

学习Python Doc第三天:函数

张朝龙

目录

1 为函数指定默认参数值	1
2 关键词参数	2
3 任意参数列表	3

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数的函数,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：

```
1 def f(a,L = None ):
2     if L is None:
3         L = []
4         L.append(a)
5     return L
6
7 print(f(1))
8 print(f(2))
9 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：

```
1 def parrot(voltage,state = 'a stiff', action = 'vroom', type = 'Norwegian Blue'):
2     print("-- This parrot wouldn't", action, end=' ')
3     print("if you put", voltage, "volts through it.")
4     print("-- Lovely plumage, the", type)
5     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：

<code>parrot(1000)</code>	# 1 positional argument
<code>parrot(voltage=1000)</code>	# 1 keyword argument
<code>parrot(voltage=1000000, action='V00000M')</code>	# 2 keyword arguments
<code>parrot(action='V00000M', voltage=1000000)</code>	# 2 keyword arguments
<code>parrot('a million', 'bereft of life', 'jump')</code>	# 3 positional arguments
<code>parrot('a thousand', state='pushing up the daisies')</code>	# 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：

```
1 def cheeseshop(kind, *argument, **keywords):
2     print("-- Do you have any", kind, "?")
3     print("-- I'm sorry, we are all out of", kind)
4     for arg in argument:
5         print(arg)
6     print('-'*40)
7     keys = sorted(keywords.keys())
8     for kw in keys:
9         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'
```