附录-函数的参数

2023 年 9 月 8 日

函数的参数

对于函数的调用者来说，只需要知道如何传递正确的参数，以及函数将返回什么样的值就够了，函数内部的复杂逻辑被封装起来，调用者无需了解。

Python 的函数定义非常简单，但灵活度却非常大。除了正常定义的位置参数外，还可以使用默认参数、可变参数和关键字参数，不但能处理复杂的参数，还可以简化调用者的代码。

1. 位置参数

我们先写一个计算 x^2 的函数：

```
def power(x):
    return x * x
```

对于 `power(x)` 函数，参数 `x` 就是一个位置参数。

当我们调用 `power` 函数时，必须传入有且仅有的一个参数 `x`：

```
>>> power(5)
25
>>> power(15)
225
```

现在，如果我们要计算 x^3 怎么办？可以再定义一个 `power3` 函数，但是如果要计算 x^4、x^5……怎么办？我们不可能定义无限多个函数。

你也许想到了，可以把 `power(x)` 修改为 `power(x, n)`，用来计算 x^n，说干就干：

```
def power(x, n):
    s = 1
    while n > 0:
```

```
        n = n - 1
        s = s * x
    return s
```

对于这个修改后的 `power(x, n)` 函数，可以计算任意 n 次方：

```
>>> power(5, 2)
25
>>> power(5, 3)
125
```

修改后的 `power(x, n)` 函数有两个参数：`x` 和 `n`，这两个参数都是位置参数，调用函数时，传入的两个值按照位置顺序依次赋给参数 `x` 和 `n`。

2. 默认参数

新的 `power(x, n)` 函数定义没有问题，但是，旧的调用代码失败了，原因是我们增加了一个参数，导致旧的代码因为缺少一个参数而无法正常调用：

```
>>> power(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: power() missing 1 required positional argument: 'n'
```

Python 的错误信息很明确：调用函数 `power()` 缺少了一个位置参数 `n`。

这个时候，默认参数就排上用场了。由于我们经常计算 x^2，所以，完全可以把第二个参数 n 的默认值设定为 2：

```
def power(x, n=2):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

这样，当我们调用 `power(5)` 时，相当于调用 `power(5, 2)`：

```
>>> power(5)
25
>>> power(5, 2)
```

```
25
```

而对于 `n > 2` 的其他情况，就必须明确地传入 `n`，比如 `power(5, 3)`。

从上面的例子可以看出，默认参数可以简化函数的调用。设置默认参数时，有几点要注意：

- 位置参数在前，默认参数在后，否则 Python 的解释器会报错（思考一下为什么默认参数不能放在位置参数前面）；
- 如何设置默认参数。

当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。

使用默认参数有什么好处？最大的好处是能降低调用函数的难度。

举个例子，我们写个一年级大学新生注册的函数，需要传入 `name` 和 `gender` 两个参数：

```
def enroll(name, gender):
    print('name:', name)
    print('gender:', gender)
```

这样，调用 `enroll()` 函数只需要传入两个参数：

```
>>> enroll('Sarah', 'Female')
name: Sarah
gender: Female
```

如果要继续传入年龄、城市等信息怎么办？这样会使得调用函数的复杂度大大增加。

我们可以把年龄和城市设为默认参数：

```
def enroll(name, gender, age=18, city='Shanghai'):
    print('name:', name)
    print('gender:', gender)
    print('age:', age)
    print('city:', city)
```

这样，大多数学生注册时不需要提供年龄和城市，只提供必须的两个参数：

