第3章 函 数

一段程序代码通常可以实现某一特定功能，当需要多次实现同样的这个功能时，都可以使用这同一段代码。但是，如果每次都要重复写相同的代码，不仅让代码看起来很累赘，而且增加了工作量。因此，可以把功能相同的代码封装到一个称为“函数”的功能模块中，当需要实现该功能时，只需要“调用”这个“函数”就可以了。

3.1 函数的定义与调用

函数是组织好的、可重复使用的、用来实现单一或相关联功能的代码段。函数能提高应用的模块性和代码的重复利用率。Python 本身有很多内置函数，如 input()、print()等，可以直接被调用。用户也可以自己创建函数，这种函数称为用户自定义函数。

在 Python 中，自定义函数的语法如下。

```
def 函数名([参数 1,参数 2,参数 3,…]):
    #注释
    函数体
```

说明：在 Python 中，函数定义以关键字 def 开头，空格后接函数标识符名称和圆括号“( )”，后面是一个冒号和换行，然后是函数体。

定义函数时要注意以下几点。

(1) 函数名后面的圆括号和冒号必不可少。用户自定义的函数名，注意不要与 Python 内置函数重名，否则相当于重写内置函数，若不需要，尽量避免重名。

(2) 参数列表是可选的，参数列表是用逗号分隔开的参数，参数个数没有限制，如果参数的个数是 0，意味着是无参函数。

(3) 函数体相对于 def 关键字必须保持一定的空格缩进。

(4) 使用“#注释”来解释该函数的主要功能，是可选的，但推荐使用，方便同行交流。

用户自定义函数后，如果想要完成函数定义的特定任务，可通过在程序中调用该函数实现，调用函数时函数里的代码就会被执行。当然如果想要多次使用同一函数，可以通过多次调用该函数来实现。调用函数的方法非常简单，直接写上函数名加上圆括号即可，但切记函数必须先定义后调用的原则。

例如，编写一个求 1～100 累加和的函数，新建文件 def_sum.py，输入如下代码。

```
def sum():
```

```
    total=0
    for count in range(1, 101):
        total=total+count
    print("sum=",total)
sum()          #调用 sum()函数
```

程序的运行结果如图 3-1 所示。

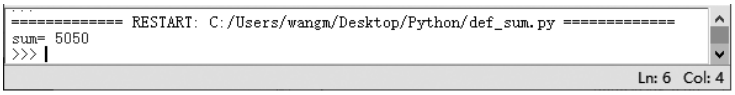

图 3-1　求和函数的运行结果

说明：

(1) 关键字 def 用于告知 Python，要定义一个名字为 sum 的函数。

(2) 紧跟在“def sum():”后面的所有缩进构成了函数体，它实现了从 1 到 100 累加求和。

(3) 最后一行代码调用 sum()函数，当程序运行时，函数 sum()被执行。

(4) 在 def_sum.py 文件中，程序代码可以仅定义函数 sum()而不输入调用语句，在运行程序时再输入调用语句 sum()。修改 def_sum.py 文件中的代码如下。

```
def sum():
    total=0
    for count in range(1, 101):
        total=total+count
    print("sum=",total)
```

运行程序，在命令行中输入 sum()并按回车键执行，结果如图 3-2 所示。

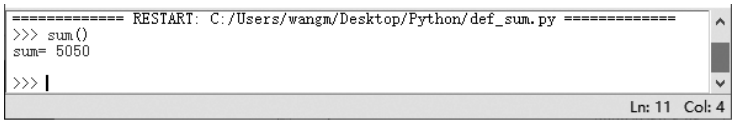

图 3-2　求和函数调用结果

3.2　函数的参数

最初的函数是没有参数概念的，函数只是对相同功能代码的打包，功能单一。例如上述求和函数，如果想分别求解 1～100 之和、100～200 之和，需要定义两个函数，这样代码重复冗余度很高，两个函数功能接近，而它们的区别仅仅是求和的数据不同。因此，可以考虑给函数一个接口，通过接口与外部交流，实现功能接近而又能满足个性化需求的函数，这个接口称为函数参数。

3.2.1　函数的形式参数与实际参数

函数的参数从调用的角度来说，分为形式参数和实际参数，简称为形参和实参。形参是

指定义函数时圆括号里的参数;实参是指函数在被调用的过程中传递过来的参数,即在调用函数时提供的具体的值或者变量。

例如,修改文件 def_sum.py 为更通用的求和函数,让其能够实现从某个数(start)到另外一个数(end)的累加求和,新建文件 sum_parm.py,输入如下代码。

```
def sum1(start,end):
    total=0
    for count in range(start, end+1):
        total=total+count
    print("sum=",total)

sum1(1,100)
sum1(101,200)
```

程序的运行结果如图 3-3 所示。

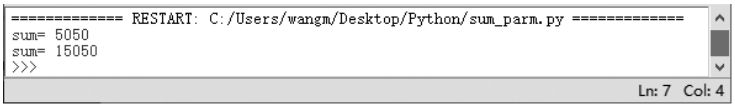

图 3-3 带参数求和函数的运行结果

说明:

(1) 程序中引入两个变量 start 和 end 来分别代表起始值和终止值,并把这两个变量添加到函数定义 def sum1()的括号内,这里的 start 和 end 为形参。当需要实现 1～100 的累加和功能时,调用函数 sum1(1,100);当需要实现 101～200 的累加和功能时,调用函数 sum1(101,200)。其中,1、100、101 和 200 为实参。

