第5章 Python 函数

观看视频

在线答题

学习目标

- 了解 Python 函数的定义与返回值，能够正确书写函数
- 掌握 Python 函数的参数传递，能够编写自定义函数来满足特定需求
- 掌握作用域与全局变量、局部变量的使用，能够应用变量的作用域
- 了解 Python 的 3 种命名空间的作用，能够理解变量的生命周期
- 掌握 Python 函数的嵌套调用，能够创建函数嵌套结构
- 掌握 Python 函数的递归调用，能够灵活地应用函数递归

Python 函数是一段可重复使用的代码块，是 Python 编程中基本的构建块，可以帮助读者更好地组织和管理代码，减少重复的工作，提高编程的效率和代码的可读性、可维护性。本章从认识 Python 函数的定义与返回值、函数参数、函数调用及变量的作用域进行讲解。

5.1 认识 Python 函数

2.3 节已经介绍过函数的定义与几种常见的内置函数，本节将针对 Python 函数的定义与调用及返回值进行讲解。

5.1.1 Python 函数的定义

Python 中的函数分为内置函数和自定义函数。下面对内置函数、自定义函数和函数调用进行介绍。

1. 内置函数

Python 的内置函数可以直接在代码中使用，无须导入任何模块，如 print()函数、input()函数等。常见的内置函数如表 5-1 所示。

表 5-1 常见的内置函数

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()

续表

bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	set()
complex()	hasattr()	max()	round()	hash()
delattr()	memoryview()			

Python 的内置函数可以提供快捷的解决方案，灵活、准确地使用内置函数可以大大提高代码的复用性和可读性。

2. 自定义函数

Python 的自定义函数是将一段有规律的、可复用的代码定义成函数，从而达到一次编写、多次复用的目的。在 Python 中，函数声明以 def 关键字开头，后跟函数名称和必要的参数列表：

```
def function_name(parameters):
    #函数体
    #执行的代码块
```

在程序中，如果需要实现多次输出“拼搏到无能为力，坚持到感动自己！”文本的功能，则可以将这个功能写成一个函数，具体代码如下：

```
def output():
    print('拼搏到无能为力,坚持到感动自己!')
```

当需要使用该函数时，则可以进行多次调用，调用函数的语句具体如下：

```
output()
```

自定义函数可以减少程序中的代码冗余，提升代码的可维护性。

3. 函数调用

定义一个函数需要为函数指定名称，并设计函数内所包含的参数和代码块结构。函数的基本结构完成以后，可以通过另一个函数调用此函数，也可以借助 Python 提示符执行此函数。使用函数一般包含两个步骤，具体如下：

(1) 定义函数，用于封装独立的功能。

(2) 调用函数，用于享受封装的结果。

下面通过一个示例演示自定义函数的定义与调用。

在创建好的“Python 程序设计基础”项目中创建 Chapter05 文件夹，在 Chapter05 文件夹中创建一个名为 squareS.py 的文件，计算已知边长的长方形的面积，具体代码如文件 5-1 所示。

文件 5-1　squareS.py

```
1  def square(length,width):
2      return length * width
3  print(square(5,6))
```

在上述代码中，第 1、2 行代码为自定义函数 square()，包含面积的计算公式；第 3 行代码为函数调用，向函数 square()传递两个参数，分别为数字 5、6。square()函数的功能是计算 5 * 6 的值，并打印出函数的返回值。

文件 squareS.py 的运行结果如图 5-1 所示。

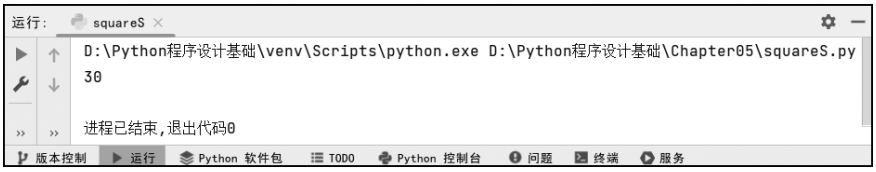

图 5-1　文件 squareS.py 的运行结果

由图 5-1 可知，函数的返回值为 30，即计算出的长方形面积为 30。

5.1.2　Python 函数的返回值

在 Python 中，使用 def 语句创建函数时，可以使用 return 语句指定函数的返回值，该返回值可以是任意数据类型。需要注意的是，return 语句在同一函数中可以出现多次，但只要满足一次执行条件，函数就会返回执行结果，并结束执行。含有返回值的函数可作为一个值赋给指定变量。return 语句的语法结构如下：

```
def function_name(parameters):
    #函数体
    #执行的代码块
    return [返回值]
```

return 语句只有一个参数，返回值是可选的，可以指定具体值，也可以省略不写，省略时将返回空值 None。

下面通过一个示例演示函数的返回值。在 Chapter05 文件夹中创建一个名为 returnS.py 的文件，计算两个数的和并打印和的值，具体代码如文件 5-2 所示。

