# 第六章 python函数高阶

# 本节所讲内容：

**6.1 递归函数**

**6.2 闭包**

**6.3 装饰器**

## 6.1 递归函数

**6.1.1 递归函数：**

**1、 什么是递归函数**

**我们先来引入一个知识点：函数的内部可以调用其它的函数**

**def fun1():  
 print("this is fun1")  
def fun2():  
 print("this is fun2")  
def main():  
 print("main is start")  
 fun1()  
 fun2()  
 print("main is down")  
main()**

**在这个例子当中，我们在main函数当中调用了fun1，fun2，最后在执行main函数的时候，按照从上到下的顺序进行了执行！**

**这种结构是一种很常见的程序设计结构，主要可以将我们的功能（这里的fun1和fun2）和执行顺序(这里的main)分开，使我们的编程思路更加清晰。**

**那么接下来研究一个问题，既然可以在函数当中能够调用函数，那么在函数当中调用自己可以吗？**

**答案：是的！**

**在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数，递归会形成一个深度循环！**

**一则小故事：从前有座山，山上有座庙，庙里有个老和尚给小和尚讲故事，从前有座山,山上有座庙，庙里有个老和尚给小和尚讲故事……**

**2、 递归函数的使用**

**举个例子：  
我们来计算阶乘 n! = 1 \* 2 \* 3 \* ... \* n**

**解决办法1:（利用while循环实现！）**

**n = int(input('请输入一个数字:'))  
result = 1  
i =1  
while i <= n:  
 result = result \*i  
 i+=1  
print(result)**

**运行结果如下：**

**请输入一个数字:4**

**24**

**紧接着我们看下阶乘的规律：**

**1! = 1**

**2! = 2 × 1 = 2 × 1!**

**3! = 3 × 2 × 1 = 3 × 2!**

**4! = 4 × 3 × 2 × 1 = 4 × 3! ...**

**n! = n × (n-1)!**

**解决办法2:（我们利用递归函数实现）**

**def test(n):  
 if n == 1:  
 return 1  
 return n \* test(n - 1)  
while True:  
 n = int(input('请输入你想要的数字'))  
 print(test(n))**

**3、递归函数经典案例（面试）：**

**题目：斐波那契数列，该数列中有n个数字，1 1 2 3 5 8**

**分析：该数列中，从第三个数字开始：数值 =前一个数字 + 前 前面一个数字**

**def feibo(n):  
 a,b = 0,1  
 c= []  
 while n>0:  
 c.append(b)  
 a,b = b,a+b  
 n-=1  
 print(c)  
feibo(5)**

**输出结果：**

**[1,1,2,3,5,8]**

**递归函数版：**

**def feibo(n):  
 if n <= 1:  
 return n  
 elif n ==2:  
 return 1  
 return (feibo(n-1)+feibo(n-2))** *# i = 1,2,3,4 lsit = [1,1,2,3]* **n = int(input('>>>'))  
*#形成一个列表*a = [feibo(i) for i in range(1,n+1)]  
print(a)**

**运行结果如下：**

**>>>5**

**[1, 1, 2, 3, 5]**

**总结：**

**1、明显看到递归函数的优点：定义简单，逻辑清晰；**

**2、实际上我们的递归都可以用循环来代替！只是不同的编程思路而已！比如：条条大路通罗马，使用于编程也是一样的！**

**拓展：**

**我们在写递归函数的时候一定要注意栈溢出情况，什么是栈溢出？**

**解释：**

**在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。**

**案例：比如刚刚我们写的阶乘  
def test(n):  
 if n == 1:  
 return 1  
 return n \* test(n - 1)  
while True:  
 n = int(input('请输入你想要的数字'))  
 print(test(n))**

**如果我输入：1000**

**运行结果如下：**

**RecursionError: maximum recursion depth exceeded in comparison**

**解决栈溢出的方法：尾递归**

**在函数返回的时候，调用自身本身，并且return语句不能包含任何表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。**

**def func\_test(n):  
 return test(n,1)  
def test(num,product):  
 if num == 1:  
 return product  
 return test(num-1,num \* product)  
print(func\_test(5))**

**很遗憾的是：对于大部分语言来说都没有对尾递归进行优化,python 同样也是！所以这也作为一个拓展让大家知道！**

**6.2 闭包（面试题）**

### 6.2.1 闭包定义

**1. 闭包函数必须有内嵌函数；**

**2. 内嵌函数需要引用该嵌套函数上一级namespace中的变量；**

**3. 闭包函数必须返回内嵌函数；**

**通过这三点，就可以创建一个闭包；Python装饰器就是使用了闭包。**

**闭包的好处：**

**使代码变得简洁；**

**提高代码的拓展性；**

**案例：**

**def outer():  
 def inner():  
 print('我是inner')  
 return inner  
outer()()#进行连续调用**

### 6.2.2 闭包分析

**def outer(num):  
 def inner(num\_in):  
 print('inner,num\_in is %d'%num\_in)  
 return num+num\_in  
 return inner  
*# a 就相当于 inner*a = outer(20) *# inner  
# print(a)  
#a(200)==inner(200)*print(a(200))**

**1、global 修改全局变量**

**x = 2  
def outer():  
 x = 0  
 def inner():  
 global x  
 x = 1  
 print(x)  
 print(x)  
 return inner  
*#调用*outer()()  
print(x)**

