第3章

函　数

本章介绍 Python 语言中的函数，内容包括函数的定义方法、函数中参数的传递方法、函数中参数的设置、匿名函数及递归函数等。

本章要点：

- 函数的定义方法。
- 函数参数的设置。
- 匿名函数的使用。

3.1　函数概述

3.1.1　模块和包

Python 程序是由函数(Function)、模块(Module)和包(Package)组成的。其中，模块是处理某一类问题的变量、函数和类的集合，包是由一系列模块组成的集合。

模块是一个 Python 文件，以.py 结尾，在模块中能定义函数、类和变量，在模块中也能包含可执行的代码，运用模块能够有逻辑地组织 Python 代码段。常用的模块导入方法有以下 3 种：①import module1(模块名)，之后在程序中就可以使用在模块中定义的变量、函数及类等，例如使用“模块名.函数名”来调用其中的函数；②from module1 import *，* 表示导入该模块中的所有函数及变量等；③from module1 import name1，表示导入模块中的一个指定部分，例如 from fib import fibonacci 代表导入模块 fib 中的 fibonacci 函数。

包是一个分层次的文件目录结构，它定义了一个由模块、子包、子包下的子包等组成的 Python 应用环境。简单来说，包就是文件夹，但该文件夹下必须存在__init__.py 文件，该文件的内容可以为空。__init__.py 用于标识当前文件夹是一个包。

3.1.2　什么是函数

如果在开发程序时需要某块代码多次，为了提高编写的效率以及方便代码的重用，把具有独立功能的代码块组织为一个小模块，这就是函数。函数是 Python 为了代码效率的最大化以及减少冗余而提供的最基本的程序结构。函数是一段代码，通过名字来进行调用，它能将一些数据(参数)传递进去进行处理，然后返回一些数据(返回值)，也可以没有返回值。

3.2 函数的定义

Python 中自定义函数的创建方法如下：使用关键字 def 定义函数，其后紧接函数名，括号内包含了将要在函数体中使用的形式参数(简称形参，调用函数时为实参，函数可以有参数，也可以没有，但必须保留括号)，以冒号结束；然后另起一行编写函数体，函数体的缩进通常为 4 个空格或者一个制表符。需要注意的是，函数在定义后如果不经调用，将不会被执行。

定义函数的格式如下：

```
def 函数名():
    函数体
```

例 3-1 函数的定义举例。

```
In:
    def add(x,y):
        x = x + 3
        y = y + 4
        return x,y
    x,y = 10,20
    add(x,y)
    print(x,y)
    x,y = add(x,y)
    print(x,y)
Out:
    10 20
    13 24
```

例 3-2 用函数实现 Fibonacci 序列。

```
In:
    def fib(n):
        a,b = 0,1
        while a < n:
            print(a,end = ' ')
    a,b = b,a + b
            print()
    fib(6)          # 调用函数
Out:
    0
    1
    1
    2
    3
    5
```

3.3 参数传递

Python 中函数参数的传递可以分为实参为不可变对象的传递和实参为可变对象的传递。字符串、元组、数值等类型是不可以更改的对象，而列表和字典等是可以修改的对象。若实参为不可变对象，即使函数体中修改了形参，实参的值在函数调用返回时仍然保持不变；如果函数调用时参数为可变对象的传递，若函数体中修改了形参，则实参的值会随之发生改变。

3.3.1 实参变量指向不可变对象

实参变量可以指向不可变的对象，例如整型。不可变对象是指对象所指向的内存中的值不能被改变，当改变这个变量时，原来指向的内存中的值不变，变量不再指向原来的值，而是开辟一块新的内存，变量指向新的内存。

例 3-3 实参变量指向不可变对象实例。

```
In:
    def change(a):
        print('<2>修改值前,形参 a 的内存地址:',id(a))
        a = a + 1           #改变 a 的值
        print('<3>修改值后: a = {}'.format(a))
        print('<3>修改值后,形参 a 的内存地址:',id(a))
    def main():
      x = 3
      print('<1>调用函数前,实参 x 的内存地址:',id(x))
      print('<1>调用函数前: x = {}'.format(x))
      change(x)
      print('<4>调用函数后: x = {}'.format(x))
      print('<4>调用函数后,实参 x 的内存地址:',id(x))
    main()
Out:
    <1>调用函数前,实参 x 的内存地址: 140716719748960
    <1>调用函数前: x = 3
    <2>修改值前,形参 a 的内存地址: 140716719748960
    <3>修改值后: a = 4
    <3>修改值后,形参 a 的内存地址: 140716719748992
    <4>调用函数后: x = 3
    <4>调用函数后,实参 x 的内存地址: 140716719748960
```

