第5章 函　数

5.1 学习要求

(1) 掌握 Python 中自定义函数的使用方法。

(2) 掌握 Python 中常见内置库函数的使用方法。

5.2 知识要点

5.2.1 使用函数的优点

函数是一组实现某一特定功能的语句集合,是可以重复调用、功能相对独立完整的程序段。使用函数的优点如下:

(1) 程序结构清晰,可读性好。

(2) 减少重复编码的工作量。

(3) 可多人共同编制一个大程序,缩短程序设计周期,提高程序设计和调试的效率。

5.2.2 函数的分类

1. 从用户的使用角度

(1) 库函数(标准函数):由系统提供,在程序前导入该函数原型所在的模块。

使用库函数应注意的事项:函数功能;函数参数的数目和顺序;各参数的意义和类型;函数返回值的意义和类型。

(2) 用户自定义函数:按照用户的需求定义一个函数。

2. 从参数传递的角度

1) 有参函数

下面定义的有参函数 average()有 3 个参数 x,y,z,因此调用时需要传递 3 个实参 a,b,c。

```
def average(x,y,z):
    aver = (x + y + z)/3
    return(aver)
a,b,c = eval(input("please input a,b,c:"))
ave = average(a,b,c)
```

```
print("average = %f" %ave)
```

2）无参函数

下面定义的 printstar()和 print_message()都是无参函数，调用时不需要传递实参。

```
def printstar():
    print("**********")
def print_message():
    print("How are you!")
def main():
    printstar()
    print_message()
    printstar()
main()
```

5.2.3 函数的定义与调用

1. 函数定义的一般形式

```
def 函数名([形式参数表]):
    函数体
    [return 表达式]
```

函数定义时要注意：

（1）采用 def 关键字定义函数，不需要指定返回值的类型。

（2）函数的参数不限，不需要指定参数类型。

（3）参数括号后面的冒号“:”必不可少。

（4）函数体相对于 def 关键字必须保持一定的空格缩进。

（5）return 语句是可选的。

2. 函数的调用的一般形式

```
函数名([实际参数表])
```

说明：

（1）实参可以是常量、变量、表达式、函数等，但在进行函数调用时必须有确定的值。

（2）函数的实参和形参应在个数、类型和顺序上一一对应。

（3）对于无参函数，调用时实参表列为空，但()不能省。

【例 5-1】 编写函数，求 3 个数中的最大值。

```
def getMax(a,b,c):
     if a > b:
          max = a
     else:
          max  = b
     if(c > max):
          max  = c
     return max
a,b,c = eval(input("input a,b,c:"))
n = getMax (a,b,c)
```

```
print("max = ",n)
```

注意,在 Python 中不允许前向引用,即在函数定义之前,不允许调用该函数。

5.2.4 函数的参数和传递方式

形式参数:即形参,定义函数时函数名后面括号中的变量名。

实际参数:即实参,调用函数时函数名后面括号中对应的参数。

说明:

(1) 实参可以是常量、变量和表达式,但必须在函数调用之前有确定的值。

(2) 形参与实参个数相同。

(3) 形参定义时编译系统并不为其分配存储空间,也无初值;只有在函数调用时,临时分配存储空间,接收来自实参的值;函数调用结束,内存空间释放。

1. 单向的值传递

实参和形参之间是单向的值传递。在函数调用时,将各实参表达式的值计算出来,赋给形参变量。因此,实参与形参必须类型相同或赋值兼容,个数相等,一一对应。在函数调用中,即使实参为变量,形参值的改变也不会改变实参变量的值。实参和形参占用不同的内存单元。

【例 5-2】 编写一个程序,将主函数中的两个变量的值传递给 swap()函数中的两个形参,交换两个形参的值。

```
def swap(a,b):
    a,b = b,a
    print("a = ",a,"b = ",b)
x,y = eval(input("input x,y:"))
swap(x,y)
print("x = ",x,"y = ",y)
```

