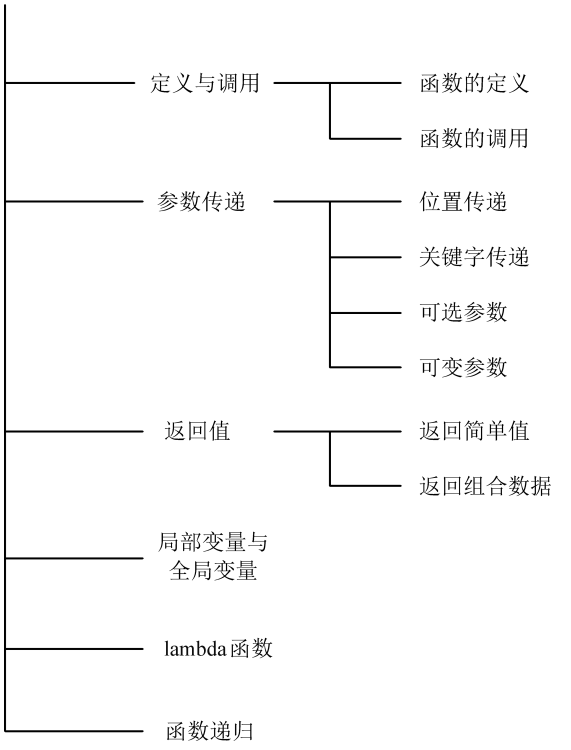
第5章 函数

学习目标

- 能根据功能定义函数并调用。
- 能理解匿名函数 lambda 的使用方法。
- 会根据需求使用全局变量和局部变量。
- 能理解递归的含义。

本章主要内容



各例题知识要点

例 5.1　定义求阶乘的函数(函数定义)

例 5.2　定义求阶乘的函数,并调用该函数计算 10 的阶乘(函数定义与调用)

例 5.3　定义一个无参数的问候函数(无参函数)
例 5.4　定义包含参数的问候函数,调用该函数输出问候语(位置传递)
例 5.5　定义函数问候列表中每一位用户(传递列表参数)
例 5.6　定义函数为列表中每项添加统一前缀(传递列表参数)
例 5.7　定义描述宠物的函数(关键字传递)
例 5.8　定义参数包含默认值的求和函数(可选参数)
例 5.9　定义参数不定长的函数(可变参数)
例 5.10　用 return 返回多个值(函数返回值)
例 5.11　用 return 返回一个列表(函数返回列表)
例 5.12　用 return 返回一个字典(函数返回字典)
例 5.13　函数内外包含同名变量(局部变量与全局变量)
例 5.14　保留字 global 修饰变量(函数内使用全局变量)
例 5.15　局部变量为组合类型(局部变量为组合类型)
例 5.16　函数内部创建局部列表变量(局部列表重新创建)
例 5.17　lambda 函数(匿名函数)
例 5.18　用递归方法定义求阶乘函数(递归)
例 5.19　用递归方法实现求斐波那契数列第 10 个数(递归)
例 5.20　用递归方法实现二分查找函数(递归)

函数是一段具有特定功能的、可重用的代码。函数是功能的抽象,一般来说,每个函数表达特定的功能。函数是带名字的代码块,用于完成具体的工作。需要使用函数的功能时,可调用该函数。在程序中多次执行同一项任务时,不需要反复编写完成该任务的代码,而只需调用执行该任务的函数即可。

使用函数的主要目的有两个:①降低编程难度;②代码复用。通过使用函数,程序的编写、阅读、测试和修复将变得更加容易。

例如,数学中的组合是从 n 个元素中不重复地选取 m 个元素的一个组合,组合数的计算如式(5-1)所示。

$$C_n^m = \frac{A_n^m}{m!} = \frac{n!}{m!(n-m)!} \tag{5-1}$$

从公式可以看出,计算组合数,需要多次计算阶乘。如果要变成实现计算组合数的功能,可以把求阶乘的代码抽象出来,即定义为一个函数实现求阶乘的功能,然后在计算组合数的过程中多次调用求阶乘的函数。使用函数既降低了编程难度,同时可以多次复用相似代码,提高了代码效率。

5.1　函数的定义与调用

5.1.1　函数的定义

函数需要先定义后调用,函数的定义形式如下。

```
def  <函数名>(<0个或多个参数>):
        <函数体>
        return  <返回值>
```

