目录

18 把盒子升级为豪宅：函数进阶

更新时间：2019-09-26 09:49:00

> 不经一翻彻骨寒，怎得梅花扑鼻香。
>
> ——宋帆

位置参数

位置参数这个东西我们并不陌生，之前所编写的函数使用的就是位置参数。**位置参数**，顾名思义，传入函数时每个参数都是通过位置来作区分的。函数调用时，传入的值需按照位置与参数一一对应。

比如：

```
def overspeed_rate(current, max, min):
    if current > max:
        return (current - max) / max    # 超过最大时速，结果为正
    elif current < min:
        return (current - min) / min    # 超过最小时速，结果为负
    else:
        return 0                        # 不超速，结果为 0
```

这个函数用来判断车辆在高速上行驶时超速的比例。它接受三个参数， current 表示当前时速， max 参数表示当前路段的允许的最大时速， min 表示所允许的最小时速。

位置参数需要按位置顺序来传递，否则结果不可预期。

```
>>> overspeed_rate(150, 120, 90)
0.25 # 超过最大时速 25%
>>> overspeed_rate(80, 100, 60)
0 # 不超速
```

参数默认值

前面的函数中，如果最大时速和最小时速比较固定，那么每次函数调用时都输入这个两个参数就显得有些繁琐，这时我们可以使用参数默认值。

参数默认值也就是给参数设置默认值，之后函数调用时便可以不传入这个参数，Python 自动以默认值来填充参数。如果一个有默认值的参数依然被传入了值，那么默认值将会被覆盖。

函数定义时，以 参数=值 来指定参数默认值。如下：

```
def 函数(参数1, 参数2=默认值):
    pass
```

例如上面的 overspeed_rate 函数， max 和 min 通常比较固定，我们可以使用一个常用值来作为默认值。

```
def overspeed_rate(current, max=120, min=90):
    if current > max:
        return (current - max) / max
    elif current < min:
        return (current - min) / min
    else:
        return 0
```

```
>>> overspeed_rate(192)
0.6
>>> overspeed_rate(45)
-0.5
```

关键字参数

对于 overspeed_rate 函数，我们还可以在**函数调用时**，以 参数名=值 的形式来向指定的参数传入值。

如：

```
overspeed_rate(100, min=80)
```

或者

```
overspeed_rate(current=100, min=80)
```

或者

```
overspeed_rate(current=100, max=100, min=80)
```

使用关键字时甚至可以打乱参数传递次序：

```
overspeed_rate(min=80, max=100, current=100)
```

```
>>> overspeed_rate(min=80, max=100, current=100)
0
```

但要注意，关键字参数需要出现在位置参数之后，否则将抛出 SyntaxError 异常：

```
>>> overspeed_rate(100, max=100, 80)
    File “ ”, line 1
SyntaxError: positional argument follows keyword argument
```

关键字参数的用法还不止如此。

当我们在定义函数时，如果参数列表中某个参数使用 **参数名 形式，那么这个参数可以接受一切关键字参数。如下：

```
def echo(string, **keywords):
    print(string)
    for kw in keywords:
        print(kw, ":", keywords[kw])
```

```
>>> echo(‘hello’, today=‘2019-09-04’, content=‘function’, section=3.6)
hello
today : 2019-09-04
content : function
section : 3.6
```

显然，我们并没有在函数定义时定义 today 、content 、section 参数，但是我们却能接收到它们，这正是 **keywords 发挥了作用。函数会将所有接收到的关键字参数组装成一个字典，并绑定到 keywords 上。验证一下：

```
>>> def foo(**keywords):
…     print(keywords)
…
>>> foo(a=1, b=2, c=3)
{‘a’: 1, ‘b’: 2, ‘c’: 3}
```

任意参数列表

如，计算任意个数的乘积：

```
def multiply(*nums):
    result = 1
    for n in nums:
        result *= n
    return result
```

```
>>> multiply(1,3,5,7)
105
```

这个函数能接收任意个参数，这正是 *nums 所发挥的作用。函数所有接收到的非关键字参数组装成一个元组，并绑定到 nums 上。来试验一下：

```
>>> def multiply(*nums):
...     print(nums)
...
>>> multiply(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5)
```

多返回值

典型情况下，函数只有一个返回值，但是 Python 也支持函数返回多个返回值。

要返回多个返回值，只需在 return 关键字后跟多个值（依次用逗号分隔）。

例如：

```
def date():
    import datetime
    d = datetime.date.today()
    return d.year, d.month, d.day
```

date() 返回了今天的日期的年、月、日。

接收函数返回值时，用对应返回值数量的变量来分别接收它们。

```
>>> year, month, day = date()
>>> year
2019
>>> month
9
>>> day
4
```

后将元组返回。来验证下：

```
>>> date()
(2019, 9, 4)
```

接收返回值时， year, month, day = date() ，这样赋值写法，会将元组解包，分别将元素赋予单独的变量中。即：

```
>>> year, month, day = (2019, 9, 4)
>>> year
2019
>>> month
9
>>> day
4
```

← 17 生成器表达式和列表生成式 19 让你的模子更好用：类进阶 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论

千学不如一看，千看不如一练