

09 | 不可或缺的自定义函数

2019-05-29 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 10:36 大小 9.73M



你好，我是景霄。

实际工作生活中，我曾见到不少初学者编写的 Python 程序，他们长达几百行的代码中，却没有一个函数，通通按顺序堆到一块儿，不仅让人读起来费时费力，往往也是错误连连。


一个规范的值得借鉴的 Python 程序，除非代码量很少（比如 10 行、20 行以下），基本都应该由多个函数组成，这样的代码才更加模块化、规范化。

函数是 Python 程序中不可或缺的一部分。事实上，在前面的学习中，我们已经用到了很多 Python 的内置函数，比如 `sorted()` 表示对一个集合序列排序，`len()` 表示返回一个集合序列的长度大小等等。这节课，我们主要来学习 Python 的自定义函数。

函数基础

那么，到底什么是函数，如何在 Python 程序中定义函数呢？

说白了，函数就是为了实现某一功能的代码段，只要写好以后，就可以重复利用。我们先来看下面一个简单的例子：

 复制代码

```
1 def my_func(message):
2     print('Got a message: {}'.format(message))
3
4 # 调用函数 my_func()
5 my_func('Hello World')
6 # 输出
7 Got a message: Hello World
```

其中：

`def` 是函数的声明；


`my_func` 是函数的名称；

括号里面的 `message` 则是函数的参数；

而 `print` 那行则是函数的主体部分，可以执行相应的语句；

在函数最后，你可以返回调用结果（`return` 或 `yield`），也可以不返回。


总结一下，大概是下面的这种形式：

 复制代码

```
1 def name(param1, param2, ..., paramN):
2     statements
3     return/yield value # optional
```

和其他需要编译的语言（比如 C 语言）不一样的是，`def` 是可执行语句，这意味着函数直到被调用前，都是不存在的。当程序调用函数时，`def` 语句才会创建一个新的函数对象，并赋予其名字。


我们一起来看几个例子，加深你对函数的印象：

 复制代码

```
1 def my_sum(a, b):
2     return a + b
3
4 result = my_sum(3, 5)
5 print(result)
6
7 # 输出
8 8
```

这里，我们定义了 `my_sum()` 这个函数，它有两个参数 `a` 和 `b`，作用是相加；随后，调用 `my_sum()` 函数，分别把 3 和 5 赋于 `a` 和 `b`；最后，返回其相加的值，赋于变量 `result`，并输出得到 8。


再来看一个例子：

 复制代码

```
1 def find_largest_element(l):
2     if not isinstance(l, list):
3         print('input is not type of list')
4         return
5     if len(l) == 0:
6         print('empty input')
7         return
8     largest_element = l[0]
9     for item in l:
10        if item > largest_element:
11            largest_element = item
12    print('largest element is: {}'.format(largest_element))
13
14 find_largest_element([8, 1, -3, 2, 0])
15
16 # 输出
17 largest element is: 8
```

这个例子中，我们定义了函数 `find_largest_element`，作用是遍历输入的列表，找出最大的值并打印。因此，当我们调用它，并传递列表 `[8, 1, -3, 2, 0]` 作为参数时，程序就会输出 `largest element is: 8`。

需要注意，主程序调用函数时，必须保证这个函数此前已经定义过，不然就会报错，比如：

 复制代码

```
1 my_func('hello world')
2 def my_func(message):
3     print('Got a message: {}'.format(message))
4
5 # 输出
6 NameError: name 'my_func' is not defined
```

但是，如果我们在函数内部调用其他函数，函数间哪个声明在前、哪个在后就无所谓，因为 `def` 是可执行语句，函数在调用之前都不存在，我们只需保证调用时，所需的函数都已经声明定义：

 复制代码

```
1 def my_func(message):
2     my_sub_func(message) # 调用 my_sub_func() 在其声明之前不影响程序执行
3
4 def my_sub_func(message):
5     print('Got a message: {}'.format(message))
6
7 my_func('hello world')
8
9 # 输出
10 Got a message: hello world
```


另外，Python 函数的参数可以设定默认值，比如下面这样的写法：

 复制代码

```
1 def func(param = 0):
2     ...
```


这样，在调用函数 `func()` 时，如果参数 `param` 没有传入，则参数默认为 0；而如果传入了参数 `param`，其就会覆盖默认值。

前面说过，Python 和其他语言相比的一大特点是，Python 是 dynamically typed 的，可以接受任何数据类型（整型，浮点，字符串等等）。对函数参数来说，这一点同样适用。比如还是刚刚的 my_sum 函数，我们也可以把列表作为参数来传递，表示将两个列表相连接：

