7.5 定义有默认参数的函数¶

问题¶

你想定义一个函数或者方法，它的一个或多个参数是可选的并且有一个默认值。

解决方案¶

定义一个有可选参数的函数是非常简单的，直接在函数定义中给参数指定一个默认值，并放到参数列表最后就行了。例如：

```
def spam(a, b=42):
    print(a, b)

spam(1) # Ok. a=1, b=42
spam(1, 2) # Ok. a=1, b=2
```

如果默认参数是一个可修改的容器比如一个列表、集合或者字典，可以使用None作为默认值，就像下面这样：

```
# Using a list as a default value
def spam(a, b=None):
    if b is None:
        b = []
    ...
```

如果你并不想提供一个默认值，而是想仅仅测试下某个默认参数是不是有传递进来，可以像下面这样写：

```
_no_value = object()

def spam(a, b=_no_value):
    if b is _no_value:
        print('No b value supplied')
    ...
```

我们测试下这个函数：

```
>>> spam(1)
No b value supplied
>>> spam(1, 2) # b = 2
>>> spam(1, None) # b = None
>>>
```

仔细观察可以发现到传递一个None值和不传值两种情况是有差别的。

讨论¶

定义带默认值参数的函数是很简单的，但绝不仅仅只是这个，还有一些东西在这里也深入讨论下。

首先，默认参数的值仅仅在函数定义的时候赋值一次。试着运行下面这个例子：

```
>>> x = 42
>>> def spam(a, b=x):
```

```
...     print(a, b)
...
>>> spam(1)
1 42
>>> x = 23 # Has no effect
>>> spam(1)
1 42
>>>
```

注意到当我们改变x的值的时候对默认参数值并没有影响，这是因为在函数定义的时候就已经确定了它的默认值了。

其次，默认参数的值应该是不可变的对象，比如None、True、False、数字或字符串。 特别的，千万不要像下面这样写代码：

```
def spam(a, b=[]): # NO!
    ...
```

如果你这么做了，当默认值在其他地方被修改后你将会遇到各种麻烦。这些修改会影响到下次调用这个函数时的默认值。比如：

```
>>> def spam(a, b=[]):
...     print(b)
...     return b
...
>>> x = spam(1)
>>> x
[]
>>> x.append(99)
>>> x.append('Yow!')
>>> x
[99, 'Yow!']
>>> spam(1) # Modified list gets returned!
[99, 'Yow!']
>>>
```

这种结果应该不是你想要的。为了避免这种情况的发生，最好是将默认值设为None， 然后在函数里面检查它，前面的例子就是这样做的。

在测试None值时使用 `is` 操作符是很重要的，也是这种方案的关键点。 有时候大家会犯下下面这样的错误：

```
def spam(a, b=None):
    if not b: # NO! Use 'b is None' instead
        b = []
    ...
```

这么写的问题在于尽管None值确实是被当成False， 但是还有其他的对象(比如长度为0的字符串、列表、元组、字典等)都会被当做False。 因此，上面的代码会误将一些其他输入也当成是没有输入。比如：

```
>>> spam(1) # OK
>>> x = []
>>> spam(1, x) # Silent error. x value overwritten by default
>>> spam(1, 0) # Silent error. 0 ignored
>>> spam(1, '') # Silent error. '' ignored
>>>
```

最后一个问题比较微妙，那就是一个函数需要测试某个可选参数是否被使用者传递进来。 这时候需要小心的是你不能用某个默认值比如None、 0或者False值来测试用户提供的值(因为这些值都是合法的值，是可能被用户传递进来的)。 因此，你需要其他的解决方案了。

为了解决这个问题，你可以创建一个独一无二的私有对象实例，就像上面的_no_value变量那样。 在函数里面，你可以通过检查被传递参数值跟这个实例是否一样来判断。 这里的思路是用户不可能去传递这个_no_value实例作为输入。 因此，这里通过检查这个值就能确定某个参数是否被传递进来了。

这里对 `object()` 的使用看上去有点不太常见。 `object` 是python中所有类的基类。 你可以创建 `object` 类的实例，但是这些实例没什么实际用处，因为它并没有任何有用的方法， 也没有任何实例数据(因为它没有任何的实例字典，你甚至都不能设置任何属性值)。 你唯一能做的就是测试同一性。这个刚好符合我的要求，因为我在函数中就只是需要一个同一性的测试而已。