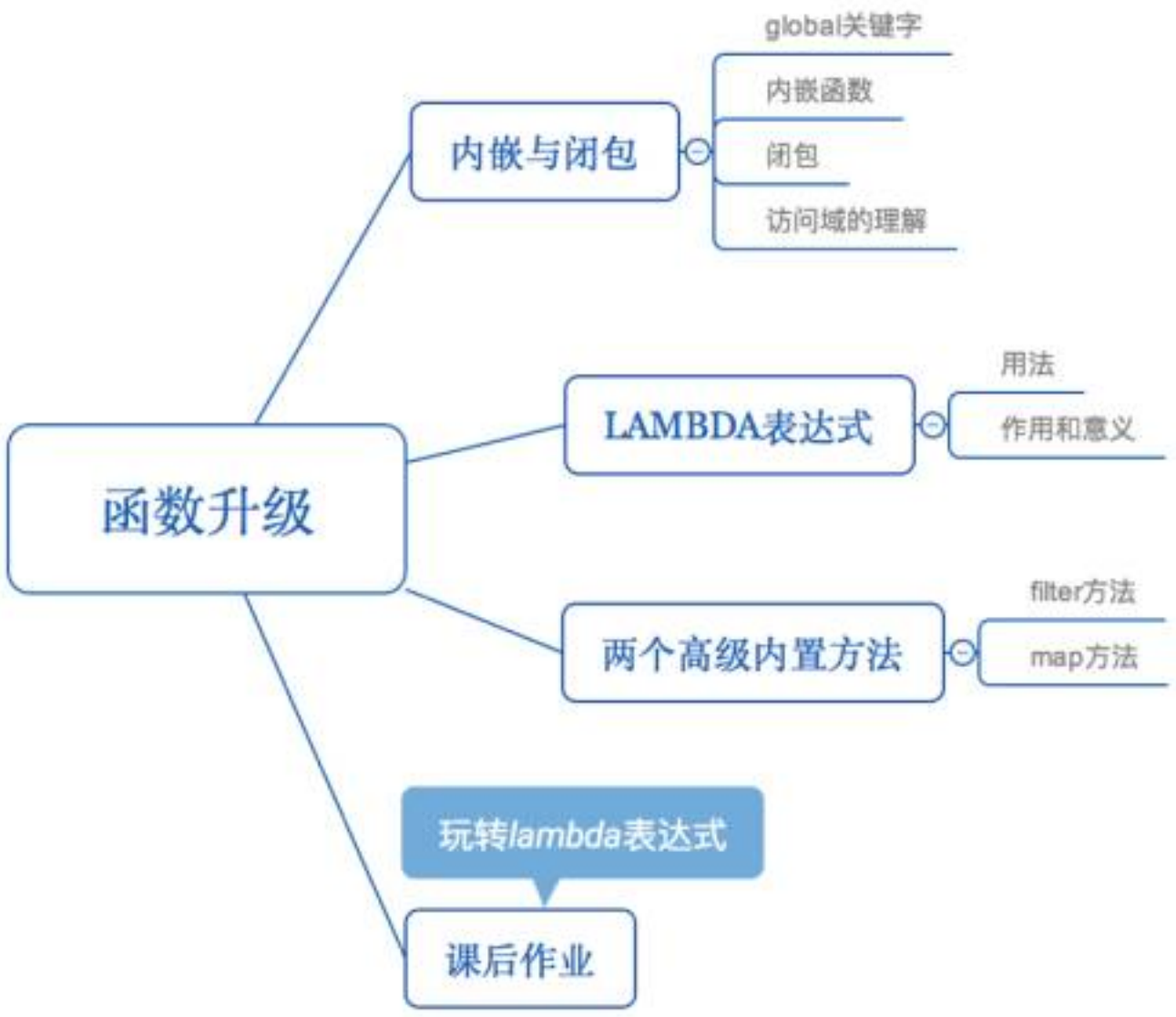
Python函数（下）－lambda表达式早有耳闻，了解之后，原来你这么强大

Original 2016-11-06 哈哈小 菜猿观世界

通过本节的学习，你将了解以下内容：



今天我们对python函数中的一些高级机制进行学习。今天的内容虽不错（后来补充：写完之后发现好像也不少），但是对于初学者来讲，在理解上，还是有一定困难的。

哈哈小要在这做个小预警，如果前面的内容你没有打扎实，这节看下去可能是一脸懵加问号的。

虽说难，但是趣味满满。要想对今天的内容一百八十度张开加合拢的大拥抱，就想想一道超难奥数题被你破解之后的快感吧。

好了，话不闲扯了，时候也不早了，我们就此进入主题吧。

1 glboal关键字

记得哈哈小前节有这么一句关键话：可以在函数内部去使用全局变量，但是不要试图去改变它，否者python之后会给你难看的脸色。

但偏有些剑走偏锋，任性自流之辈，想走一走这内部函数改全局变量的路子。

要想实现，倒也不难，glboal关键字来助你臂力。

举个例子：

```
>>> laiya=6
>>> def UseGlobal():
global laiya
laiya=4
print(laiya)

>>> UseGlobal()
4
>>> print(laiya)
4
>>>
```

laiya这个变量处于全局 ，比较得瑟，尝试在UseGlobal函数中改变它，在该变量前面使用了global关键字。global laiya 这句代码好像在对python说：嘿，python哥们，是不是有个叫laiya的变量在你的全局被定义了，把他带过来，我来好好修整他。

python一看到global这张“召唤令牌”，不得不从啊。于是乎，laiya全局变量就被好好修整了一番。最后就不奇怪调用UseGlobal函数把laiya打印出来的值和print直接打印值都为4了。

插播一句：这个例子告诉我们，做人不要自以为6了，否者就得4，哈哈～（玩笑ing）

2 内嵌函数

内嵌函数小名叫内部函数，听这名字应该对其意思猜出一二。

内嵌函数，顾他小名思他意，就是函数内部还有函数，并且在外部函数中调用内部函数。

可能有点绕，看下面这个例子：

```
>>> def Outside():
print("outside function is calling...")
def Inside():
print("inside function is calling...")
Inside()
```

```
>>> Outside()
outside function is calling...
inside function is calling...
>>>
```

在Outside函数中定义了一个叫Inside的函数，并在外部函数中的最后一行调用了在自己函数体定义的内部函数。

按照执行顺序，最后结果表明两个函数都执行了，这就是内嵌函数。

最后，哈哈小手贱多加了这么一句：

```
>>> Inside()