文件 5-2　returnS.py

```
1  def return_sum(x,y):
2      c =x + y
3      return c
```

```
4   res =return_sum(4,5)
5   print(res)
```

在上述代码中，第 1～3 行代码为自定义函数 return_sum()，用于计算两个数之和，并返回结果；第 4 行代码为函数调用，使用变量 res 接收函数的返回结果，并向函数 return_sum()传递了两个参数，分别为数字 4、5；第 5 行代码用于打印函数的返回值。

文件 returnS.py 的运行结果如图 5-2 所示。

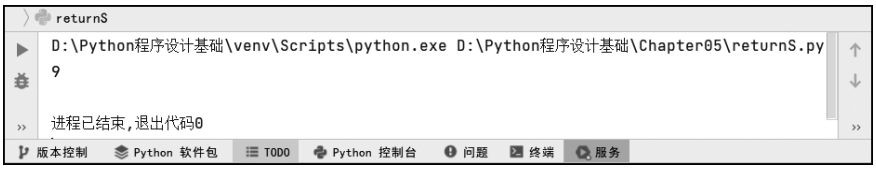

图 5-2　文件 returnS.py 的运行结果

由图 5-2 可知，函数的返回值为 9，即两个数的和为 9。

下面通过一个示例演示函数返回空值 None 时 return 语句的使用。

在 Chapter05 文件夹中创建一个名为 returnN.py 的文件，用于计算两个数的值但不返回数值，具体代码如文件 5-3 所示。

文件 5-3　returnN.py

```
1   def empty_return(x,y):
2       c =x + y
3       return
4   res =empty_return(4,5)
5   print(res)
```

文件 returnN.py 的运行结果如图 5-3 所示。

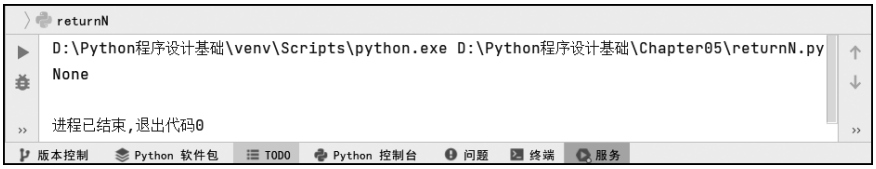

图 5-3　文件 returnN.py 的运行结果

由图 5-3 可知，函数的返回值为 None，函数计算两个数的和后，不返回数值，因此调用此函数时返回 None。

拓展阅读：Python 的常量 None

在 Python 中，有一个特殊的常量 None(N 必须大写)。None 和 False 不同，既不表示 0，也不表示空字符串，而是表示没有值，即空值。None 有自己的数据类型，属于 NoneType 类型。

需要注意的是，None 是 NoneType 数据类型的唯一值，即不能再创建其他 NoneType 类型的变量，但是可以将 None 赋值给任何变量。如果希望避免变量中存储的数据与任何其他数据混淆，就可以使用 None。

除此之外，None 常用于 assert、判断及函数无返回值的情况。

例如，使用 print()函数输出数据时，该函数的返回值就是 None，因为 print()函数的功能是在屏幕上显示文本，根本不需要返回任何值，所以 print()返回 None。

另外，对于所有没有 return 语句的函数定义，Python 都会在末尾加上 return None，使用不带值的 return 语句，即只有 return 关键字本身，那么该函数就会返回 None。

5.2 Python 函数的参数传递

在学习定义一个函数时，可能会设置多个形式参数，此时在调用函数时也会传递多个实际参数。本节将从函数参数及传递实际参数两方面进行讲解。

5.2.1 函数参数

1. 函数参数的分类

通常情况下，定义函数时都会选择有参数的函数形式，函数参数的作用是向函数传递数据，使函数对接收的数据进行具体的处理。函数参数可以是一个，也可以是多个。多个参数被称为参数列表，参数之间需要用逗号进行间隔。

