第 5 章

Python 函数

在前面的章节中，所有编写的代码都是从上到下依次执行的，如果某段代码需要多次使用，则需要将该段代码复制多次，这种做法势必会影响到开发效率。在实际开发中，如果有若干段代码的执行逻辑完全相同，那么可以考虑将这些代码抽象成一个函数，这样不仅可以提高代码的重用性，而且条理会更加清晰，可靠性更高。

从本质上来说，函数就是将一段具有独立功能的代码块整合到一个整体并命名，在需要的位置调用这个名称即可完成对应的需求。函数在开发过程中，可以更高效地实现代码重用。但是如果不主动调用函数，代码是不会执行的。在调用函数的过程中需要外部代码将数据传入函数，而函数又要将内部的数据传给外部的代码。因此，完成数据交换需要两个要素：参数和返回值。如果外部代码需要调用函数，也需要有个名字，即函数名。因此在 Python 中理解函数的定义即一个拥有名称、参数和返回值的代码块。

本章将对如何定义和调用函数，以及函数的参数、变量的作用域等进行详细介绍。

5.1 函数的定义和调用

在 Python 中，函数的应用非常广泛。在前面已经多次接触过函数。例如，用于输出的 print()函数、用于输入的 input()函数，以及用于生成一系列整数的 range()函数。这些都是 Python 内置的标准函数，可以直接使用。除了可以直接使用的标准函数外，Python 还支持自定义函数，函数是组织好的，可重复使用的，用来实现单一或相关联功能的代码段，它能够提高应用的模块化和代码的重复利用率。

5.1.1 定义函数

由前面的描述可知，函数是可以调用的，而且是可以交互的。因此函数要有三个重要元素：函数名，函数参数和返回值，其中，函数名是必需的，函数参数和返回值是可选的。Python 定义函数以 def 开头，定义函数的基本格式如下：

```
def  函数名(参数列表):
    '''函数注释字符串'''
    函数体
```

参数说明：

- 函数代码块以 def 开头，后面紧跟的是函数名和圆括号()，以冒号(：)结束。因此

函数内部的代码需要用缩进量来与外部代码分开。

- 函数名：在调用函数时使用，其命名规则跟变量的名字是一样的，即只能是字母、数字和下画线的任何组合，但是不能以数字开头，并且不能跟关键字重名。
- 函数的参数：可选参数，用于指定向函数中传递的参数。如果有多个参数，各参数间使用逗号“,”分隔。如果不指定，则表示该函数没有参数。在调用时，也不指定参数。参数必须放在圆括号中，即使函数没有参数，也必须保留一对空的小括号“()”，否则将显示语法错误。由于 Python 是动态语言，所以函数参数与返回值不需要事先指定数据类型。
- 函数注释字符串（函数的说明文档）：函数的说明文档也叫函数的文档说明，可选参数，表示为函数指定注释，注释的内容通常是说明该函数的功能、要传递的参数的作用等。在调用函数时，输入函数名称及左侧的小括号，可以为用户提供友好提示和帮助的内容。通常使用一对单引号或双引号将多行注释内括起来。通常可以由 help（函数名）查看。
- 函数体：可选参数，用于指定函数体，即该函数被调用后，要执行的功能代码。如果函数有返回值，可以使用 return 语句返回，结束函数，返回值传给调用方。return 语句可以返回任何值，可以是一个值、一个变量，或者另外一个函数的返回值。不带表达式的 return 相当于返回 None。如果想定义一个什么也不做的空函数，可以使用 pass 语句作为占位符。

函数体和注释相对于 def 关键字必须保持一定的缩进。

需要注意的是，如果参数列表包含多个参数，默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配的。

【例 5-1】 定义一个打印信息的函数。

```
def printInfo():
    '''定义一个函数,能够完成打印信息的功能。
    '''
    print('-----------------------------------')
    print('不忘初心,牢记使命')
    print('-----------------------------------')
```

