛啃了一波，就连 python 大佬讲解装饰器起来也需要大半天的时间。据说是两个老师轮流讲解，还得吃药才能讲完。详情请见老男孩的第 14 期视频 day4。

听完两位老师的讲解，我算是系统而又全面的理解了装饰器，并将其总结。

一、定义

想要彻底的弄清楚装饰器，就得彻底的弄明白 Python 装饰器的定义，百科里的定义说：装饰器是一个很著名的设计模式，经常被用于有切面需求的场景，较为经典的有插入日志、性能测试、事务处理等。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量函数中与函数功能本身无关的雷同代码并继续重用。之前的都是扯淡，具体的等深入后才能了解，我们目前只需要明白装饰器的作用就是为已经存在的对象添加额外的功能。

简单来说，装饰器的作用就是给已经存在的函数附加功能。

二、原则

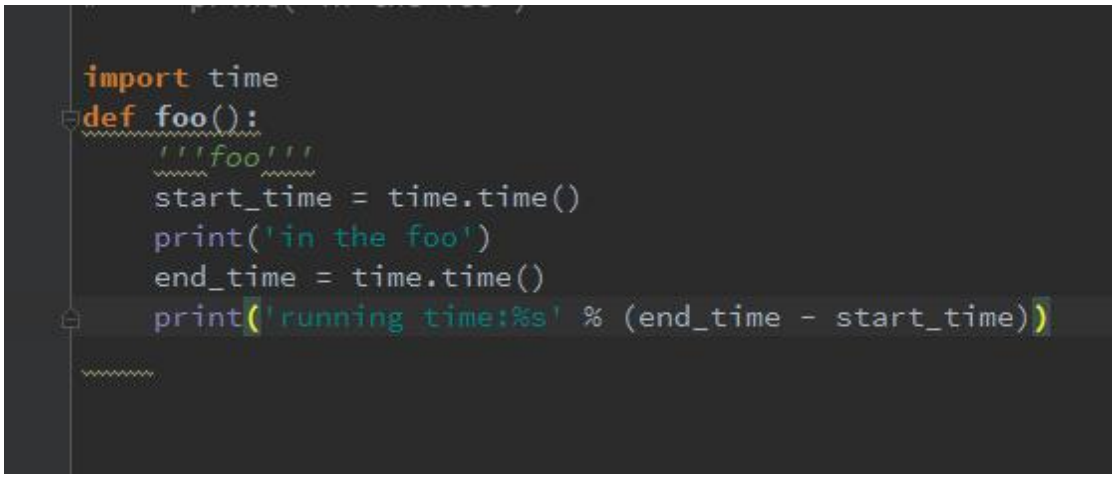
做任何事都需要讲究原则。编写装饰器也需要按照相应的原则出现，如果不遵守其原则就不能算是装饰器。

装饰器的原则：

A、不能修改被装饰函数的源代码

B、不能修改装饰器的函数的调用方式

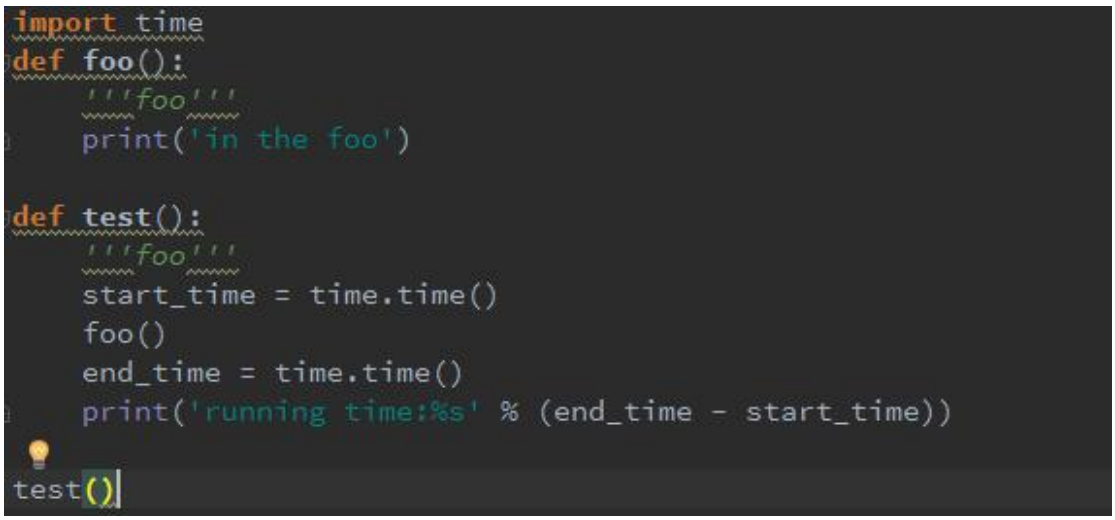
图 2.1 需求



```
import time
def foo():
    '''foo'''
    start_time = time.time()
    print('in the foo')
    end_time = time.time()
    print('running time:%s' % (end_time - start_time))
```

图 2.2 修改一

如图 2.2 中的 foo（）函数。虽然实现了图 2.1 中所需求的功能，但是图 2.2 中的 foo 修改了代码，这便违反了装饰器原则一，不能称之为装饰器。



```
import time
def foo():
    '''foo'''
    print('in the foo')

def test():
    '''foo'''
    start_time = time.time()
    foo()
    end_time = time.time()
    print('running time:%s' % (end_time - start_time))

test()
```