Python 中的函数传递参数的形式主要有 4 种，分别为位置传递、关键字传递、默认值传递和不定参数传递。

2. 形式参数与实际参数

在定义函数时，函数名后紧跟着的括号中的参数称为形式参数；在调用函数时，函数名后紧跟着的括号中的参数称为实际参数，即函数的调用者传递给函数的参数；实际参数可以向形式参数传递数据，形式参数接收实际参数的值。下面通过一个示例演示函数的形式参数与实际参数的含义。

在 Chapter05 文件夹中创建一个名为 demo.py 的文件，具体代码如文件 5-4 所示。

文件 5-4　demo.py

```
1  def demo(obj):
2      print(obj)
3  obj ="动辄不衰，用则不退，生命在于运动。"
4  demo(obj)
```

在上述代码中，第 1 行代码中的 obj 为形式参数，而第 4 行代码中的 obj 为实际参数。文件 demo.py 的运行结果如图 5-4 所示。

图 5-4　文件 demo.py 的运行结果

由图 5-4 可知，调用函数后，执行函数 demo()的 print(obj)语句，打印的结果是传递实

际参数值后的形式参数。

3. 函数参数的传递方式

在 Python 中，根据实际参数的类型不同，函数参数的传递方式可分为两种，分别为值传递和引用（地址）传递。值传递适用于实际参数类型为不可变类型的情况，如字符串、数字、元组等；引用（地址）传递适用于实际参数类型为可变类型的情况，如列表与字典等。下面通过一个示例演示函数参数的值传递与引用传递的过程。

在 Chapter05 文件夹中创建一个名为 demo01.py 的文件，定义函数 demo()后调用函数，采用不同的参数传递方式并查看执行函数的结果，具体代码如文件 5-5 所示。

文件 5-5　demo01.py

```
1   def demo(obj):
2       obj +=obj
3       print(obj)
4   print("_____值传递_____")
5   a =" 健康的身体乃是灵魂的客厅,有病的身体则是灵魂的禁闭室。"
6   print("形式参数:",a)
7   demo(a)
8   print("实际参数:",a)
9   print("_____引用传递_____")
10  a=[1,"跑步",3,"游泳"]
11  print("形式参数:",a)
12  demo(a)
13  print("实际参数:",a)
```

在上述代码中，第 5 行代码中的变量 a 传递的数据类型是字符串，属于不可变类型，因此第 7 行代码调用函数时为值传递，第 10 行代码中的变量存储的数据类型是列表，属于可变类型，因此第 12 行代码调用函数时为引用传递。

文件 demo01.py 的运行结果如图 5-5 所示。

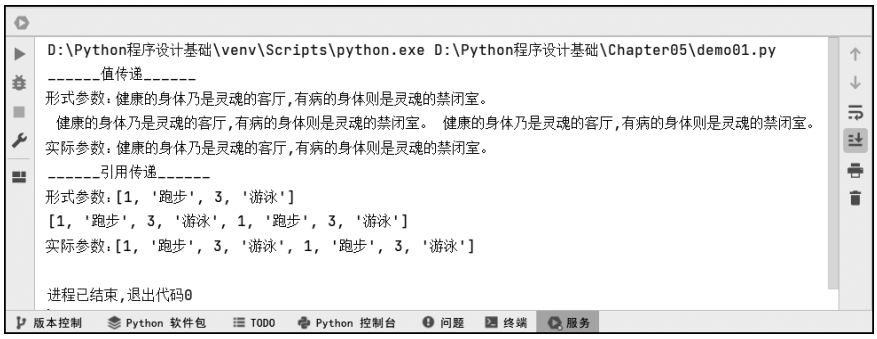

图 5-5　文件 demo01.py 的运行结果

由图 5-5 可知，在执行值传递时，读者将形式参数的值由字符串“健康的身体乃是灵魂的客厅，有病的身体则是灵魂的禁闭室。”修改为字符串“健康的身体乃是灵魂的客厅，有病的身体则是灵魂的禁闭室。健康的身体乃是灵魂的客厅，有病的身体则是灵魂的禁闭室。”后，实际参数并不会发生改变，依然是“健康的身体乃是灵魂的客厅，有病的身体则是灵魂的