运行结果:

```
input x,y:3,5
a = 5 b = 3
x = 3 y = 5
```

2. 传地址方式

【例 5-3】 函数调用时,将实参数据的存储地址作为参数传递给形参。

```
def swap(a_list):
    a_list[0],a_list[1] = a_list[1],a_list[0]
    print("a_list[0] = ",a_list[0],"a_list[1] = ",a_list[1])
x_list = [3,5]
swap(x_list)
print("x_list[0] = ",x_list[0],"x_list[1] = ",x_list[1])
```

运行结果:

```
a_list[0] = 5 a_list[1] = 3
x_list[0] = 5 x_list[1] = 3
```

3. 位置参数、关键字参数、默认值参数、可变参数的区别

位置参数，即按出现的位置进行普通形式的传参。例如：

```
def f(a,b):                    # 实参 3 传给形参 a,实参 4 传给形参 b
    c = a+b
    return c
ret = f(3,4)
```

关键字参数通过“=”明确指定将某个实参传递给某个形参。例如：

```
def f(a,b,c)
    e = a+b+c
f(c=1,b=2,a=3)                 # 实参不再按照位置对应传参
```

默认值参数：定义时直接对形参赋值，如果函数调用时没有对应的实参，则使用默认值，否则仍使用实参值。如果同时有位置参数，默认值参数必须出现在形参表的右端。

可变参数：当需要传入多个参数时，可以用 * args 代表多个参数，不用分别在括号里指定多个参数。

(1) 可变参数可以输入任何类型的数据，数据和数据直接用逗号隔开。

(2) 接收的实参会组合成元组。

例如：

```
def greet( * names):
    print(names)
>>> greet()                    # 没有参数,返回空元组
()
>>> greet('Jordan', 'James', 'Kobe')
('Jordan', 'James', 'Kobe')
```

** kwargs：当需要传入键值对类型的参数时就可以用 ** kwargs。

例如：

```
def greet( ** all_star):
    print(all_star)
>>> greet()                    # 没有参数,返回空字典
{}
>>> greet(name = 'James', age = 18)
{'name': 'James', 'age': 18}
```

5.2.5 函数的返回

指函数被调用、执行完后，返回给主调函数的值。

函数的返回语句的一般形式：

```
return  表达式
```

功能：使程序控制从被调用函数返回到调用函数中，同时把返回值带给调用函数。

说明：

(1) 函数内可有多条返回语句。

(2) 如果没有 return 语句,会自动返回 None; 如果有 return 语句,但是 return 后面没有表达式也返回 None。

【例 5-4】 编写函数,判断一个数是否是素数。

```
def isprime(n):
    for i in range(2,n):
        if(n % i == 0):
            return 0
    return 1
m = int(input("请输入一个整数:"))
flag = isprime(m)
if(flag == 1):
    print(" % d 是素数" % m)
else:
    print(" % d 不是素数" % m)
```

【例 5-5】 求一个数列中的最大值和最小值。

```
def getMaxMin(x):
    max = x[0]
    min = x[0]
    for i in range(0, len(x)):
        if max < x[i]:
            max = x[i]
        if min > x[i]:
            min = x[i]
    return (max,min)
a_list = [ - 1,28, - 15,5, 10 ]          # 测试数据为列表类型
x,y  = getMaxMin(a_list)
print("a_list = ", a_list)
print("最大元素 = ",x, "最小元素 = ",y)
```