(2) 在调用函数 sum1(1,100)时,将实参 1 和 100 传递给函数 sum1(start,end)里的形参 start 和 end,1 和 100 这两个值被存储在形参中,执行求和函数时便输出 1 到 100 的累加和;同理,在调用函数 sum1(101,200)时,将实参 101 和 200 传递给函数 sum1(start,end)里的形参 start 和 end,101 和 200 这两个值被存储在形参中,执行求和函数时便输出 101 到 200 的累加和。

(3) 形参的数据类型取决于调用时输入的实参的数据类型。例如,定义 sum1(start,end)函数时没有指定形参 start 和 end 的数据类型,在调用函数 sum1(1,100)时,两个实参的类型都是数值,所以形参 start 和 end 的数据类型也是数字值类型。

(4) 为方便别人调用,也可以在函数定义时为形参指定数据类型,参考代码如下。

```
def sum1(start:int,end:int):
    total=0
    for count in range(start, end+1):
        total=total+count
    print("sum=",total)

sum1(1,100)
sum1(101,200)
```

3.2.2 函数的参数类型

Python 函数的形参和实参根据参数的类型特点分成 4 种：位置参数(又称必选参数)、默认参数、可变参数(又称不定长参数)、关键字参数。不同类型的参数各有特色，如果使用恰当，不但能处理复杂的参数，还可以简化调用者的代码。

1. 位置参数

位置参数，顾名思义，就是位置固定的参数，也称为必选参数，调用函数时需要一一对应为其赋值，不能不赋值，更不能不按设定类型赋值。

3.2.1 节 sum_parm.py 文件定义的 sum1(start，end)函数中 start 和 end 两个参数就是位置参数。在定义 sum1(start，end)函数时有两个形参 start 和 end，因此在调用函数 sum1(1，100)时也必须有两个实参 1 和 100，如果传入的实参数量多于或少于形参数量，运行程序都会报错。

例如，将调用函数 sum1(1，100)语句改为 sum1(1)，传入实参数量少于形参数量，运行程序就会报错，如图 3-4 所示。

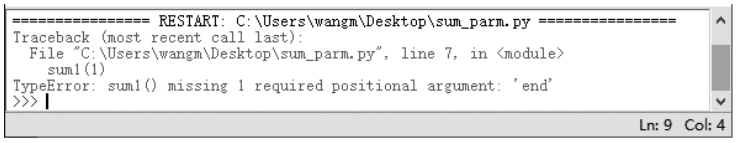

图 3-4 参数报错

同时，需要编程者注意实参的顺序，确保与形参一一对应。当然，编程者也可以通过关键字＝值的方式一一对应，如 sum1(start＝1，end＝100)或者 sum1(end＝100，start＝1)，将实参与形参关联映射，不需要考虑函数调用过程中实参的顺序。

2. 默认参数

有时函数中的某些参数存在默认值，或者为了简化调用过程某些参数省略不写，需要使用默认值运行函数，因此需要使用默认参数。默认参数的用法是直接在定义函数时对其进行赋值，调用时如果没有特殊需要可以不对其再次赋值。特别需要注意的是，如果有位置参数，默认参数必须在位置参数后面声明。

例如，编写一个大学一年级新生注册的函数，新建文件 enroll.py，输入如下代码。

```
def enroll(name, gender):
    print("姓名:", name)
    print("性别:", gender)
```

以上定义的 enroll(name，gender)函数中有 name 和 gender 两个参数，因此调用函数 enroll()时需要传入两个参数。

如果还要继续注册年龄、健康状况，进一步考虑到大学一年级新生大多是 20 岁且健康状况良好，所以可以把年龄和健康状况设为默认参数，添加代码如下。

```
def enroll(name, gender, age=20, health='良好'):
    print("姓名:", name)
    print("性别:", gender)
    print("年龄:", age)
    print("健康状况:", health)
```

这样大多数学生注册时不需要提供年龄和健康状况，只需提供两个必选参数即可。只有与默认参数不符的学生才需要提供额外的信息。例如，李四的年龄 21 岁，王五的健康状态一般。分别运行程序，输入调用函数语句 enroll("张三","男")、enroll("李四","男",21)和 enroll("王五","男",health="一般")，查看执行结果如图 3-5 所示。

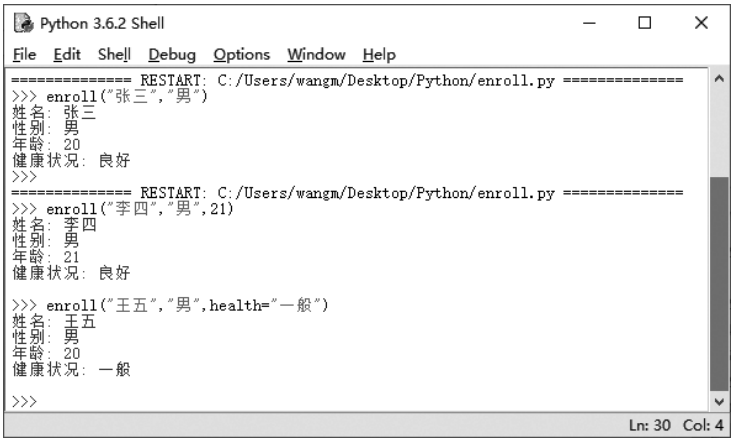

图 3-5　默认参数值执行结果

3. 可变参数

可变参数，顾名思义，就是指传入的参数的个数是可变的，可以是 1 个、2 个，也可以是任意个或 0 个。

例如，给定一组数字 a、b、c…，请计算 $a^2+b^2+c^2+\cdots$。要定义这个函数，必须确定输入的参数。由于参数的个数不确定，首先想到可以把 a、b、c…作为一个列表或元组传进来。新建文件 calc1.py，输入如下代码。

```
def calc(numbers):
    sum=0
    for n in numbers:
        sum=sum+n*n
    print("sum=",sum)
```

