Python编程案例教程

第7章 函 数

本章导读

在实际开发过程中，经常会遇到很多完全相同或者非常相似的操作，这时，可以将实现类似操作的代码封装为函数，然后在需要的地方调用该函数。这样不仅可以实现代码的复用，还可以使代码更有条理性，增加代码的可靠性。

本章首先介绍函数的定义和调用，并详细介绍函数的参数和返回值；然后介绍函数的嵌套、递归函数、匿名函数以及变量的作用域；接着介绍模块的导入和创建方法；最后通过两个典型案例，让读者进一步掌握函数在程序设计中的应用。

学习目标

- 掌握函数的定义和调用方法
- 熟悉函数的返回值和函数参数传递的过程
- 理解函数的嵌套调用与递归调用的定义和使用方法
- 了解匿名函数
- 掌握局部变量和全局变量的区别和典型用法
- 掌握模块的导入与创建方法以及包的使用方法

7.1 函数的定义和使用

7.2 函数的参数

7.3 函数的嵌套

7.4 递归函数

7.5 变量作用域

7.6 匿名函数

7.7 模块

7.8 典型案例

7.1 函数的定义和使用

- 7.1.1 函数的定义
- 7.1.2 函数的调用
- 7.1.3 函数的返回值

函数的定义和使用

函数是一段具有特定功能的、可重复使用的代码段，它能够提高程序的模块化和代码的复用率。

- Python提供了很多内建函数（如print()、input()、int()函数等）
- 标准库函数（如math库中的sqrt()函数）
- 用户还可以自己编写函数，称为自定义函数。

7.1.1 函数的定义

在Python中，定义函数的一般形式为：

```
def 函数名([形式参数列表]):
    函数体
```

- 使用def关键字来定义函数
- 函数名可以是任何有效的Python标识符
- 函数名后的圆括号内是形式参数列表（简称形参），形式参数列表是调用该函数时传递给它的值，可以有零个、一个或多个，当传递多个参数时各参数之间由逗号分隔。
- 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。

7.1.1 函数的定义

注意

（1）即使该函数不需要接收任何参数，也必须保留一对空的圆括号。

（2）括号后面的冒号不能省略。

（3）函数体相对于def关键字必须保持一定的空格缩进。

例如：定义名为“print_text()”的函数，该函数的功能是输出“Hello world!”

```
def print_text():
    print("Hello world!")
```

7.1.2 函数的调用

定义了函数后，就相当于有了一段具有特定功能的代码，要想执行这些代码，需要调用函数。函数调用的一般形式为：

```
函数名([实际参数列表])
```

7.1.2 函数的调用

函数名([实际参数列表])

例：编程实现输出如下内容，要求使用函数。

```
def print_line():
    print("**********")
def print_text():
    print("Hello world!")
print_line()                    #调用函数
print_line()                    #调用函数
print_text()                    #调用函数
print_line()                    #调用函数
print_line()                    #调用函数
```

```
**********
**********
Hello world!
**********
**********
```

7.1.3 函数的返回值

函数并非总是直接输出数据，它还可以处理一些数据，并返回一个或一组值。

函数返回的值被称为返回值。在Python中，函数使用return语句返回值。

◆return语句用来退出函数并将程序返回到函数被调用的位置继续执行。

◆return语句可以同时返回0个、1个或多个结果给函数被调用处的变量。

7.1.3 函数的返回值

例：编写函数实现求两数中较小数。

```
def minimal(x, y):                          #自定义计算较小值函数
    if x > y:                               #如果x>y成立，返回y的值
        return y
    else:                                   #否则返回x的值
        return x
a = float(input('输入第一个数据：'))          #显示提示语并接收a的值
b = float(input('输入第二个数据：'))          #显示提示语并接收b的值
c = minimal(a, b)                           #调用函数，将较小值赋给c
print('较小值为：',c)                        #输出c的值
```

