第 5 章
函数与模块化

本章学习目标

- 熟悉模块化编程思想
- 掌握函数的定义、调用和参数传递
- 理解递归函数的思想
- 掌握递归函数的定义和应用
- 掌握程序打包的第三方库


本章源代码

本章首先引入模块的概念，以及 Python 模块的管理方式；然后介绍模块化编程中函数的定义、调用和参数传递的方法；接着介绍特殊类型函数递归函数的编程思想、定义和应用，以及特殊函数匿名函数的定义和调用；再介绍 Python 第三方库，PyInstaller 程序打包库的使用方法；最后基于“中医体质辨识小助手”的实践案例的探索，让读者进一步掌握函数和模块化在医疗健康领域应用开发中的使用。

5.1 模块和包

将一段程序代码保存为一个扩展名为.py 的文件，该文件就是一个模块。Python 中的模块分为 3 种，分别是内置模块、第三方模块和自定义模块。内置模块和第三方模块又可称为库。内置模块是由 Python 解释器自带的，不用单独安装，而第三方模块是需要下载后手动安装的，如果内置模块和第三方模块没有所需的功能，就需要用户自己编写程序，将程序保存为.py 文件，即为自定义模块。无论哪种模块，都需要使用 import 语句引用后，才能在程序中调用其中的功能函数。

如果一个项目中，包含了大量的 Python 文件（模块），就会存在两个问题。第一，Python 模块过多，会导致某个模块不方便查找；第二，大型项目中多个角色的模块会出现命名冲突的问题。为了防止以上两种问题发生，将功能相似的、关联较强的模块组织起来，形成一个目录，叫作包。

Python 中的包是模块包，即程序包的简称，主要是用来包含多个相同或相似功能的 Python 模块的文件夹。因此，包是一个目录，包含一组模块和__init__.py 文件，支持多层嵌套，.py 程序文件中可以直接定义一些变量、函数和类，使用时，通过 import 语句导入，如 import myPackage.file1。

包、模块、函数的关系如图 5.1 所示。

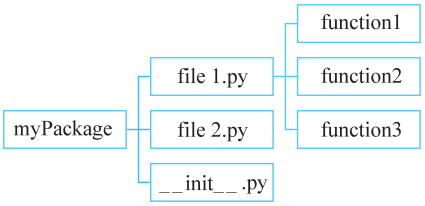

图 5.1　包、模块、函数的关系

在设计较复杂的程序时，一般采用自顶向下的方法，将总问题划分为几个子问题，每个子问题再细化为功能模块，直到分解到不能再分为止。以功能模块为单位进行程序设计，实现其求解算法的方法称为模块化。

模块化程序设计就是指在进行程序设计时将一个大程序按照功能划分为若干小程序模块，每个小程序模块完成一个确定的功能，并在这些模块之间建立必要的联系，通过模块的互相协作完成整个功能的程序设计方法。模块化的目的是降低程序复杂度，使程序设计、调试和维护等操作简单化。

函数是实现程序模块化的最小功能单元，它使得程序设计更加简单和直观，从而提高程序的易读性和可维护性，把程序中经常用到的一些计算或操作封装成通用函数，可以供其他程序随时调用。

5.2　函数

一个程序一般会分为若干程序块，每一个程序块用来实现一个特定的功能，完成某个特定功能的程序块，称为函数。函数可以直接被另一段程序或代码引用，而有效利用函数，可以减少重复编写程序的工作量。本节主要介绍自定义函数的定义和使用。

5.2.1　函数的定义和调用

Python 中定义函数的语法格式如下：

```
def 函数名([参数 1,参数 2,…,参数 n]):
    语句块
    [return [返回值 1,返回值 2,…,返回值 n]]
```

其中，def 为 Python 保留字，用来定义函数，函数名的命名规则与变量的一致；括号中是函数的参数，参数可以为空，此时，括号()不能省略，函数参数也可以是多个，多个参数之间用英文逗号隔开；语句块是函数体，可以对函数进行调用，完成函数的主要功能；return 是函数的返回保留字，当函数没有返回值时，return 保留字可以省略，函数有多个返回值时，在 return 保留字后面，将多个返回值以英文逗号隔开；函数的类型和函数的返回值类型相同，当返回值是多个时，函数的返回值是元组类型。

函数调用可以作为单独一行语句，也可以作为表达式的一部分，当作为表达式的一部分时，函数需要有返回值，如例 5.1 所示。

【例 5.1】　定义一个求和函数，对于任意整数 n，返回值是 1～n 中整数的和。

```
#例 5.1
def sumn(n):
    s=0
    for i in range(1,n+1):
        s=s+i
    return s
print(sumn(10))
```