运行程序，在调用函数的时候，需要先组装出一个列表或元组，因此分别输入调用函数语句 calc([1,2,3])和 calc((1,3,5,7))，按回车键执行，其结果如图 3-6 所示。

接下来，如果把函数的参数 numbers 改为可变参数，定义可变参数与定义列表或元组参数相比，仅仅在参数前面加了一个星号“*”，因此改为 *numbers，在函数体内部，可变参数 numbers 接收到的是一个列表或元组。新建文件 calc2.py，输入如下代码。

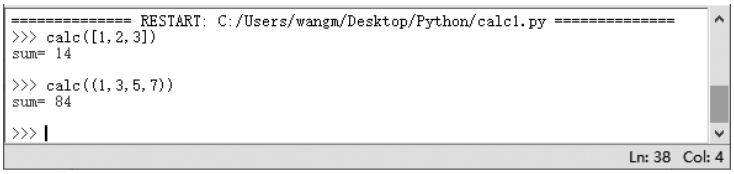

图 3-6　列表/元组参数的运行结果

```
def calc(*numbers):
    sum=0
    for n in numbers:
        sum=sum+n*n
    print("sum=",sum)
```

利用可变参数，调用函数时不需要先组装出一个列表或元组，调用函数语句可以简化成 calc(1,2,3)和 calc(1,3,5,7)，还包括 0 个参数，输入调用函数语句 calc(1,2)和 calc()，按回车键执行，其结果如图 3-7 所示。

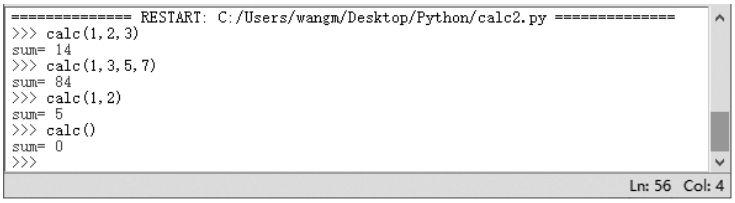

图 3-7　可变参数的运行结果

4. 关键字参数

可变参数允许传入 0 个或任意个参数，这些可变参数在函数调用时自动组装为一个列表或元组。而关键字参数允许传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装成一个字典。调用时可以直接输入一个字典或者按照 key＝value 的格式进行赋值，声明关键字参数时只需在参数名前边加上两个星号“**”。

修改 enroll.py 函数，要求实现大学一年级新生注册，除了姓名和性别是必填项外，其他项都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。新建文件 enrollstu.py，输入如下代码。

```
def enroll(name, gender, **kw):
    print("姓名:", name)
    print("性别:", gender)
    print("其他信息:", kw)
```

函数 enroll()除了必选参数 name 和 gender 之外，还接收关键字参数 kw。如果未指定关键字参数 kw，则 kw 为空，否则 kw 为参数指定值，运行结果如图 3-8 所示。

在 Python 中定义函数时，可以任选必选参数、默认参数、可变参数和关键字参数这 4 种参数中的一种或多种。但在定义的过程中需要注意，参数定义的顺序必须是必选参数、默认

参数、可变参数和关键字参数。

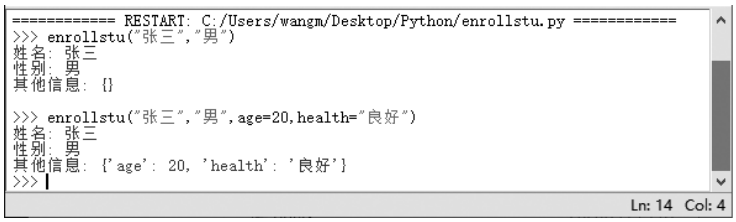

图 3-8　关键字参数的运行结果

3.3　函数的返回值

Python 中，用 def 语句创建函数时，可以用 return 语句指定函数的返回值。在函数中，使用 return 语句的语法格式一般形式为

```
return [返回值]
```

说明：return 语句的作用是结束函数调用并返回值。

3.3.1　指定返回值和隐含返回值

Python 函数使用 return 语句返回“返回值”，可以将其赋给其他变量或作其他用处，所有的函数都有返回值，如果没有 return 语句，函数运行结束会隐含返回一个 None 作为返回值，类型是 NoneType，与 return、return None 等效，都是返回 None。

修改文件 sum_parm.py 代码如下，另存为文件 sum_parm_return.py。

```
def sum2(start, end):
    total=0
    for count in range(start, end+1):
        total=total+count
    return total

s=sum2(1, 100)  #将 sum2()函数返回值赋给变量 s
```

程序运行结果如图 3-9 所示。

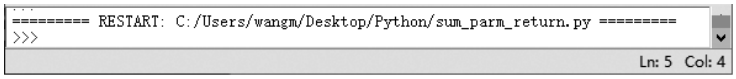

图 3-9　函数值返回赋值结果

上例只是将函数 sum2(1,100)执行结果返回并赋给变量 s，程序没有指定任何输出。继续增加输出语句，代码如下。

```
def sum2(start, end):
    total=0
    for count in range(start, end+1):
        total=total+count
    return total

s=sum2(1, 100)  #将 sum2()函数返回值赋给变量 s
print("return_sum=",s)
```