分析：在 Python 中，类型属于对象，变量是没有类型的，变量只是指向了对象。本例函数调用时，在参数传递后，实参变量 x 和形参变量 a 都指向对象 3(整型)。在 change()函数内，对 a 赋新值 4，即 a+1 后，由于不可变对象的值不能变化，所以为 4 分配新的内存单元，同时使 a 指向这个对象。函数调用返回时，由于变量 x 和 a 指向了不同的对象，而 x 所指向的不可变对象的内存地址一直没变，所以输出的 x 的值也不变。

3.3.2 实参变量指向可变对象

实参变量可以指向可变对象，例如列表。可变对象是指对象的值可以改变，当更改这个

变量时还是指向原来的内存地址，只是在原来的内存地址进行值的修改并没有开辟新的内存。

例 3-4 实参变量指向可变对象实例。

```
In:
    def change(a):
        print('<2>修改值前,形参 a 的内存地址:',id(a))
        a.append(999)  #a 和 x 指向同一个地址,因为 a 是可变类型,故不创建对象副本,直接修改
        print('<3>修改值后: a = {}'.format(a))
        print('<3>修改值后,形参 a 的内存地址:',id(a))
    def main():
      x = [1,2]
      print('<1>调用函数前,实参 x 的内存地址:',id(x))
      print('<1>调用函数前: x = {}'.format(x))
      change(x)
      print('<4>调用函数后: x = {}'.format(x))
      print('<4>调用函数后,实参 x 的内存地址:',id(x))
    main()
Out:
    <1>调用函数前,实参 x 的内存地址: 2188578220032
    <1>调用函数前: x = [1, 2]
    <2>修改值前,形参 a 的内存地址: 2188578220032
    <3>修改值后: a = [1, 2, 999]
    <3>修改值后,形参 a 的内存地址: 2188578220032
    <4>调用函数后: x = [1, 2, 999]
    <4>调用函数后,实参 x 的内存地址: 2188578220032
```

分析：本例函数调用时，在参数传递后，实参变量 x 和形参变量 a 都指向同一个列表(list)对象[1，2]。由于列表对象本身是可以改变的，因此在 change()函数内向列表中加入一个元素不会重新创建对象，而是直接在原对象中添加了新的元素。调用结束后，变量 x 和 a 仍然指向同一个对象，改变 a 指向对象的值也就改变了 x 指向对象的值。

注意：若在函数中给形参赋予了全新的值，即让形参指向了新的对象，则形参的改变就不会影响到实参，例如在例 3-4 的 change()函数中，令 a=['Apple']。如果形参的改变是直接在原指向对象上进行操作，函数调用后形参和实参仍然指向同一个对象，则形参的改变会同步到实参。

3.4 函数参数的设置

3.4.1 函数参数的类型

在 Python 中函数参数主要有 4 种，即位置参数、默认参数、关键字参数、可变参数。

1. 位置参数

在调用函数时，根据函数定义的参数位置来传递参数，实参的个数、顺序必须和形参保持一致，否则会抛出异常。

2. 默认参数

在定义函数时为部分形参设置了默认值，在调用函数时可传或不传该默认参数的值(注意，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)。

3. 关键字参数

在调用函数时可以通过“键-值”形式指定参数，即传参时把定义函数时的参数名和对应的值一起传入函数中，此时可不考虑传入参数的顺序。

4. 可变参数

可变参数主要包括任意数量的可变位置参数和任意数量的关键字可变参数，* args 参数传入时存储在元组中，** kwargs 参数传入时存储在字典内。

在 Python 中定义函数时参数的顺序一般为位置参数、默认参数、可变参数、关键字参数，否则会造成程序不能被正确解析。此外，在定义函数时应尽量避免 3 种以上的参数类型混合使用，否则会造成函数的可读性较差。