禁闭室。”;而在进行引用传递时,读者将形式参数的值由列表“[1,"跑步",3,"游泳"]”修改为列表“[1,"跑步",3,"游泳",1,"跑步",3,"游泳"]”后,实际参数也会发生同样的改变。

5.2.2 传递实际参数

实际参数的传递方式包括位置参数传递,要求实际参数的顺序与形式参数的顺序相同;关键字参数传递,实际参数由变量名和值组成;除此之外,还可以通过参数的包裹传递来传递任意个数的参数等。下面对位置参数、关键字参数、默认值参数以及包裹参数 4 种参数传递方式进行讲解。

1. 位置参数

位置参数也称必备参数,是函数参数中最常用的形式。位置参数要求,调用函数时,Python 必须将函数调用中的每个实际参数都关联到函数定义中的一个形式参数,即调用函数时传入实际参数的数量和位置都必须和定义函数时的形式参数保持一致,否则 Python 解释器会抛出 TypeError 异常,并提示缺少必要的位置参数。下面通过一个示例演示函数的位置参数的使用。

在 Chapter05 文件夹中创建一个名为 locationP.py 的文件,具体代码如文件 5-6 所示。

文件 5-6 locationP.py

```
1  def print_Course(a, b, c, d):
2      print("Course: {} {} {} {}".format(a, b, c, d))
3  print_Course("语文", "高等数学", "计算机基础", "物理")
```

在上述代码中,a、b、c、d 是位置参数。

文件 locationP.py 的运行结果如图 5-6 所示。

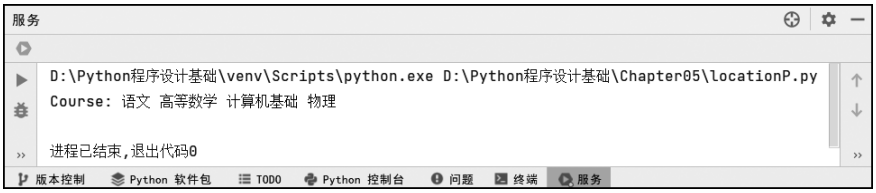

图 5-6 文件 locationP.py 的运行结果

由图 5-6 可知,将实际参数的字符串数据“语文、高等数学、计算机基础、物理”依次传递给了形式参数 a、b、c、d。

2. 关键字参数

关键字参数指的是使用形式参数的名字来确定输入的参数值。此方式不再要求实际参数与形式参数在位置上完全一致,读者只要将参数名写正确即可。关键字参数和函数调用的关系十分紧密,函数调用需要使用关键字参数来确定传入的参数值。下面通过一个示例演示函数的关键字参数的使用。

在 Chapter05 文件夹中创建一个名为 keyP.py 的文件,具体代码如文件 5-7 所示。

文件 5-7 keyP.py

```
1  vdef printinfo(name, age):
```

```
    print("名字: ", name)
    print("年龄: ", age)
    return
printinfo(age=50, name="小千")
```

在上述代码中，第 5 行代码在传递实际参数时，使用关键字参数调换了参数的顺序，导致实际参数与形式参数的顺序不同。

文件 keyP.py 的运行结果如图 5-7 所示。

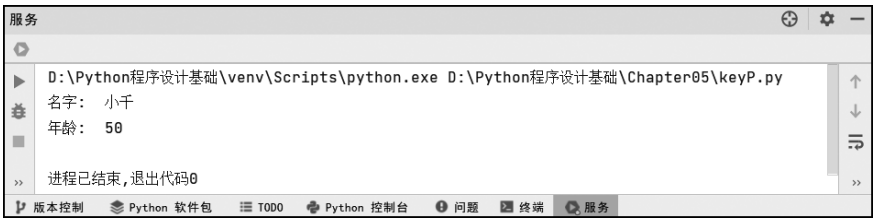

图 5-7　文件 keyP.py 的运行结果

由图 5-7 可知，打印的参数与形式参数的顺序一致。

使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。混合传参时，关键字参数必须位于所有的位置参数之后。

3. 默认值参数

定义 Python 函数时，可给每个形式参数指定默认值。在调用函数时，如果读者为形式参数提供实际参数，Python 将优先使用指定的实际参数值；否则，将使用形式参数的默认值。给形式参数指定默认值后，可省略函数调用中相应的实际参数传递。使用默认值不仅可以简化函数调用，还可以清楚地指出函数的典型用法。下面通过一个示例演示函数的默认值参数的使用。