7.1.3 函数的返回值

注意

如果函数没有return语句，Python将认为该函数以return None结束，即返回空值。函数也可以用return语句返回多个值，多个值以元组类型保存。

7.1.3 函数的返回值

例：编写程序，要求输入字符串，计算字符串中大写字母和小写字母的个数并输出，计算过程用函数实现。

```
def demo(s):                          #定义函数
    a = 0                             #变量a用于存储大写字母个数
    b = 0                             #变量b用于存储小写字母个数
    for ch in s:                      #循环判断字符串中的每个字母
        if ch.isupper():              #调用isupper()方法判断是否为大写字母
            a += 1                    #如果是a加1
        elif ch.islower():            #调用islower()方法判断是否为小写字母
            b += 1                    #如果是b加1
    return a,b          #返回a和b的值
s = input('请输入字符串')  #输入字符串
c = demo(s)             #调用函数返回a和b的值给变量c
print(c,type(c))                      #输出变量c及变量c的类型
print('大写字母的个数为：',c[0],'，小写字母的个数为：',c[1])#输出结果
```

7.2 函数的参数

- 7.2.1 实参和形参
- 7.2.2 参数类型

7.2.1 实参和形参

定义函数时，圆括号内是使用逗号分隔的形式参数列表（简称形参），调用函数时向其传递实参，根据不同的参数类型，将实参的值或引用传递给形参。

1．传值

当参数类型为固定数据类型（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。

例：阅读以下程序，分析输出结果。

```
def ChangeInt(a):
    a = 10
a = 2
ChangeInt(a)
print('a =',a)
```

7.2.1 实参和形参

▶ 2．传引用

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时，修改后的结果是可以反映到函数之外的，即实参也会得到相应的修改。

例：阅读以下程序，分析输出结果。

```
def changeme(mylist):
    mylist.append([1,2,3,4])
    print("函数内取值: ", mylist)
    return
mylist = [10,20,30]
changeme(mylist)
print ("函数外取值: ",mylist)
```

7.2.2 参数类型

3．位置参数

注意：调用函数时，实参和形参的顺序必须严格一致，并且实参和形参的数量必须相同。

例：运行以下程序，分析运行结果。

```
def printme(a, b, c):
    print (a, b, c)
printme(1, 2, 3)
printme(1, 2)
```

7.2.2 参数类型

4．默认值参数

在定义函数时，可以为函数的参数设置默认值，这个参数被称为默认值参数。带有默认值参数的函数定义语法如下：

```
def 函数名(...,形参名=默认值):
    函数体
```

注意：（1）在调用带有默认值参数的函数时，可以不用为设置了默认值的形参进行传值，此时函数将会直接使用函数定义时设置的默认值，也可以通过显式赋值来替换其默认值。（2）默认值参数必修放在参数列表的末尾

知识库

可以使用“函数名.__defaults__”随时查看函数所有默认值参数的当前值，其返回值为一个元组，其中的元素依次表示每个默认值参数的当前值。

7.2.2 参数类型

例：默认值参数。

```
def printinfo( name, age = 35 ):#定义函数，打印任何传入的字符串
    print ("名字: ", name)
    print ("年龄: ", age)
    return
#调用printinfo函数
print(printinfo.__defaults__)              #输出函数默认值参数
printinfo("root" ,50)                  #显式赋值
print ("------------------------")
printinfo("root" )                     #使用默认值参数
```

```
(35,)
名字:  root
年龄:  50
------------------------
名字:  root
年龄:  35
[Finished in 0.3s]
```

9 lines, 269 characters selected

7.2.2 参数类型

提示

在定义带有默认值参数的函数时，默认值参数必须出现在函数形参列表的最右端，否则会提示语法错误。

例如：以下函数的定义是错误的：

```
def f(a = 1, b, c = 3):
    print(a, b, c)
```

注意:多次调用函数并不为默认值参数传递值时，默认值参数只在定义时进行一次解释和初始化，对于列表、字典这样可变类型的默认值参数，这一点可能会导致逻辑错误。因此，一般来说，要避免使用列表、字典、集合或其他可变数据类型作为函数参数的默认值。

7.2.2 参数类型

例：阅读以下程序，分析运行结

```
def demo(newitem,old_list = []):
    old_list.append(newitem)
    return old_list
print(demo('5',[1,2,3,4]))
print(demo('a'))
print(demo('b'))
```

```
[1, 2, 3, 4, '5']
['a']
['a', 'b']
[Finished in 0.2s]
```

