第5章　函　　数

本章学习目标

- 理解函数的概念及分类。
- 掌握函数的定义和调用方法。
- 掌握函数传参的两种方式和三类形参的用法。
- 理解变量的4级作用域，掌握作用域规则。
- 理解递归函数的概念及用法。
- 掌握常用的内置函数的调用方法。
- 掌握 datetime 库的常用类 datetime 的用法。

本章主要介绍 Python 函数概述、函数的基本操作、函数的参数和变量的作用域4个内容，同时介绍递归函数和常用的 Python 内置函数的用法，穿插讲解一个标准库——datetime 库的知识。

5.1　函 数 概 述

本节包括以下两方面内容。

① 函数的基本概念。

② 使用函数编程的目的。

5.1.1　函数的基本概念

函数是具有特定书写格式、可重复使用的，用来实现单一，或相关联功能的代码段，是可以用来构建更大程序的一小部分。该代码段用函数名表示并通过函数名进行功能调用。可以在需要的地方调用执行，不需要在每个执行的地方重复编写这些语句。每次使用函数可以提供不同的参数作为输入，以实现对不同数据的处理；函数执行后，可以反馈相应的处理结果。

Python 中的函数，通常分为以下四大类。

1. 内置函数

Python 提供了许多内置函数，比如前面章节使用的 str()、list()、print()、range() 函数等，它们是 Python 官方已经为程序开发人员设计好的模块化的代码段，可以被反复使用(调用)，程序开发人员不需要了解函数的实现代码，只要了解函数的功能、参数和调用语法即可直接使用。

2. 标准库函数

Python 标准库是用 Python 和 C 语言预先编写的模块，提供了系统管理、网络通信、文本处理、数据库接口、图形系统、XML 处理等额外的功能，如 random(随机数)、math(数学

运算)、datetime(时间处理)、file(文件处理)、os(和操作系统交互)、sys(和解释器交互)等。这些模块随着 Python 解释器一起自动安装,程序员可以通过 import 语句导入对应功能的库,然后使用其中定义的函数。

3. 第三方库函数

除了内置函数和标准库函数,还可以获取和安装一些第三方库函数,第三方库函数很多是对标准库函数的优化和再封装,如 NumPy、Django 等。PyPI(Python Package Index)是 Python 官方第三方库的仓库,所有人都可以下载第三方库或上传自己开发的库到 PyPI。世界各地的程序员通过该开源社区贡献了十几万个第三方函数库,功能覆盖了我们能想象到的所有领域,如科学计算、Web 开发、大数据、人工智能、图形系统等。与标准库不同,Python 的第三方库需要下载后安装到 Python 的安装目录下,第三方库的获取和安装将在后续章节介绍。安装完成后,再通过 import 语句导入,然后才可以使用这些第三方库的函数。

4. 用户自定义函数

用户自定义函数,是开发中为适应用户自身需求定义的函数,也是本章介绍的重点内容。

5.1.2　使用函数编程的目的

使用函数主要有以下 4 个目的。

1. 降低代码的复杂性

程序员的目标之一是,编写简单的代码来完成任务,而函数有助于实现这样的目标。函数是一种功能抽象,利用它可以将一个复杂的大问题分解成一系列简单的小问题,然后将小问题继续划分成更小的问题,当问题细化到足够简单时,就可以分而治之。为了实现这种分而治之的设想,就要通过编写函数,将各个小问题逐个击破,为每个小问题编写程序,再集合起来,解决大的问题。

2. 实现代码复用

在编程的过程中,比较忌讳同样一段代码不断重复。因此,可以定义一个函数,在程序的多个位置使用,也可以将其用于多个程序。需要运行函数中的代码时,只需编写一行函数调用代码,就可让函数完成其工作。当然,还可以把功能相近的函数放到一个模块中供其他程序员使用。也可以使用其他程序员定义的函数,这就避免了重复劳动,提高了工作效率。

3. 降低代码维护的工作量

修改函数的功能时,只需在函数中修改一次,所有调用位置的功能都将得到更新。函数式程序的测试和调试相对来说更容易。调试很简单是因为函数通常都很小,而且清晰、明确。当程序无法工作的时候,每个函数都是一个可以检查数据是否正确的接入点。程序员可以通过查看中间输入和输出迅速找到出错的函数。测试更容易是因为每个函数都是单元测试的潜在目标。在执行测试前,函数并不依赖于需要重现的系统状态;相反,程序员只需要给出正确的输入,然后检查输出是否和期望的结果一致。