代码执行结果如下：

```
55
```

本例中，自定义函数名为 sumn(注意：由于不能与 Python 内置函数 sum()重名，函数名不能命名为 sum)；变量 n 是函数的参数，在函数定义时，不代表任何具体的值；函数体由 3 行语句构成，其中，s 为临时变量，存放整数和，for 循环执行 n 次，每次循环将循环变量 i 累加到和变量 s 中；最后，使用 return 保留字返回变量 s 的值。该函数定义后，并不能马上被执行，只有被后面的代码调用时才被执行；语句 print(sumn(10))，即表示将整数 10 传递给参数 n，对函数进行调用。

例 5.1 中，函数的返回值只有一个 s。值得注意的是，函数的返回值也可以是多个，如例 5.2 所示。

【例 5.2】 定义一个函数，对于半径 r，返回圆的面积和周长。

```
#例 5.2
def circle_sc(r):
    s=3.14*r**2
    c=2*3.14*r
    return s,c
result=circle_sc(10)
print("半径为 10 的圆面积是：{},周长是：{}".format(round(result[0],1),
round(result[1],1)))
```

代码执行结果如下：

```
半径为 10 的圆面积是：314.0,周长是：62.8
```

本例中，函数返回值是面积变量 s 和周长变量 c，变量 result 是元组(s,c)，因此，可以通过索引的方式获取单个值，语句 round(result[0],1)，round(result[1],1))分别获取面积和周长的值，并保留 1 位小数位。

另外，在函数中，可以根据不同的条件进行判断，决定函数不同的出口和返回值，如例 5.3 所示。

【例 5.3】 定义一个函数，对于给定的身份证号码，判断是否满足 18 位，如果不满足，则提示重新输入，返回空值；如果满足 18 位，则获取第 17 位上的数，根据奇偶数判断性别，奇数为男性，偶数为女性；获取第 6～13 位上的数，表示出生日期的年(6～9 位)、月(10～11 位)、日(12～13 位)，将出生日期表示为“*年*月*日”的形式。函数返回出生日期和性别。

```
#例 5.3
def birth(birthnum):
    if len(birthnum)!=18:
```

```
            print("身份证不是 18 位,请重新输入!")
            return
        if int(birthnum[-2])%2==0:
            gender="女"
        else:
            gender="男"
        birthday=birthnum[6:10]+"年"+birthnum[10:12]+"月"+birthnum[12:14]+"日"
        return birthday,gender
print(birth("41052519890219112X"))
print(birth("410525198902191112"))
```

代码执行结果如下：

```
('1989 年 02 月 19 日', '女')
身份证不是 18 位,请重新输入!
None
```

本例中，两次调用 birth()函数，当函数参数传递错误时，函数返回值为空；正确时，返回值为一个元组(birthday,gender)。

5.2.2 函数的参数

1. 形参和实参

函数定义时，圆括号内参数列表的参数，不代表具体的数据，称为形参(形式参数)；函数调用时，将具体的数据传递给形式参数，此时的参数称为实参。根据参数的不同数据类型，将实参的值或者引用传递给形参。

当参数类型为不可变数据类型(如整数、浮点数、字符串、元组等)时，在函数内部直接修改形参的值不会影响实参；但当参数类型为可变数据类型(如列表、字典、集合等)时，在函数内部使用下标或其他方式为其增加、删除元素或修改元素值，修改后的结果是可以传递到函数之外的，即实参也会得到相应的修改，如例 5.4 所示。

【例 5.4】 体质指数，即身体质量指数(简称 BMI，本例中使用小写形式 bmi)，是国际最常用来量度体重与身高比例的工具，也是国际上常用的衡量人体胖瘦程度以及是否健康的一个标准。该指数利用身高和体重之间的比例去衡量一个人是否过瘦、标准或过胖。

bmi 通用计算公式是 bmi＝身高(m)/体重$(kg)^2$。要求根据个人信息列表的身高和体重数据，结合正常体重上限，编程定义函数计算 bmi 的值，并判断体重是否超标，最后更新到个人信息列表中。

```
#例 5.4
person1=["张三","男",20,"170cm","60kg"]
bmi_m=23.9
def bmi_caculate(person,bmi_max):
    height=int(person1[3][:-2])/100
    weight=int(person1[4][:-2])
    bmi=round(weight/height**2,1)
    person1.append(bmi)
    if bmi>bmi_max:
        person1.append("超重")
    else:
```

```
        person1.append("不超重")
    bmi_max=23.5
bmi_caculate(person1,bmi_m)
print(person1,bmi_m)
```