在 Chapter05 文件夹中创建一个名为 defaultP.py 的文件，比较使用默认值参数时形式参数的值，具体代码如文件 5-8 所示。

文件 5-8　defaultP.py

```
def printinfo(name, age=35):
    print("名字: ", name)
    print("年龄: ", age)
    return
printinfo(age=15, name="小千")
print("------------------------")
printinfo(name="小千")
```

由文件 defaultP.py 可知，定义函数时，形式参数 age 设置了默认值；第 5 行代码调用函数时指定了实际参数的值，当定义一个有默认值参数的函数时，有默认值的参数必须位于所有无默认值参数的后面；第 7 行代码调用函数时仅指定了实际参数 name，而没有指定 age。

文件 defaultP.py 的运行结果如图 5-8 所示。

由图 5-8 可知，调用函数时指定了实际参数值，因此使用指定的值，如果没有传递实际参数，则会使用参数默认值。

图 5-8　文件 defaultP.py 的运行结果

4. 包裹参数

当一个函数接收的参数数量超出已声明的参数数量时，Python 提供了能够接收任意数量实际参数的传参方式，这些参数叫作包裹参数。在包裹参数中，需要使用 * args 接收实际参数传递参数时转换为元组的形式，**kwargs 接收实际参数传递参数时转换为字典的形式。下面通过一个示例演示函数的包裹参数的使用。

在 Chapter05 文件夹中创建一个名为 Variable-lengthP.py 的文件，展示使用包裹参数时参数的传递情况，具体代码如文件 5-9 所示。

文件 5-9　Variable-lengthP.py

```
print("-----元组-----")
def info(arg1, *vartuple):
    print(arg1)
    print(vartuple)
info(70, 80, 75)
print("-----空元组-----")
def print_info(arg1, *vartuple):
    print(arg1)
    for var in vartuple:
        print(var)
    return
print_info(60)
print_info(98, 89, 78)
print("-----字典-----")
def print_dict(arg1, **vardict):
    "打印任何传入的参数"
    print(arg1)
    print(vardict)
print_dict(1, a=2, b=3)
```

在上述代码中，第 2、7 行代码加了星号 * 的参数会以元组的形式导入；第 12～13 行代码没有指定参数，此时未向函数传递未命名的变量；第 15 行代码加了双星号** 的参数会以字典的形式导入，存放所有未命名的变量参数。

文件 Variable-lengthP.py 的运行结果如图 5-9 所示。

由图 5-9 可知，加星号 * 的参数会以元组的形式导入，如果在函数调用时没有指定参数，它就是一个空元组；加双星号** 的参数会以字典的形式导入。

声明函数时，参数中的星号 * 可以单独出现。此时，其后的参数必须使用关键字传入。

图 5-9　文件 Variable-lengthP.py 的运行结果

5. 匿名函数

在 Python 中,需要使用 lambda 函数来创建匿名函数,lambda 函数的特点如下:

(1) lambda 只是一个表达式,函数体比 def 简单很多。

(2) lambda 的主体是一个表达式,而非一个代码块。读者仅能在 lambda 表达式中封装有限的逻辑。

(3) lambda 函数拥有自己的命名空间,并且不能访问自有参数列表之外或全局命名空间中的参数。

匿名函数的语法结构如下:

```
lambda [arg1[,arg2,...argn]]:expression
```

lambda 函数设计用于编写简洁的单行语句。下面通过一个示例演示匿名函数的使用。

在 Chapter05 文件夹中创建一个名为 lambdafunc.py 的文件,具体代码如文件 5-10 所示。

文件 5-10　lambdafunc.py

```
1  def func(n):
2      return lambda a: a * n
3  a =func(2)
4  b =func(5)
5  print(a(11))
6  print(b(12))
```

在上述代码中,将匿名函数封装在 func()函数中,通过传入不同的参数来创建不同的匿名函数。

文件 lambdafunc.py 的运行结果如图 5-10 所示。

由图 5-10 可知,使用匿名函数计算两个数的乘法,计算 a * n 的值并将结果赋值给变量 a,作为函数的返回值。