

## 10 | 简约不简单的匿名函数

2019-05-31 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 09:55 大小 9.09M



你好，我是景霄。

上一节，我们一起学习了 Python 中的“常规”函数，用途十分广泛。不过，除了常规函数，你应该也会在代码中见到一些“非常规”函数，它们往往很简短，就一行，并且有个很酷炫的名字——lambda，没错，这就是匿名函数。

匿名函数在实际工作中同样举足轻重，正确地运用匿名函数，能让我们的代码更简洁、易读。这节课，我们继续 Python 的函数之旅，一起来学习这个简约而不简单的匿名函数。

### 匿名函数基础

首先，什么是匿名函数呢？以下是匿名函数的格式：

```
1 lambda argument1, argument2,... argumentN : expression
```

我们可以看到，匿名函数的关键字是 `lambda`，之后是一系列的参数，然后用冒号隔开，最后则是由这些参数组成的表达式。我们通过几个例子看一下它的用法：

```
1 square = lambda x: x**2
2 square(3)
3
4 9
```

这里的匿名函数只输入一个参数 `x`，输出则是输入 `x` 的平方。因此当输入是 3 时，输出便是 9。如果把这个匿名函数写成常规函数的形式，则是下面这样：

```
1 def square(x):
2     return x**2
3 square(3)
4
5 9
```


可以看到，匿名函数 `lambda` 和常规函数一样，返回的都是一个函数对象（function object），它们的用法也极其相似，不过还是有下面几点区别。

**第一，`lambda` 是一个表达式（expression），并不是一个语句（statement）。**

所谓的表达式，就是用一系列“公式”去表达一个东西，比如 `x + 2`、`x**2` 等等；


而所谓的语句，则一定是完成了某些功能，比如赋值语句 `x = 1` 完成了赋值，`print` 语句 `print(x)` 完成了打印，条件语句 `if x < 0:` 完成了选择功能等等。

因此，lambda 可以用在一些常规函数 def 不能用的地方，比如，lambda 可以用在列表内部，而常规函数却不能：

 复制代码

```
1 [(lambda x: x*x)(x) for x in range(10)]
2 # 输出
3 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

再比如，lambda 可以被用作某些函数的参数，而常规函数 def 也不能：

 复制代码

```
1 l = [(1, 20), (3, 0), (9, 10), (2, -1)]
2 l.sort(key=lambda x: x[1]) # 按列表中元祖的第二个元素排序
3 print(l)
4 # 输出
5 [(2, -1), (3, 0), (9, 10), (1, 20)]
```

常规函数 def 必须通过其函数名被调用，因此必须首先被定义。但是作为一个表达式的 lambda，返回的函数对象就不需要名字了。

**第二，lambda 的主体是只有一行的简单表达式，并不能扩展成一个多行的代码块。**

这其实是出于设计的考虑。Python 之所以发明 lambda，就是为了让它和常规函数各司其职：lambda 专注于简单的任务，而常规函数则负责更复杂的多行逻辑。关于这点，Python 之父 Guido van Rossum 曾发了一篇[文章](#)解释，你有兴趣的话可以自己阅读。

## 为什么要使用匿名函数？

理论上来说，Python 中有匿名函数的地方，都可以被替换成等价的其他表达形式。一个 Python 程序是可以不用任何匿名函数的。不过，在一些情况下，使用匿名函数 lambda，可以帮助我们大大简化代码的复杂度，提高代码的可读性。

通常，我们用函数的目的无非是这么几点：

1. 减少代码的重复性；

## 2. 模块化代码。


对于第一点，如果你的程序在不同地方包含了相同的代码，那么我们会把这部分相同的代码写成一个函数，并为它取一个名字，方便在相对应的不同地方调用。

对于第二点，如果你的一块儿代码是为了实现一个功能，但内容非常多，写在一起降低了代码的可读性，那么通常我们也会把这部分代码单独写成一个函数，然后加以调用。

不过，再试想一下这样的情况。你需要一个函数，但它非常简短，只需要一行就能完成；同时它在程序中只被调用一次而已。那么请问，你还需要像常规函数一样，给它一个定义和名字吗？


答案当然是否定的。这种情况下，函数就可以是匿名的，你只需要在适当的地方定义并使用，就能让匿名函数发挥作用了。

举个例子，如果你想对一个列表中的所有元素做平方操作，而这个操作在你的程序中只需要进行一次，用 `lambda` 函数可以表示成下面这样：

 复制代码