Traceback (most recent call last):
 File "<pyshell#16>", line 1, in <module>
   Inside()
NameError: name 'Inside' is not defined
>>>
```

NameError: name 'Inside' is not defined，报错信息告诉我们，在Outside函数外部去调用Inside函数是行不通的，用官方的话说就是Inside函数函数定义在Outside函数栈内，对外部是不可见的。

有些人可能在这会对python函数这种机制稍稍提出质疑，要实现上面这个过程，去定义两个独立的函数，分别去调用不就行了？使用内嵌函数，这一层内嵌还好，要是多个内嵌，这不是独增困惑嘛?

哈哈小要在这回答的是，既然存在，自有其存在的因果。所有的编程范式都是前人通过不断地写代码，不断地实践，提炼，概括，封装，总结而来的，一种一种范式的产生，一种一种地被推翻，为的就是代码的复用性提高，结构性提高。通俗地讲，就是使你的编程工作变得更简单。

记住，包括后面类似的问题，哈哈小都会这么回答的哦。

3 闭包（closure）

既然在函数内部定义和调用函数是可行的，由此而引出的一个重要的概念是：闭包（closure）。

前面都是直接在外部函数中直接调用内部函数，并且我们在全局是调用不到内部函数的。

由此我们把思维往前迈进一步，何不将内部函数作为外部函数的返回值返回？闭包就是这么一个概念。

我们用闭包来计算一个矩形的面积。

定义

```
>>> def closure1(width):
def closure2(height):
return width*height
return closure2
```

有发现，closure1这个函数的返回值是closure2内部函数。

调用

```
>>> closure1(5)
```

<function closure1.<locals>.closure2 at 0x1056a38c8>

发现输出的是一串看不懂，type关键字一探究竟。

```
>>> get=closure1(5)
>>> type(get)
<class 'function'>
```

返回的是函数类型的变量，没错，按道理也因该是的。

这回真正地调用一下

```
>>> get(8)
40
>>> closure1(5)(8)
40
>>>
```

两种方式都是没问题的。第二种输出方式就把closure1(5)看作一个整体（看作get）去理解。

对上述过程来一个总结：当调用 closure1(5)的时候（很明显，closure1函数的width参数传入的值是5），closure1函数执完，返回closure2函数，我们用get去接受这个函数，这个get于是就closure2附体了，所以，当我们get（8）的时候就相当于去调用了内部函数closure2并赋height的值为8。可以看到，closure2的返回值是width＊height，所以最后的结果就是40了。