4. 增加程序的易读性

使用函数让程序更容易阅读,而良好的函数名概述了程序各个部分的作用。相对于阅读一系列的代码块,阅读一系列函数调用能够让使用者更快地明白程序的作用。

5.2 函数的基本操作

本节包括以下 4 个内容。

① 函数的定义。

② 函数的返回值。

③ 函数的调用。

④ lambda 表达式和匿名函数。

5.2.1 函数的定义

Python 定义函数的语法格式如下。

```
def <函数名>([形式参数列表]) :
    '''文档字符串'''
    <函数体>
    [return [返回值列表]]
```

说明:

① def 是定义函数的关键词,这个简写来自英文单词 define。Python 执行 def 时,会创建一个函数对象,并绑定到函数名变量上。函数名可以是任何有效的 Python 标识符。函数名通常使用小写字母,如果想提高可读性,可以用下画线分隔。大小写混合仅在兼容原来主要以大小写混合风格的情况下使用,为了保持向后兼容性。

② 函数的形式参数列表用于接收调用该函数时传递给它的值,放在一对圆括号中,参数的个数可以有零个、一个或多个,多个参数之间用逗号间隔,这种参数称为形式参数,简称为“形参”。函数形参无须声明类型,完全由调用者传递的实参类型以及 Python 解释器的理解和推断决定。即使该函数无须接收任何参数,也必须保留一对空的圆括号。

③ 括号后面以冒号结束,冒号必须使用英文输入法输入,否则会提示 SyntaxError 异常。

④ 冒号后面所有的缩进行构成了函数体。函数体是函数每次被调用时执行的代码,由一行或多行语句组成。函数体相对于 def 关键字必须保持一定的缩进。

⑤ 程序的可读性非常重要,一般建议在函数体开始的部分附上函数定义说明,即文档字符串(docstring),又称为“函数注释”。通过三个单引号或者三个双引号界定,中间可以加入多行文字进行说明,描述函数的功能或用途。Python 使用它们生成有关程序中函数的文档。需要注意的是,Python PEP8 编码规范中约定,对于单行的文档说明,尾部的三引号应该和文档在同一行。多行文档说明使用的结尾三引号应该自成一行。

⑥ 当需要返回值时,使用关键字 return 和返回值列表,执行 return 语句会结束对函数的调用,并带回返回值。否则,函数可以没有 return 语句,在函数体结束位置将控制权返回给调用者。例如,第 4 章**微实例 4.6** 讲述了如何用辗转相除法求任意两个整数的最大公约数。为了实现代码复用,可以将该功能定义成 gcd() 函数,需要时直接调用该函数,即可实现求任意两个正整数的最大公约数,程序代码如下。

```
>>>def gcd(m, n):
```

```
    '''求任意两个正整数的最大公约数'''
    while m %n:
        m, n =n, m %n
    return n
>>>print(gcd.__doc__)
求任意两个正整数的最大公约数
>>>x, y =12, 16
>>>print("两个数的最大公约数是: {}".format(gcd(x, y)))
两个数的最大公约数是: 4
>>>print(type(gcd), type(gcd(x, y)))
<class 'function'><class 'int'>
```

5.2.2 节将会详细讲述 return 语句的相关语法。

⑦ 定义函数时只检测语法，不执行函数体代码。定义好的函数只有被调用的时候才会执行函数体代码。

⑧ Python 允许嵌套定义函数。即在一个函数的定义中再定义另一个函数，该操作称为“闭包”。除非特别必要，一般不建议过多使用嵌套定义函数，因为每次调用外部函数时，都会重新定义内层函数，程序运行效率较低，如下述代码所示。

```
def f1():
    print('f1 running...')

    def f2():
        print('f2 running...')

    f2()

f1()
```

运行该程序，结果如下。

```
f1 running...
f2 running...
```

在函数的内部嵌套定义函数时，内部函数的定义和调用都要在该函数的内部完成。上例中，函数 f2() 定义在函数 f1() 内部，函数 f2() 的定义和调用都必须在函数 f1() 内部。如果在函数 f1() 外部调用函数 f2()，则会提示 NameError 异常，如下述代码所示。

```
def f1():
    print('f1 running...')

    def f2():
        print('f2 running...')

f2()
```