Line 6, Column 17 Tab Size: 4

7.2.2 参数类型

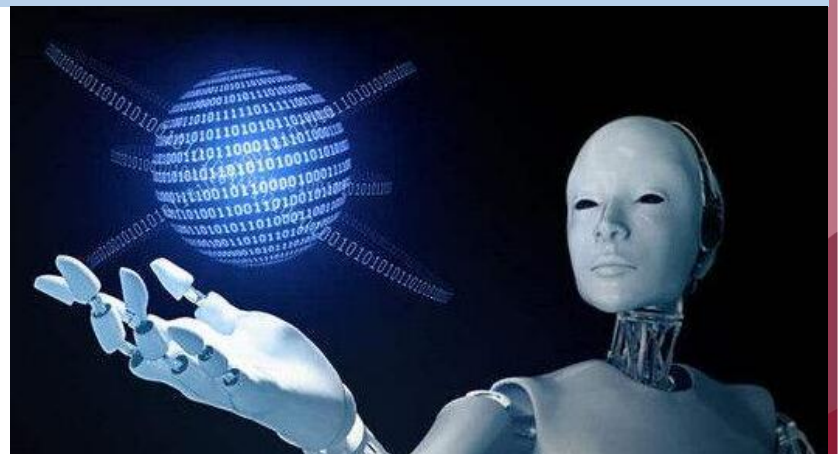
注意:如果在定义函数时某个参数的默认值为一个变量，那么参数的默认值只依赖于函数定义时该变量的值。

例：运行以下程序，分析运行结果。

```
a = 1
def f(n = a):
    print(n)
a = 5
f()
```

```
1
[Finished in 0.2s]
```

5 lines, 38 characters selected Tab



7.2.2 参数类型

2．关键字参数

关键字参数是指调用函数时的参数传递方式，是一种按参数名字传递值的方式。使用关键字参数允许函数调用时参数的顺序与定义时不一致，Python解释器能够用参数名匹配参数值。

例：关键字参数。

```
def printinfo(name, age):
    print ("名字: ", name)
    print ("年龄: ", age)
    return
#调用printinfo函数
printinfo(age=50, name="runoob")
```

```
名字:  runoob
年龄:  50
[Finished in 0.2s]
```

Line 6, Column 49

7.2.2 参数类型

▶ 3．不定长参数

通常在定义一个函数时，若希望函数能够处理比定义时更多的参数，此时可以

```
def 函数名([形参列表,] *args, **kwargs):
    函数体
```

◆*args 用来接收任意多个实参并将其放在一个元组中

◆**kwargs用来接收类似于关键字参数一样显式赋值形式的多个实参并将其放入字典中

例：不定长参数。

```
def f(a, b, *args, **kwargs):
    print(a)
    print(b)
    print(args)
    print(kwargs)
f(1, 2, 3, 4, 5, x = 6, z = 7)
```

7.3 函数的嵌套

7.3 函数的嵌套

1．嵌套定义

Python允许函数的嵌套定义，即在函数内部可以再定义另外一个函数。

例如： 以下代码就是在print_line()函数中嵌套定义了print_text()函数

```
def print_line():                          #定义print_line()函数
    print("**********")
    def print_text():                      #嵌套定义print_text()函数
        print("Hello world!")
    print_text()                           #调用print_text()函数
print_line()                          #调用print_line()函数
```

7.3 函数的嵌套

1．嵌套定义

注意：在函数内部可以再定义另外一个函数，该内部函数为局部函数，在外层函数外不能调用该函数。

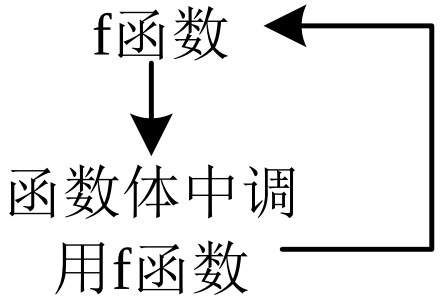
例如： 以下代码就是在print_line()函数中嵌套定义了print_text()函数

```
def print_line():                          #定义print_line()函数
    print("**********")
    def print_text():                      #嵌套定义print_text()函数
        print("Hello world!")
    print_text()                           #调用print_text()函数
print_line()                          #调用print_line()函数
print_text()                          #该调用print_text()函数为错误
```