4 访问域（access level）

讲到这里，哈哈小又想到一个问题，想想下面输出的结果会是什么？

```
>>> def closure1():
area=40
def closure2():
area=area+10
return area
return closure2()
>>> closure1()
```

[马赛克打这，看不到我]

结果不尽如人意啊，是这么一行报错信息：UnboundLocalError: local variable 'area' referenced before assignment

还是哈哈小前面多次讲到的访问域的问题。请注意这一行area＝area＋10，后面一个area来自全局变量值为40，当area加10变50，这时要把这个值赋值给又一个叫area的变量，前面讲过，pyhton机制会重新创建一个叫area的同名局部变量。如果是这样，左边的area是局部area，右边的是全局的area，python傻傻分不清，只好给你报错啰。

要想解决这一问题，其实很简单，只需要用到今天自global之后的又一关键字nonlocal

```
>>> def closure1():
area=40
def closure2():
nonlocal area
area=area+10
return area
return closure2()
>>> closure1()
50
>>>
```

完全没问题，似global，nonlocal是这样跟python对话的：嘿，python老哥，虽然我定义在局部，但是看在nonlocal大人的面子上，就把我变为全局变量吧。见到nonlocal，python二话不说，将area从局部升级为全局，area因此访问域也上升了一个等级。

再次插播：不管是学习编程还是任何其他东西，我们要不断争取到nonlocal使用权，以达到一个更高的可视访问，因为只有站得更高，我们才能看得更远，看得更广（哈哈鸡汤ing）

二 lambda

1 lambda语法

进入今天的第二part，也就是今天的主题，好玩的lambda表达式。

以前，我们要计算一个圆的面积，基本上都会这样做：

```
>>> def CircleArea(radius):
return 3.14*radius*radius
>>> CircleArea(4)
50.24
>>>
```

有了lambda表达式，再也不用担心自己的代码冗长了。

```
>>> get=lambda r : 3.14*r*r
>>> get(4)
50.24
>>>
```

lambda表达式的语法很简单：lambda （相当于函数参数放这，多个用逗号隔开）：（放相当于在函数体中要return的语句或语句块）

细心的伙伴可能已经发现，lambda不过是函数的变相，与函数不同的是，它没有函数名。对于没有名字的函数我们把它叫做匿名函数（anonymous function）

上面使用这个匿名函数的方法是创建一个变量，然后让这个变量传参去调用。

不难想到，除了上面的方式，下面这种方式也是可行的。

```
>>> (lambda r : 3.14*r*r)(4)
50.24
```

在强调语法时，有提到语句块一词，我们不妨用一例子来看看lambda表示式怎么操作语句块：

```
>>> isOdd=lambda x : print('yes') if (x+1)%2 else print('no')
>>> isOdd(8)
yes
>>> isOdd(11)
```

```
no
>>>
```

分析一下

lambda x : print('yes') if (x+1)%2 else print('no')，分号前面的x就是参数，分号后面print('yes') if (x+1)%2 else print('no')的意思就是说，如果（x+1）被2整除，也就是奇数，最后的值就是0（对if判断来讲，0就是false），if条件不成立，所以就执行else后面的内容，相反，如果为偶数就执行print（'yes'）语句。把整个一句当作人类语言的顺序去理解，难度应该也不会太大。

一行代码就完成了判断奇偶的函数，比起一般的函数，有了lambda表达式，有没有一种以前抓到的虫鱼鸟兽直接拿来生吃而现在烹蒸烤焖各种吃法的感觉?