运行上面的代码，将不显示任何内容，也不会抛出异常，因为 printInfo ()函数还没有调用。

【例 5-2】 定义一个查看函数说明文档的函数。

```
#函数的说明文档的高级使用
def sum_num1(a, b):
    """
    求和函数 sum_num1
    :param a: 参数 1
    :param b: 参数 2
    :return: 返回值
```

```
    """
    return a +b

help(sum_num1)
```

运行结果：

```
Help on function sum_num1 in module __main__:

sum_num1(a, b)
求和函数 sum_num1
    :param a: 参数 1
    :param b: 参数 2
    :return: 返回值
```

由例 5-2 可以看出，help 函数的作用是查看函数的说明文档（函数的解释说明的信息）。

5.1.2　调用函数

定义了函数之后，就相当于有了一段具有某些功能的代码，要想让这些代码能够执行，需要调用它。调用函数的基本语法格式如下：

```
函数名称([函数参数])
```

参数说明：

- 函数名称：要调用的函数名称必须是已经创建好的。
- 可选参数：用于指定各个参数的值。如果需要传递多个参数值，则各参数值间使用逗号“,”分隔。如果该函数没有参数，则直接写一对小括号即可。

【例 5-3】 调用 printInfo()函数。

```
printInfo()
```

运行结果：

```
-------------------------------------
不忘初心，牢记使命
-------------------------------------
```

5.1.3　函数的返回值

到目前为止，上述创建的函数都只是为做一些事，做完了就结束了。但实际上，有时还需要对事情的结果进行获取。这类似于主管向下级员工下达命令，员工去做，最后需要将结果报告给主管。假设把函数看作工厂里的机器，往机器里输送生产用的材料，最终机器会将产品生产好。这里可以把产品的材料看作函数的参数，做好的产品看作函数的输出，而生产过程就是函数体的代码了。

为函数设置返回值的作用就是将函数的处理结果返回给调用它的函数。所谓“返回值”,就是程序中的函数完成一件事情后,最后给调用者的结果。

在 Python 中,可以在函数体内使用 return 语句为函数指定返回值。该返回值可以是任意类型,并且无论 return 语句出现在函数的什么位置,只要得到执行,就会直接结束函数的执行。

return 语句的语法格式:

```
return [value]
```

参数说明:

- return:为函数指定返回值后,在调用函数时,可以把它赋给一个变量(如 result),用于保存函数的返回结果。如果返回一个值,那么 result 中保存的就是返回的一个值,该值可以是任意类型。如果返回多个值,那么这些值会聚集起来并以元组类型返回。
- value:可选参数,用于指定要返回的值,可以返回一个值,也可以返回多个值。当函数中没有 return 语句时,或者省略了 return 语句的参数时,将返回 None,即返回空值。

【例 5-4】 return 语句应用示例:定义一个两个数相加的函数。

```
#定义函数
def add2num(a, b):
    c=a+b
    return c
    print("函数运行中")
#调用函数
sumValue =add2num(2,3)
print(sumValue)
```

运行结果:

```
5
```

由例 5-4 可以看出,return 语句的作用如下。

(1) 负责函数返回值。

(2) 退出当前函数:导致 return 下方的所有代码(函数体内部)不执行。

一般情况下,每个函数都有一个 return 语句,如果函数没有定义返回值,那么返回值就是 None,None 表示没有任何值,属于 NoneType 类型。返回值个数与返回值类型的对应关系如表 5-1 所示。

表 5-1 返回值对应类型

返回值个数	返回值类型
0	None
1	Object
大于 1	Tuple

如果一个函数要有多个返回值，可以采取“return a，b”的写法，返回多个数据的时候，默认是元组类型。return 后面也可以连接列表、元组或字典，以返回多个值。