在函数的定义中，函数名是函数的标识。函数名后面是用圆括号“()”括起来的形式参数列表，参数列表可以包含0个或多个形式参数。形式参数用来接收外部传递给函数的数据，如果函数的运行不需要提供参数，则可以不要形式参数，此时圆括号中是空的。但是，不管圆括号内的参数是否为空，圆括号本身都不能省略。函数体是实现函数功能所需的若干条语句。

如果函数需要向外界返回结果，可以使用return语句。return后面可以列出0个或多个数据。return后面没有数据时，return的作用是返回到主调语句，且不带回任何数据。当return后面有一个数据时，将该数据作为结果带回。当return后面有多个数据时，将把多个数据作为一个元组带回。

【例5.1】 定义求阶乘的函数。

【分析】 求阶乘函数要实现求某个数的阶乘，需要一个参数，求得的阶乘值，需要通过return返回。

程序代码：

```
def fact(n):
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
```

程序说明：变量 n 是形式参数，计算得到的 n 的阶乘，存放在变量 s 中，通过return返回 s 的值。

函数定义时，参数是输入、函数体是处理、return返回的结果是输出，即遵循IPO流程。定义函数时，圆括号中的参数是形式参数，没有具体的值。当函数被调用时，才会把实际参数值传递给形式参数。

需要注意，函数定义后，如果不经过调用，不会被执行。

5.1.2 函数的调用

【例5.2】 定义求阶乘的函数fact，调用函数fact，求10的阶乘并输出。

【分析】 先定义一个求阶乘函数fact(n)，然后将10作为实际参数，调用fact函数，输出函数返回值。

程序代码：

```
#定义函数 fact
def fact(n):
    s = 1
```

```
    for i in range(1, n+1):
        s *= i
    return s
#调用函数 fact,求 10!
m=fact(10)
print("10!={}".format(m))
```

程序运行结果：

```
10!=3628800
```

程序说明：

(1) 该程序运行从 $m=$fact(10)开始，由于运行时调用了 fact 函数，则转向定义函数的部分，将 10 传递给 n。

(2) 函数定义中的形参与函数调用中的实参是两个重要概念。函数定义里的变量 n 是形式参数。通过语句 $m=$fact(10)调用函数，这里的 10 称为实际参数。

整个调用过程中数据的传递情况如图 5-1 所示。在函数调用过程中，实际参数 10 传递给形式参数 n，经过计算得到结果 $s=36288000$，通过 return 将 s 的值 36288000 返回，赋值给 m 并输出。

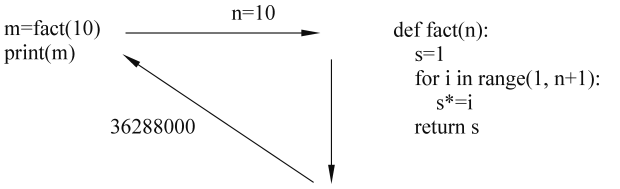


图 5-1　函数的调用过程

为了调用该函数求其他数的阶乘，可以先输入一个数，然后调用 fact()函数求该数的阶乘。即函数定义部分不变，调用的代码修改为：

```
#调用函数 fact,求 n!
n=eval(input())
m=fact(n)
print("{}!={}".format(n,m))
```

如果输入 5，则求得 5 的阶乘并输出。此时要注意，虽然这里使用了 n 作为输入的变量，与函数定义中的形式参数 n 名称相同，但是含义不同，调用时输入的 n 有实际的值，是实际参数。函数内外同名变量的区别会在 5.4 节中介绍。

5.2　函数的参数传递

函数可以有参数，也可以无参数。无参数传递的函数定义形式如下。

```
def <函数名>() :
    <函数体>
    return <返回值>
```

注意：即使没有参数，也必须保留圆括号。

【例 5.3】 定义无参数的问候函数，调用该函数，输出问候语“Hello!”。

```
deffunc():
    print("Hello!")
func()          #调用 func()函数
```

程序运行结果：

```
Hello!
```

这是一个无参数的函数，调用时不需要传入参数值。

5.2.1 位置传递

将例 5.3 稍做修改，就可以让函数 func()不仅显示“Hello!”，还将打招呼的对象名字显示出来。为此，可在函数定义 func()的括号内添加一个参数 username。通过在这里添加 username，就可以让函数接收给 username 指定的值。调用 func()时，可提供名字作为实际参数。