 复制代码


```
1 print(my_sum([1, 2], [3, 4]))
2
3 # 输出
4 [1, 2, 3, 4]
```

同样，也可以把字符串作为参数传递，表示字符串的合并拼接：

 复制代码

```
1 print(my_sum('hello ', 'world'))
2
3 # 输出
4 hello world
```

当然，如果两个参数的数据类型不同，比如一个是列表、一个是字符串，两者无法相加，那就会报错：


 复制代码

```
1 print(my_sum([1, 2], 'hello'))
2 TypeError: can only concatenate list (not "str") to list
```

我们可以看到，Python 不用考虑输入的数据类型，而是将其交给具体的代码去判断执行，同样的一个函数（比如这边的相加函数 my_sum()），可以同时应用在整型、列表、字符串等等的操作中。

在编程语言中，我们把这种行为称为**多态**。这也是 Python 和其他语言，比如 Java、C 等很大的一个不同点。当然，Python 这种方便的特性，在实际使用中也会带来诸多问题。因此，必要时请你在开头加上数据的类型检查。

Python 函数的另一大特性，是 Python 支持函数的嵌套。所谓的函数嵌套，就是指函数里面又有函数，比如：


 复制代码

```
1 def f1():
2     print('hello')
3     def f2():
4         print('world')
5     f2()
6 f1()
7
8 # 输出
9 hello
10 world
```

这里函数 f1() 的内部，又定义了函数 f2()。在调用函数 f1() 时，会先打印字符串 'hello'，然后 f1() 内部再调用 f2()，打印字符串 'world'。你也许会问，为什么需要函数嵌套？这样做有什么好处呢？

其实，函数的嵌套，主要有下面两个方面的作用。

第一，函数的嵌套能够保证内部函数的隐私。内部函数只能被外部函数所调用和访问，不会暴露在全局作用域，因此，如果你的函数内部有一些隐私数据（比如数据库的用户、密码等），不想暴露在外，那你就可以使用函数的嵌套，将其封装在内部函数中，只通过外部函数来访问。比如：

 复制代码

```
1 def connect_DB():
2     def get_DB_configuration():
3         ...
4         return host, username, password
5     conn = connector.connect(get_DB_configuration())
6     return conn
```

这里的函数 get_DB_configuration，便是内部函数，它无法在 connect_DB() 函数以外被单独调用。也就是说，下面这样的外部直接调用是错误的：


```
1 get_DB_configuration()  
2  
3 # 输出  
4 NameError: name 'get_DB_configuration' is not defined
```

我们只能通过调用外部函数 `connect_DB()` 来访问它，这样一来，程序的安全性便有了很大的提高。

第二，合理的使用函数嵌套，能够提高程序的运行效率。我们来看下面这个例子：


```
1 def factorial(input):  
2     # validation check  
3     if not isinstance(input, int):  
4         raise Exception('input must be an integer.')  
5     if input < 0:  
6         raise Exception('input must be greater or equal to 0' )  
7     ...  
8  
9     def inner_factorial(input):  
10         if input <= 1:  
11             return 1  
12         return input * inner_factorial(input-1)  
13     return inner_factorial(input)  
14  
15  
16 print(factorial(5))
```

这里，我们使用递归的方式计算一个数的阶乘。因为在计算之前，需要检查输入是否合法，所以我写成了函数嵌套的形式，这样一来，输入是否合法就只用检查一次。而如果我们不使用函数嵌套，那么每调用一次递归便会检查一次，这是没有必要的，也会降低程序的运行效率。

实际工作中，如果你遇到相似的情况，输入检查不是很快，还会耗费一定的资源，那么运用函数的嵌套就十分必要了。

函数变量作用域


Python 函数中变量的作用域和其他语言类似。如果变量是在函数内部定义的，就称为局部变量，只在函数内部有效。一旦函数执行完毕，局部变量就会被回收，无法访问，比如下面的例子：

 复制代码

```
1 def read_text_from_file(file_path):
2     with open(file_path) as file:
3         ...
```


我们在函数内部定义了 file 这个变量，这个变量只在 read_text_from_file 这个函数里有效，在函数外部则无法访问。

相对应的，全局变量则是定义在整个文件层次上的，比如下面这段代码：

 复制代码

```
1 MIN_VALUE = 1
2 MAX_VALUE = 10
3 def validation_check(value):
4     if value < MIN_VALUE or value > MAX_VALUE:
5         raise Exception('validation check fails')
```