代码执行结果如下：

```
['张三', '男', 20, '170cm', '60kg', 20.8, '不超重'] 23.9
```

本例中，个人信息列表 person1 是可变类型，函数 bmi_caculate()定义的参数有两个，person 和 bmi_max 是形参，函数调用时的参数 person1 和 bmi_m 是实参。参数 person1 是列表类型，属于可变数据类型，参数传递时将 person1 的引用传递给形参 person；bmi_m 是浮点类型，属于不可变类型，参数传递时将 bmi_m 的引用传递给形参 bmi_max。

在函数体内部，计算 bmi 值后，将 bmi 的值与是否超重信息追加到了形参 person1 的末尾，实际上也同时修改了 person 列表变量的内容，虽然在函数体内修改了 bmi_max 的值，但不会影响函数体外 bmi_m 的值。

2. 必备参数、默认参数、可变参数和关键字参数

函数调用时，必须传递的参数，称为必备参数，也称必选参数；函数定义时，设置了默认值的参数，在函数调用时，可以传递实参，也可以不传递参数，不传递时，参数取定义时的默认值，这种参数，称为默认参数，也称可选参数。需要注意的是，必备参数必须在前面定义，其他参数在后面定义，如例 5.5 所示。

【例 5.5】 定义函数时使用必备参数和默认参数。

```
#例 5.5
def power(x,n=2):
    s = x * n
    return s
print(power(4))
print(power(3,n=3))
print(power(2,4))
```

代码执行结果如下：

```
8
9
8
```

本例中，函数定义时的形参 x 是必备参数，在函数调用时，必须传递值，3 次的调用，分别传递了实参 4、3、2。函数定义时的形参 n 是默认参数，第一次函数调用，第二个参数没有传递参数，使用默认值 2，则函数返回值是 4 * 2＝8；第二次函数调用，第二个参数传递值是 3，则函数返回值是 3 * 3＝9；第三次函数调用，第二个参数传递值是 4，则函数的返回值是 2 * 4＝8。

如果函数调用时传入的参数个数是可变的，可以是 0 个、1 个、2 个或更多，这种参数称为可变参数，可变参数通常以元组的形式传递，如例 5.6 所示。

【例 5.6】 定义个人信息，包含姓名、性别和其他信息。

```
#例 5.6
def person_info(name,gender, * hobby):
```

```
    print('姓名:',name,' 性别:',gender,'爱好:',hobby)
person_info('张林','女',"唱歌","跳舞")
person_info('王妍','女',"摄影","跳舞","弹琴")
```

代码执行结果如下：

```
姓名: 张林性别: 女爱好: ('唱歌', '跳舞')
姓名: 王妍性别: 女爱好: ('摄影', '跳舞', '弹琴')
```

本例中，函数定义时的形参 * hobby 是可变参数，在函数调用参数传递时，可以将若干参数以元组的形式保存，因此第一次调用，hobby＝('唱歌', '跳舞')；第二次调用，hobby＝('摄影', '跳舞', '弹琴')。

函数调用时传入的参数个数是可变的，且传入的每个参数是键值对的形式，这种参数称为关键字参数，关键字参数通常以字典的形式传递，如例 5.7 所示。

【例 5.7】 定义个人信息，包含姓名、性别和其他信息。

```
#例 5.7
def person_info(name,gender,**other):
    print('姓名:',name,' 性别:',gender,' 其他:',other)
person_info('张林','女',年龄=17,城市='北京')
person_info('王亮','男',城市='上海')
```

代码执行结果如下：

```
姓名: 张林性别: 女其他: {'年龄': 17, '城市': '北京'}
姓名: 王亮性别: 男其他: {'城市': '上海'}
```

本例中，函数定义时的形参**other 是关键字参数，在函数调用参数传递时，可以将若干参数作为键值对以字典的形式保存，因此第一次调用，other＝{'年龄': 17, '城市': '北京'}；第二次调用，other＝{'城市': '上海'}。

3. 参数传递

在函数调用参数传递时，形参和实参按位置一一对应地传递，称为位置传递；在函数调用参数传递时，参数按照名称显式地传递，称为名称传递。按位置传递，不需要给出参数的名称，但形参和实参的位置相同、类型相同，否则容易出错；按名称传递参数，则不关心参数的前后顺序，在参数较多时，不容易混淆，如例 5.8 所示。

