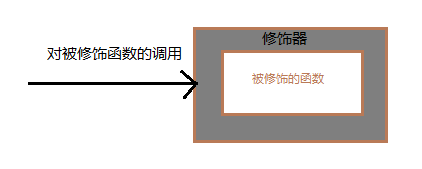
***Python函数装饰器***

函数装饰器的作用：用于修饰函数，从而在执行被修饰的函数前以及之后可以有额外的功能。并且不直接修改原函数，保持了黑盒子特性。



1. 函数调用以及函数引用

def calculate():

pass

在运行时，Python编译器遇到def关键字,会在内存中保存次函数的代码块。不执行函数。并且以把函数引用赋值给calculate. 在接下来的代码中, calculate指向该函数代码块但是不会调用函数。而calculate()才会调用函数。

1. 函数引用传递

def printWord():

print(“Hello World”)

def calculate(func):

func()

calculate(printWord) # 输出Hello World

这段代码将printWord的引用传入了calculate中作为参数，并且以此引用调用函数。

1. 修饰器的无参数使用

def Test\_Decorator(func):  
 def return\_function():  
 print("Before Call Original")  
 func()  
 print("After Call Original")  
 return return\_function  
  
@Test\_Decorator  
def func1():  
 print("I am Original Function")

func1() #程序输出Before Call Original

I am Original Function

After Call Original

修饰器的两大运行特点：

1. [当Python遇到@时，将判断为修饰器使用。Python](mailto:当Python遇到@时，将判断为修饰器使用。Python)将运行修饰器函数，并且把修饰器下面的函数引用作为参数，传入修饰器函数。
2. Python将被修饰的函数引用指向修饰器函数的返回值。

* 首先这段代码申明一个修饰器函数Test\_Decorator. 然后当Python解析到@Test\_Decorator的时候，识别为修饰器函数，从而马上执行修饰器函数。并且将下面的func1函数引用作为参数传入修饰器函数中。在修饰器函数中将返回 return\_function所指向的内容。而return\_function所指向的是一个function,所以等于func1将被重写为return\_function的内容。从而，以后再调用func1,也等于执行return\_function中的逻辑，即我们打印字符串，然后再调用被修饰前的函数func1，然后再打印字符串。

基本此原理特点；

* 如果我们在修饰器函数中，不返回任何内容，则Python默认返回None,那么一样重写被修饰函数，那么所有被修饰的函数都等于被擦写成None.
* 如果我们在修饰器函数中定义，返回的函数中不调用原来被修饰函数。那么也可以，只是被修饰完的函数不会执行任何原来的功能。

4. 修饰器的含参数使用

def Test\_Decorator2(func):  
 def return\_function(\*args,\*\*kwargs):  
 print("Before Call Original")  
 func(\*args,\*\*kwargs)  
 print("After Call Original")  
 return return\_function  
  
@Test\_Decorator2  
def func2(content):  
 print(content)

此时被修饰函数func2 需要接收参数，那么我们修饰器函数内返回的函数定义也要接收参数。 因为我们将此返回方法传给func2，重写func2. 如果我们不接受参数，那么以后使用者在func2(content)会传入一个参数去调用func2,但是我们重写的方法不接受任何参数，那么Python变会报错。

* 我们可以有两种写法。我们可以为return\_function也指定接受一个参数，是可运行的。但是如果我们用这个修饰器去修饰一个需要两个参数的函数时，变又出错。我们又要写一个接受两个参数的修饰器函数。
* 如果我们在定义修饰器函数时候以\*args,\*\*kargs这样万能参数的方法。那么Python会将自动检测并把参数传入原被修饰函数中。这样被修饰的函数，不管有多少个参数，即使被同一个修饰器修饰的函数需要的参数个数不一样，这些被修饰函数一样可以在修饰后无障碍的调用。

1. 多层修饰器

我们还可以对一个函数进行加载多层修饰器。语法为：

@decorator1

@decorator2

def function1():

pass

在Python对函数的多层修饰器进行解释时，从下往上进行绑定。即使首先decorator2首先对原函数进行装饰，然后以此被包装过的原函数作为函数再给decorator1进行再次绑定。而对于逻辑，以及进行执行顺序时从上往下执行。先执行decorator1中的外码，再执行decorator2中的代码，最后再调用函数funcion1. 当function1有返回值时，function1返回给decorator2,decorator2再返回给decorator1再返回给调用者。