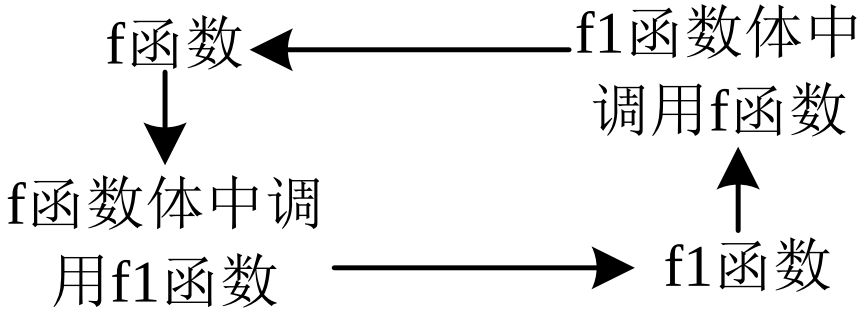
7.4 递归函数

7.4 递归函数

Python支持函数的递归调用，所谓递归就是函数直接或间接地调用其本身。



直接递归调用



间接递归调用

提示

这两种递归调用都是无休止地调用自身。因此，为了防止无限递归，所有递归函数都需要设定终止条件。

7.4 递归函数

例：计算n的阶乘。

```
def f(n):                                    #定义递归函数
    if n==1:                                 #当n等于1时返回1
        return 1
    else:                                    #当n不为1是返回f(n-1)*n
        return f(n-1)*n
n = int(input('请输入一个正整数：'))          #输入一个整数
print(n,'的阶乘结果为：',f(n))                #调用函数f并输出结果
```

程序运行效果

7.5 变量作用域

- 7.5.1 局部变量和全局变量
- 7.5.2 global和nonlocal关键字

7.5.1 局部变量和全局变量

- 所谓局部变量，指的是定义在函数内的变量，只能在函数内使用，它与函数外具有相同名称的其他变量没有任何关系。不同函数中，可以使用相同名字的局部变量，它们代表不同对象，互不干扰。此外，函数的形式参数也属于局部变量，作用范围仅限于函数内部。
- 在函数之外定义的变量称为全局变量，全局变量在整个程序范围内有效。

7.5.1 局部变量和全局变量

例：分析以下程序的运行结果。

```
total = 0
def sum( arg1, arg2 ):
    total = arg1 + arg2
    print ("函数内是局部变量 : ", total)
sum(10, 20)
print ("函数外是全局变量 : ", total)
```

```
函数内是局部变量 :  30
函数外是全局变量 :  0
[Finished in 0.2s]
```

7.5.2 global和nonlocal关键字

当内部作用域想要使用外部作用域的变量时，可使用global和nonlocal关键字。

1．global关键字

在函数内部修改一个定义在函数外的变量时，须使用global关键字明确声明变量。在函数内部通过global关键字来声明或定义全局变量，可分为两种情况：

（1）一个变量已在函数外定义，如果在函数内需要使用该变量的值或修改该变量的值，并将修改结果反映到函数外，可以在函数内用关键字global明确声明该全局变量。

（2）在函数内部直接使用global关键字将一个变量声明为全局变量，如果在函数外没有定义该全局变量，在调用该函数后，会创建新的全局变量。

7.5.2 global和nonlocal关键字

例：分析以下程序的运行结果。

```
num = 1
def fun():
    global num
    num += 1
    print('函数内num的值为',num)
fun()
print('函数外num的值为',num)
```

```
函数内num的值为 2
函数外num的值为 2
[Finished in 0.2s]
```

7.5.2 global和nonlocal关键字

2．nonlocal关键字

如果要在一个嵌套的函数中修改嵌套作用域中的变量，则须使用nonlocal关键字。

例：分析以下程序的运行结果。

```
def outer():
    num = 1
    def inner():
        nonlocal num
        num = 2
        print(' inner函数中num的值为',num)
    inner()
    print(' outer函数中num的值为',num)
outer()
```

程序运行效果

```
 inner函数中num的值为 2
 outer函数中num的值为 2
[Finished in 0.3s]
```

Line 10, Column 1; Build finished

```
 inner函数中num的值为 2
 outer函数中num的值为 1
[Finished in 0.2s]
```

Line 3, Column 17 Tab Size: 4

7.6 匿名函数

7.6 匿名函数

关键字lambda用于定义一种特殊的函数——匿名函数，又称lambda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 [参数列表]:表达式
```

简单地说，lambda函数用于定义简单的、能够在一行内表示的函数，返回一个函数类型。

例如：