5.1.4　函数的嵌套调用

在一个函数中调用了另外一个函数，这就是函数嵌套调用。其执行流程为如果在函数 A 中调用了另外一个函数 B，那么先把函数 B 中的任务都执行完毕之后才会回到上次函数 A 执行的位置。接下来，通过一个示例来演示。

【例 5-5】 函数的嵌套调用。

```
#计算三个数之和
def sum_num(a, b, c):
    return a +b +c

#求三个数的平均值
def average_num(a, b, c):
    sumResult =sum_num(a, b, c)
    return sumResult / 3

result =average_num(1, 2, 3)
print(result)
```

运行结果：

```
2.0
```

5.2　函数的参数与值传递

在调用函数时，大多数情况下，主调函数和被调用函数之间有数据传递关系，这就是有参数的函数形式。函数参数的作用是传递数据给函数使用，函数利用接收的数据进行具体的操作处理。函数参数在定义函数时放在函数名称后面的一对小括号中。函数的参数的优势是函数调用的时候可以传入真实数据，增大函数的使用的灵活性。

5.2.1　函数的形参和实参

在使用函数时，经常会用到形式参数（形参）和实际参数（实参），二者的关系类似于剧本选主角一样，剧本的角色相当于形参，而演角色的演员就相当于实参。其中，

（1）形式参数：在定义函数时，函数名后面括号中的参数为“形式参数”，简称“形参”。形参对于函数调用者来说是透明的，即形参叫什么，与调用者无关。形参是在函数内部使用的，函数外部并不可见。

（2）实际参数：在调用一个函数时，函数名后面括号中的参数为“实际参数”。也就是将函数的调用者提供给函数的参数称为实际参数，简称“实参”。

根据实际参数的类型不同，可以分为将实际参数的值传递给形式参数，和将实际参数

的引用传递给形式参数两种情况。其中,当实际参数为不可变对象时,进行的是值传递;当实际参数为可变对象时,进行的是引用传递。

实际上,值传递和引用传递的基本区别就是,进行值传递后,改变形式参数的值,实际参数的值不变;而进行引用传递后,改变形式参数的值,实际参数的值也一同改变。

【例 5-6】 定义一个名称为 printString()的函数,然后为 printString()函数传递一个字符串类型的变量作为参数(代表值传递),并在函数调用前后分别输出该字符串变量,再为 printString()函数传递列表类型的变量作为参数(代表引用传递),并在函数调用前后分别输出该列表。

```
#定义函数
def printString(obj):
    print("原值:",obj)
    obj +=obj
#调用函数
print("------------值传递------------")
strslogan ="不忘初心,牢记使命"
print("函数调用前:",strslogan)
printString(strslogan)                          #采用不可变对象:字符串
print("函数调用后:",strslogan)
print("------------引用传递------------")
listslogan =["不忘初心","牢记使命"]
print("函数调用前:",listslogan)
printString(listslogan)                         #采用可变对象:列表
print("函数调用后:",listslogan)
```

运行结果:

```
------------值传递------------
函数调用前:不忘初心,牢记使命
原值:不忘初心,牢记使命
函数调用后:不忘初心,牢记使命
------------引用传递------------
函数调用前: ['不忘初心', '牢记使命']
原值: ['不忘初心', '牢记使命']
函数调用后: ['不忘初心', '牢记使命', '不忘初心', '牢记使命']
```

从上面的执行结果中可以看出,在进行值传递时,改变形式参数的值后,实际参数的值不改变;在进行引用传递时,改变形式参数的值后,实际参数的值也发生改变。

如果传递的变量类型是数值、字符串、布尔等类型,那就是值传递;如果传递的变量类型为序列、对象(后边章节会介绍)等复合类型,就是引用传递。由于值传递就是在传递时将自身复制一份,而在函数内部接触的参数实际上是传递给函数的变量的副本,修改副本的值自然不会影响到原始变量。而像序列、对象这样的复合类型的变量,在传入函数时,实际上也将其复制了一份,但复制的不是变量中的数据,而是变量的引用。因为这些复合

