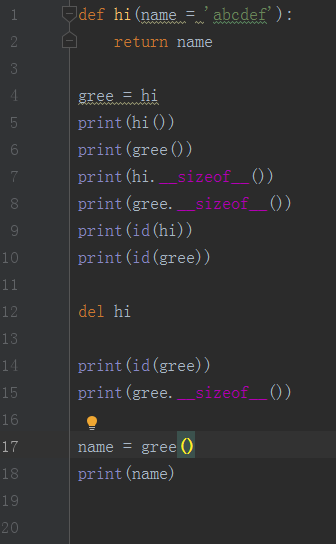
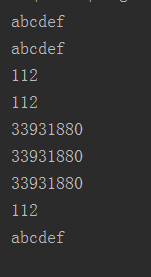
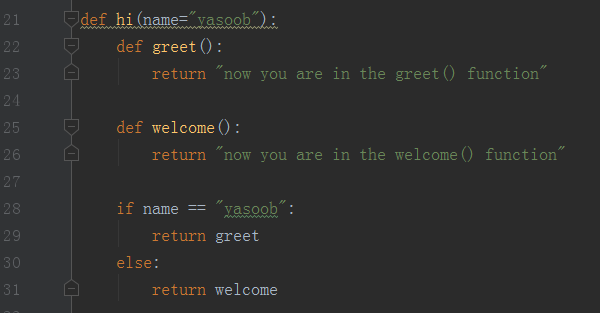
装饰器(Decorators)是 Python 的一个重要部分。简单地说：他们是**修改**其他函数的功能的函数。他们有助于让我们的代码更简短，也更Pythonic（Python范儿）。大多数初学者不知道在哪儿使用它们，所以我将要分享下，哪些区域里装饰器可以让你的代码更简洁。 首先，让我们讨论下如何写你自己的装饰器。

**1、python中一切皆对象**

 结果：

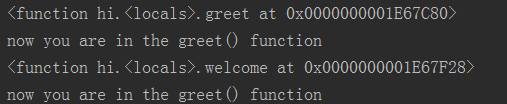
### **在函数中定义函数，从函数中返回函数**



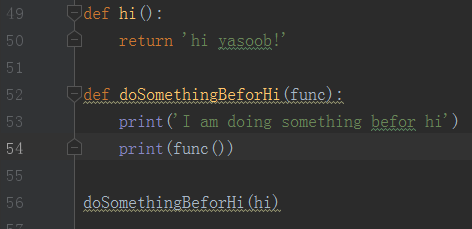
**如何调用？**



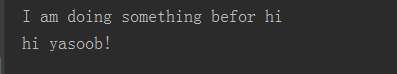
**结果：**



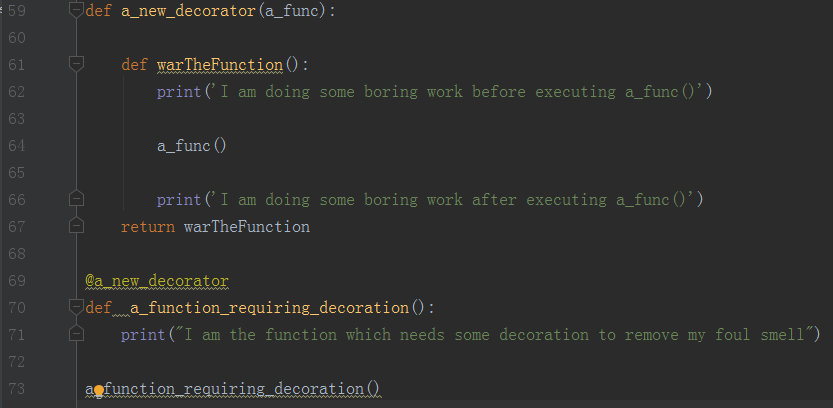
**把函数作为参数传给另一个函数**



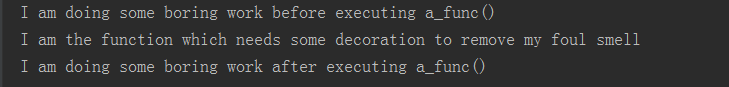
结果：



**第一个装饰器**



结果：



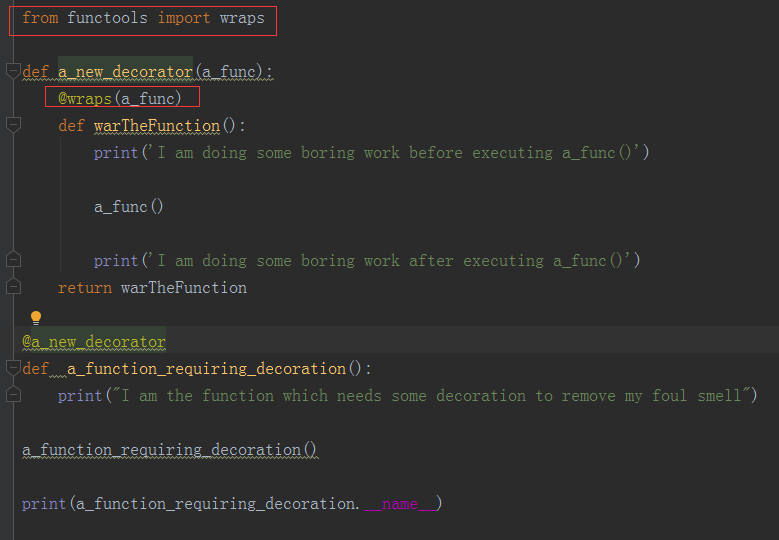
这样又会出现一个问题，在第一个装饰器的代码下，加上：



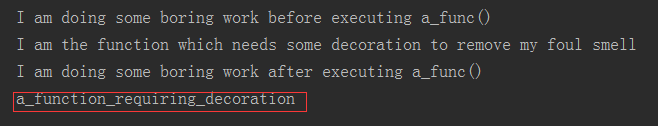
期望的结果是：a\_function\_requiring\_decoration