运行该程序，结果如下。

```
NameError: name 'f2' is not defined
```

5.2.2　函数的返回值

Python 中，用 def 关键字定义函数时，函数并非总是直接显示输出，相反，它可以处理

一些数据,并返回一个或一组值。函数返回的值被称为返回值。在函数中,可使用 return 语句将值返回到函数调用处。通过返回值,能够实现将程序的大部分繁重工作转移至函数中完成,从而简化主程序。return 语句的语法格式如下。

```
return [返回值列表]
```

说明:

① 返回值可以是任意类型,return 语句在同一函数中可以出现多次,但只要有一条 return 语句执行,就会直接结束当前函数的执行。

② 一条 return 语句也可以同时带回一个或多个返回值,多个值以元组方式返回,如下述代码所示。

```
>>>def median (*data):
        data =sorted(data)
        number =len(data)
        if number %2 ==0:
            return data[number // 2 -1], data[number // 2]          #带回多个返回值
        else:
            return data[number // 2]                                #带回一个返回值

>>>median(1, 8, 5, 4, 9)
5
>>>median(1000, 100, 10, 1)
(10, 100)
```

③ 对于以下三种情况,Python 将认为该函数以 return None 结束,即返回空值。

- 若函数内没有 return 语句,Python 解释器将默认为函数体最后添加了一条 return None 语句。

本书实例经常使用 print()函数输出数据,其实该函数的返回值就是 None。因为它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()函数就返回 None,如下述代码所示。

```
>>>demon =print("China Pharmaceutical University")
China Pharmaceutical University
>>>demon ==None
True
```

- 函数内有 return 语句,但是没有执行到。
- 函数内有 return 语句,也执行到了,但 return 语句后面的表达式列表省略,即当前代码行只有 return 关键字本身。

需要特别说明的是,None(N 必须大写)是 Python 中一个特殊的常量,和 False 不同,它既不表示 0,也不表示空字符串,而表示没有值,也就是空值。None 有自己的数据类型,读者可以在 IDLE 中使用 type() 函数查看它的类型,执行代码如下。

```
>>>type(None)
<class 'NoneType'>
```

可以看到,它属于 NoneType 类型。NoneType 是 Python 的特殊类型,它只有一个取值 None。可以将 None 赋值给任何变量,但它不支持任何运算,也没有任何内建方法,和其

他的数据类型比较是否相等时永远返回 False。通常，如果希望变量中存储的内容不与其他值混淆，就可以使用 None。

5.2.3　函数的调用

函数创建成功后，可以在当前文件中调用，也可以在其他模块中调用。Python 在实际调用函数的过程中非常灵活，不必为不同类型的参数定义多个函数，在处理不同类型的数据时可以调用相同的函数。函数调用或执行的一般格式如下。

<函数名>(<实际参数列表>)

说明：

① 函数的调用采用函数名加一对圆括号的方式，圆括号内的参数列表是调用函数时，用来将列表中的参数分别赋值给函数中对应的形参。这类参数称为实际参数，简称为“实参”。实参和形参的绑定关系在函数调用时生效，函数调用结束后解除绑定关系（释放内存空间）。

② 函数应该先定义、后调用。如果函数定义在调用该函数的语句之后，执行时将提示异常。

微实例 5.1：计算血药的浓度。

某药品口服后，1 小时测定，口服剂量与血药浓度的关系是线性关系，其计算方程为 $y=0.8x+3.25$，其中，x 表示口服剂量（单位：片），y 表示血药浓度（单位：mg/L）。

要求：两个患者分别摄入该药品 2 片和 3 片，请分别计算血药的浓度。

程序代码如下。

```
#定义函数
def linearregression(x):
    '''计算一小时后的血药浓度'''
    return 0.8 * x +3.25

dosage1 =2
dosage2 =3
#调用函数计算 1 号患者的血药浓度
concentration1 =linearregression(dosage1)
#再次调用函数计算 2 号患者的血药浓度
concentration2 =linearregression(dosage2)
print("1 号患者口服剂量为: ", dosage1, ",一小时后的血药浓度为: ", concentration1)
print("2 号患者口服剂量为: ", dosage2, ",一小时后的血药浓度为: ", concentration2)
```

运行该程序，结果如下。

```
1 号患者口服剂量为：2，一小时后的血药浓度为：4.85
2 号患者口服剂量为：3，一小时后的血药浓度为：5.65
```

本例在函数 linearregression()的定义中，圆括号中的变量 x 是一个形参，形参是在定义函数时使用的，用来获得函数完成其工作所需的数据。主程序中两次调用函数 linearregression()计算患者的血药浓度，在函数调用语句中，dosage1 和 dosage2 为实参。即分别将实参 dosage1 和 dosage2 中的数据传递给形参 x，函数返回值被分别存储在变量 concentration1 和 concentration2 中。