类型在内存是一块连续或不连续的内存空间保存，要想找到这些复合类型的变量传入函数，复制的是内存空间的首地址，而这个首地址就是复合类型数据的引用。

【例 5-7】 根据身高、体重计算 BMI 指数。

定义一个名称为 fun_bmi()的函数，该函数包括 3 个参数，分别用于指定姓名、身高和体重，再根据公式：BMI＝体重/(身高×身高)，计算 BMI 指数，并输出结果。

```
def fun_bmi(person,height,weight):
    '''功能:根据身高和体重计算 BMI 指数
    :param person: 姓名
    :param height: 身高
    :param weight: 体重
    :return: none
    '''
    print(person+"的身高为:"+str(height)+"米体重:"+str(weight)+"千克")
    bmi =weight/(height * height)
    print(person+"的 BMI 指数为:"+str(bmi))
    #判断身材是否正常
    if bmi <18.5:
        print("你的体重过轻!")
    if bmi >=18.5 and bmi <24.9:
        print("正常范围,注意保持。")
    if bmi >=24.9 and bmi <29.9:
        print("你的体重过重!")
    if bmi >=29.9:
        print("肥胖!")
#函数定义
fun_bmi("李福" , 1.78 , 75)
```

运行结果：

```
李福的身高为:1.78 米体重:75 千克
李福的 BMI 指数为:23.671253629592222
正常范围,注意保持。
```

5.2.2 位置参数

位置参数也称必备参数，调用函数时根据函数定义的参数位置来传递参数。当调用函数时，传入的参数位置是和定义函数的参数位置对应的，即调用时的数量和位置必须和定义时是一样的。

1. 数量必须与定义时一致

在调用函数时，指定的实际参数的数量必须与形式参数的数量一致，否则将抛出 TypeError 异常，提示缺少必要的位置参数。

例如，调用根据身高、体重计算 BMI 指数的函数 fun_bmi(person, height, weight)，若参

数少传一个，即只传递两个参数：fun_bmi("高晓萌", 1.75)，抛出 TypeError 异常类型。

2. 位置必须与定义时一致

在调用函数时，指定的实际参数的位置必须与形式参数的位置一致，否则将产生以下两种结果。

（1）形参和实参的类型一致，而产生的结果和预期不一致。

（2）实际参数的类型与形式参数的类型不一致，类型不能正常转换，抛出 TypeError 异常类型。

由于调用函数时，传递的实参位置与形参位置不一致时，并不会总是抛出异常，所以在调用函数时一定要确定好位置，否则容易产生 Bug，而且不容易被发现。

【例 5-8】 位置参数的应用示例。

```
def winprize(name, prize):
    return "{}荣获{}".format(name, prize)
#调用函数
print(winprize("李二毛", "特等奖"))
```

在上述例子中，如果使用代码 print(winprize("特等奖","李二毛"))调用函数，并不会抛出异常，但会输出如下内容："特等奖荣获李二毛"，输出内容并不符合要求。就是位置参数所致，因此位置参数传递和定义参数的顺序及个数必须一致。

5.2.3 关键字参数

关键字参数是指使用形式参数的名字来确定输入的参数值。函数调用，通过"键＝值"形式加以指定。可以让函数更加清晰、容易使用，同时也清除了参数的顺序需求。通过该方式指定实际参数时，不再需要与形式参数的位置完全一致。只要将参数名写正确即可。这样可以避免用户需要牢记参数位置的麻烦，使得函数的调用和参数传递更加灵活方便。例如：

```
print(winprize(prize="特等奖",name="李二毛"))
```

显示结果为"李二毛荣获特等奖"，显然是正确的。

关键参数也可以与位置参数混合使用，例如：

```
"print(winprize("李二毛", prize="特等奖"))"
```

在混合使用时，关键字参数必须放在位置参数后面，否则会抛出异常。

【例 5-9】 关键字参数应用示例。

```
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')
user_info('Rose', age=20, gender='女')
user_info('huayi', gender='男', age=6)
```