【例 5.8】 定义个人信息函数，函数调用时分别按位置和按名称传递参数。

```
#例 5.8
def person_info(name,gender,age,city):
    print('姓名:',name,' 性别:',gender,'年龄: ',age,'城市:',city)
person_info('张林','女',17,'北京')
person_info(name='王亮',city='上海',gender="男",age="20")
```

代码执行结果如下：

```
姓名: 张林性别: 女年龄: 17 城市: 北京
姓名: 王亮性别: 男年龄: 20 城市: 上海
```

本例中，第一次调用时，按照参数位置将值对应传递到形参中；第二次调用时，打乱实参顺序后，显式地对形参进行赋值传递。

练一练

以下函数打印结果正确的是_____。

```
def chanageInt(number2):
      number2 = number2+1
      print("changeInt: number2= ",number2)
#调用
number1 = 2
chanageInt(number1)
print("number:",number1)
```

A. changeInt：number2＝3　　number：3

B. changeInt：number2＝3　　number：2

C. number：2　　changeInt：number2＝2

D. number：2　　changeInt：number2＝3

5.2.3　全局变量与局部变量

在函数外部定义的变量，称为全局变量；在函数内部定义的变量，称为局部变量。

全局变量的生命周期是程序的整个运行周期，只有程序被关闭后，全局变量才会被销毁并释放内存空间，因此，全局变量在整个程序中，都可以使用；而局部变量的生命周期是函数调用时间，函数调用结束返回后，局部变量就销毁并释放内存空间，因此，局部变量的作用范围是这个函数内部，即只能在这个函数中使用，在函数的外部是不能使用的。

在函数体内，当局部变量与全局变量同名时，函数体对局部变量的修改、删除等不会影响全局变量，即相当于创建了一个新的变量，全局变量不会被使用；而当需要在函数体内使用全局变量时，则需要在局部变量前加 global 关键字声明，此时该局部变量即是全局变量，如例 5.9 所示。

【例 5.9】　全局变量与局部变量。

```
#例 5.9
def abs_xy(x,y):
    if x>y:
        s=x-y
    else:
        s=y-x
    print("函数内部 s 的值是: ",s)
    return
s=0
abs_xy(4,13)
print("函数外部 s 的值是: ",s)
```

代码执行结果如下：

```
函数内部 s 的值是: 9
函数外部 s 的值是: 0
```

本例中，语句 s＝0 定义了全局变量 s，但是当函数调用时，s 是局部变量，是函数内部变量，s 被赋了 x 和 y 的差值，当函数返回后，局部变量 s 释放，最后打印语句仍然是全局变量

s 的值 0。

当全局变量是列表等可变数据类型时，如果函数体中没有声明同名的新的局部变量，可以认为局部变量被当作全局变量使用，也不需要 global 声明。如例 5.4 中，列表变量 person 在函数外部被定义，在函数体内，并没有重新对 person 声明，而是对 person 列表进行了修改追加了新内容。需要注意的是，当列表变量在函数体内被声明后，就不再是全局变量了，如例 5.10 所示。

【例 5.10】 定义函数时使用全局变量。

```
#例 5.10
def func(n):
    ls=[]
    global s
    for i in range(n):
        s=s+i
        ls.append(s)
    print("函数内部 s 的值是: ",s)
    print("函数内部 ls 的值是: ",ls)
    return
ls=[1,2]
s=0
func(5)
print("函数外部 s 的值是: ",s)
print("函数外部 ls 的值是: ",ls)
```

码执行结果如下：

```
函数内部 s 的值是: 10
函数内部 ls 的值是: [0, 1, 3, 6, 10]
函数外部 s 的值是: 10
函数外部 ls 的值是: [1, 2]
```

本例中，函数体外部定义全局变量 s=0，函数体内部通过 global 关键字将 s 声明为全局变量，因此 s 值在函数内部被修改；函数体外部定义全局变量 ls=[1,2]，函数体内部，重新定义了同名的列表变量 ls，此时，ls 就是局部变量，对 ls 的修改不会改变外部同名全局变量 ls 的值。

练一练

1. 以下关于 Python 全局变量和局部变量的描述中，错误的是_____。

   A. 局部变量在函数内部创建和使用，函数退出后变量被释放

   B. 全局变量一般指定义在函数之外的变量

   C. 使用 global 关键字声明后，变量可以作为全局变量使用

   D. 当函数退出时，局部变量依然存在，下次函数调用可以继续使用

2. 有以下函数，下面对 list 的值输出正确的是_____。

```
def chanageList(list):
    list.append(" end")
    print("list",list)
```

```
#调用
strs=['1','2']
chanageList(strs)
print("strs",strs)
```

