5 个很好的 Python 面试题 (/a/1190000000618513)

yexiaobai (/u/yexiaobai) 2014年07月26日 发布

推荐 **4** 推荐

收藏 **46** 收藏，14.2k 浏览

> 注：本文的原文是 5 Great Python Interview Questions (http://www.toptal.com/python/interview-questions)，同时谢谢 @非乌龟 指出我的疏漏，没有来源标记，也赞其细心，希望看文章的同时大家都能看下原文，因为每个人的理解不一致，原汁原味的最有帮助，我翻译很多文章的目的一是为了自己以后找资料方便；二是作为一个索引，以后再看原文的时候，能更加快捷。其目的还是希望大家能看原文的。

问题一：以下的代码的输出将是什么？说出你的答案并解释。

```
class Parent(object):
    x = 1

class Child1(Parent):
    pass

class Child2(Parent):
    pass

print Parent.x, Child1.x, Child2.x
Child1.x = 2
print Parent.x, Child1.x, Child2.x
Parent.x = 3
print Parent.x, Child1.x, Child2.x
```

答案

以上代码的输出是：

```
1 1 1
1 2 1
3 2 3
```

使你困惑或是惊奇的是关于最后一行的输出是 `3 2 3` 而不是 `3 2 1`。为什么改变了 `Parent.x` 的值还会改变 `Child2.x` 的值，但是同时 `Child1.x` 值却没有改变？

这个答案的关键是，在 Python 中，类变量在内部是作为字典处理的。如果一个变量的名字没有在当前类的字典中发现，将搜索祖先类（比如父类）直到被引用的变量名被找到（如果这个被引用的变量名既没有在自己所在的类又没有在祖先类中找到，会引发一个 `AttributeError` 异常 ）。

因此，在父类中设置 `x = 1` 会使得类变量 `X` 在引用该类和其任何子类中的值为 1。这就是因为第一个 print 语句的输出是 `1 1 1` 。

随后，如果任何它的子类重写了该值（例如，我们执行语句 `Child1.x = 2` ），然后，该值仅仅在子类中被改变。这就是为什么第二个 `print` 语句的输出是 `1 2 1` 。

最后，如果该值在父类中被改变（例如，我们执行语句 `Parent.x = 3` ），这个改变会影响到任何未重写该值的子类当中的值（在这个示例中被影响的子类是 `Child2` ）。这就是为什么第三个 `print` 输出是 `3 2 3` 。

问题二：以下的代码的输出将是什么？说出你的答案并解释？

```
def div1(x,y):
    print("%s/%s = %s" % (x, y, x/y))

def div2(x,y):
    print("%s//%s = %s" % (x, y, x//y))

div1(5,2)
div1(5.,2)
div2(5,2)
div2(5.,2.)
```

答案

这个答案实际依赖于你使用的是 Python 2 还是 Python 3。

在 Python 3 中，期望的输出是：

```
5/2 = 2.5
5.0/2 = 2.5
5//2 = 2
5.0//2.0 = 2.0
```

在 Python 2 中，尽管如此，以上代码的输出将是：

```
5/2 = 2
5.0/2 = 2.5
5//2 = 2
5.0//2.0 = 2.0
```

默认，如果两个操作数都是整数，Python 2 自动执行整型计算。结果， `5/2` 值为 `2` ，然而 `5./2` 值为 ```2.5``。

注意，尽管如此，你可以在 Python 2 中重载这一行为（比如达到你想在 Python 3 中的同样结果），通过添加以下导入：

```
from __future__ import division
```

也需要注意的是“双划线”（ // ）操作符将一直执行整除，而不管操作数的类型，这就是为什么 `5.0//2.0` 值为 `2.0` 。

> 注： 在 Python 3 中， / 操作符是做浮点除法，而 // 是做整除（即商没有余数，比如 10 // 3 其结果就为 3，余数会被截除掉，而 (-7) // 3 的结果却是 -3 。这个算法与其它很多编程语言不一样，需要注意，它们的整除运算会向0的方向取值。而在 Python 2 中， / 就是整除，即和 Python 3 中的 // 操作符一样， ）

问题三：以下代码将输出什么?

```
list = ['a', 'b', 'c', 'd', 'e']
print list[10:]
```

答案

以上代码将输出 [] ，并且不会导致一个 IndexError 。

正如人们所期望的，试图访问一个超过列表索引值的成员将导致 IndexError （比如访问以上列表的 list[10] ）。尽管如此，试图访问一个列表的以超出列表成员数作为开始索引的切片将不会导致 IndexError ，并且将仅仅返回一个空列表。

一个讨厌的小问题是它会导致出现 bug ，并且这个问题是难以追踪的，因为它在运行时不会引发错误。

问题四：以下的代码的输出将是什么? 说出你的答案并解释？

```
def multipliers():
    return [lambda x : i * x for i in range(4)]

print [m(2) for m in multipliers()]
```