```
1 squared = map(lambda x: x**2, [1, 2, 3, 4, 5])
```


如果用常规函数，则表示为这几行代码：

 复制代码

```
1 def square(x):  
2     return x**2  
3  
4 squared = map(square, [1, 2, 3, 4, 5])
```


这里我简单解释一下。函数 `map(function, iterable)` 的第一个参数是函数对象，第二个参数是一个可以遍历的集合，它表示对 `iterable` 的每一个元素，都运用 `function` 这个函数。两者一对比，我们很明显地发现，`lambda` 函数让代码更加简洁明了。

再举一个例子，在 Python 的 Tkinter GUI 应用中，我们想实现这样一个简单的功能：创建显示一个按钮，每当用户点击时，就打印出一段文字。如果使用 lambda 函数可以表示成下面这样：

 复制代码

```
1 from tkinter import Button, mainloop
2 button = Button(
3     text='This is a button',
4     command=lambda: print('being pressed')) # 点击时调用 lambda 函数
5 button.pack()
6 mainloop()
```

而如果我们用常规函数 def，那么需要写更多的代码：

 复制代码

```
1 from tkinter import Button, mainloop
2
3 def print_message():
4     print('being pressed')
5
6 button = Button(
7     text='This is a button',
8     command=print_message) # 点击时调用 lambda 函数
9 button.pack()
10 mainloop()
```


显然，运用匿名函数的代码简洁很多，也更加符合 Python 的编程习惯。

## Python 函数式编程

最后，我们一起来看一下，Python 的函数式编程特性，这与我们今天所讲的匿名函数 lambda，有着密切的联系。


所谓函数式编程，是指代码中每一块都是不可变的（immutable），都由纯函数（pure function）的形式组成。这里的纯函数，是指函数本身相互独立、互不影响，对于相同的输入，总会有相同的输出，没有任何副作用。

举个很简单的例子，比如对于一个列表，我想让列表中的元素值都变为原来的两倍，我们可以写成下面的形式：

 复制代码

```
1 def multiply_2(l):
2     for index in range(0, len(l)):
3         l[index] *= 2
4     return l
```

这段代码就不是一个纯函数的形式，因为列表中元素的值被改变了，如果我多次调用 `multiply_2()` 这个函数，那么每次得到的结果都不一样。要想让它成为一个纯函数的形式，就得写成下面这种形式，重新创建一个新的列表并返回。


 复制代码

```
1 def multiply_2_pure(l):
2     new_list = []
3     for item in l:
4         new_list.append(item * 2)
5     return new_list
```

函数式编程的优点，主要在于其纯函数和不可变的特性使程序更加健壮，易于调试（debug）和测试；缺点主要在于限制多，难写。当然，Python 不同于一些语言（比如 Scala），它并不是一门函数式编程语言，不过，Python 也提供了一些函数式编程的特性，值得我们了解和学习。

Python 主要提供了这么几个函数：`map()`、`filter()` 和 `reduce()`，通常结合匿名函数 `lambda` 一起使用。这些都是你需要掌握的东西，接下来我逐一介绍。


首先是 `map(function, iterable)` 函数，前面的例子提到过，它表示，对 `iterable` 中的每个元素，都运用 `function` 这个函数，最后返回一个新的可遍历的集合。比如刚才列表的例子，要对列表中的每个元素乘以 2，那么用 `map` 就可以表示为下面这样：

 复制代码

```
1 l = [1, 2, 3, 4, 5]
2 new_list = map(lambda x: x * 2, l) # [2, 4, 6, 8, 10]
```



我们可以以 `map()` 函数为例，看一下 Python 提供的函数式编程接口的性能。还是同样的列表例子，它还可以用 `for` 循环和 `list comprehension`（目前没有统一中文叫法，你也可以直译为列表理解等）实现，我们来比较一下它们的速度：


 复制代码

```
1 python3 -mtimeit -s'xs=range(1000000)' 'map(lambda x: x*2, xs)'  
2 2000000 loops, best of 5: 171 nsec per loop  
3  
4 python3 -mtimeit -s'xs=range(1000000)' '[x * 2 for x in xs]'  
5 5 loops, best of 5: 62.9 msec per loop  
6  
7 python3 -mtimeit -s'xs=range(1000000)' 'l = []' 'for i in xs: l.append(i * 2)'  
8 5 loops, best of 5: 92.7 msec per loop
```

你可以看到，`map()` 是最快的。因为 `map()` 函数直接由 C 语言写的，运行时不需要通过 Python 解释器间接调用，并且内部做了诸多优化，所以运行速度最快。

接下来来看 `filter(function, iterable)` 函数，它和 `map` 函数类似，`function` 同样表示一个函数对象。`filter()` 函数表示对 `iterable` 中的每个元素，都使用 `function` 判断，并返回 `True` 或者 `False`，最后将返回 `True` 的元素组成一个新的可遍历的集合。

举个例子，比如我要返回一个列表中的所有偶数，可以写成下面这样：

 复制代码