运行结果如图 3-10 所示。

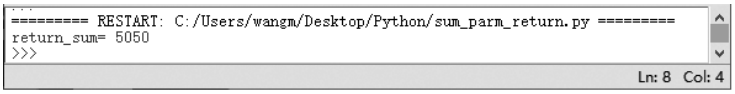

图 3-10　函数值返回输出结果

说明：

(1) 要想输出求和结果，仅仅使用 return 语句是不行的，return 语句只是将 return 后面的返回值作为函数的输出，而 print 语句是打印在控制台，因此需要使用 print 语句打印求和结果，可见 return 和 print 语句是有区别的。

(2) sum_parm.py 文件中定义 sum1()函数时，函数体内虽然没有使用 return 语句，但隐含返回值为 None。

(3) sum_parm_return.py 文件中定义 sum2()函数时，函数体中使用 return 语句指定返回值为 total。

(4) 程序中使用 s＝sum2(1,100)语句将 sum2()函数返回值赋给了变量 s。

3.3.2　多条 return 语句

一个函数可以存在多条 return 语句，但只有一条 return 语句可以被执行。如果函数执行了 return 语句，函数会立刻返回，结束调用，函数体中 return 之后的其他语句都不会被执行了；如果函数体执行完都没有一条 reutrn 语句被执行，同样会隐式调用 return None 作为返回值。

例如，修改文件 sum_parm_return.py 代码如下，另存为文件 sum_parm_returns.py。

```
def sum3(start, end):
    total=0
    for count in range(start, end+1):
        total=total+count
    return "累加求和结果是:"
    return total
    print("这是 sum3 函数最后一条语句。")

print(sum3(1, 100))
```

程序的执行结果，如图 3-11 所示。

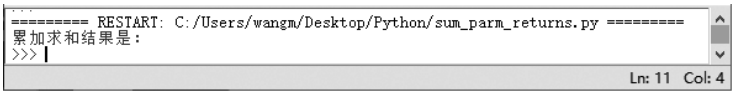

图 3-11　多条 return 语句，只有一条被执行

说明：sum3(start, end)函数中有两条 return 语句，调用函数时只执行第一条“return "累加求和结果是："”语句，其后所有的语句都不执行。一般情况，顺序结构的多条 return 语句往往存在冗余，因此多条 return 语句通常和分支结构结合使用，分别返回不同的函数执行结果，程序修改如下。

```
def sum3(start, end):
    total=0
    for count in range(start, end+1):
        total=total+count
    if total >1000:
        return total
    else:
        return "累加求和结果小于 1000"
    print("这是 sum3 函数最后一条语句。")

print("sum3(1,100)=",sum3(1,100))
print("sum3(1,10)=",sum3(1,10))
```

程序的执行结果，如图 3-12 所示。

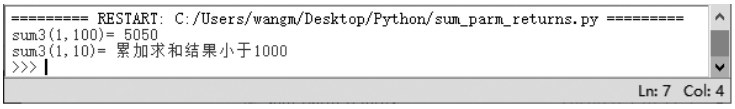

图 3-12　多分支 return 语句执行结果

3.3.3　返回值类型

函数的返回值类型除了可以是数值、字符串外，还可以是任何类型的值，包括列表、元组、字典等复杂的数据结构。

例如，定义 infor_stu(name,sex,age)函数，函数返回值类型为字典，新建文件 infor_stu.py，输入如下代码。

```
def infor_stu(name,sex,age):
    student={"姓名":name,"性别":sex,"年龄":age}
    return student
print(infor_stu("张三","男",20))
```

程序执行结果，如图 3-13 所示。

图 3-13　return 返回值为字典类型

前面的实例中 return 语句都仅返回了一个值，其实无论定义 return 返回什么类型，return 只能返回一个值，但这个值可以存在多个元素。

例如，求 1～100 的累加和，并输出起始数、终止数、累加和。新建文件 start_end_sum.py，输入如下代码。

```
def sum4(start, end):
    total=0
    for count in range(start, end+1):
        total=total+count
    return start,end,total

print(sum4(1,100))
```

程序的执行结果，如图 3-14 所示。

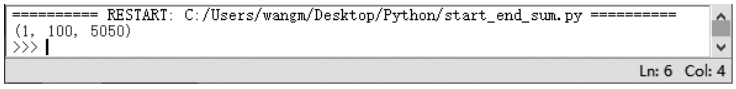

图 3-14　return 返回值类型为元组

说明：函数 sum4()使用“return start，end，total”语句看似返回多个值，其实隐式地被 Python 封装成了一个元组返回，因此返回值为(1，100，5050)。

3.4　函数的嵌套

Python 支持函数的嵌套，即在一个函数内部定义子函数，而且支持多层嵌套。例如，在 fun1()函数中定义 fun2()函数。

```
def fun1():
    num1=20
    def fun2():
        num2=30
```

如果想在调用 fun1()函数的时候输出 num1＝20 或 num2＝30，那么可以给上述代码添加输出语句如下，并另存为文件 fun.py。

```
def fun1():
    num1=20
```

```
    def fun2():
        num2=30
    print("num1=",num1)
    print("num2=",num2)
```

运行程序，在命令行中输入 fun1()，即调用 fun1()函数，按回车键执行，结果如图 3-15 所示。

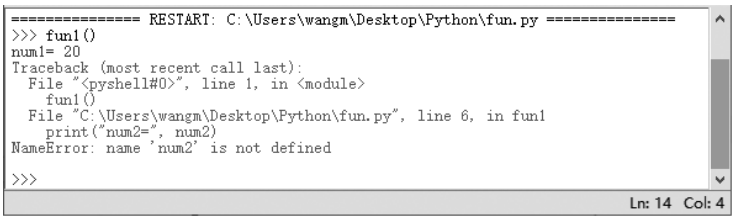

图 3-15　嵌套函数输出变量报错

程序执行会报错，提示没有定义变量 num2。这说明，程序中定义的变量是有作用范围的。虽然在 fun1()函数中定义了变量 num2，但是 num2 是定义在 fun1()函数的嵌套函数 fun2()中的。