```
>>>sum = lambda arg1, arg2: arg1 + arg2          #定义lambda函数
>>>print('相加后的值为:', sum(10, 20))          #调用sum函数
相加后的值为: 30
```

lambda函数常用在临时需要一个类似于函数的功能，但又不想定义函数的场合。例如，作为内置函数sorted()和列表方法sort()的key参数。

7.6 匿名函数

例：给定多位学生信息（包括学号、姓名和成绩），按成绩排序后输出。

```
stu = [
    {'num':'201801','name':'Wangwu','score':89},
    {'num':'201802','name':'Liujun','score':95},
    {'num':'201803','name':'Limeng','score':85}]                    #定义学生信息
stu.sort(key = lambda x:x['score'])                                 #按成绩排序
for s in stu:
    print('学号:',s['num'],'姓名:',s['name'],'成绩:',s['score'])    #输出列表
```

```
学号: 201803 姓名: Limeng 成绩: 85
学号: 201801 姓名: Wangwu 成绩: 89
学号: 201802 姓名: Liujun 成绩: 95
```

7.7 模块

- 7.7.1 模块的导入
- 7.7.2 模块的创建
- 7.7.3 __name__属性
- 7.7.4 包

7.7.1 模块的导入

1．导入整个模块

使用import导入整个模块的基本格式如下：

```
import 模块名 [as 别名]
```

使用这种方式导入模块后，调用模块中的函数时需要在函数名前加上模块名作为前缀：

```
模块名.函数名
```

```
>>>import math   #导入标准库math
>>>math.sqrt(4)   #求4的开方
2.0
```

提示

当模块名字很长时，可以使用语句“import 模块名 as 别名”为导入的模块设置别名，然后使用“别名.函数名”的方式调用函数。

```
>>>import random as r
>>>r.randint(1,100)
47
```

7.7.1 模块的导入

▶ 2．导入特定的函数

当我们只需要用到模块中的某个函数时，可只导入特定的函数。

```
from 模块名 import 函数名 [as 别名]
```

在调用函数时，不需要使用模块名作为前缀。

例如：

```
>>>from math import sqrt                    #只导入模块中的指定函数
>>>sqrt(9)                                  #调用函数，求9的开方
3.0
>>>from random import randint as r#给导入的函数指定别名r
>>>r(1,10)                                  #调用函数，获得[1,10]区间的随机整数
10
```

7.7.1 模块的导入

▶ 3．导入模块中所有函数

使用星号“*”可以导入模块中的所有内容（包括函数和变量等）。

```
from 模块名 import *
```

这是“导入特定的函数”用法的一种极端情况，可以一次导入模块中的所有内容。

例如：

```
>>>from math import *          #导入标准库math中的所有内容
>>>pi                          #常数π
>>>log2(8)                     #计算以2为底的对数值
>>>sqrt(16)                    #计算16的开方
```

7.7.2 模块的创建

在Python中，每个Python文件都可以作为一个模块，模块的名字就是文件名。

例如： 现有一个文件“test.py”，其中定义了求较小值的函数minimal()。

```
def minimal(x, y):      #自定义计算较小值函数
    if x > y:       #如果x>y成立，返回y的值
        return y
    else:                   #否则返回x的值
        return x
```

例：编写程序，通过调用上述“test”模块中的minimal()函数，计算两数中的较小值。

```
import test                                  #导入test模块
a = float(input('输入第一个数据：'))
b = float(input('输入第二个数据：'))
c = test.minimal(a, b)
print('较小值为：',c)
```

7.7.3 __name__属性

在实际开发过程中，开发人员为了让模块能够在项目中达到想要的效果，会自行在模块中添加测试信息。

例如：在“test_1.py”文件中添加测试代码如下：

```
def minimal(x, y):          #自定义计算较小值函数
    if x > y:               #如果x>y成立，返回y的值
        return y
    else:                   #否则返回x的值
        return x
#用来测试
r = minimal(2,3)
print('测试2和3的较小值为:',r)
```

例：运行以下程序，分析运行结果。

```
import test_1          #导入test_1模块
c = test_1.minimal(5,6)
print('较小值为：',c)
```

```
测试2和3的较小值为: 2
较小值为：  5
[Finished in 0.2s]
```

7.7.3 __name__属性

Python提供了一个__name__属性，通过__name__属性可以识别程序的使用方式：

- 当作为模块导入时，则其__name__属性的值被自动设置为模块名；
- 如果作为程序直接运行，则其__name__属性被自动设置为字符串“__main__”。

例如：