**运行结果如下：**

**0**

**1**

**1**

**注意：我们对不可变类型的数据例如：字符串，数字等修改时，我们要声明global全局变量！可变类型的数据例如：字典，列表等可以不用global声明！  
2、nonlocal修改嵌套变量**

**def outer():  
 x = 2  
 def inner():  
 nonlocal x  
 x = x + 1  
 print(x)  
 inner()  
 print(x)  
outer()**

**运行结果如下：**

**3**

**3**

**注意：nonlocal此时是对嵌套变量也就是x = 2 进行操作处理。**

## 6.3 装饰器(面试题)

### 6.3.1 装饰器的定义

**装饰器是程序开发中经常会用到的一个功能，用好了装饰器，开发效率如虎添翼。**

**应用场景：**

**1、当我们已经写好一个函数时，项目也已经上线了，突然客户想要增添一个需求，让这个函数处理的更加细致，这个就可以用到装饰器了；**

**2、淘宝登录验证；**

**3、log日志；**

**执行顺序：**

**先返回内嵌函数🡪有参数直接传参给内嵌函数🡪执行内嵌函数**

**案例：**

**#装饰器快速入门假如现在我们定义了两个函数，分别来打印他们的名字。**

**def func1():  
 print('this is xurgod1')  
def func2():  
 print('this is xuegod2')  
func1()  
func2()**

**运行结果如下：**

**this is xurgod1**

**this is xuegod2**

**现在，我们要对这两个做个修改，要分别打印他们的国家，那我们第一时间想到的自然是给每个函数都去加上这个功能**

**def func1():  
 print('this is xuegod1')  
 print('I come from china')  
def func2():  
 print('this is xuegod2')  
 print('I come from china')**

**运行结果如下：**

**this is xuegod1**

**I come from china**

**this is xuegod2**

**I come from china**

**那如何有n个函数都要这么打印呢？还要一个个的去修改，显然很费劲，那我们想想如何用装饰器来实现这个新增的功能。**

**def func1():  
 print('this is xuegod1')  
def func2():  
 print('this is xuegod2')  
def outer(func):  
 def inner():  
 func()  
 print('come from china')  
 return inner**

**#inner 赋值给f1  
f1 = outer(func1)**

**#f1() 相当于inner的调用  
f1()**

**运行结果如下：**

**this is xuegod1**

**come from china**

**但是上面的操作我们在给一个函数进行装饰时候会调用f1这个函数！**

**python认为上面的写法不简洁，所以提出了@语法糖**

**def outer(func):  
 def inner():  
 func()  
 print('come from china')  
 return inner  
@outer # 等同于func1 = outer(func1)**

**#第一个func1 是为了接收返回的inner**

**#第二个func1是def func1()函数，作为一个实参传递给形参func。  
def func1():  
 print('this is xuegod1')  
@outer  
def func2():  
 print('this is xuegod2')  
func1()  
func2()**

**运行结果如下：**

**this is xuegod1**

**come from china**

**this is xuegod2**

**come from china**

**问题：**

**现在想调用的函数的时候传递name参数，并的func1和func2中调用name，来获取我们传进去的名字，怎么办？**

**def outer(func):  
 def inner(name):  
 *# 直接将name = for 传递给 p（name），调用p函数后，直接打印出name* func(name)  
 print('inner中的%s'%name)  
 return inner  
@outer # func1 = outer(func1)  
def func1(name): #name = ‘for’  
 print('this is xuegod1')  
 print('I am %s'%name)  
*# func1(name) = Dec(func1(name))  
#此时第一个func1中的name=‘for*func1('for') #inner(‘for’)   
#第一个func1 相当于inner 所以inner需要参数来接受for值**

**更加通用的装饰器：**

**def outer(func):  
 def inner(\*args,\*\*kwargs):  
 *# func(\*args,\*\*kwargs)相当于func1(5,5)* c = func(\*args,\*\*kwargs)  
 print(c)  
 return inner  
@outer  
*#func1 = dec(func1)*def func1(a,b=1):  
 return a\*b  
func1(5,5)**

**运行结果如下：**

**25**

**实战：利用结合本节课知识点，让我们进入装饰器的分析，模拟淘宝加入购物车时候我们进行客户登录验证！**

 **import time  
a = ['while','for','django']  
def outer(func):  
 def inner(name):  
 *# name = 'for'* func(name)  
 print('开始判断你有没有登录')   
 time.sleep(2)  
 if name in a:  
 print('已经登录成功，请尽情访问')  
 time.sleep(1)  
 else:  
 print('你没有登录，没有访问权限')  
 time.sleep(1)  
 return inner  
@outer  
def login(name):  
 print('我要浏览')  
login('fo')**

**总结：**

**6.1 递归函数**

**6.2 闭包**

**6.3 装饰器**

**练习作业：**

**如何给装饰器传参呢？**

**def Out(info):  
 def Dec(func):  
 def inner(name):  
 print('info is %s'%info)  
 func(name)  
 print('come from china')  
 return inner  
 return Dec  
*#直接调用 返回 Dec*@Out('装饰器传递参数')  
def func1(name):  
 print('this is xuegod1')  
 print('I am %s'%name)  
@Out('装饰器传递参数')  
def func2(name):  
 print('this is xuegod2')  
 print('I am %s' % name)  
*#func1 = Dec(func1)*func1('for')  
*#func2 = Dec(func2)*func2('while')**