【例 5.4】 定义问候函数，输入姓名和城市名，输出形如“你好，来自 * 的 # 同学!”的问候语。其中，* 表示城市名，# 表示姓名。

程序代码：

```
#定义函数
def func(name,city):
     return "你好,来自"+city+'的'+ name+"同学!"

#调用函数
n=input()
c=input()
print(func(n,c))
```

程序运行结果：

```
张华
北京
你好,来自北京的张华同学!
```

程序说明：

(1) 分两行分别输入“张华”和“北京”,调用 func()函数,返回指定的问候语并输出。

(2) 函数中包括两个参数,参数传递时,按照形式参数定义的顺序,将 n 和 c 分别传递给 name 和 city。这种位置传递是使用较为普遍的方式。

除了传递简单的变量,还可以向函数传递组合数据,例如列表。将列表传递给函数后,函数就能直接访问其内容。

【例 5.5】 假设有一个用户列表,定义函数问候其中每一位用户。

【分析】 将一个名字列表传递给一个名为 user()的函数,这个函数的功能是问候列表中的每个人。

程序代码:

```
def user (names):
#向列表中每位用户都发出问候
    for i in names:
        msg = "Hello, "+i+"!"
        print(msg)

usernames=["Mary","Kitty","Bill"]
user(usernames)
```

程序运行结果:

```
Hello, Mary!
Hello, Kitty!
Hello, Bill!
```

将列表传递给函数后,函数就可以对其进行修改。在函数中对这个列表所做的任何修改都是永久性的,这让你能够高效地处理大量的数据。

【例 5.6】 假设有一个魔术师名字列表,定义 make_great()函数,为魔术师列表中的每个魔术师名字前都加入“the Great”。

程序代码:

```
def make_great(names, new_names):
    for item in names:
        item = "the Great "+item
        new_names.append(item)

names=['Bill','Andy','Joy','Lily','Bin']
new_names=[]
make_great(names, new_names)
for item in new_names:
    print(item)
```

程序运行结果：

```
the Great Bill
the Great Andy
the Great Joy
the Great Lily
the Great Bin
```

5.2.2 关键字传递

关键字实参传递是给函数同时传递形参名和实参值。直接在实参中将名称和值关联起来，因此向函数传递实参时不会混淆。关键字实参让你无须考虑函数调用中的实参顺序，还清楚地指出了函数调用中各个值的用途。

【例 5.7】 定义一个描述宠物的函数 describe_pet，包含宠物类型和名字两个参数，调用该函数，分别输出宠物类型和名字。

程序代码：

```
def describe_pet(animal_type, pet_name):
    print("I have a "+animal_type +".")
    print("My "+ animal_type+"'s name is "+pet_name+".")
describe_pet(pet_name="kitty",animal_type="cat")
```

程序运行结果：

```
I have a cat.
My cat's name is kitty.
```

程序说明：定义函数 describe_pet()时，有两个形式参数，分别是 animal_type 和 pet_name。调用时，通过赋值直接将实际参数与形式参数的名称对应起来，根据每个参数的名称传递，此时，关键字不需要遵守位置的对应关系。

用关键字实参传递时，关键字实际参数的顺序无关紧要。下面两个函数调用是等效的。

```
describe_pet(animal_type="cat", pet_name="kitty")
describe_pet(pet_name="kitty", animal_type="cat")
```

注意：关键字实参传递时务必提供正确的形参名。

5.2.3 可选参数

可选参数是定义函数时就指定默认值的参数。可选参数函数的定义形式如下。

```
def  <函数名>(<非可选参数>, <可选参数>=<默认值>) :
    <函数体>
    return  <返回值>
```

注意：可选参数必须放在非可选参数的后面。

例如，下面例 5.8 的代码中 b 和 c 是可选参数，如果调用时没有提供新值，则 b 和 c 取值为默认值 3 和 5，如果调用时提供了新值，则可选参数取新值。

【例 5.8】 定义求三个数和的函数，包含三个参数，其中两个是可选参数，默认值分别是 3 和 5。

【分析】 可选参数要放在非可选参数之后。

程序代码：

```
def Sum(a,b=3,c=5):
    return a+b+c
print(Sum(8))
print(Sum(8,2))
```