2 lambda表达式的作业

两个例子，大家应该也能感受到lambda的作用不容小视。

尽管如此，哈哈小还是要在这对lambda表达式不吝赞美之词对其大赞一番：

当我们写的程序变得日益壮大的时候

- 当你头疼于给函数起名时，lambda绝对是你的不二选择
- lambda表达式给代码编写带来的精简性是不容质疑，有目共睹的
- 当你调用某个函数而忘记其具体实现时，你可能要跑到def这个函数的地方去看看，lambda表达式可以助你省去这样的步骤，给你的编码效率和代码可读性带来同步的提高

三 两个高级内置方法

lambda表达式已如此强大，再结合两个内置方法来使用它。这时，哈哈小会告诉你，前方高能，因为你将一睹它核弹爆发般的能量。同时，也请带好你的大脑，因为你的感知系统都在那。

1 filter内置方法

filter要是直接翻译过来的话，就是过滤器的意思。

既然时filter方法，那就help一下，看看这个方法怎么用。

```
>>> help(filter)
Help on class filter in module builtins:
class filter(object)
|  filter(function or None, iterable) --> filter object
```

（部分显示）

filter(function or None, iterable) 三个参数，第一个为函数或None，第二个为iterable对象，记得哈哈小曾经说过，三大序列，字符串，列表，元组都是可迭代对象。

我们以具体的例子来看看filter方法怎么用

如果传入第一个参数为None，过滤对象是['haha','',None,True,False]，最后并以列表的形式把过滤后的结果显示出来：

```
>>> list(filter(None,['haha','',None,True,False]))
['haha', True]
>>>
```

从输出结果可以得出一个结论：可迭代对象中的元素凡是判断为假的都被过滤掉，为真的留下（知识点回顾：在python中，除了False判断结果为假之外，还有'',"",None,[],(),{}也都会判断为假，其他的判断为真）

接下来，我们第一个参数传入一个函数试试，在这种情况下，lambda可谓是派上大用场了。

与前面判断一个数是否为偶数的lambda表达式不同，这次我们用元组来打印一定范围内的所有偶数。

```
>>> tuple(filter(lambda x:(x+1)%2,range(1,101)))
(2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100)
>>>
```

1到100的所有偶数就这样轻松被打印出来了。

我们来分析一下filter的两个参数，第一个参数是一个匿名函数lambda表达式，函数内容是需要传入一个参数x，执行的语句是(x+1)%2 。第二个参数是range（1，101），即1到100的数字序列。当使用filter函数，后面的数字序列会一个一个地去调用前面的lambda表达式，二lambda最后执行的结果如果为0就会被过滤掉（所有奇数会被过滤掉），留下来的就是偶数了。

2 map内置方法

基本知道了filter函数的使用，使用map函数起来就得心应手多了。

map如果要在这翻译成汉语的话，哈哈小觉得最合适的翻译应该是：映射

```
>>> help(map)
Help on class map in module builtins:
class map(object)
|  map(func, *iterables) --> map object
```

（部分显示）

通过help可以知道map的第一个参数为函数对象，第二个参数为可变参数，可以传入多个可迭代对象。

车江中学七年级二班，现在有这么一道数学题：

已知函数f(x)=x*x-2x+1，x为－10到12的整数，求对应的函数值。

我们如果用程序来实现，会变得如此滴简单：

```
>>> list(map((lambda x:x*x-2*x+1),range(-10,13)))
[121, 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
>>>
```

map用起来也是很有feel啊，通过一个函数，将参数x映射成x*x-2x+1，最后把映射后的结果显示。

如果传入多个参数，看看回是什么情况。

```
>>> list(map((lambda x,y:[x,y]),[1,2,3,4],[0,1,4,9]))
[(1, 0), (2, 1), (3, 4), (4, 9)]
>>>
```

map，filter配limbda函数，还是哈哈小那句话，就好比流水配清风，星星配月亮，白天配黑夜，和谐而具有感召力。

四 课后作业

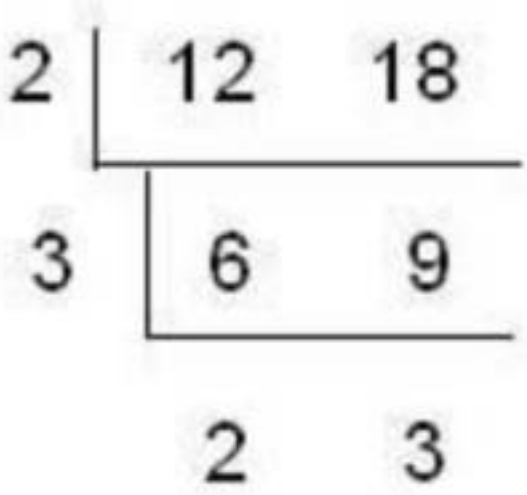
limbda与内置方法用的恰到好处，可以实现一些高级的数学计算。如果你表示怀疑，我表示有力向你证明。