微实例 5.2：输出某区间内的所有素数。

第 4 章**微实例 4.7** 讲述了如何判断一个数是否为素数。本例将定义一个 isPrime()函数，用来判断一个数是否为素数。

要求：输出 101～199 的所有素数，每一行最多输出 5 个数字。

程序代码如下。

```
def isprime(x):
    for i in range(2, x):
        if x %i ==0:
            return False
    return True

k =0
for j in range(101, 200):
    if isprime(j):
        print(j, end="")
        k +=1
        if k %5 ==0:
            print()
```

运行该程序，结果如下。

```
101 103 107 109 113
127 131 137 139 149
151 157 163 167 173
179 181 191 193 197
199
```

5.2.4 lambda 表达式和匿名函数

Python 还提供了一个关键字 lambda，用于定义一种特殊的函数——匿名函数。匿名函数并非没有名字，而是将函数名作为函数结果返回，其语法格式如下。

```
<函数名>=lambda <形参列表>: <表达式>
```

匿名函数等价于下面的函数定义。

```
def <函数名>(<形参列表>) :
    return <表达式>
```

匿名函数是一种简单的、在同一行中定义函数的方法。lambda 表达式实际生成了一个函数对象。lambda 表达式只允许包含一个表达式，不能包含复杂语句，该表达式的计算结果就是函数的返回值。例如：

```
>>>f =lambda x, y : x +y
>>>type(f)
<class 'function'>
>>>f(20, 16)
36
```

本例使用语句 f = lambda x, y : x + y 定义了一个函数，赋值给变量 f，因此变量 f 成为一个函数的函数名，可以在需要调用该函数对象的场景使用 f(实参列表)的形式。该匿

名函数等价于下面的函数定义。

```
>>>def f(x, y):
        return x +y
```

本章**微实例 5.1** 中实现计算血药浓度的函数也可以使用匿名函数来定义，程序代码如下。

```
linearregression =lambda x : 0.8 * x +3.25
```

5.3　函数的参数

本节包括以下两方面内容。

① 参数传递的方式。

② 函数形参的分类。

5.3.1　参数传递的方式

前面讲述了函数的定义和调用，顺便简要介绍了函数定义时的“形参”和调用函数时的“实参”，本节将介绍与参数相关的细节——参数传递的方式。

1. 位置传递

调用函数时需要将实参传递给被调用函数的形参，默认的传递形式就是位置传递，实参默认按位置顺序传递给形参，因此，采用位置传递时实参和形参的顺序必须严格一致，如下述代码所示。

```
>>>def demo(a, b, c):
        print(a, b, c)

>>>#按位置传递参数
>>>demo(3, 4, 5)
3 4 5
>>>#实参与形参数量必须相同
>>>demo(1, 2, 3, 4)
TypeError: demo() takes 3 positional arguments but 4 were given
```

上例中，第一次调用函数时，实参和形参数量相同，第一个实参“3”传递给第一个形参“a”，第二个实参“4”传递给第二个形参“b”，第三个实参“5”传递给第三个形参“c”。第二次调用函数，因为实参和形参数量不同，Python 解释器将提示 TypeError 异常。

2. 关键字传递

在规模稍大的程序中，函数定义可能在函数库中，也可能与函数调用处相距较远。尤其当参数很多时，如果仅看实际调用而不看函数定义，很难理解这些输入参数的含义，从而导致程序的可读性较差。为了解决上述问题，Python 提供了关键字传递的方式。

在调用函数时，实参可以是“形参名 = value”的形式，这种以形参名称作为一一对应的参数传入方式被称作关键字传递。采用关键字传递的方式传递参数时，实参顺序可以和形参顺序不一致，但不影响传递结果，避免了用户需要牢记形参列表顺序的麻烦，如下述代码所示。

```
>>>def demo(a, b, c):
        print(a, b, c)

>>>#按位置传递参数
>>>demo(3, 4, 5)
3 4 5
>>>#后两个参数指定形参名称传递参数
>>>demo(7, c =3, b =6)
7 6 3
>>>#所有参数均按关键字传递方式传递
>>>demo(c =7, a =9, b =8)
9 8 7
```