```
>>> enroll('Sarah', 'Female')
name: Sarah
gender: Female
```

```
age: 6
city: Shanghai
```

只有与默认参数不符的学生才需要提供额外的信息：

```
enroll('Bob', 'Male', 19)
enroll('Adam', 'Male', city='Hangzhou')
```

可见，默认参数降低了函数调用的难度。无论是简单调用还是复杂调用，函数只需要定义一个。

有多个默认参数时，调用的时候，既可以按顺序提供默认参数，比如调用 `enroll('Bob', 'Male', 19)`，意思是，除了 `name`，`gender` 这两个参数外，最后 1 个参数应用在参数 `age` 上，`city` 参数由于没有提供，仍然使用默认值。

也可以不按顺序提供部分默认参数。当不按顺序提供部分默认参数时，需要把参数名写上。比如调用 `enroll('Adam', 'M', city='Tianjin')`，意思是，`city` 参数用传进去的值，其他默认参数继续使用默认值。

3. 可变参数

在 Python 函数中，还可以定义可变参数。顾名思义，可变参数就是传入的参数个数是可变的，可以是 1 个、2 个到任意个，还可以是 0 个。

我们以数学题为例子，给定一组数字 a，b，c……，请计算 $a^2 + b^2 + c^2 + ...$。

要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把 a，b，c……作为一个 list 或 tuple 传进来，这样，函数可以定义如下：

```
def calc(numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

但是调用的时候，需要先组装出一个 list 或 tuple：

```
>>> calc([1, 2, 3])
14
>>> calc((1, 3, 5, 7))
84
```

如果利用可变参数，调用函数的方式可以简化成这样：

```
>>> calc(1, 2, 3)
14
>>> calc(1, 3, 5, 7)
84
```

所以，我们把函数的参数改为可变参数：

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

定义可变参数和定义一个 list 或 tuple 参数相比，仅仅在参数前面加了一个 * 号。在函数内部，参数 numbers 接收到的是一个 tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括 0 个参数：

```
>>> calc(1, 2)
5
>>> calc()
0
```

如果已经有一个 list 或者 tuple，要调用一个可变参数怎么办？可以这样做：

```
>>> nums = [1, 2, 3]
>>> calc(nums[0], nums[1], nums[2])
14
```

这种写法当然是可行的，问题是太繁琐，所以 Python 允许你在 list 或 tuple 前面加一个 * 号，把 list 或 tuple 的元素变成可变参数传进去：

```
>>> nums = [1, 2, 3]
>>> calc(*nums)
14
```

*nums 表示把 nums 这个 list 的所有元素作为可变参数传进去。这种写法相当有用，而且很常见。

4. 关键字参数

试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，而且我们在设计函数时，并不知道调用者是不是还想填写更多的信息，比如课程信息。利用关键字参数来定义

这个函数可以满足这个需求。

关键字参数允许你传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 dict。请看示例：

```
def enroll(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)
```

函数 `enroll` 除了位置参数 `name` 和 `age` 外，还接受关键字参数 `kw`。在调用该函数时，可以只传入位置参数：

```
>>> enroll('Michael', 18)
name: Michael age: 18 other: {}
```

也可以传入任意个数的关键字参数：

```
>>> enroll('Bob', 19, city='Shanghai')
name: Bob age: 19 other: {'city': 'Shanghai'}
>>> enroll('Adam', 17, gender='Male', course='Accounting')
name: Adam age: 17 other: {'gender': 'Male', 'course': 'Accounting'}
```

关键字参数有什么用？它可以扩展函数的功能。比如，在 `enroll` 函数里，我们保证能接收到 `name` 和 `age` 这两个参数，但是，如果调用者愿意提供更多的参数，我们也能收到。

和可变参数类似，也可以先组装出一个 dict，然后，把该 dict 转换为关键字参数传进去：

```
>>> extra = {'city': 'Shanghai', 'course': 'Accounting'}
>>> enroll('Jack', 18, **extra)
name: Jack age: 18 other: {'city': 'Shanghai', 'course': 'Accounting'}
```

5. 命名关键字参数

对于关键字参数，函数的调用者可以传入任意不受限制的关键字参数。如果我们想要调用者传入恰当的信息，就可以使用命名关键字参数。

仍以 `enroll()` 函数为例，我们希望检查是否有 `city` 和 `course` 参数：

```
def enroll(name, age, **kw):
    if 'city' in kw:
        # 有 city 参数
        pass
    if 'course' in kw:
```

```
        # 有 course 参数
        pass
    print('name:', name, 'age:', age, 'other:', kw)
```