5.2.6 函数的递归调用

在函数的执行过程中可以直接或间接调用该函数本身。

1. 直接递归调用

在函数中直接调用函数本身,如图 5-1 所示。

2. 间接递归调用

在函数中调用其他函数,其他函数又调用原函数,如图 5-2 所示。

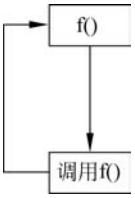

图 5-1 直接递归调用

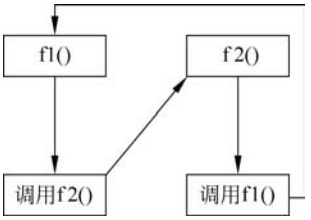

图 5-2 间接递归调用

递归算法有两个基本特征。

(1) 递推归纳：将问题转换为比原问题小的同类规模，归纳出一般递推公式，故所处理的对象要有规律地递增或递减。

(2) 递归终止：当规模小到一定程度时应结束递归调用，逐层返回，常用条件语句来控制何时结束递归。

【例 5-6】 用递归方法求 n 的阶乘。

$$n!=\begin{cases}1 & n=0,1\\ n*(n-1)! & n>1\end{cases}$$

```
def fac(n):
    if  n == 0:
        f = 1
    else:
        f = fac(n - 1) * n;
    return f
n = int(input("please input n: "))
f = fac(n)
print(" % d!= % d" % (n,f))
```

分析：

递推归纳：$n!\rightarrow(n-1)!\rightarrow(n-2)!\rightarrow\cdots\rightarrow2!\rightarrow1!$

递归终止：$n=0$ 时，$0!=1$

执行过程(两个阶段)：

第一阶段：逐层调用，调用函数自身。

第二阶段：逐层返回，返回到调用该层的位置。

递归调用是多重嵌套调用的一种特殊情况。

设计递归算法的前提如下所述。

(1) 原问题可以层层分解为类似的子问题，且子问题比原问题规模更小。

(2) 规模最小的问题具有直接解。

方法如下所述。

(1) 寻找分解方法：将原问题转换为子问题求解，例如，$n!=n*(n-1)!$。

(2) 设计递归出口：根据规模最小的子问题确定递归终止条件，例如，求解 $n!$，当 $n=0$ 时，$n!=1$。

5.2.7 变量的作用域

当程序中有多个函数时，定义的每个变量只能在一定的范围内访问，称之为变量的作用域。按作用域划分，将变量分为局部变量和全局变量。

1. 局部变量

在一个函数内或者语句块内定义的变量称为局部变量。局部变量的作用域仅限于定义它的函数体或语句块中。不同函数可以定义同名的局部变量，虽然同名，但却代表不同的变量。

```
def fun1(a):
```

```
    x = a + 10                                    # x 为局部变量
    …
def fun2(a,b):
    x,y = a,b                                     # x,y 为局部变量
    …
```

2. 全局变量

在所有函数之外定义的变量称为全局变量,它可以在多个函数中被引用。

```
x = 30
def func():
    global x                                      # 定义 x 为全局变量
    print('x 的值是', x)                          # x 的值是 30
    x = 20
    print('全局变量 x 改为', x)                   # x 的值是 20
func()
print('x 的值是', x)                              # x 的值是 20
```

5.2.8 模块

将一些常用的功能单独放置到一个文件中,方便其他文件来调用,这些文件即为模块。

从用户的角度看,模块也分为标准库模块和用户自定义模块。

标准库模块:Python 自带的函数模块,包括文本处理、文件处理、操作系统功能、网络通信、网络协议等。

用户自定义模块:用户建立的一个模块,就是建立扩展名为.py 的 Python 程序。

```
def printer(x):
    print(x)
```

将以上程序代码保存成.py 程序,例如 module.py。

导入模块:给出一个访问模块提供的函数、对象和类的方法。

(1) 引入模块。

```
import 模块
```

(2) 引入模块中的函数。

```
from 模块名 import 函数名
```

(3) 引入模块中的所有函数。

```
from 模块名 import *
```

5.2.9 匿名函数、enumerate()函数、zip()函数

对于只有一条表达式语句的函数,可以用关键字 lambda 将其定义为匿名函数(anonymous functions),使得程序简洁,提高可读性。匿名函数定义形式如下:

```
lambda [参数列表]:表达式
```

匿名函数没有函数名，参数可有可无，有参的匿名函数参数个数任意。但是作为函数体的表达式限定为仅能包含一条表达式语句，因此只能表达有限的逻辑。这条表达式的运行结果就作为函数的值返回。

```
s = lambda : "python".upper()          # 定义无参匿名函数,将字母改成大写
f = lambda x : x * 10                  # 定义有参匿名函数,将数字扩大 10 倍
print(s())                             # 调用无参匿名函数,注意要加一对()
print(f(7.5))                          # 调用有参匿名函数,传入参数
```