```
def minimal(x, y):
    if x > y:
        return y
    else:
        return x
#用来测试
if __name__ == '__main__':                    #识别当前的运行方式
    r = minimal(2,3)
    print('测试2和3的较小值为:',r)
```

7.7.4 包

为了组织好模块，通常会将多个模块放在一个目录中。我们将Python模块文件所在的目录称为包，且该目录下必须存在__init__.py文件（文件内容可以为空）。

例如：有如下包结构：

```
package_1
|—— __init__.py
|—— module_1.py
|—— module_2py
|—— module_3py
main.py
```

如果main.py想要调用package_1包中模块module_1中的f()函数：

```
import package_1.module_1     #导入模块
package_1.module_1.f()        #调用函数
#也可以用以下语句实现
from package_1 import module_1     #导入模块
module_1.f()                  #调用函数
```

7.8 典型案例

典型案例

编写程序验

```
def prime(i):                    #定义函数，判断i是否为素数
    if i<=1:                     #如果小于等于1，返回0（i不是素数）
        return 0
    if i==2:                     #如果等于2，返回1（i是素数）
        return 1
    for j in range(2,i):         #判断i是否为素数
        if i%j==0:               #i可以被j除尽，余数为0
            return 0             #返回0，i不是素数
        elif i!=j+1:             #如果i不等于j+1，继续
            continue
        else:
            return 1             #否则，i等于j+1，返回1（i是素数）
n=0
for i in range(6,21,2):
    k=2
    while k<=i/2:
        j=i-k
        flag1=prime(k)           #调用prime函数
        if flag1:                #如果k为素数
            flag2=prime(j)       #调用prime函数
            if flag2:            #如果k和j都是素数
                print(i,'=',k,'+',j)   #输出结果
                n+=1
        k=k+1
```

程序运行效果

```
6 = 3 + 3
8 = 3 + 5
10 = 3 + 7
10 = 5 + 5
12 = 5 + 7
14 = 3 + 11
14 = 7 + 7
16 = 3 + 13
16 = 5 + 11
18 = 5 + 13
18 = 7 + 11
20 = 3 + 17
20 = 7 + 13
[Finished in 0.2s]
```

Line 13, Column 5 Tab Size: 4

案例

```
stuInfos=[]                                    #用来保存学生的所有
def printMenu():                               #打印功能提示
    print("="*20)
    print(" 学生管理系统V1.0 ")
    print("1.添加学生信息")
    print("2.删除学生信息")
    print("3.显示所有学生信息")
    print("0.退出系统")
    print("="*20)
#添加一个学生信息
def addStuInfo():
    newNum = input("请输入新学生的学号：")          #提示并获
    newName = input("请输入新学生的姓名：")         #提示并获
    newSex = input("请输入新学生的性别（男/女）：")  #提示并获
    newInfo = {}                                #定义字典
    #赋值
    newInfo['num'] = newNum
    newInfo['name'] = newName
    newInfo['sex'] = newSex
    stuInfos.append(newInfo)                    #将元素添
#删除一个学生信息
def delStuInfo(student):
    del_num = input("请输入要删除的学生的学号：")    #提示并获
    for stu in student:                         #遍历列表
        if stu['num'] == del_num:               #判断是否
            student.remove(stu)                 #删除该学
```

```
#显示所有学生信息
def showStuInfo():
    print("=" * 20)
    print("学生的信息如下:")
    print("=" * 20)
    print("序号    学号    姓名    性别")
    i = 1
    #遍历存储学生信息的列表，输出每个学生的详细信息
    for tempInfo in stuInfos:
        print("%d      %s      %s      %s" % (i, tempInfo['num'],tempInfo['name'],
tempInfo['sex']))
        i += 1
#main函数控制整个程序的流程
def main():
    while True:
        printMenu()                                 #打印功能菜单
        key = input("请输入功能对应的数字")          #获取用户输入
        if key == '1':                              #添加学生信息
            addStuInfo()
        if key == '2':                              #删除学生信息
            delStuInfo(stuInfos)
        elif key == '3':                            #显示学生信息
            showStuInfo()
        elif key == '0':                            #退出循环
            quit_con = input("确定退出吗？（Yes or No）：")
            if quit_con == 'Yes':
                break
main()                                              #调用main函数
```

7.8.2 学生管理系统

程序运行效果

感谢您的观看