Python编程过程中的注意事项

# 可变数据类型作为函数定义中的默认参数

## 实例演示及问题展示

def fun(url,urlsList = []):  
 urlsList.append(url)  
 return urlsList  
  
print(fun("http://www.baidu.com"))  
print(fun("http://www.sina.com.cn"))  
print(fun("http://www.sohu.com"))

结果本来可能想的是：

["http://www.baidu.com"]  
["http://www.sina.com.cn"]  
["http://www.sohu.com"]

但是，结果却是：

['http://www.baidu.com']  
['http://www.baidu.com', 'http://www.sina.com.cn']  
['http://www.baidu.com', 'http://www.sina.com.cn', 'http://www.sohu.com']

## 原因剖析

为什么呢？如你所见，**每次都使用的是同一个列表**，输出为什么会是这样？**在 Python 中，当我们编写这样的函数时，这个列表被实例化为函数定义的一部分。当函数运行时，它并不是每次都被实例化。**这意味着，这个函数会一直使用完全一样的列表对象，除非我们提供一个新的对象：

print(fun("http://www.baidu.com",[]))  
print(fun("http://www.sina.com.cn",[]))  
print(fun("http://www.sohu.com",[]))

结果如下：  
['http://www.baidu.com']  
['http://www.sina.com.cn']  
['http://www.sohu.com']

## 改变方法

对默认可变数据参数类型进行判断**是否为空**。

def fun2(url,urlsList = []):  
 if not urlsList:  
 urlsList = []  
 urlsList.append(url)  
 return urlsList

print(fun2("http://www.baidu.com"))  
print(fun2("http://www.sina.com.cn"))  
print(fun2("http://www.sohu.com"))

结果为：

['http://www.baidu.com']  
['http://www.sina.com.cn']  
['http://www.sohu.com']

这将在模块加载的时候移走实例化的内容，以便每次运行函数时都会发生列表实例化。请注意，**对于不可变数据类型，比如元组、字符串、整型，是不需要考虑这种情况的**。

# 可变数据类型作为类变量

## 实例演示及问题展示

class URLCatcher(object):  
 urls = []  
 def add\_url(self, url):  
 self.urls.append(url)  
  
a = URLCatcher()  
a.add\_url('http://www.google.com')  
print(a.urls)  
  
b = URLCatcher()

# print(b.urls)#['http://www.google.com']  
b.add\_url('http://www.bbc.co.hk')  
print(b.urls)

预期结果为：

['http://www.google.com']  
['http://www.bbc.co.hk']

但是，结果为

['http://www.google.com']  
['http://www.google.com', 'http://www.bbc.co.hk']

## 原因剖析

我们实例化了两个单独的对象 a 和 b。把一个 URL 给了 a，另一个给了 b。这两个对象怎么会都有这两个 URL 呢？

这和第一个错例是同样的问题。创建类定义时，URL 列表将被实例化。该类所有的实例使用相同的列表。在有些时候这种情况是有用的，但大多数时候你并不想这样做。你希望每个对象有一个单独的储存。为此，我们修改代码为：

class URLCatcher(object):  
 def \_\_init\_\_(self):  
 self.urls = []  
 def add\_url(self, url):  
 self.urls.append(url)  
  
a = URLCatcher()  
a.add\_url('http://www.google.com')  
print(a.urls)  
  
b = URLCatcher()  
# print(b.urls)#[]  
b.add\_url('http://www.bbc.co.hk')  
print(b.urls)

结果为：

['http://www.google.com']  
['http://www.bbc.co.hk']

现在，当创建对象时，URL 列表被实例化。当我们实例化两个单独的对象时，它们将分别使用两个单独的列表

# 可变的分配错误

## 实例演示及问题展示

字典类型：

dict1 = {'1':'one','2':'two'}  
dict2 = dict1  
dict2['3'] = 'three'  
print(dict1)  
print(dict2)

预期结果：

{'1': 'one', '2': 'two'}  
{'1': 'one', '2': 'two', '3': 'three'}

但是，结果却是：

{'1': 'one', '2': 'two', '3': 'three'}  
{'1': 'one', '2': 'two', '3': 'three'}

列表类型：

list1 = ['one','two']  
list2 = list1  
list2.append('three')  
print(list1)#['one', 'two', 'three']  
print(list2)#['one', 'two', 'three']

假如是不可变数据类型，如元组：

tuple1 = (1,2)  
tuple2 = tuple1  
tuple2 = (3,4)  
print(tuple1)#(1, 2)  
print(tuple2)#(3, 4)

## 原因剖析

那么，在之前的例子中到底发生了什么？**当使用可变类型时，其行为有点像 C 语言的一个指针**。在上面的代码中，我们令 b = a，我们真正表达的意思是：**b 成为 a 的一个引用**。**它们都指向 Python 内存中的同一个对象**。听起来有些熟悉？那是因为这个问题与先前的相似。其实，这篇文章应该被称为「**可变引发的麻烦**」。

## 解决办法

列表和字典都会发生这种情况。如何解决呢？

如果我们真的需要复制一个列表进行处理，我们可以这样做：

list2 = list1[:]

这将遍历并复制列表中的每个对象的引用，并且把它放在一个新的列表中。但是要注意：如果列表中的每个对象都是可变的，我们将再次获得它们的引用，而不是完整的副本。

假设在一张纸上列清单。在原来的例子中相当于，A 某和 B 某正在看着同一张纸。**如果有个人修改了这个清单，两个人都将看到相同的变化**。当我们复制引用时，每个人现在有了他们自己的清单。但是，我们假设这个清单包括寻找食物的地方。如果“冰箱”是列表中的第一个，即使它被复制，两个列表中的条目也都指向同一个冰箱。所以，如果冰箱被 A 修改，吃掉了里面的大蛋糕，B 也将看到这个蛋糕的消失。这里没有简单的方法解决它。只要你记住它，并编写代码的时候，使用不会造成这个问题的方式。

字典以相同的方式工作，并且你可以通过以下方式创建一个昂贵副本：

dict2 = dict1.copy()

## 避免这些问题的出现

可变数据类型的麻烦也是它们强大的地方。**以上都不是实际中的问题；它们是一些要注意防止出现的问题**。在第三个项目中使用昂贵复制操作作为解决方案在 99% 的时候是没有必要的。你的程序或许应该被改改，所以在第一个例子中，这些副本甚至是不需要的。