变量的作用范围称为该变量的作用域，即一般常说的变量的可见范围。根据定义变量的位置，变量的可见范围分为两种：局部变量和全局变量。

(1) 全局变量：是指在函数外面、全局范围内定义的变量。全局变量的作用域是在整个程序运行环境中都可见，意味着它们可以在所有函数内被访问。

(2) 局部变量：是指在函数中定义的变量，包括参数。局部变量的作用域是在函数内部可见，局部变量的使用范围不能超过其所在的局部作用域。

分析上例，num2 是 fun2()函数内部定义的变量，它是局部变量，只在 fun2()函数内部是可见的，因此在 fun2()函数外没法使用 num2。同理，num1 是 fun1()函数内部定义的变量，它是局部变量，只在 fun1()函数内部是可见的，因此在 fun1()函数外没法使用 num1。另外，fun2()函数在 fun1()函数内部，因此 num1 在 fun2()函数内部也是可见的。

需要注意的是，Python 中的函数也可以被当作变量来对待，因此函数也有作用域。例如，fun2()函数的作用域是在 fun1()函数内部，因此只能在 fun1()内部调用 fun2()函数。

综上所述，如果想输出 num1＝20 或 num2＝30，程序 fun.py 代码应修改如下。

```
def fun1():
    num1=20
    def fun2():
        num2=30
        print("num2=", num2)
    print("num1=", num1)
    fun2()
```

运行程序，在命令行中输入 fun1()，即调用 fun1()函数，按回车键执行，结果如图 3-16 所示。

有时需要定义一个在整个程序中任意位置都可以使用的变量，即定义一个全局变量。

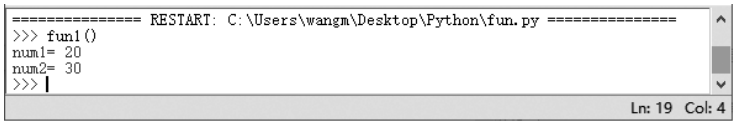

图 3-16　嵌套函数变量作用域

例如，在以上程序的基础上，定义全局变量 num，其在整个程序中都可见，因此在 fun1()和 fun2()函数中甚至程序的任何位置都可以使用 num。程序代码修改如下。

```
num=10
def fun1():
    num1=20
    def fun2():
        num2=30
        print("num2=", num2)
    print("num=", num)
    print("num1=", num1)
    fun2()
```

运行程序并输入 fun1()，即调用 fun1()函数，按回车键执行，结果如图 3-17 所示。

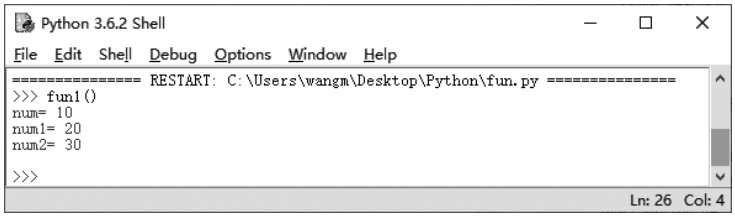

图 3-17　全局变量和局部变量

3.5　精 选 案 例

【例 3-1】　编程实现求任意一个正整数 n 的阶乘。

新建实现正整数 n 的阶乘的文件 factorial.py，输入如下代码。

```
def factorial(n):
    result=n
    for i in range(1,n):        #for 循环实现求 n 的阶乘
        result=result*i
    return  result

number=int(input("请输入一个正整数:"))
result=factorial(number)
print("%d 的阶乘是%d"%(number,result))
```

执行程序，结果如图 3-18 所示。

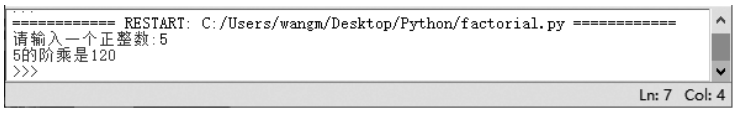

图 3-18　求正整数 n 的阶乘的运行结果

【例 3-2】 编程实现判断输入的年份是否为闰年。提示：闰年的条件是能被 4 整除但是不能被 100 整除，或者能被 400 整除。

新建采用循环判断闰年的文件 leap.py，输入如下代码。

```
def isLeap(y):
    if y%400==0 or (y%100!=0 and y%4==0):#判断闰年条件
        return True
    else:
        return False
year=int(input("请输入一个年份:"))
if isLeap(year)==True:
    print("%d年是闰年"%year)
else:
    print("%d年不是闰年"%year)
```

程序执行结果，如图 3-19 所示。

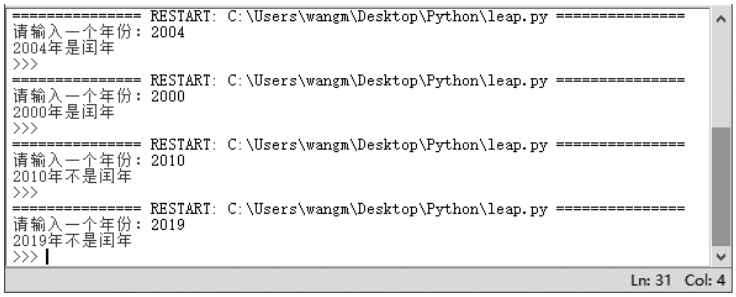

图 3-19　判断闰年的运行结果

说明：

(1) isLeap(y)函数用来判断是否是闰年，y 是形参用来接收输入的年份，y 是否为闰年的判定条件为：y%400==0 or (y%100!=0 and y%4==0)。