【例 5-7】 把匿名函数作为参数传递的使用方法。

```
points = [(1,7),(3,4),(5,6)]
# 调用 sort()按元素第二列进行升序排序
points.sort(key = lambda point: point[1])
print(points)
```

运行结果：

```
[(3, 4), (5, 6), (1, 7)]
```

【例 5-8】 匿名函数的其他使用方法。

```
>>> f = lambda x,y,z:x + y + z          # 可以给 lambda 表达式起名字
>>> f(1,2,3)                            # 像函数一样调用
6
>>> g = lambda x, y = 2,z = 3: x + y + z  # 默认值参数
>>> g(1)
6
>>> g(2, z = 4, y = 5)                  # 关键字参数
11
>>> L = [(lambda x: x ** 2), (lambda x: x ** 3), (lambda x: x ** 4)]
>>> print(L[0](2),L[1](2),L[2](2))
4 8 16
>>> D = {'f1':(lambda:2 + 3), 'f2':(lambda:2 * 3),
         'f3':(lambda:2 ** 3)}
>>> print(D['f1'](), D['f2'](), D['f3']())
5 6 8
```

【例 5-9】 匿名函数在 sorted()、sort()中的应用。

```
list1 = [('m',12),('b',23),('c',3)]
print(sorted(list1))
print(sorted(list1,key = lambda x:x[1],reverse = True))
dic1 = {'m':12,'b':23,'c':3}
print(sorted(dic1))
print(sorted(dic1.items(),key = lambda x:x[1],reverse = True))
d1 = [
    {'name': 'alice', 'age': 38},
    {'name': 'bob', 'age': 18},
    {'name': 'ctrl', 'age': 28}
     ]
d1.sort(key = lambda x: x['age'])
```

```
print(d1)
```

运行结果：

```
[('b', 23), ('c', 3), ('m', 12)]                    # 按照第一个元素升序
[('b', 23), ('m', 12), ('c', 3)]                    # 按照第二个元素降序
['b', 'c', 'm']                                     # 按照字典的关键字升序
[('b', 23), ('m', 12), ('c', 3)]                    # 按照字典的值降序
[{'name': 'bob', 'age': 18}, {'name': 'ctrl', 'age': 28}, {'name': 'alice', 'age': 38}]
                                                    # 按照 age 升序
```

enumerate()函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列。

格式：enumerate (sequence, [start = 0])

参数：

sequence：一个序列、迭代器或其他支持迭代对象。

start：下标起始位置。

【例 5-10】 enumerate()函数举例。

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
print(list(enumerate(seasons)))
print(list(enumerate(seasons, start = 1)))          # 下标从 1 开始
```

运行结果：

```
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

zip()函数用于将可迭代对象对应元素打包成元组，返回由这些元组组成的列表或迭代器，返回列表长度与最短对象相同。

【例 5-11】 zip()函数举例。

```
a = [1,2,3]
b = [4,5,6]
c = [4,5,6,7,8]
print(type(zip(a,b)))                               # <class 'zip'>
print(list(zip(a,b)))                               # [(1,4),(2,5),(3,6)]
print(list(zip(a,c)))                               # [(1,4),(2,5),(3,6)]
```

【例 5-12】 利用 zip()函数实现字典键值互换。

```
d = {'a':1,'b':2,'c':3}
dic2 = dict(zip(d.values(),d.keys()))
print(dic2)                                         # {1: 'a', 2: 'b', 3: 'c'}
```

5.2.10 高阶函数

高阶函数是在 Python 中一个非常有用的功能函数。一个函数可以用来接收另一个函数作为参数，这样的函数叫作高阶函数。

1. map ()函数

map()函数是 Python 内置的高阶函数,它接收一个函数和可迭代对象,并通过把函数依次作用于后者,得到一个迭代器并返回。

【例 5-13】 假设用户输入的英文名字不规范,没有按照首字母大写,可以利用 map()函数进行规范。

```
def format_name(s):
    return s.capitalize()
result = map(format_name, ['adam', 'LISA', 'barT'])
print(list(result))
```

