

INDEX



- Python教程
 - Python简介
 - 安装Python
 - 第一个Python程序
 - Python基础
 - 函数
 - 高级特性
 - 函数式编程
 - 高阶函数
 - 返回函数**
 - 匿名函数
 - 装饰器
 - 偏函数
 - 模块
 - 面向对象编程
 - 面向对象高级编程
 - 错误、调试和测试
 - IO编程
 - 进程和线程
 - 正则表达式
 - 常用内建模块
 - 常用第三方模块
 - virtualenv
 - 图形界面
 - 网络编程
 - 电子邮件
 - 访问数据库
 - Web开发
 - 异步IO
 - 实战
 - FAQ
 - 期末总结

关于作者

返回函数

Reads: 25927541

函数作为返回值

高阶函数除了可以接受函数作为参数外，还可以把函数作为结果值返回。

我们来实现一个可变参数的求和。通常情况下，求和的函数是这样定义的：

```
def calc_sum(*args):
    ax = 0
    for n in args:
        ax = ax + n
    return ax
```

但是，如果不需要立刻求和，而是在后面的代码中，根据需要再计算怎么办？可以不返回求和的结果，而是返回求和的函数：

```
def lazy_sum(*args):
    def sum():
        ax = 0
        for n in args:
            ax = ax + n
        return ax
    return sum
```

当我们调用 `lazy_sum()` 时，返回的并不是求和结果，而是求和函数：

```
>>> f = lazy_sum(1, 3, 5, 7, 9)
>>> f
<function lazy_sum.<locals>.sum at 0x101c6ed90>
```

调用函数 `f` 时，才真正计算求和的结果：

```
>>> f()
25
```

在这个例子中，我们在函数 `lazy_sum` 中又定义了函数 `sum`，并且，内部函数 `sum` 可以引用外部函数 `lazy_sum` 的参数和局部变量，当 `lazy_sum` 返回函数 `sum` 时，相关参数和变量都保存在返回的函数中，这种称为“**闭包 (Closure)**”的程序结构拥有极大的威力。

请再注意一点，当我们调用 `lazy_sum()` 时，每次调用都会返回一个新的函数，即使传入相同的参数：

```
>>> f1 = lazy_sum(1, 3, 5, 7, 9)
>>> f2 = lazy_sum(1, 3, 5, 7, 9)
>>> f1==f2
False
```

`f1()` 和 `f2()` 的调用结果互不影响。

闭包

注意到返回的函数在其定义内部引用了局部变量 `args`，所以，当一个函数返回了一个函数后，其内部的局部变量还被新函数引用，所以，闭包用起来简单，实现起来可不容易。

另一个需要注意的问题是，返回的函数并没有立刻执行，而是直到调用了 `f()` 才执行。我们来看一个例子：

```
def count():
    fs = []
    for i in range(1, 4):
        def f():
            return i*i
        fs.append(f)
    return fs

f1, f2, f3 = count()
```

在上面的例子中，每次循环，都创建了一个新的函数，然后，把创建的3个函数都返回了。

你可能认为调用 `f1()`，`f2()` 和 `f3()` 结果应该是 `1`，`4`，`9`，但实际结果是：

```
>>> f1()
9
>>> f2()
9
>>> f3()
9
```

全部都是 `9`！原因就在于返回的函数引用了变量 `i`，但它并非立刻执行。等到3个函数都返回时，它们所引用的变量 `i` 已经变成了 `3`，因此最终结果为 `9`。

▲ 返回闭包时牢记一点：返回函数不要引用任何循环变量，或者后续会发生变化的变量。

如果一定要引用循环变量怎么办？方法是再创建一個函数，用该函数的参数绑定循环变量当前的值，无论该循环变量后续如何更改，已绑定到函数参数的值不变：

```
def count():
    def f(j):
        def g():
            return j*j
        return g
    fs = []
    for i in range(1, 4):
        fs.append(f(i)) # f(i) 立刻被执行，因此i的当前值被传入f()
    return fs
```

再看看结果：

```
>>> f1, f2, f3 = count()
>>> f1()
1
>>> f2()
4
>>> f3()
9
```

缺点是代码较长，可利用lambda函数缩短代码。

练习

利用闭包返回一个计数器函数，每次调用它返回递增整数：

```
# -*- coding: utf-8 -*-

def createCounter():
    def counter():
        return 1
    return counter

# 测试：
counterA = createCounter()
print(counterA(), counterA(), counterA(), counterA()) # 1 2 3 4 5
counterB = createCounter()
if [counterB(), counterB(), counterB(), counterB()] == [1, 2, 3, 4]:
    print('测试通过!')
else:
    print('测试失败!')
```

▶ Run

小结

一个函数可以返回一个计算结果，也可以返回一个函数。

返回一个函数时，牢记该函数并未执行，返回函数中不要引用任何可能会变化的变量。

参考源码

[return_func.py](#)

读后有收获可以支付宝请作者喝咖啡，读后有疑问请加微信群讨论：



还可以分享给朋友：

分享到微博

◀ Previous Page

Next Page ▶

Comments

Make a comment

Sign in to make a comment