(2) isLeap(y)函数的返回值为 bool 型 True 或 False，因此程序的条件判断语句 isLeap(year)==True 中将 isLeap(y)函数返回值与 True 比较，如果 y 是闰年，则 isLeap(y)函数返回为 True，所以该条件判断语句的判断结果为 True；否则，为 False。此处代码也可直接将条件判断语句写成 if isLeap(year)，执行效果相同。

尽管定义了判断闰年的功能函数 isLeap(y)，但是每次判断都要执行程序，会影响效率。修改文件 leap.py 的调用函数可以实现循环查询，修改代码如下。

```
def isLeap(y):
```

```
    if y%400==0 or (y%100!=0 and y%4==0):#判断闰年条件
        return True
    else:
        return False

while True:
    year=int(input("请输入一个年份(小于 0 结束):"))
    if year>=0:
        if isLeap(year)==True:
            print("%d 年是闰年"%year)
        else:
            print("%d 年不是闰年"%year)
    else:
        print("GoodBye!")
        break
```

程序执行结果,如图 3-20 所示。

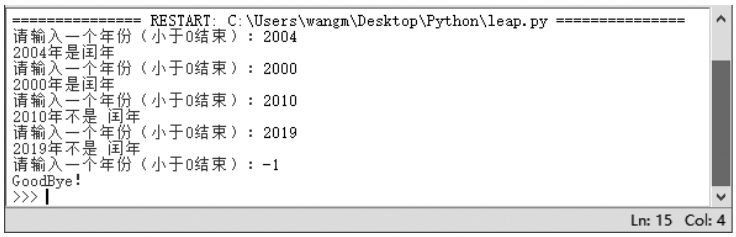

图 3-20　循环判断是否是闰年

【例 3-3】 编程实现输出 100～200 的所有素数。提示：素数的定义是一个大于 1 的自然数,除了 1 和它本身外,不能被其他自然数整除。

新建实现输出素数的文件 primer.py,代码如下。

```
import math
def isPrimer(n):
    if n<=1:
        return False
    for i in range(2,int(math.sqrt(n))+1):
        if n%i==0:
          return False
    return True

CR=1
for num in range(100,200):
    if isPrimer(num) ==True:
        print(num , end='   ')
        if CR%5==0:
            print("\n")
```

```
        CR=CR+1
```

程序执行结果，如图 3-21 所示。

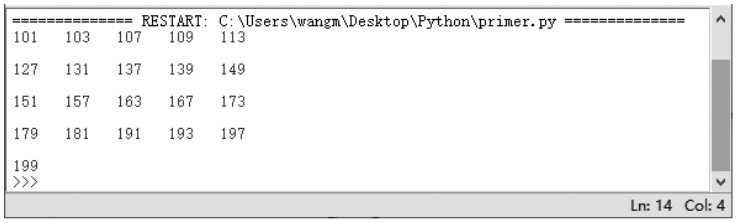

图 3-21　输出 1～100 的所有素数

说明：

(1) 根据素数的定义，判断 n 是否是素数，首先想到如果 n 不能被 2～n－1 的数整除，n 就是素数。其实该算法可以优化，因为如果一个数不是素数是合数，那么一定可以由两个自然数相乘得到，其中一个大于或等于它的平方根，一个小于或等于它的平方根，并且成对出现，所以对上述算法进行优化，只要 n 不能被 2～sqrt(n)的数整除，n 就是素数。

(2) 程序中需要使用平方根函数，因此通过 import math 语句导入标准库 math，通过 math.sqrt(n)调用 math 库中的 sqrt(n)函数，即可返回 n 的平方根。

【例 3-4】 编程实现，从键盘录入一个班级学生的姓名及其大学计算机基础课程的考试成绩，然后统计该班级学生成绩的平均分、最高分、最低分，并且按照分数从高到低对学生的成绩进行排序。

新建实现学生成绩统计的文件 count_stu.py，代码如下。

```
def sorted_score(y):
    score_list=[(y[k],k) for k in y]
    score_sorted=sorted(score_list,reverse=True)
    return [(k[1],k[0]) for k in score_sorted]

n=int(input("请输入班级人数:"))
scores={}
for count in range(1, n+1):
    i=input("请输入学生姓名:")
    j=input("请输入%s 学生分数:"%i)
    scores[i]=float(j)
print("请输入全班大学计算机基础的成绩:",scores)
print("全班大学计算机基础的平均分数:",end='')
print(sum(scores.values())/len(scores))
print("大学计算机基础的最高成绩:",max(scores.values()))
print("大学计算机基础的最低成绩:",min(scores.values()))
print("全班分数从高到低的顺序是:",sorted_score(scores))
```