这里的 MIN_VALUE 和 MAX_VALUE 就是全局变量，可以在文件内的任何地方被访问，当然在函数内部也是可以的。不过，我们**不能在函数内部随意改变全局变量的值**。比如，下面的写法就是错误的：

 复制代码

```
1 MIN_VALUE = 1
2 MAX_VALUE = 10
3 def validation_check(value):
4     ...
5     MIN_VALUE += 1
6     ...
7 validation_check(5)
```

如果运行这段代码，程序便会报错：


```
1 UnboundLocalError: local variable 'MIN_VALUE' referenced before assignment
```

这是因为，Python 的解释器会默认函数内部的变量为局部变量，但是又发现局部变量 `MIN_VALUE` 并没有声明，因此就无法执行相关操作。所以，如果我们一定要在函数内部改变全局变量的值，就必须加上 `global` 这个声明：

```
1 MIN_VALUE = 1
2 MAX_VALUE = 10
3 def validation_check(value):
4     global MIN_VALUE
5     ...
6     MIN_VALUE += 1
7     ...
8 validation_check(5)
```


这里的 `global` 关键字，并不表示重新创建了一个全局变量 `MIN_VALUE`，而是告诉 Python 解释器，函数内部的变量 `MIN_VALUE`，就是之前定义的全局变量，并不是新的全局变量，也不是局部变量。这样，程序就可以在函数内部访问全局变量，并修改它的值了。

另外，如果遇到函数内部局部变量和全局变量同名的情况，那么在函数内部，局部变量会覆盖全局变量，比如下面这种：

```
1 MIN_VALUE = 1
2 MAX_VALUE = 10
3 def validation_check(value):
4     MIN_VALUE = 3
5     ...
```


在函数 `validation_check()` 内部，我们定义了和全局变量同名的局部变量 `MIN_VALUE`，那么，`MIN_VALUE` 在函数内部的值，就应该是 3 而不是 1 了。

类似的，对于嵌套函数来说，内部函数可以访问外部函数定义的变量，但是无法修改，若要修改，必须加上 `nonlocal` 这个关键字：

 复制代码

```
1 def outer():
2     x = "local"
3     def inner():
4         nonlocal x # nonlocal 关键字表示这里的 x 就是外部函数 outer 定义的变量 x
5         x = 'nonlocal'
6         print("inner:", x)
7     inner()
8     print("outer:", x)
9 outer()
10 # 输出
11 inner: nonlocal
12 outer: nonlocal
```

如果不加上 `nonlocal` 这个关键字，而内部函数的变量又和外部函数变量同名，那么同样的，内部函数变量会覆盖外部函数的变量。


 复制代码

```
1 def outer():
2     x = "local"
3     def inner():
4         x = 'nonlocal' # 这里的 x 是 inner 这个函数的局部变量
5         print("inner:", x)
6     inner()
7     print("outer:", x)
8 outer()
9 # 输出
10 inner: nonlocal
11 outer: local
```

闭包

这节课的第三个重点，我想再来介绍一下闭包（closure）。闭包其实和刚刚讲的嵌套函数类似，不同的是，这里外部函数返回的是一个函数，而不是一个具体的值。返回的函数通常赋于一个变量，这个变量可以在后面被继续执行调用。

举个例子你就更容易理解了。比如，我们想计算一个数的 n 次幂，用闭包可以写成下面的代码：

 复制代码

```
1 def nth_power(exponent):
2     def exponent_of(base):
3         return base ** exponent
4     return exponent_of # 返回值是 exponent_of 函数
5
6 square = nth_power(2) # 计算一个数的平方
7 cube = nth_power(3) # 计算一个数的立方
8 square
9 # 输出
10 <function __main__.nth_power.<locals>.exponent(base)>
11
12 cube
13 # 输出
14 <function __main__.nth_power.<locals>.exponent(base)>
15
16 print(square(2)) # 计算 2 的平方
17 print(cube(2)) # 计算 2 的立方
18 # 输出
19 4 # 2^2
20 8 # 2^3
```


这里外部函数 `nth_power()` 返回值，是函数 `exponent_of()`，而不是一个具体的数值。需要注意的是，在执行完 `square = nth_power(2)` 和 `cube = nth_power(3)` 后，外部函数 `nth_power()` 的参数 `exponent`，仍然会被内部函数 `exponent_of()` 记住。这样，之后我们调用 `square(2)` 或者 `cube(2)` 时，程序就能顺利地输出结果，而不会报错说参数 `exponent` 没有定义了。

看到这里，你也许会思考，为什么要闭包呢？上面的程序，我也可以写成下面的形式啊！

 复制代码

```
1 def nth_power_rewrite(base, exponent):
2     return base ** exponent
3
```