例 3-5 没有参数的函数。

```
In:
    def printInfo():
        '定义一个函数,能够完成打印信息的功能'       #定义函数的注释信息
        print('-------------------------------------')
        print('       人生苦短,我用 Python       ')
        print('-------------------------------------')
    print('The function name is:',printInfo.__name__) #获取函数的名称
    print('The function note is:',printInfo.__doc__)  #获取函数的注释信息
    printInfo()
Out:
    The function name is: printInfo
    The function note is: 定义一个函数,能够完成打印信息的功能
    -------------------------------------
        人生苦短,我用 Python
    -------------------------------------
```

注意：__name__可以获取函数的名称，__doc__可以获取函数的注释信息。

3.4.2 位置参数

在传递位置参数时，实参依次按顺序传递给形参即可，注意，实参与形参必须按顺序一一对应。

例 3-6 编写带两个位置参数的函数，输出一个数据区间内所有的素数。

```
In:
    import math
    def getPrimeList(lower,upper):              #返回数据区间[lower,upper]内的所有素数
         primeList = []                         #primeList 为返回的素数列表
```

```
        for num in range(lower, upper + 1):
            if num > 1:                                  #所有的素数都要大于 1
                for i in range(2, num + 1):
                    if num % i == 0:                     #判断 num 能否被 i 整除
                        break                            #不是素数,则退出 for 循环
                if i == num:                             #若 i 等于 num,说明 num 是素数
                    primeList.append(num)
        return primeList
    start = int(input('请输入区间的最小值: '))           #输入区间的最小值
    end = int(input('请输入区间的最大值: '))             #输入区间的最大值
    pList = getPrimeList(start,end)
    print("在区间", start, "和", end, "之间的素数有:",pList)
Out:
    请输入区间的最小值: 10
    请输入区间的最大值: 20
    在区间 10 和 20 之间的素数有: [11, 13, 17, 19]
```

3.4.3 默认参数

默认参数指在定义函数时就为参数设置了默认值,在调用函数时可以不传递这个默认参数。需要注意的是,默认参数一般要放到参数列表的最后,即默认参数后面不能再有不带默认值的参数。

例 3-7 在定义函数时设置参数的默认值。

```
In:
    def f(x = True):
        if x:
            print("{} is a correct word".format(x))
        else:
            print("Not right!")
    f()                  #使用参数的默认值调用函数
    x = False
    f(x)                 #不使用默认值调用函数
Out:
    True is a correct word
    Not right!
```

例 3-8 计算椭球的表面积和体积,在调用时修改默认参数的值。

```
In:
    import math
    def cal(a = 1,b = 1,c = 1):
        s = 4/3 * a * b * math.pi
        v = 4/3 * a * b * c * math.pi
        return s,v
    print("请输入椭球方程的 3 个系数: ")
    a,b,c = eval(input("请输入 a,b,c: "))
    s,v = cal(a,b,c)
    print("x1 = {0:.3f},x2 = {1:.3f}".format(s,v))
```

```
Out:
    请输入椭球方程的 3 个系数:
    请输入 a,b,c: 1,2,3
    x1 = 8.378,x2 = 25.133
```

注意：默认参数一般要放到参数列表的最后，例如下例会报错。

```
In:
    def f(x = True,y):      #默认参数的例子,默认参数一般要放到参数列表的最后
        if x:
            print("{} is a correct word".format(x))
        else:
            print("Not right!")
    f(False)                #报错
Out:
    File "< ipython - input - 13 - 32adc56681e6 >", line 1
        def f(x = True,y):
    SyntaxError: non - default argument follows default argument
```

修改为如下格式即可：

```
In:
    def f(x,y = True):#默认参数的例子,默认参数一般要放到参数列表的最后
    if x:
              print("{} is a correct word".format(x))
        else:
              print("Not right!")
    f(False)
Out:
    Not right!
```

说明：默认值只计算一次。当默认值是可变对象(例如列表、字典或大多数类的实例)时，这会有所不同。以下函数将在后续调用中累积传递给它的参数值，默认值在定义范围内的函数定义点进行计算，例如下例。

例 3-9 默认值在定义范围内的函数定义点进行计算的实例。

方法一：

```
In:
    i = 5
    def f(arg = i):
        print(arg)
    i = 6
    f()           #调用时使用默认值
Out:
    5
```

方法二：

```
In:
    i = 5
    def f(arg = i):
```

```
        print(arg)
    i = 6
    f(i)          #调用时传入参数值
Out:
    6
```