程序执行结果，如图 3-22 所示。

说明：

(1) scores＝{}定义了一个字典 scores。

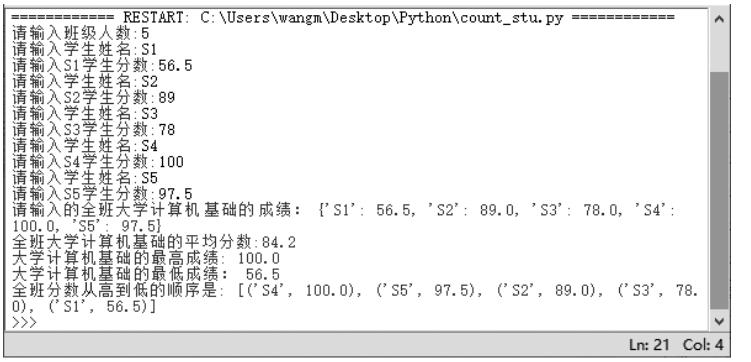

图 3-22　学生成绩统计

(2) scores[i]=float(j)实现了键-值关联,将 i 键对应的值设置为 float(j)。

(3) len(scores)计算 scores 字典的长度,即键-值对个数。

(4) scores.values()返回包含所有值的列表。

(5) sum(scores.values())调用内置函数 sum 求所有值的和。

(6) max(scores.values()))调用内置函数 max 求所有值的最大值。

(7) min(scores.values()))调用内置函数 min 求所有值的最小值。

(8) 调用 sorted_score(scores))函数对字典排序,该函数的实参是字典 scores。

【例 3-5】 编写函数,封装例 2-15 的程序,实现函数 count_fun 分别统计出字符串中英文字母、数字和其他字符的个数,并编写程序调用该函数。

修改文件 count_char.py 并另存为文件 count_char_fun.py,其代码如下。

```
def count_fun(string):
    char=0
    number=0
    space=0
    other=0
    for i in string:
        if i.isdigit():
            number =number +1
        elif i.isalpha():
            char =char +1
        elif i ==' ':
            space =space +1
        else:
            other =other +1
    print("The number_count : ",number)
    print("The char_count : ",char)
    print("The space_count : ",space)
    print("The other_count : ",other)

while True:
    s =input("please input a string : ")
```

```
    if s =="quit":
        break
    else:
        count_fun(s)
```

程序执行结果，如图 3-23 所示。

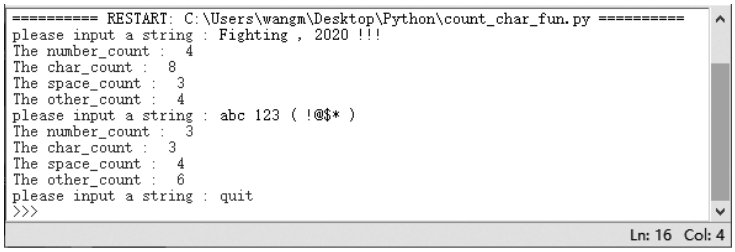

图 3-23　统计字符串中英文字母、数字和其他字符的个数

由此可见，函数的定义是对代码的封装，可以通过重复调用实现功能相同但参数不同的需求。与例 2-15 相比，count_fun(string)函数可以在同一个执行中重复被调用，简化了代码。

3.6　单 元 实 验

请完成如下实验。

(1) 填空完善函数 MAX、MIN、SUM，计算出数字列表的最大值、最小值及总和。要求：利用 range()函数创建一个整数列表，其中列表起始、终止值从键盘输入，并编写程序调用该函数。

```
def func(list_a):
    print("列表的长度是:",…???…)
    print("最大的数字是:",…???…)
    print("最小的数字是:",…???…)
    print("和是:",…???…)

a=int(input("请输入一个整数列表的起始值:"))
b=int(input("请输入一个整数列表的结束值:"))
list_x=list(…???…)
func(…???…)
```

例如，程序执行结果如图 3-24 所示。

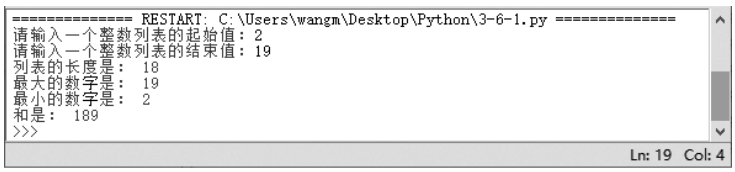

图 3-24　统计列表的最大值、最小值及总和

(2) 填空完善函数 cube()。要求：创建一个列表，其中列表元素包含 1～N 这 N 个整数的三次方，执行调用函数 cube()的结果是将每个元素的三次方值打印出来。

```
def cube(l):
…???…
    for c in l :
        print(c)

N =int(input("please in put a number:"))
list_cube =…???…
cube(list_cube)
```

例如，程序执行结果如图 3-25 所示。

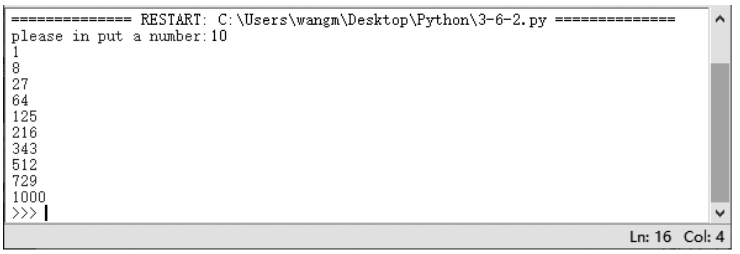

图 3-25　求 N 个整数的三次方值的运算

(3) 编写函数将两个矩阵相加。提示：用列表存储矩阵，首先创建一个新的矩阵 Z，然后使用 for 迭代并取出 X 和 Y 矩阵中对应位置的值，将其相加后放到新矩阵的对应位置中。

例如：

$$X=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix},\quad Y=\begin{bmatrix}2 & 4 & 6\\ 1 & 3 & 5\\ 7 & 8 & 9\end{bmatrix}$$