程序运行结果：

```
16
15
```

程序说明：在本例中，函数 Sum 共有三个参数，分别是 a、b、c。其中，b 和 c 是可选参数，在定义时给定了默认值，分别是 3 和 5。通过 print(Sum(8))调用 Sum 函数时，只给了一个参数 8，赋给 a，因此 b 和 c 取默认值。而在通过 print(Sum(8,2))调用时，给了两个参数，则 c 会取默认值 5。

5.2.4 可变参数

如果函数定义时参数数量不确定，则可以将形式参数指定为可变数量参数。

可变参数传递函数形式如下。

```
def  <函数名>(<参数>, * b) :
    <函数体>
    return  <返回值>
```

其中，带星号 * 的参数即为可变参数，可变参数只能出现在参数列表的最后。一个可变参数代表一个元组。有关元组的知识将会在本章后续小节中讲解。

【例 5.9】 定义一个可变参数函数，输出学生的学号、姓名和喜爱的运动项目。

【分析】 每个学生喜爱的运动项目可能有多个，因此，运动项目考虑用可变参数存储。

程序代码：

```
def stu(num,name,*sports):
    print("num:",num,"name:",name,"sports:",sports)
stu("2020001","Wang")
stu("2020002","Li","running")
stu("2020003","Zhao","running","skating")
```

程序运行结果：

```
num: 2020001 name: Wang sports: ()
num: 2020002 name: Li sports: ('running',)
num: 2020003 name: Zhao sports: ('running', 'skating')
```

程序说明：参数 sports 前面加星号，表示 sports 是可变参数。调用 stu 函数时，可以不提供值给 sports 参数；也可以为 sports 指定一个参数值，如'running'；也可以为 sports 提供两个或多个参数值，如'running'和'skating'两个值。

5.3 函数的返回值

函数可以返回 0 个或多个结果，传递返回值用 return 保留字。如果不需要返回值，则可以没有 return 语句。return 可以传递 0 个返回值，也可以传递任意多个返回值。

5.3.1 返回多个值

return 可以传递一个或多个返回值，下面举例说明如何利用 return 传递多个返回值。

【例 5.10】 编写函数，计算 1～n 的总和以及平均值，并返回总和值及平均值。

【分析】 函数有两个返回值，分别是总和值和平均值。使用 return 返回两个值，并用逗号分隔两个值。

程序代码：

```
def func(n,m):
    s=0
    for i in range(1,n+1):
        s+=i
    return s,s/m
s,ave=func(10,10)
print(s, ave)
```

程序运行结果：

```
55 5.5
```

程序说明：

(1) 在定义 func()函数时，首先将两个值分别存放到形式参数 n 和 m 中，然后通过 for 循环求得 1～n 的和，并存放在变量 s 中，最后函数返回 s 的值及 s 除以 m 的商。程序通过一个 return 返回了两个值。

(2) 在调用 func 函数时，实际参数 10，10 分别传递给形式参数 n 和 m。func 函数计算并返回和值 55 及商 5.5。最后，输出带回的总和值及平均值。

5.3.2 返回组合数据

函数可以返回任何类型的值，包括列表和字典等较复杂的数据结构。

【例 5.11】 假设列表中为一个班级所有学生的成绩，用函数返回三个最高分数。

程序代码：

```
def func(arg):
#对成绩列表进行排序，返回前三个成绩
    arg.sort(reverse=True)
    return [arg[0],arg[1],arg[2]]

ls= func([68, 86, 56, 98, 57, 95, 56, 45, 97, 56, 45, 87, 2, 87, 54, 97, 56, 45, 77, 65, 99, 22,
37, 48, 55, 64])
print('前三名成绩分别为{}、{}和{}'.format(ls[0],ls[1],ls[2]))
```

程序运行结果：

```
前三名成绩分别为 99、98 和 97
```

下面举一个返回字典的例子。

【例 5.12】 编写函数，接收姓名中名和姓两部分，返回一个表示人名的字典。

程序代码：

```
#返回一个人名信息的字典
def build_namedic(first_name, last_name):
    name={'first':first_name, 'last': last_name }
    return name
name = build_namedic('Weiwei', 'Wang')
print(name)
```

程序运行结果：