确实可以，不过，要知道，使用闭包的一个原因，是让程序变得更简洁易读。设想一下，比如你需要计算很多个数的平方，那么你觉得写成下面哪一种形式更好呢？

 复制代码

```
1 # 不适用闭包
2 res1 = nth_power_rewrite(base1, 2)
3 res2 = nth_power_rewrite(base2, 2)
4 res3 = nth_power_rewrite(base3, 2)
5 ...
6
7 # 使用闭包
8 square = nth_power(2)
9 res1 = square(base1)
10 res2 = square(base2)
11 res3 = square(base3)
12 ...
```

显然是第二种，是不是？首先直观来看，第二种形式，让你每次调用函数都可以少输入一个参数，表达更为简洁。

其次，和上面讲到的嵌套函数优点类似，函数开头需要做一些额外工作，而你又需要多次调用这个函数时，将那些额外工作的代码放在外部函数，就可以减少多次调用导致的不必要的开销，提高程序的运行效率。

另外还有一点，我们后面会讲到，闭包常常和装饰器（decorator）一起使用。

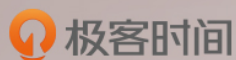
总结

这节课，我们一起学习了 Python 函数的概念及其应用，有这么几点你需要注意：

1. Python 中函数的参数可以接受任意的数据类型，使用起来需要注意，必要时请在函数开头加入数据类型的检查；
2. 和其他语言不同，Python 中函数的参数可以设定默认值；
3. 嵌套函数的使用，能保证数据的隐私性，提高程序运行效率；
4. 合理地使用闭包，则可以简化程序的复杂度，提高可读性。

思考题

最后给你留一道思考题。在实际的学习工作中，你遇到过哪些使用嵌套函数或者是闭包的例子呢？欢迎在下方留言，与我讨论，也欢迎你把这篇文章分享给你的同事、朋友。



Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 异常处理：如何提高程序的稳定性？

下一篇 10 | 简约不简单的匿名函数

精选留言 (45)

写留言



不瘦到140...

2019-05-29

17

其实函数也可以看做成是一个变量，函数名就是变量名，函数体就是值。函数虽然在不被调用的情况下不会执行，但是python解释器会做一些变量检测、或者类型检测，比如是不是有yield，如果有那么就会被标记为生成器，这个在编译成字节码的时候就已经确定了。而有些东西则是只有在解释执行的时候才会被发现。

比如说：...

展开 ∨



pyhhou

2019-05-29

👍 15

闭包必须使用嵌套函数，一看到闭包我首先想到的是 JavaScript 里面的回调函数。闭包这里看似仅仅返回了一个嵌套函数，但是需要注意的是，它其实连同嵌套函数的外部环境变量也一同保存返回回来了（例子中的 `exponent` 变量），这个环境变量是在调用其外部函数时设定的，这样一来，对于一些参数性，不常改变的设定变量，我们可以通过这个形式来设定，这样返回的闭包函数仅需要关注那些核心输入变量，节省了效率，这样做也大...

展开 ▾



farFlight

2019-05-29

👍 10

谢谢老师，这节课的内容非常有意思！

有两个问题：

1. python自己判断类型的多态和子类继承的多态Polymorphism应该不是一个意思吧？
2. 函数内部不能直接用`+=`等修改全局变量，但是对于list全局变量，可以使用`append`, `extend`之类修改，这是为什么呢？

展开 ▾



Brigand

2019-05-29

👍 3

我建议，以后文中不要放代码，放代码截图，有需要代码去github，这样移动端体验会好点。



进击的菜鸟...

2019-05-29

👍 3

老师，您说的“函数的调用和声明哪个在前哪个在后是无所谓的。”请问这句话怎么理解呢？

如下是会报异常`NameError: name 'f' is not defined`:

```
f()
```

```
def f():...
```

展开 ▾

作者回复: 文中已经更新了。可能之前表达的不准确，意思是主程序调用函数时，必须保证这个函数此前已经定义过，但是，如果我们在函数内部调用其他函数，函数间哪个声明在前、哪个在后就无所谓，因为`def`是可执行语句，函数调用前都不存在，我们只需保证调用时，所需的函数都已经声明定义



Vincent

2019-05-29

👍 3

关于嵌套函数：“我们只能通过调用外部函数 `connect_DB()` 来访问它，这样一来，程序的安全性便有了很大的提高。” 这个怎么就安全了呢？这个安全指的是什么安全呢？