A. strs['1','2']　　B. list['1','2']

C. list['1','2','end']　　D. strs['1','2','end']

5.2.4　匿名函数

当一次性使用函数时，函数体语句较少，如只有一个表达式，为了节省内存中的变量定义空间，就不需要定义函数名，此时，可以定义匿名函数。匿名函数的语法格式如下：

```
函数名=lambda [参数]:返回值
```

匿名函数可以有 0 个或多个参数，但必须有返回值。匿名函数的调用和一般函数的调用相似，可以单独使用，也可以作为表达式的一部分，如例 5.11 所示。

【例 5.11】　匿名函数的定义和调用。

```
#例 5.11
c=lambda x,y,z:x*y*z
m=lambda :print("欢迎学习匿名函数!")
m()
print(c(1,2,3))
```

代码执行结果如下：

```
欢迎学习匿名函数!
6
```

本例中，定义了两个匿名函数，第一个匿名函数有 3 个参数 x、y、z，返回值是表达式 x*y*z；第二个匿名函数没有参数，函数功能是打印一行文字。注意，匿名函数不论多复杂，只能是一行语句或表达式，不能是多行语句。

练一练

运行如下代码，程序的输出结果是_____。

```
c = lambda x, y: y if x < y else x*y
print(c(6,2))
```

A. 2　　B. 6　　C. 8　　D. 12

5.3　递归函数

在计算机算法中，递归是一种解决复杂问题的有效算法。递归算法就是将原问题分解为规模缩小的子问题，然后利用递归调用的方法来表示问题的解，即用同一种方法解决规模不同的问题。递归算法有两个阶段：递和归。递是对复杂问题分解，直到找到确定解为止（递归出口）；归是将确定的解带入上一层问题的求解式中，直到最上层问题得到解决。

递归思维是一种从下而上的思维方式，使用递归算法往往可以让代码更简洁。设计递

归算法时，逻辑性非常重要，需要注意以下几个关键点。

（1）明确递归的终止条件。

（2）确定递归终止时问题的确定解。

（3）程序逻辑有重复性。

在程序设计中，递归函数是实现递归算法的有效方法。

5.3.1 递归函数的定义

函数在定义时调用了函数本身，就称为递归函数。每次函数调用都会开辟新的内存空间，递归调用可以看成函数的嵌套调用。按照递归算法的原理，递归函数必须自己调用自己，递归函数在函数体定义时，必须有明确的结束条件，即递归出口，如例 5.12 所示。

【例 5.12】 计算 1＋2＋…＋n 的和。

```
#例 5.12
def sum_n(n):
    if n == 1:
        return 1
    return n + sum_n(n - 1)
print(sum_n(3))
```

代码执行结果如下：

```
6
```

本例中，递归的结束条件是 n＝＝1，明确的出口解是 1；在函数返回值的表达式中，调用了函数本身，递归过程如图 5.2 所示。

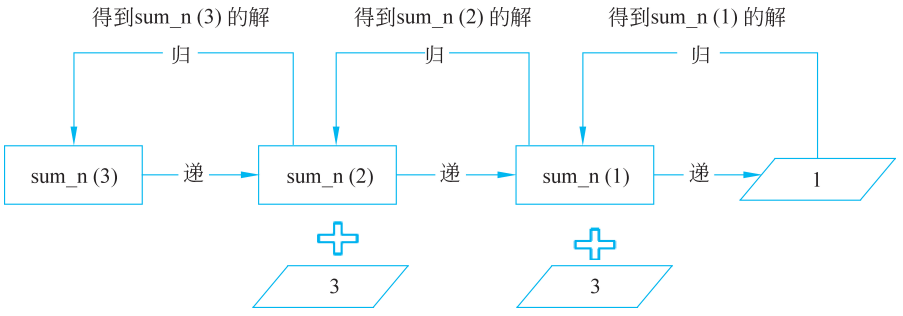

图 5.2 递归计算过程

注意： 递归函数的结束条件，至少有 1 个，也可以有多个；函数对自身的调用，可以在函数体中，也可以在 return 返回值表达式中。

练一练

1. 以下程序中没有用到递归函数的是_____。

A.

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1) num = eval(input("请输入一个整数："))
```