请分析方法一和方法二的不同之处。

例 3-10 默认值对后续调用影响的实例。

```
In:
    def f(a, L=[]):
        L.append(a)
        print('参数 L 的内存地址',id(L))
        return L
    print('第一次调用后的返回值:',f(1))
    print('第二次调用后的返回值:',f(2))
    print('第三次调用后的返回值:',f(3))
Out:
    参数 L 的内存地址 2188578218176
    第一次调用后的返回值: [1]
    参数 L 的内存地址 2188578218176
    第二次调用后的返回值: [1, 2]
    参数 L 的内存地址 2188578218176
    第三次调用后的返回值: [1, 2, 3]
```

分析：f()函数在定义时，默认参数 L 的值就被计算出来了，即[]。因为默认参数 L 也是一个变量，它指向列表类型对象[]，每次调用该函数，L 指向的对象不变，对象的内容可变。若改变了 L 的内容，则下次调用时默认参数的内容就变了，不再是函数定义时的[]了。

如果不希望在后续调用之间共享默认值，则每次调用函数时让 L 指向新的对象，如下例所示：

```
In:
    def f(a, L=None):
        if L is None:
            L = []          #L 定义为新的列表
        L.append(a)
        print('参数 L 的内存地址',id(L))
        return L
    print('第一次调用后的返回值:',f(1))
    print('第二次调用后的返回值:',f(2))
    print('第三次调用后的返回值:',f(3))
Out:
    参数 L 的内存地址 2188577083520
    第一次调用后: [1]
    参数 L 的内存地址 2188578364544
    第二次调用后: [2]
    参数 L 的内存地址 2188578365504
    第三次调用后: [3]
```

3.4.4 关键字参数

如果希望函数调用时不用固定参数的顺序，可以在调用的同时指定形参和实参，这些参数称为关键字参数。使用关键字参数是指在函数调用时可以明确地为形参指定实参，可以不用固定参数的顺序。

例 3-11 使用关键字参数的实例。

```
In:
    def parrot(age, name, color):
        print('-- Parrot name is:{},Age is:{},Color is:{} -- '.format(name,age,color))
    parrot(8,color = 'Red',name = 'Jerry')   #传入一个位置参数,传入两个关键字参数,顺序与形
                                             #参可以不一致
Out:
    -- Parrot name is:Jerry,Age is:8,Color is:Red --
```

3.4.5 可变参数

在 Python 函数中还有一种参数类型——可变参数，即传入函数的参数个数是可变的。可变参数可以分为形参前使用 * args 和使用 ** kwargs 两种形式，若定义函数时二者同时存在，一定要将 * args 放在 ** kwargs 之前。

1. 在形参前使用 * 号

在定义函数时，在形参 args 前添加一个 * 号，则 * args 参数收集所有未匹配的位置参数组成一个元组(tuple)对象，形参 args 指向此元组(tuple)对象。在调用函数时， * args 参数用于解包元组(tuple)对象的每个元素，作为一个一个位置参数传入函数中。

例 3-12 可变参数实例，传入的参数组成一个元组。

```
In:
    def calc( * number):            #将传入的数累加
        sum = 0
        print('Type of number is:',type(number),'Value of number is:',number)
        for n in number:
            sum = sum + n
        return sum
    print(calc(1,2,3))              #将 1,2,3 作为参数传递给函数 calc()
    l = [1,2,3,4,5]
    print(calc( * l))               #将列表[1,2,3,4,5]作为参数传递给函数 calc()
Out:
    Type of number is: <class 'tuple'> Value of number is: (1, 2, 3)
    6
    Type of number is: <class 'tuple'> Value of number is: (1, 2, 3, 4, 5)
    15
```

2. 在形参前使用 ** 号

在定义函数时，在形参 kwargs 前添加两个 ** 号，则 ** kwargs 参数收集所有未匹配的

位置参数组成一个字典(dict)对象,形参 kwargs 指向此字典(dict)对象。在调用函数时,** kwargs 参数用于解包字典(dict)对象的每个元素,作为一个一个关键字参数传入函数中。

可变长度参数具有可以扩展函数的功能。例如,在 person()函数中,函数保证能接收到 name 和 age 两个参数,但是如果调用者愿意提供更多的参数,函数也能收到。假设用户正在实现账号注册的功能,除了用户名和年龄是必填项以外,其他都是可选项,利用可变长度参数来定义这个函数就能满足注册的需求。