但是调用者仍可以传入不受限制的关键字参数：

```
>>> enroll('Jack', 18, city='Shanghai', addr='Dongfang Road No.32', zipcode=200000)
```

如果要限制关键字参数的名字，就可以用命名关键字参数，例如，只接收 city 和 course 作为关键字参数。这种方式定义的函数如下：

```
def enroll(name, age, *, city, course):
    print(name, age, city, course)
```

和关键字参数 **kw 不同，命名关键字参数需要一个特殊分隔符 *，* 后面的参数被视为命名关键字参数。

调用方式如下：

```
>>> enroll('Jack', 18, city='Shanghai', course='Accounting')
Jack 18 Shanghai Accounting
```

如果没有传入参数名，调用将报错：

```
>>> enroll('Jack', 18, 'Shanghai', 'Accounting')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: enroll() takes 2 positional arguments but 4 were given
```

由于调用时缺少参数名 city 和 course，Python 解释器把这 4 个参数均视为位置参数，但 enroll() 函数仅接受 2 个位置参数。

命名关键字参数可以有缺省值，从而简化调用：

```
def enroll(name, age, *, city='Shanghai', course):
    print(name, age, city, course)
```

由于命名关键字参数 city 具有默认值，调用时，可不传入 city 参数：

```
>>> enroll('Jack', 18, course='Accounting')
Jack 18 Shanghai Accounting
```

使用命名关键字参数时，要特别注意，如果没有可变参数，就必须加一个 * 作为特殊分隔符。如果缺少 *，Python 解释器将无法识别位置参数和命名关键字参数：

```
def enroll(name, age, city, course):
    # 缺少 *，city 和 course 被视为位置参数
    pass
```

6. 参数组合

在 Python 中定义函数，可以用位置参数、默认参数、可变参数、关键字参数和命名关键字参数，这 5 种参数都可以组合使用。但是请注意，参数定义的顺序必须是：位置参数、默认参数、可变参数、命名关键字参数和关键字参数。

比如定义一个函数，包含上述若干种参数：

```
def f1(a, b, c=0, *args, **kw):
    print('a =', a)
    print('b =', b)
    print('c =', c)
    print('args =', args)
    print('kw =', kw)


def f2(a, b, c=0, *, d, **kw):
    print('a =', a)
    print('b =', b)
    print('c =', c)
    print('d =', d)
    print('kw =', kw)
```

在函数调用的时候，Python 解释器自动按照参数位置和参数名把对应的参数传进去。

```
>>> f1(1, 2)
>>> f1(1, 2, c=3)
>>> f1(1, 2, 3, 'a', 'b')
>>> f1(1, 2, 3, 'a', 'b', x=99)
>>> f2(1, 2, d=99, ext=None)
```

最神奇的是通过一个 tuple 和 dict，你也可以调用上述函数，请在 jupyter 尝试这些调用，看看运行结果：

```
>>> args = (1, 2, 3, 4)
>>> kw = {'d': 99, 'x': '#'}
>>> f1(*args, **kw)
```

```
>>> args = (1, 2, 3)
>>> kw = {'d': 88, 'x': '#'}
>>> f2(*args, **kw)
```

虽然可以组合多达 5 种参数，但不要同时使用太多的组合，否则函数接口的可理解性很差。

!!! Note

Python的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的参数。要注意定义

- `*args`是可变参数，args接收的是一个tuple；

- `**kw`是关键字参数，kw接收的是一个dict。

以及调用函数时如何传入可变参数和关键字参数的语法：

- 可变参数既可以直接传入：`func(1, 2, 3)`；

- 关键字参数既可以直接传入：`func(a=1, b=2)`.

使用`*args`和`**kw`是Python的习惯写法，当然也可以用其他参数名，但最好使用习惯用法。

[ ]: