学习Python Doc第三天:函数

张朝龙

目录

在定义函数的时候，我们可以为输入参数指定默认值，也可以使输入参数的个数可变，等等。今天，我们深入讨论一下函数。

1 为函数指定默认参数值

在定义函数的时候可以为函数的一个或者多个参数指定默认值。通过这种方法，我们定义了一个可变参量的函数，在调用的时候，如果给定了默认值的参数没有被赋值则用默认值代替，看代码：

```
1  def ask_ok(prompt,retries=4,reminder='please␣try␣again'):
2      while True:
3          ok = input(prompt)
4          if ok in ('y','ye','yes'):
5              return True
6          if ok in ('n','no','nop','nope'):
7              return False
8          retries = retries -1
9          if retries < 0 :
10             raise ValueError('invalid␣user␣response')
11         print(reminder)
```

这个函数可以用三种方式调用：

1. 给定一个参数， `ask_ok('Do you really want to quit?')`
2. 给一个可选参数赋值， `ask_ok('OK to overwrite the file',2)`
3. 给所有参数赋值， `ask_ok('OK to overwrite the file',2,'come on, only yes or no!')`

这个函数在第 4 行引入了一个关键词 `in` ,这个关键词用来测试一个序列是不是包含了特定值。

参数的默认值在定义函数的时候就被执行了，看代码：

```
i = 5
def f(arg = i):
    print(arg)

i = 6
f()
```

输出为：

```
5
```

可以看到 `arg=i` 中的 `i` 在定义的时候被赋值为 `5` 这个函数定义的时候就把 `arg` 赋值为 `5` 。

注意：默认值只被赋值一次。但是当默认参量是一个list，dictionary或者大多数class的instance时执行过程会稍有不同，看代码：

```
1  def f(a,L = []):
2      L.append(a)
3      return L
4
5  print(f(1))
6  print(f(2))
7  print(f(3))
```

输出为：

```
[1]
[1, 2]
[1, 2, 3]
```

如果你不想在多次调用过程中共享默认值参量，你可以这样写:

```
def f(a,L = None ):
    if L is None:
        L = [];
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))
```

输出为：

```
[1]
[2]
[3]
```

2 关键词参数

关键词参数我还不是很理解，在python doc里有一个例子：

```
def parrot(voltage,state = 'a stiff', action = 'voom', type = 'Norwegian Blue'):
    print("-- This parrot wouldn't ", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's ",state,"!")
```

这个函数接受一个必选参数 voltage 和三个可选参数 state action type ，这个参数通过以下语句来调用：

```
parrot(1000)                                          # 1 positional argument
parrot(voltage=1000)                                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
```

但是下面的几种调用方式都是非法的：

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument after a keyword argument
parrot(110, voltage=220)     # duplicate value for the same argument
parrot(actor='John Cleese')  # unknown keyword argument
```

当一个函数的最后一个参数是 **name 这种类型时，该函数接受字典类型数据作为输入，看代码:

```
def cheeseshop(kind, *argument, **keywords):
    print("-- Do you have any",kind,"?")
    print("-- I'am sorry, we are all out of",kind)
    for arg in argument:
        print(arg)
    print('-'*40)
    keys = sorted(keywords.keys())
    for kw in keys:
        print(kw,":",keywords[kw])
```

调用时，可以这样子：

```
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
```

```
        shopkeeper="Michael Palin",
        client="John Cleese",
        sketch="Cheese Shop Sketch")
```

输出为：

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
client : John Cleese
shopkeeper : Michael Palin
sketch : Cheese Shop Sketch
```

3 任意参数列表

一个函数可以被设计的支持任意参数。这些参数被放置到一个tuple里。看代码：

```
1 def write_multiple_items(file, separator, *args):
2     file.write(separator.join(args))
```

任何在 *args 之后出现的参数都被 *args 接受。另外在定义函数时， *args 之后的参数必须以 keyword-value 的形式出现。

比如：

```
1 def concat(*args,sep = "/"):
2     return sep.join(args)
```

调用和输出为:

```
In [333]: concat("earth","mars","venus")
Out[337]:
'earth/mars/venus'

In [338]: concat("earth","mars","venus",sep='.')
Out[356]:
'earth.mars.venus'
```

4 Lambda 表达式

可以使用 lambda 来表示一些小的匿名函数。比如 lambda a,b: a+b 返回 a,b 的和。 lambda 可以被用在对象调用的地方。从语法上来看， lambda 表达式是一句话。 python 中的 lambda 有点语法糖的味道。一个简单的例子，看代码：

```
1 def make_incrementor(n):
2     return lambda x: x+n
```

调用和输出为：

```
In [1]:
In [2]: f = make_incrementor(42)
```

```
In [33]: f(0)
Out[33]:
42

In [34]: f(1)
Out[34]:
43

In [35]: f(4)
Out[38]:
46
```

make_incrementor 生成的是一个 generator .

5 代码文档格式规范化

在写 Python 代码的过程中，可以顺便把代码文档也给完成，颇有文学编程的感觉。对于函数来说可以通过以下方式告知该函数的信息：

```
1 def my_function():
2     """␣Do␣nothing,␣it␣doesn't␣do␣anything
3
4 ␣␣␣␣no,␣really,␣it␣doesn't␣do␣anything
5 ␣␣␣␣"""
6
7     pass
```

通过 print(my_function.__doc__) 查看 my_function 的文档：

```
In [40]: print(my_function.__doc__)
 Do nothing, it doesn't do anything

    no, really, it doesn't do anything
```

当 Python 工程较大时，更需要规范化的文档和编程风格。所幸 PEP 8 提供了高度可读的编码风格，每个 python 程序员都应当遵守这个规范（尤其参与大项目的时候，像我平常做的小实验，我就不那么严格的要求了）。