传递参数时,实参中一旦启用关键字传递,后续参数必须都按关键字传递方式传递参数,否则 Python 解释器将提示 SyntaxError 异常,如下述代码所示。

```
>>>def demo(a, b, c):
        print(a, b, c)

>>>demo(c =3, 4, a =6)
SyntaxError: positional argument follows keyword argument
```

无论采用何种方式传递参数,都不能给某个形参重复传递参数,即同一形参不能被匹配两次,否则 Python 解释器将提示 TypeError 异常,如下述代码所示。

```
>>>def demo(a, b, c):
        print(a, b, c)

>>>demo(3, 4, a =5)
TypeError: demo() got multiple values for argument 'a'
```

5.3.2 函数形参的分类

5.2.3 节讲述到 Python 在实际调用函数的过程中非常灵活。例如,在调用某些函数时,可以向其传递实参,也可以不传递;传递给函数的实参的数量不确定,可能是一个,也可能是几个,甚至几十个;传递给函数的实参数据类型各不相同,等等。但函数依然可以被正确调用。

上述功能的实现,主要依靠程序员定义函数时,对不同类型的形参及其组合的选择和使用,本节将介绍以下三种类型的形参。

1. 默认值参数

对于一些函数,程序员可能希望它的一些参数是可选的,如果用户不想为这些参数提供值,这些参数就使用默认值。这个功能借助于函数的默认值参数完成。默认值参数又称可选参数或默认参数,可以在函数定义的形参名后加上赋值运算符(=)和默认值,从而给形参指定默认参数值。这些设置过默认值的参数在传递时是可选的,具有很大的灵活性。调用带有默认值参数的函数时,如果未传递对应的实参值,则用默认值,如实参传递了新的值,则用新的值覆盖默认值,如下述代码所示。

```
>>>def demo(a, b, c =10 , d =20):
        print(a, b, c, d)
```

```
>>>demo(8, 9)                    #第 3 个和第 4 个形参采用默认值
8 9 10 20
>>>demo(8, 9, 19)                #第 3 个形参用新的值覆盖默认值,第 4 个形参采用默认值
8 9 19 20
>>>demo(8, 9, 19, 29)            #第 3 个和第 4 个形参均用新的值覆盖默认值
8 9 19 29
>>>demo(8, 9, d =29)             #第 3 个形参采用默认值,第 4 个形参用新的值覆盖默认值
8 9 10 29
>>>#使用"函数名.__defaults__"查看函数默认值参数的当前值
>>>demo.__defaults__             #返回值是一个元组
(10, 20)
```

如果定义某一函数时,其形参列表中既包含非默认值参数,又包含默认值参数,那么,在声明函数的形参时,必须先声明非默认值参数,后声明默认值参数,如下述代码所示。

```
>>>def demo2(a =5, b, c =20):
       print(a, b, c )

SyntaxError: non-default argument follows default argument
```

以上错误提示说明:在定义函数 demo2()时,形参列表中,默认值参数后面有非默认值参数。

为参数设置默认值时,建议使用不可变对象,如整数、浮点数、字符串、True、False、None 或以上类型组成的元组等,因为默认值只会在函数定义时被设定一次,如果是可变对象,一旦在函数内部被原地修改,效果会保留至以后每次的函数调用,不会被重新初始化。如果非要使用某个可变对象作为默认值,比如列表,或者要设定依赖于其他参数的默认值,建议设为 None,如下述代码所示。

```
>>>def join(lst, sep =None):
       return (sep or ' ').join(lst)

>>>aList =['a', 'b', 'c']
>>>join(aList)
'a b c'
>>>join(aList, ', ')
'a,b,c'
```

上例定义的函数 join()的功能是使用指定分隔符将列表中的所有字符串元素连接成一个新的字符串。等第 6 章学习完毕,读者会对这个知识点会有更高层次的认识。

2. 可变数量参数

在使用函数的过程中,时常存在传递给函数的实参数量不确定的情况。这种情况下,应该如何定义形参呢? Python 利用可变数量参数解决实参个数不确定的问题。此处内容涉及元组和字典相关概念,建议学完第 6 章,再阅读此处内容。

可变数量参数又称可变参数,主要有两种形式。

① *args(一个星号),将多个参数收集到一个"元组"对象中,如下述代码所示。

```
>>>def f1(*a):
       print(a)

>>>f1(1 , 2 , 3)
(1, 2, 3)
>>>def f2(a , b , *c):
```