运行结果：

```
您的名字是 Rose, 年龄是 20, 性别是女
```

```
您的名字是 huayi, 年龄是 6, 性别是男
```

由例 5-9 可以看出，函数调用时，如果有位置参数时，位置参数必须在关键字参数的前面，但关键字参数之间不存在先后顺序。

5.2.4　默认参数

定义函数时，可以给函数的参数设置默认值，这个参数就被称为默认参数。调用函数时，如果没有指定某个参数将抛出异常，为了解决这个问题，可以为参数设置默认值，即在定义函数时，直接指定形式参数的默认值。这样，当没有传入参数时，则直接使用定义函数时设置的默认值。定义带有默认值参数的函数的语法格式如下：

```
def  函数名(…,[参数 n=默认值]):
    "函数注释字符串"
    函数体
```

在定义函数时，指定默认的形式参数必须在所有参数的最后，否则将产生语法错误。

当调用函数的时候，由于默认参数在定义时已经被赋值，所以可以直接忽略，而其他参数是必须要传入值的。如果默认参数没有传入值，则直接使用默认的值；如果默认参数传入了值，则使用传入的新值替代。

【例 5-10】　函数定义时设置默认参数，调用时验证其功能。

```
def printinfo( name, age =100 ):
    #打印任何传入的字符串
    print("Name:", name)
    print("Age:", age)
#调用 printinfo 函数
printinfo(name="cau" )
printinfo(name="cau",age=110 )
```

运行结果：

```
Name: cau
Age: 100
Name: cau
Age: 110
```

定义函数时，为形式参数设置默认值要牢记一点：默认参数必须指向不可变对象。若使用可变对象作为函数参数的默认值时，多次调用可能会导致意料之外的情况。若参数中有位置参数，所有位置参数必须出现在默认参数前，包括函数定义和调用。

5.2.5　不定长可变参数

在 Python 中，还可以定义可变参数。不定长参数也叫可变参数，用于不确定调用的时候会传递多少个参数(不传参也可以)的场景。可变参数即传入函数中的实际参数可以是零个、一个、两个到任意个。通常在定义一个函数时，若希望函数能够处理的参数个数

比当初定义的参数个数多，此时可以在函数中使用不定长参数。

定义可变参数时，主要有两种形式：一种是 * args(args 也可以是别的标识符)；另一种是**kwargs。

1. * args 形式

这种形式表示接收任意多个实际参数并将其放到一个元组中。

【例 5-11】 定义一个函数，让其可以接收任意多个实际参数。

```
#定义函数
def printschool(*name):
    print("\n我梦想的大学:")
    for item in name:
        print(item)
#调用函数
printschool('清华大学')
printschool('清华大学','北京大学')
printschool('清华大学','北京大学','中国农业大学')
```

运行结果：

```
我梦想的大学:
清华大学

我梦想的大学:
清华大学
北京大学

我梦想的大学:
清华大学
北京大学
中国农业大学
```

如果想要使用一个已经存在的列表作为函数的可变参数，可以在列表的名称前加“*”。例如：

```
schoolname=['清华大学','北京大学','中国农业大学']
printschool(*schoolname)
```

使用可变参数需要考虑形参位置的问题。如果在函数中既有普通参数，也有可变参数，通常可变参数会放在最后。若可变参数放在函数参数的中间或者最前面，只是在调用函数时，可变参数后面的普通参数要用关键字参数形式传递参数。如果可变参数在函数参数的中间位置，而且为可变参数后面的普通参数传值时也不想使用关键字参数，那么就必须为这些普通参数指定默认值。

2. **kwargs 形式

这种形式表示接收任意多个类似关键字参数显式赋值的实际参数，并将其放到一个字典中。