7.7 匿名函数捕获变量值¶

问题¶

你用lambda定义了一个匿名函数，并想在定义时捕获到某些变量的值。

解决方案¶

先看下下面代码的效果：

```
>>> x = 10
>>> a = lambda y: x + y
>>> x = 20
>>> b = lambda y: x + y
>>>
```

现在我问你，a(10)和b(10)返回的结果是什么？如果你认为结果是20和30，那么你就错了：

```
>>> a(10)
30
>>> b(10)
30
>>>
```

这其中的奥妙在于lambda表达式中的x是一个自由变量， 在运行时绑定值，而不是定义时就绑定，这跟函数的默认值参数定义是不同的。 因此，在调用这个lambda表达式的时候，x的值是执行时的值。例如：

```
>>> x = 15
>>> a(10)
25
>>> x = 3
>>> a(10)
13
>>>
```

如果你想让某个匿名函数在定义时就捕获到值，可以将那个参数值定义成默认参数即可，就像下面这样：

```
>>> x = 10
>>> a = lambda y, x=x: x + y
>>> x = 20
>>> b = lambda y, x=x: x + y
>>> a(10)
20
>>> b(10)
30
>>>
```

讨论¶

在这里列出来的问题是新手很容易犯的错误，有些新手可能会不恰当的使用lambda表达式。 比如，通过在一个循环或列表推导中创建一个lambda表达式列表，并期望函数能在定义时就记住每次的迭代值。例如：

```
>>> funcs = [lambda x: x+n for n in range(5)]
```

```
>>> for f in funcs:
... print(f(0))
...
4
4
4
4
4
>>>
```

但是实际效果是运行是n的值为迭代的最后一个值。现在我们用另一种方式修改一下：

```
>>> funcs = [lambda x, n=n: x+n for n in range(5)]
>>> for f in funcs:
... print(f(0))
...
0
1
2
3
4
>>>
```

通过使用函数默认值参数形式，lambda函数在定义时就能绑定到值。