两矩阵相加的结果如图 3-26 所示。

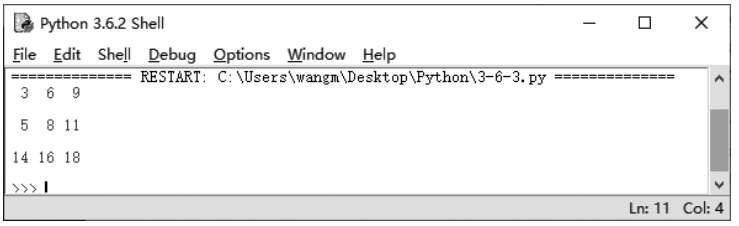

图 3-26　矩阵相加

(4) 编写函数，判断一个多位数是否是回文数。所谓回文数，是指一个像 121、1465641 这样“对称”的数，即将这个数的数字按相反的顺序重新排列后，所得到的数和原来的数一样。如 12345、12312 就不是回文数。

例如，程序执行结果如图 3-27 所示。

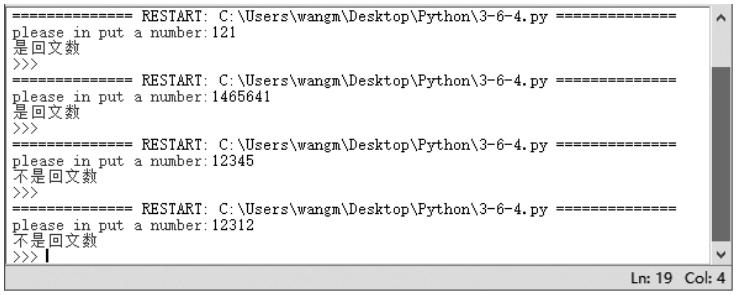

图 3-27　判断回文数

(5) 编写函数,输出 1000 以内的所有完全数。一个自然数如果它的所有真因子(即除了自身以外的约数)的和等于该数,那么这个数就是完全数。例如,6 的真因子有 1、2、3,6＝1＋2＋3,所以 6 是一个完全数。

提示:

① 用 for 循环分别列出 1000 以内所有整数。

② 用每一个整数分别除以比它小的整数,若整除,则记为该整数因子,并将所有因子相加求和,求和后判断和这个整数是否相等,若相等则该整数是完全数。

③ 定义一个空列表,用以储存 1000 以内的完全数。

例如,程序执行结果如图 3-28 所示。

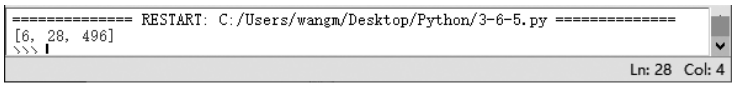

图 3-28　求完全数

(6) 一个已经排好序的列表,从键盘输入一个数字,要求按列表原来的规律将输入的数字插入列表中。提示:首先判断此数是否大于最后一个数,然后再考虑插入中间的数的情况,插入后此元素之后的数依次后移一个位置。

例如,一个有序列表 numlist＝[1,10,100,1000,10000,100000],从键盘输入 34,将其插入有序列表后的结果如图 3-29 所示。

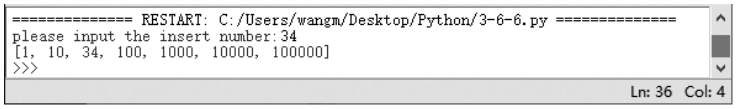

图 3-29　有序列表插入元素

(7) 有 5 个人坐在一起,问第五个人多少岁?他说比第四个人大 2 岁。问第四个人岁数,他说比第三个人大 2 岁。问第三个人,又说比第二人大 2 岁。问第二个人,说比第一个人大 2 岁。最后问第一个人,他说是 10 岁。请问第五个人多大?请编写函数求第五个人的年龄。

(8) 简单选择排序算法是一种典型的交换排序算法,通过交换数据元素的位置进行排序。具体方法是:

① 设所排序序列的记录个数为 n；

② i 取 1,2,…,n－1,从所有 n－i＋1 个记录(a_i,a_{i+1},…,a_n)中找出排序码最小(大)的记录,与第 i(n－i)个记录交换。

③ 执行 n－1 趟后就完成了记录序列的排序,如图 3-30 所示。

编写程序实现简单排序算法。

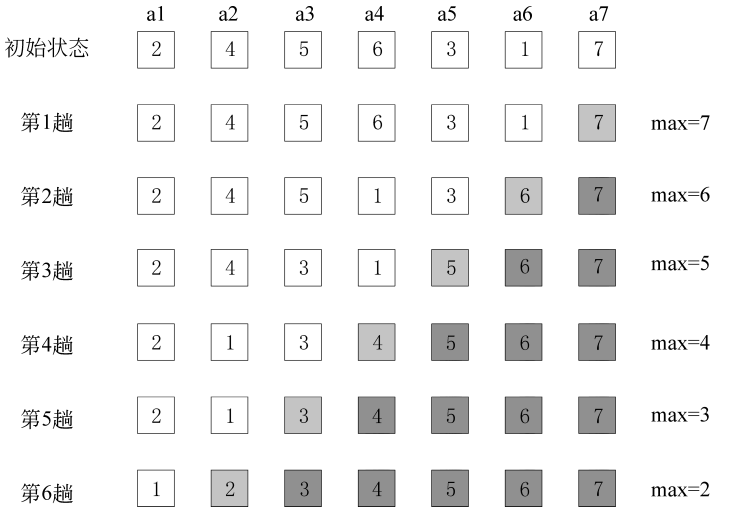

图 3-30　简单选择排序