例 3-13 可变参数实例,传入的参数组成一个字典。

```
In:
    def person(name,age, ** kw):
        print('Name is:',name,',Age is:',age,',Other parameters:',kw)
    person('Michael',30)                    #传入两个位置参数
    person('Michael',30,City = 'Beijing')   #传入两个位置参数及可变参数
    person('Michael',30,City = 'Beijing',Gender = 'Male',Job = 'Engineer')   #传入两个位置参数
                                                                            #及可变参数
    extra = {'Gender':'Male','Job':'Engineer'}
    print('传入的字典数据: ',extra)
    person('Michael',30, ** extra) #传入两个位置参数,一个字典类型作为可变参数
Out:
    Name is: Michael ,Age is: 30 ,Other parameters: {}
    Name is: Michael ,Age is: 30 ,Other parameters: {'City': 'Beijing'}
    Name is: Michael ,Age is: 30 ,Other parameters: {'City': 'Beijing', 'Gender': 'Male', 'Job':
'Engineer'}
    传入的字典数据: {'Gender': 'Male', 'Job': 'Engineer'}
    Name is: Michael ,Age is: 30 ,Other parameters: {'Gender': 'Male', 'Job': 'Engineer'}
```

** extra 表示把 extra 这个 dict 的所有 key-value 用关键字参数传入函数的 ** kw 参数,kw 将获得一个 dict,注意 kw 获得的 dict 是 extra 的一个副本,对 kw 的改动不会影响到函数外的 extra。

3.5 匿名函数

Python 允许使用 lambda 语句创建匿名函数,常用于临时需要一个函数的功能,但又不想定义函数的场合,即函数没有具体的名称。在 lambda 语句中,冒号前是函数的参数列表,若有多个参数,要使用逗号分隔,冒号右边是返回值。如此便构建了一个函数对象,与使用 def 语句创建的函数相比,使用 lambda 语句创建的函数对象没有名字。

lambda 函数的语法只包含一个语句,如下:

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

实例如下:

```
    sum = lambda arg1, arg2: arg1 + arg2
    print( "Value of total : ", sum( 10, 20 ))       #调用 sum()函数
```

lambda 函数能接收任何数量的参数,但只能返回一个表达式,匿名函数不能直接调用

print(),因为 lambda 需要一个表达式。

例 3-14 匿名函数实例 1。

```
In:
    f = lambda x,y:x + y
    print(type(f))
    k = f(10,12)
    print(k)
Out:
    <class 'function'>
    22
```

例 3-15 匿名函数实例 2。

```
In:
    pairs = [(5, 'five'), (2, 'two'), (4, 'four'), (3, 'three'),(1,'one')]
    pairs.sort(key = lambda pair: pair[0])
    print(pairs)
Out:
    [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five')]
```

3.6 递归函数

如果一个函数在执行过程中又调用了函数本身,则称之为递归函数。递归函数可以将一个大的复杂问题层层转换为一个问题本质相同但规模更小的问题。需要注意以下问题:递归必须有一个明确的结束条件;每次进入更深一层的递归时,问题的规模相比上次递归都应有所减少;函数递归的深度不能太大,否则容易引起内存崩溃。

例 3-16 用递归函数实现返回 Fibonacci 序列的前 n 个数。

```
In:
    result = []
    def fac(n):
        if(n <= 1):
            return n
        else:
            return(fac(n-1) + fac(n-2))
    n = int(input("请输入 n:"))
    print("Fibonacci 序列:")
    for i in range(n):
    result.append(fac(i))
    print(result)
Out:
    请输入 n:10
    Fibonacci 序列:
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

3.7 本章实战例题

例 3-17 定义无参函数和有参函数的实例。

```
In:
    def happyA():
        print('Happy New Year')
    def happyB(name):
    happyA()
    happyA()
        print('Hello {}'.format(name))
    happyA()
    happyB('Tom')
```

例 3-18 定义与调用函数的实例。

```
In:
    def happyA(name):
        print('Happy birthday {}!'.format(name))
    def happyB(person):
        happyA(person)
        happyA(person)
        print("Happy Happy {}".format(person))
        happyA(person)
        happyA(person)
    def main():
        #name = 'Tom'
        person = input('input name:')
    happyB(person)
    main()
```

例 3-19 定义计算平面上两点间距离的函数 distance(),计算三角形的周长,点在横坐标和纵坐标上的位置分别用 x 和 y 表示。