```
1 l = [1, 2, 3, 4, 5]  
2 new_list = filter(lambda x: x % 2 == 0, l) # [2, 4]
```

最后我们来看 `reduce(function, iterable)` 函数，它通常用来对一个集合做一些累积操作。

`function` 同样是一个函数对象，规定它有两个参数，表示对 `iterable` 中的每个元素以及上一次调用后的结果，运用 `function` 进行计算，所以最后返回的是一个单独的数值。

举个例子，我想要计算某个列表元素的乘积，就可以用 `reduce()` 函数来表示：

```
1 l = [1, 2, 3, 4, 5]
2 product = reduce(lambda x, y: x * y, l) # 1*2*3*4*5 = 120
```

当然，类似的，`filter()` 和 `reduce()` 的功能，也可以用 `for` 循环或者 `list comprehension` 来实现。

通常来说，在我们想对集合中的元素进行一些操作时，如果操作非常简单，比如相加、累积这种，那么我们优先考虑 `map()`、`filter()`、`reduce()` 这类或者 `list comprehension` 的形式。至于这两种方式的选择：

在数据量非常多的情况下，比如机器学习的应用，那我们一般更倾向于函数式编程的表示，因为效率更高；

在数据量不多的情况下，并且你想要程序更加 `Pythonic` 的话，那么 `list comprehension` 也不失为一个好选择。

不过，如果你要对集合中的元素，做一些比较复杂的操作，那么，考虑到代码的可读性，我们通常会使用 `for` 循环，这样更加清晰明了。

## 总结

这节课，我们一起学习了 Python 中的匿名函数 `lambda`，它的主要用途是减少代码的复杂度。需要注意的是 `lambda` 是一个表达式，并不是一个语句；它只能写成一行的表达式，语法上并不支持多行。匿名函数通常的使用场景是：程序中需要使用一个函数完成一个简单的功能，并且该函数只调用一次。

其次，我们也入门了 Python 的函数式编程，主要了解了常见的 `map()`、`filter()` 和 `reduce()` 三个函数，并比较了它们与其他形式（`for` 循环，`comprehension`）的性能，显然，它们的性能效率是最优的。

## 思考题

最后，我想给你留下两道思考题。



第一问：如果让你对一个字典，根据值进行由高到底的排序，该怎么做呢？以下面这段代码为例，你可以思考一下。

 复制代码

```
1 d = {'mike': 10, 'lucy': 2, 'ben': 30}
```

第二问：在实际工作学习中，你遇到过哪些使用匿名函数的场景呢？

欢迎在留言区写下你的答案想法，与我讨论，也欢迎你把这篇文章分享给你的同事、朋友。

 极客时间

# 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 09 | 不可或缺的自定义函数

下一篇 11 | 面向对象（上）：从生活中的类比说起

精选留言 (46)

 写留言



lmingzhi

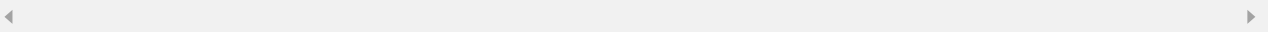
2019-05-31

👍 15