实际上是：

我们需要使用也个 functools.wraps函数来纠正

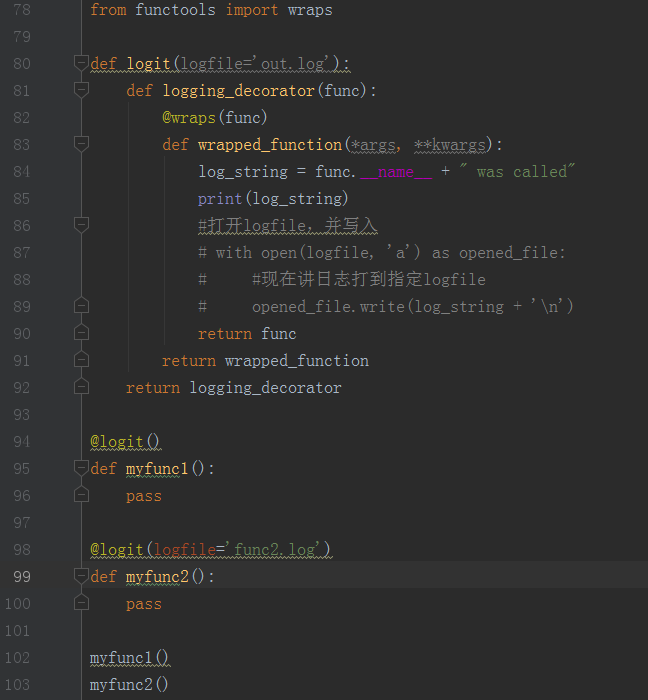


输出结果：



**带参数的装饰器**

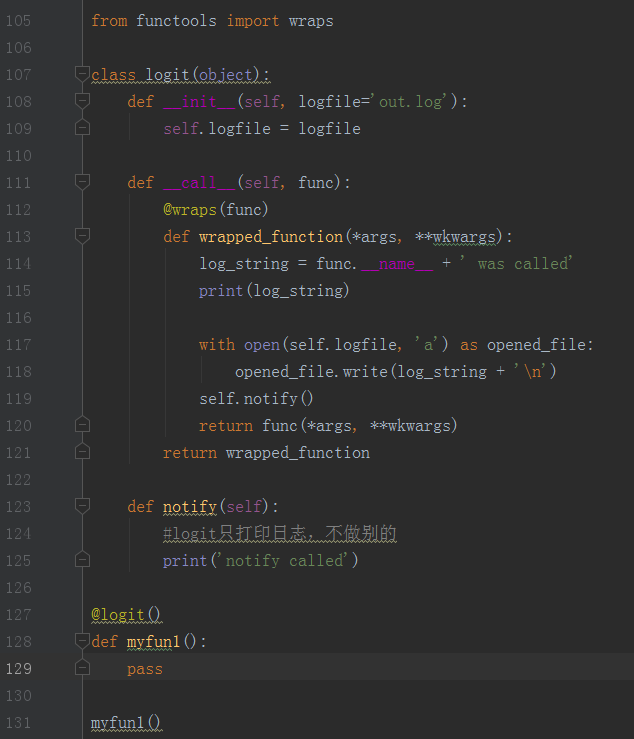
来想想这个问题，难道@wraps不也是个装饰器吗？但是，它接收一个参数，就像任何普通的函数能做的那样。那么，为什么我们不也那样做呢？ 这是因为，当你使用@my\_decorator语法时，你是在应用一个以单个函数作为参数的一个包裹函数。记住，Python里每个东西都是一个对象，而且这包括函数！记住了这些，我们可以编写一下能返回一个包裹函数的函数。



**装饰器类**

现在我们有了能用于正式环境的logit装饰器，但当我们的应用的某些部分还比较脆弱时，异常也许是需要更紧急关注的事情。比方说有时你只想打日志到一个文件。而有时你想把引起你注意的问题发送到一个email，同时也保留日志，留个记录。这是一个使用继承的场景，但目前为止我们只看到过用来构建装饰器的函数。

幸运的是，类也可以用来构建装饰器。那我们现在以一个类而不是一个函数的方式，来重新构建logit。



总结：

装饰器的功能就跟他名字讲的那样，用来装饰另一个函数，也就是说在调用这个函数的前后再加上一些东西，让这个函数更完整，更具体

还有一点，我们甚至可以在装饰器中不调用原函数的，这样就可以完全修改函数内容

装饰器本质上是一个 Python 函数或类，它可以让其他函数或类在不需要做任何代码修改的前提下增加额外功能，装饰器的返回值也是一个函数/类对象。它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景，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与函数功能本身无关的雷同代码到装饰器中并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能

[

{'instance\_id': '683e4739-c5ed-420f-9e6d-fcd87048b598',

'cloudenv\_id': 'dfd596d0-4921-4577-87ae-31d8ff92827a'}

]

[

{u'instance\_id': u'60024711-f602-46fb-8785-abb4bf8ebe84',

u'cloudenv\_id': u'ccb24f7d-fffc-4901-af5d-35b7bed7b7e9'},

{u'instance\_id': u'33dac5f8-4088-486d-8811-ce98bd880958',

u'cloudenv\_id': u'ccb24f7d-fffc-4901-af5d-35b7bed7b7e9'}

]