续上节：

三年级的小明昨晚的家庭作业中有这么两道题，哈哈小觉得用我们今天所学，以程序的思维来解答，会有点意思。

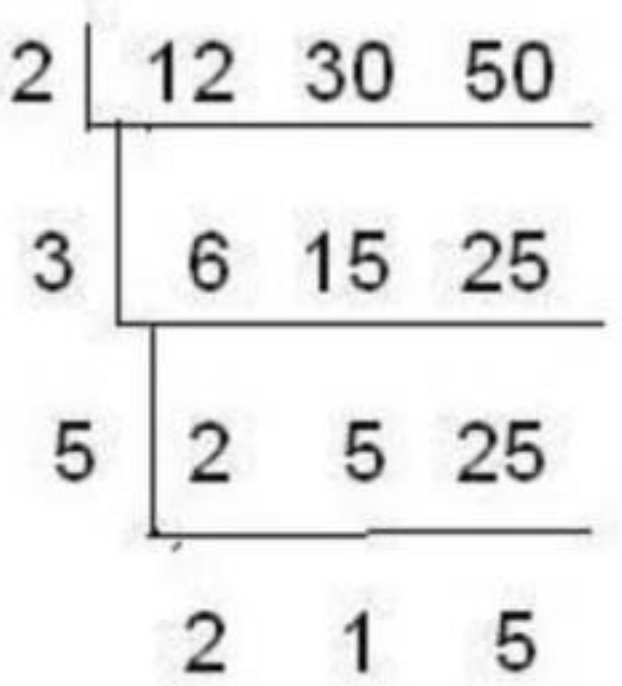
题目是，求12，30，50的最小公倍数以及12和18的最大公约数。

如果有对最大公约数和最小公倍数概念模糊的同学，下图一见明朗：



最大公约数
2×3=6

图1



最小公倍数
2×3×5×2×1×5=300

图2

对于第一题，哈哈小的思路是这样的：

先创建这么一个过滤器

```
filter((lambda x: x%12 and x%30 and x%50),range(1,12*30*50+1))
```

通过lambda表达式，可以很容易知道，1到12*30*50之间的整数，凡是既能被12能被30也能被50同时整除的数将被过滤。

好像最后list出来的结果与我们预期的结果刚好相反，看来这个过滤器看来还得优化一下。

既然相反，那好办，加个not在前面就好了。

这里哈哈小要补充逻辑运算上的一个知识点：

A and B and C <=> not (A or B or C)

知道这点了，最后我们需要的过滤器会是这样的：

```
filter((lambda x: not (x%12 or x%30 or x%50)),range(1,12*30*50+1))
```

万事俱备，用list打印出来看看

```
>>> result=list(filter((lambda x: not (x%12 or x%30 or x%50)),range(1,12*30*50)))
>>> result
[300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700, 3000, 3300, 3600, 3900, 4200, 4500, 4800,
5100, 5400, 5700, 6000, 6300, 6600, 6900, 7200, 7500, 7800, 8100, 8400, 8700, 9000, 9300, 9600,
9900, 10200, 10500, 10800, 11100, 11400, 11700, 12000, 12300, 12600, 12900, 13200, 13500, 13800,
14100, 14400, 14700, 15000, 15300, 15600, 15900, 16200, 16500, 16800, 17100, 17400, 17700, 18000]
>>>
```

哎，对了我们已经看到了我们想要的结果300，处于第一位，拿出来，直接索引：

```
>>> result[0]
300
>>>
```

把上述过程用一句代码表示：

```
>>> list(filter((lambda x: not (x%12 or x%30 or x%50)),range(1,12*30*50)))[0]
300
>>>
```

求12，30，50的最小公倍数这一题哈哈小的解题思路这就展示完了。

接下来就到你们解答第二题的时间了，最后也像哈哈小这样，以一行代码表示出来。赶快行动吧～(回复1025，见课后习题答案。)

函数的内容到这儿基本就已告一段落了，讲完函数，哈哈小也是感慨万千啊。一方面感受到python强大数据

处理能力的魅力，另一方面，有一种与python相见恨晚的感觉，要是青春年少时，就掌握了python，那些数学课后作业还不是个个手到擒来啊，哈哈～

最后插播一句，由于哈哈小的文章是每周末更新，所以最后说句周内愉快应该不会被打吧？

结束语：我们不想给你带来多少多少的知识，我们只想尽绵薄之力给你带来可能的智慧的启迪。

Read more

Read more