```
python3 -mtimeit -s'xs=range(1000000)' 'map(lambda x: x*2, xs)'
```

这个地方map生成的是生成器，与后面的2个做比较感觉不大合适，是否更改为测试list(map(lambda x: x\*2, xs))更恰当？

作者回复: 实际情况中，Map返回的对象依然可以直接遍历，所以直接比较从实用的角度来说也是可以的，Map在Python3中变为Lazy了以后，速度得到了很大的提升。当然，如果以返回的类型一致为标准，你的建议也是可以的



Hoo-Ah

2019-05-31

👍 11

第一问: sorted(d.items(), key=lambda x: x[1], reverse=True);

第二问: 最开始接触 lambda 匿名函数的时候觉得蛮不理解的，觉得这个函数没有灵魂，用完一次就扔掉。后来在和高阶函数、列表生成式搭配使用以及一些小功能的使用上觉得很好用，这样代码即简洁又易于阅读。

注: 匿名函数最难理解的地方就是要传入的参数是一个可迭代的对象，lambda 内部会调...  
展开 ▼

作者回复: 你说的对。关于迭代器生成器后面会讲到，所以这篇文章没有提及。



Geek\_59f23...

2019-05-31

👍 5

1. sorted(d.item(), key = lambda x: x[1], reverse = True)

2. 一般想偷懒和装X的时候用（来个玩笑☺）

展开 ▼



Geek\_59f23...

2019-05-31

👍 2

一楼说的对，list(map(###))和列表推导式对比更科学，显然后者生成列表速度更快，另外我实测圆括号生成器和map生成器速度在一个数量级，性能差别很小，结果如下：

函数generator被调用了1000000次，共计用时：2.248 秒

<generator object generator.<locals>.<genexpr> at 0x000002D735AE5ED0> ...

展开 ∨



catshitfiv...

2019-05-31

👍 2

应用场景举个栗子：比如在 pandas 中对二维数据进行数据分析时，对于某些数据块我们需要用函数如apply applymap transform 等进行临时性一次性的转换变更以得到最终的分析结果，那么就可以用匿名函数配合着来使用，使代码更简洁易读高效



爬行的蜗牛

2019-06-04

👍 1

1.reduce报错问题，python3需要加

from functools import reduce

2.filter，print变量时，需要加list()

如下：

l=[1,2,3,4,5]...

展开 ∨



风居住的街

2019-06-02

👍 1

# map 生成序列的效率是最高的，若要保证类型一致，赋值效率更高

a = timeit.timeit("map(lambda x: x\*2, range(100))")

b = timeit.timeit("list(map(lambda x: x\*2, range(100)))")

c = timeit.timeit("[i\*2 for i in range(100)]")

d = timeit.timeit("list1 = map(lambda x: x\*2, range(100))")...

展开 ∨



Steven

2019-06-02

👍 1

我还以为列表推导式已经是公认的说法了，原来还没有统一呀😁。

不过之前也有看过流畅的python，好像说也是列表推导式比map更快。



Jove

2019-05-31

👍 1

在python3中，map、filter函数返回的是迭代器，不是集合

展开 ∨



宾果

2019-06-06



前面有讲过这句： `sorted(d.items(),key=lambda x:x[1], reverse=True)`，问老师一个问题，这条语句中匿名函数的参数是怎么传递进去的呢，有点不太清楚？

展开 ∨



小侠龙旋风

2019-06-06



`sorted(d.items(),key=lambda x:x[1],reverse=True)`

展开 ∨



跑跑

2019-06-04



`dict(sorted(d.items(),key=lambda x:x[1], reverse=True))`

展开 ∨



lllong33

2019-06-04



第一问： `sorted(d.items(), key=lambda x,y: y, reverse=True)`，记得还有一种zip将字典k和v置换的方法。

第二问：pandas中的`apply()`，需要筛选某个元素的时候，判断是否包含特殊字符。



羽球码农

2019-06-03



2、reduce 在python 3的内置函数中不是被废弃了吗？

展开 ∨



羽球码农

2019-06-03



1、列表解析里是可以放常规函数的

展开 ∨

---



周平

2019-06-03



```
d = {'mike': 10, 'lucy': 2, 'ben': 30}
```

```
e=sorted(d.items(),key=lambda x:x[1],reverse=True)
```

```
print(dict(e))
```

展开 ∨

---



rogerr

2019-06-02



可耻地抄了一遍答案

```
sorted(d.items(),key=lambda x:x[1],reverse=True)
```

对lambda函数的语法还是不太清楚，感觉可读性没有普通函数那么清晰易懂呢，只是为了简洁么

展开 ∨

---



KK

2019-06-02



```
sort(d.items(d), key=lambda x: x[1], reverses=True)
```

接着打印出来，sort 反向进行排序

展开 ∨

---



阿坤儿00

2019-06-02



```
sorted(d.items(),key=lambda x:x[1],reverse=True)
```

这样返回的是列表，而不是字典，但是如果使用dict()，又会变成无序的字典，那是不是不能够生成有序字典呢？前面的写法就是第一问的正解吗？

展开 ∨

---





istock浅笑

2019-06-02



```
l = [1, 2, 3, 4, 5]
```

```
new_list = map(lambda x: x * 2, l) # [2, 4, 6, 8, 10]
```

这个运行

```
<map at 0x22028fb9898>...
```

展开 ∨