图 2.3 修改二

如图 2.3 中的 test（）函数。它也实现了图 2.1 中所需求的功能，但是图 2.3 中需要调用 test（）函数才能实现其功能，这样就违反了原则二（如果程序中有一千个 foo 函数就得将这 1000 个 foo 全改成 test），这么修改也不能称之为装饰器。

三、掌握装饰器所需的知识储备

想要掌握 python 的装饰器，就得先掌握如下知识作为掌握装饰器成储备知识：

1、函数即“变量”（一切皆对象）

2、高阶函数

3、嵌套函数

只有掌握了这三个知识，才能掌握装饰器。

3.1 函数即“变量”（一切皆对象）

在 python 中，一切皆对象（这就是为什么这么多屌丝来学 python 的原因）。

变量是对象，函数是对象，类是对象，所有的一切都是对象。如图 3.1 所示，这里定义函数 func，于是内存中就开辟属于 func 的内存空间，写入 func 的函数功能。随后在解释器上输入 func 函数的函数名，这时会返回一串指针地址，在 python 中所有的对象都会对应的指针地址，即指向内存中 func 函数的位置。如果在 func 函数名后加上（），这就说明 func 根据指针地址调用了 func 函数。

如图，将 func 函数的变量名赋值给 f，就是将 func 的内存地址赋值给 f，f 就会指向内存红 func 函数的位置。在解释器中输入 f()时，就能调用对应地址的 func 函数。

```
>>> def func():
...     print("in the func")
...
>>> func
<function func at 0x0000000000503E18>
>>> f = func
>>> f()
in the func
>>> f
<function func at 0x0000000000503E18>
>>> _
```

图 3.1 一切皆对象

3.2 高阶函数

高阶函数即一个函数的函数名作为参数传入另一个函数。如图 3.2 所示。定义 func 函数有 x,y,f 三个参数，执行 func 函数时将实参 abs(abs 是求绝对值的内置函数)传形参 f，func 函数最后 return 两数的绝对值和。

```
>>> def func(x,y,f):
...     return f(x)+f(y)
...
>>> print(func(1,-3,abs))
4
```

图 3.2 高阶函数

3.3 嵌套函数

嵌套函数是在一个函数的函数体内声明一个函数,再调用

```
def foo():
    def bar():
        print('in the foo.bar')
    bar()
    print('in the foo')

foo()
```

图 3.3 嵌套函数

四、装饰器初成

如图 4.1 所示，这样写就能够实现图 2.1 中的需求，并且遵循了装饰器的两个原则。其中就涉及到了装饰器的三个储备知识，由此可见三个储备知识的重要性。

```
import time
def foo():
    print('in the foo')
def test(func):
    def warpper():
        start_time = time.time()
        func()
        end_time = time.time()
        print('running time : %s' % (end_time - start_time))
    return warpper
foo = test(foo)
foo()
```

图 4.1 装饰器初成

但是图 4.1 中的 test 只是刚好装饰没有返回值的函数，如图 4.2 则可以装饰存在返回值的函数。

```
import time

def test(func):
    def warpper():
        start_time = time.time()
        res = func()
        end_time = time.time()
        print('running time : %s' % (end_time - start_time))
        return res
    return warpper
@test # 此处的@test 就等同于 foo = test(foo)
def foo():
    print('in the foo')
    return 'in the foo'

foo()
```

图 4.2 装饰器初成修改

五、装饰器小成

四中描述的装饰器还不够全面，如果函数 foo 需要传入参数时，图 4.2 中的装饰函数就会报错。于是这里就涉及到了函数的可变长参数和可变长关键字参数。这样就能实现任意的参数传入。于是装饰器的功法就小有所成了。如图 5.1 所示。

```python
import time

def test(func):
    def warpper(*args,**kwargs):
        start_time = time.time()
        res = func(*args,**kwargs)
        end_time = time.time()
        print('running time : %s' % (end_time - start_time))
        return res
    return warpper
@test # 此处的@test 就等同于 foo = test（foo）
def foo(*args,**kwargs):
    print(args)
    print('name is {name}'.format(**kwargs))
    print('in the foo')
    return 'in the foo'
foo(*[1,2,3],name = 'Evin')
```

图 5.1 装饰器小成

六、装饰器大成

其实装饰器学到五的时候，我就觉得已经很厉害了，想着自己终于弄懂了，但是老男孩里的 alex 老师有继续补充了一个关于装饰函数传参的知识点，突然间我就觉得这节课值 500 块，感谢 Alex 老师。老师最后的代码是这样的。这是一个模拟网站不同登陆方式的装饰器，使用一个装饰器，对不同的函数的附加不同的功能。Home 函数使用密码登陆方式，bbs 函数使用另一种登入方式（此处只是模拟一下），如图 6.1 所示。

```python
user, passwd = 'zsc', '123'
def auth(auth_type):
    def outer_wrapper(func):
        def wrapper(*args, **kwargs):
            if auth_type == 'local':
                username = input('Username:')
                password = input('Password:')
                if user == username and passwd == password:
                    print("\033[32;1mUser has passed authentication\033[0m")
                    return func(*args,**kwargs)
                else:
                    exit("\033[31;1mInvalid username or password\033[0m")

            elif auth_type == 'ldap':
                print('搞毛线的ldap，不会搞。。。')
        return wrapper
    return outer_wrapper
@auth(auth_type = 'local')
def home():
    print('welcome to home page')
    return 'from home'
@auth(auth_type = 'ldap')
def bbs():
    print('welcome to bbs page')
```

图 6.1 装饰器大成

七、总结

以上就是我学习装饰器之后对其的总结，根据上述内容，我绘制了一个概念图，便于对装饰器的理解和掌握。