展开 ▾

作者回复: 数据库的用户名密码等一些信息不会暴露在外部的API中



William

2019-05-29

👍 2

快排封装，增加index参数会用到嵌套。

```
```python
```

```
def quickSort(arr):
```

```
 def partition(arr, left, right):
```

```
 pivot = arr[left]...
```

展开 ▾

作者回复: 嗯嗯，学习很细心



**Aspirin**

2019-05-29

👍 2

关于闭包，我想到一个案例，就是卷积神经网络模型的实现。我们知道，在CNN模型推理时，所有卷积层或全连接层的权重weights都是已知的、确定的，也就是说我们实例化一个模型之后，所有layer的weights都是确定好的，只需要处理不同的输入就可以了。所以，我们可以写一个闭包函数，输入不同的权重，返回使用该权重进行卷积运算的函数即可。伪代码如下： ...

展开 ▾



**路伴友行**

2019-05-29

👍 2

我有个项目需要将很多不规则的列表展平  
但没有找到推荐的方式，就自己写了个



希望大佬们多多指出缺点，谢谢

```
def getSmoothList(lst):...
```

展开 ▾

---



**KaitoShy**

2019-05-29

👍 2

```
a = {'shanghai':50000, 'hangzhou':300000}
```

```
def func():
```

```
 a['beijing'] = 17500
```

```
 ...
```

展开 ▾

---



**michel**

2019-05-30

👍 1

关于函数申明及调用关系，以及变量范围，做了几个测试，终于理解的比较透彻了。

def本身就是一个申明，如果不执行，不涉及对对象的引用，则不会报错，即使在函数内部引用了一个不存在的变量。关键在于执行的时候，被引用的变量或者函数是否被加载。

...

展开 ▾

---



**付剑津**

2019-05-30

👍 1

一开始看完，对闭包的概念有了，但比较糙，不知道闭包究竟指的是哪个变量。

这篇文章对大家理解闭包有一定帮助：<https://zhuanlan.zhihu.com/p/26934085>

---



**路伴友行**

2019-05-29

👍 1

顺便我想多问一句，在Python里是不推荐使用递归的，是因为Python没有对递归做优化，那使用 yield from 来代替递归会不会好些呢？

其实我上一个例子就是一个尝试，我之前只尝试过打印栈信息，只看到有2层，就是不清楚有些其他什么弊端。

作者回复: 你说的没错



third

2019-05-29

👍 1

1. Python中...是啥意思? 发现在代码中运行没有错误。也没有百度到

2. #不是说全局变量可以在文件的任意地方都可以被访问吗? ,我试了下, 去掉x的赋值, 就可以访问了。这是什么原因呢?

#x=10...

展开 ▾

作者回复: 1. 我只是用 '...' 表示省略

2. 全局变量在任何地方都可以访问, 但是访问之前你必须得定义赋值他啊



潇洒哥er

2019-05-29

👍 1

看到评论区经常有同学在问手机用什么软件写代码, 推荐一下:

苹果系统的: Pythonista 3

安卓系统的: PyDroid3

两个都有用, 但感觉苹果的pythonista 特别的好用, 打一半提示一半, 半智能, 自动格式化。

展开 ▾



且听疯吟

2019-06-06

👍

闭包是不是可以理解为C语言里面的一个函数, 返回的就是一个函数指针?

可以用一个指针接受函数指针。



小侠龙旋风

2019-06-06

👍

1. Python 中函数的参数可以接受任意的数据类型, 使用起来需要注意, 必要时请在函数开头加入数据类型的检查;

- 2.和其他语言不同, Python 中函数的参数可以设定默认值;
- 3.嵌套函数的使用, 能保证数据的隐私性, 提高程序运行效率;
- 4.合理地使用闭包, 则可以简化程序的复杂度, 提高可读性。...

展开 ∨



蚂蚁

2019-06-05



"其实函数也可以看做成是一个变量, 函数名就是变量名, 函数体就是值。函数虽然在不被调用的情况下不会执行, 但是python解释器会做一些变量检测、或者类型检测, 比如是不是有yield, 如果有那么就会被标记为生成器, 这个在编译成字节码的时候就已经确定了。而有些东西则是只有在解释执行的时候才会被发现。

比如说: ...

展开 ∨



while (1...

2019-06-05



"如果不加上 nonlocal 这个关键字, 而内部函数的变量又和外部函数变量同名, 那么同样的, 内部函数变量会覆盖外部函数的变量。" 这个是不覆盖吧



wanj

2019-06-01



默认值参数再有新的参数也必须是带有默认值的, 另外\*args \*\*kwargs也比较常用, 希望能够补充讲解