你将如何修改 multipliers 的定义来产生期望的结果

答案

以上代码的输出是 [6, 6, 6, 6] （而不是 [0, 2, 4, 6] ）。

这个的原因是 Python 的闭包的后期绑定导致的 late binding (http://en.wikipedia.org/wiki/Late_binding)，这意味着在闭包中的变量是在内部函数被调用的时候被查找。所以结果是，当任何 multipliers() 返回的函数被调用，在那时， i 的值是在**它被调用时的周围作用域中查找**，到那时，无论哪个返回的函数被调用， for 循环都已经完成了， i 最后的值是 3 ，因此，每个返回的函数 multiplies 的值都是 3。因此一个等于 2 的值被传递进以上代码，它们将返回一个值 6 （比如： 3 x 2 ）。

（顺便说下，正如在 The Hitchhiker's Guide to Python (http://docs.python-guide.org/en/latest/writing/gotchas/) 中指出的，这里有一点普遍的误解，是关于 lambda 表达式的一些东西。一个 lambda 表达式创建的函数不是特殊的，和使用一个普通的 def 创建的函数展示的表现是一样的。 ）

这里有两种方法解决这个问题。

最普遍的解决方案是创建一个闭包，通过使用默认参数立即绑定它的参数。例如：

```
def multipliers():
    return [lambda x, i=i : i * x for i in range(4)]
```

另外一个选择是，你可以使用 `functools.partial` 函数：

```
from functools import partial
from operator import mul

def multipliers():
    return [partial(mul, i) for i in range(4)]
```

问题五：以下的代码的输出将是什么？说出你的答案并解释？

```
def extendList(val, list=[]):
    list.append(val)
    return list

list1 = extendList(10)
list2 = extendList(123,[])
list3 = extendList('a')

print "list1 = %s" % list1
print "list2 = %s" % list2
print "list3 = %s" % list3
```

你将如何修改 `extendList` 的定义来产生期望的结果

以上代码的输出为：

```
list1 = [10, 'a']
list2 = [123]
list3 = [10, 'a']
```

许多人会错误的认为 `list1` 应该等于 `[10]` 以及 `list3` 应该等于 `['a']`。认为 `list` 的参数会在 `extendList` 每次被调用的时候会被设置成它的默认值 `[]`。

尽管如此，实际发生的事情是，新的默认列表仅仅只在函数被定义时创建一次。随后当 `extendList` 没有被指定的列表参数调用的时候，其使用的是同一个列表。这就是为什么当函数被定义的时候，表达式是用默认参数被计算，而不是它被调用的时候。

因此， `list1` 和 `list3` 是操作的相同的列表。而 ````list2 是操作的它创建的独立的列表（通过传递它自己的空列表作为 list``` 参数的值）。

`extendList` 函数的定义可以做如下修改，但，当没有新的 `list` 参数被指定的时候，会总是开始一个新列表，这更加可能是一直期望的行为。

```
def extendList(val, list=None):
    if list is None:
        list = []
    list.append(val)
    return list
```

使用这个改进的实现，输出将是：

```
list1 = [10]
list2 = [123]
list3 = ['a']
```

python (/t/python/blogs)　闭包 (/t/%E9%97%AD%E5%8C%85/blogs)

链接 (/a/1190000000618513)　更多

4 推荐　收藏

你可能感兴趣的文章

python 中的闭包 (/a/1190000000344681)　1.8k 浏览

Python 闭包的理解 (/a/1190000002965736)　11 收藏，1k 浏览

swift基础之_闭包 (/a/1190000004047682)　112 浏览

讨论区

看来水平还是不行啊，就对了一题。

#1 (/c/1050000000618589) **andong777** (/u/andong777) · 2014年07月26日 · 回复

对不起，我读出了文章里的翻译腔，然后我查到了原文http://www.toptal.com/python/interview-questions (http://www.toptal.com/python/interview-questions) 。请注明翻译转载，谢谢。

#2 (/c/1050000000619187) **非乌龟** (/u/feiwugui) · 2014年07月27日 · 回复

我是在微博推送上看到的。您其他文章都是注明了来源的，这篇可能只是疏漏了，倒个歉。但觉得还是标一下好。

#3 (/c/1050000000619195) **非乌龟** (/u/feiwugui) · 2014年07月27日 · 回复

回复 非乌龟 (/u/feiwugui) :
已经修正，谢谢指出这个问题。

#4 (/c/1050000000619355) **yexiaobai** (/u/yexiaobai) · 2014年07月27日 · 回复

问题四 应该这么修改是最简单的吧。修改撑生成器
def multipliers():
return （ lambda x : i * x for i in range(4))

print [m(2) for m in multipliers()]

#5 (/c/1050000000707019) **menghuan072** (/u/menghuan072) · 2014年10月03日 · 回复

请先 登录 后评论

本文隶属于专栏

yexiaoxiaobai (/blog/yexiaobai)

刀锋上的跳舞的 SRE，喜爱 Perl, Python, Go，常被正则婊戏耍，热衷于鼓捣一些小玩具。

关注专栏

分享扩散：

http://segmentfault.com/a/1190000000618513 缩短