```
In:
    import math
    def square(x):
        return x * x
    def distance(x1,y1,x2,y2):
        dist = math.sqrt(square(x1 - x2) + square(y1 - y2))
        return dist
    def isTig(x1,y1,x2,y2,x3,y3):
        flag = ((x1 - x2) * (y3 - y2) - (x3 - x2) * (y1 - y2))!= 0
        return flag
    def main():
        print('x,y')
        x1,y1 = eval(input('x1,y1:'))
        x2,y2 = eval(input('x2,y2:'))
```

```
        x3,y3 = eval(input('x3,y3:'))
        if(isTig(x1,y1,x2,y2,x3,y3)):
            per = distance(x1,y1,x2,y2) + \
                distance(x3,y3,x2,y2) + distance(x1,y1,x3,y3)
            print('the per is:{}'.format(per))
        else:
            print('no triangle!s')
main()
```

例 3-20 输入一个数字,用递归函数求出该数字的各位数之和。

```
In:
l = []
def sum_digits(b):
  if(b == 0):
     return l
  dig = b % 10
  l.append(dig)
  sum_digits(b//10)
n = int(input("请输入一个数字: "))
sum_digits(n)
print(sum(l))
```

例 3-21 设置函数参数默认值的实例。

```
In:
def ask_ok(prompt, retries = 4, reminder = 'Please try again!'):          #默认参数值
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

例 3-22 编写函数,求输入的正整数的阶乘。

```
In:
def getFac(num):
    factorial = 1
    if num < 0:
        print("无法求负数的阶乘!")
    elif num == 0:
        print("0 的阶乘是 1!")
    else:
        for i in range(1,num + 1):
```

```
                    factorial = factorial * i
        return factorial
    num = int(input('请输入一个正整数: '))
    fac = getFac(num)
    print("{}的阶乘是: {}".format(num,fac))
```

例 3-23 从键盘输入 n 个数组成列表(输入－1 结束),编写带可变参数的函数 calc(输入参数为该列表),求该列表的总和及均值。

```
In:
    def calc( * numList):
    sumList = 0
        for i in range(len(numList)):
            sumList += numList[i]
        return sumList,sumList/len(numList)
    numList = []              #用来存放用户输入的数据
    k = eval(input('input k:'))
    while k!= -1:             #以 -1 结束输入
    numList.append(k)
        k = eval(input('input k:'))
    print('The sum and avg is:',calc( * numList))
```

例 3-24 使用函数判断输入字符串是否为回文串。

```
In:
    def main():
        s = input("input a string:").strip()
        if iss(s):
              print("Yes")
        else:
              print("No")
    def iss(s):
        low = 0
        high = len(s) - 1
        while low < high:
              if s[low] != s[high]:
                    return False
              low += 1
              high -= 1
        return True
    main()
```

例 3-25 使用函数根据中奖概率模拟中奖次数。

```
In:
    import random
    count1, count2, count3 = 0, 0, 0
    n = eval(input("请输入模拟抽奖次数:"))
    def reward():
```

```
        global count1, count2, count3
        for i in range(n):
            number = random.uniform(0, 1)
            if 0 < number <= 0.1:
                count1 += 1
            elif 0.1 < number <= 0.3:
                count2 += 1
            else:
                count3 += 1
        print("一等奖:", count1, "二等奖:", count2, "三等奖:", count3)
    reward()
```

3.8 本章小结

本章主要介绍了 Python 语言中的函数，并结合实例讲解了函数的定义方法、函数参数的传递方法、函数参数的设置，以及匿名函数、递归函数等内容。

本章需重点掌握的内容包括函数的定义方法、函数参数的设置和匿名函数的使用。

3.9 本章习题

1. 定义函数 triangle(a,b,c)，其可以根据用户输入的 3 条边判断三角形的类型。

2. 编写带可变参数的函数，分别用来计算用户输入的任意个非 0 数据的平均值、方差、中位数。

3. 编写函数 fun(filename)，从文件中(文件中是由空格分开的数字)读出所有数字，求所有数字的和。

4. 编写函数 getbin(number)，返回输入整数的二进制形式。

5. Fibonacci 序列实例，定义函数，返回由斐波那契数列中前 n 个数组成的列表。

6. 编写函数，删除传入字符串中的标点符号。