运行结果:

```
['Adam', 'Lisa', 'Bart']
```

【例 5-14】 map()函数接收 lambda 函数。

```
a = [4,5,6,7]
b = map(lambda x:x ** 2,a)
print(list(b))
```

运行结果:

```
[16, 25, 36, 49]
```

2. filter ()函数

filter()函数接收一个函数和可迭代对象,并通过函数依次作用后者上,但是只有函数返回为真时才会保留。

【例 5-15】 filter()函数接收 lambda 函数。

```
a = [4,5,6,7]
b = filter(lambda x:x % 2 == 0,a)
print(list(b))
```

运行结果:

```
[4, 6]
```

可以看出,和 map()函数全部保留相比,filter()函数只会保留符合要求的部分。

3. reduce ()函数

reduce()函数和 map()函数一样,可以接收两个参数:第一个是函数;第二个是可迭代对象 iterable,不同的是 reduce()函数把结果继续和可迭代对象的下一个元素做积累运算。

【例 5-16】 利用 reduce()函数进行累加。

```
from functools import reduce
def f(x, y):
    return x + y
result = reduce(f, [1, 3, 5, 7, 9])
print(result)
```

运行结果:

```
25
```

分析：reduce()函数使用起来比较特殊，为了更好地理解例 5-16，下面详细拆解进行介绍。当调用 reduce(f,[1,3,5,7,9])时，reduce()函数将做如下计算。

由于 f()函数的功能是计算两个元素的值，因此先计算前两个元素 f(1,3)，结果为 4；

再把结果和第 3 个元素计算 f(4,5)，结果为 9；

再把结果和第 4 个元素计算 f(9,7)，结果为 16；

再把结果和第 5 个元素计算 f(16,9)，结果为 25；

由于没有更多的元素了，计算结束，返回结果 25。

reduce()还可以接收第 3 个可选参数，作为计算的初始值。如果把初始值设为 500，计算 reduce(f,[1,3,5,7,9],500)结果将变为 525，因为第一轮计算是计算初始值和第一个元素 f(500,1)，结果为 501。

视频讲解

5.3 应 用 举 例

【例 5-17】 利用函数采取插入排序法将 10 个数据从小到大进行排序。

```
def insert_sort(array):
    for i in range(1, len(array)):
        if array[i - 1] > array[i]:
            temp = array[i]
            index = i
            while index > 0 and array[index - 1] > temp:
                array[index] = array[index - 1]
                index -= 1
            array[index] = temp
b = input("请输入一组用逗号分隔的数据: ")
array = [ ]
for i in b.split(','):
    array.append(int(i))
print("排序前的数据: ")
print(array)
insert_sort(array)
print("排序后的数据: ")
print(array)
```

习　　题

一、基础题

1. 编写函数，计算圆的面积。

2. 编写函数，计算传入字符串中数字、字母、空格以及其他类型字符的个数。

3. 编写函数，判断用户传入的对象(字符串、列表、元组)长度是否大于 5。

4. 编写函数，检查传入列表的长度，如果大于 2，那么仅保留前两个长度的内容，并将新内容返回给调用者。

5. 编写函数，检查传入字典的每一个 value 的长度，如果大于 2，那么仅保留前两个长度的内容，并将新内容返回给调用者。

6. 编写函数，检查获取传入列表或元组对象的所有奇数位索引对应的元素，并将其作为新列表返回给调用者。

二、提高题

1. 编写函数，接收任意多个实数，返回一个元组，其中第一个元素为所有参数的平均值，其他元素为所有参数中大于平均值的实数。

2. 编写函数，接收字符串参数，返回一个列表，其中第一个元素为大写字母个数，第二个元素为小写字母个数。

3. 编写程序，将一个由键盘输入的十进制数转换为十六进制数并输出，进制转换设计成函数形式。