第5章 chapter 5

函　数

导读

函数是用来实现单一或相关联功能的程序代码段，也可以说函数是指将一组语句通过一个函数名封装起来，如果要运行这个函数仅需调用函数名即可。在计算机中，函数也被称为子程序(subroutine)或过程(procedure)。函数能提高应用的模块性和代码的复用率，方便程序的修改与功能的扩展。Python提供了许多内置函数，供编程人员使用，当然编程人员也可以自己编写函数，然后进行调用，这种函数被称为自定义函数。函数是Python编程非常重要的内容，本章介绍函数相关的内容。

5.1　函数的创建

5.1.1　函数的定义

1. 定义函数的规则

定义函数的规则如下。

(1) 函数代码块以 def 关键词开头，后跟函数名和圆括号()。

(2) 需要传入的参数放在函数名后的圆括号中。

(3) 函数的第一行语句可以选择性地使用文档字符串对函数进行说明。

(4) 函数内容以冒号起始，冒号跟在函数的()之后，函数的语句体要按要求进行缩进。

(5) return 语句用于结束函数返回函数值给调用者。不带返回值的 return 相当于返回 None。

在程序开发过程中，编程人员创建函数时通常希望函数执行结束后返回给调用者一个结果，以便调用者针对具体的结果做出后续的处理。返回值就是函数执行完后给调用者的一个结果。

2. 函数定义的格式

函数定义的格式如下：

```
def functionname( parameters ):
    "函数_文档字符串"                #函数说明文本串
    function_statement(s)          #函数语句体
    return [expression]            #函数的返回语句
```

函数的命名应该符合标识符的命名规则，可由字母、下画线和数字组成，不能以数字开头，不能与关键字同名。在函数名中不能使用任何标点符号，函数名严格区分大小写。在默认情况下，参数名称是按函数声明中定义的顺序匹配。

例如，下述函数能生成日志记录：

```
def logger():
    '生成日志记录函数'
    time_format='%y-%m-%d %x'
    time_current=time.strftime(time_format)
    with open('日志记录','a')as f:
        f.write('%s end action\n'%time_current)
#上述为函数的定义部分
logger()          #此处是调用定义的函数
```

又如，下述函数能输出个人相关的信息：

```
def info_output(name,age,sex):
    '输出个人信息函数'
    print('name:%s'%name)
    print('age:%d'%age)
    print('sex:%s'%sex)
#上述为函数的定义部分
info_output("lili",20,"male")        #此处是调用定义的函数
```

5.1.2 函数调用

定义一个函数就是定义函数的名称，指定函数需要的参数和编写函数的业务逻辑代码块。函数定义好后，就可以调用函数。

注意：在通常情况下，函数的定义必须在函数的调用之前。如果是在函数中调用另一个函数，被调用的函数可以定义在调用函数之后。

函数的调用方法：

```
函数名([实参 1][,实参 2][,实参 3][,…)
```

调用上述两个函数的方法分别是 logger()与 info_output("lili",20,"male")。看上面函数定义框中的代码。

示例：定义一个函数，调用该函数能获取正整数列表中最大数与次大数值。

编写思路如下。

(1) 首先,函数要获取一个 data_list 列表,即 data_list 是函数的参数。

(2) 把 data_list[0]这个值作为最大值(max_num)的参照,把 0 作为 second_num 的参照。

(3) 用列表后续的值和最大值(max_num)做比较,如果比最大值(max_num)大,则把该值赋值给 max_num,同时把原来的 max_num 赋值给 second_num。如果后继值比最大值(max_num)小,则与 second_num 比较,把该值赋给 second_num。

(4) 然后继续做比较,如果找到比 max_num 大的值,重复(3)。

(5) 定义一个字典存放最大值和次大值。

程序代码如下:

```
def find_max_and_second_large_num(data_list):
    max_num=list[0]                #存放最大值
    second_num =0                  #存放次大值
    for i in range(1,len(list)): #从第二个元素开始对比,所以 i 从 1 开始
        if data_list[i] >max_num:
            second_num=max_num
            max_num=list[i]
        elif list[i] >second_num:
            second_num =data_list[i]
    return {"max":max_num, "second": second_num}
#上述为函数部分
list =[100,50,60,70,30,45]         #创建一个列表
#下面为函数调用部分
max_and_second_large_num =find_max_and_second_large_num(list)
print(max_and_second_large_num)
#输出结果为
  {'max': 100, 'second': 70}
```

在上述示例中,find_max_and_second_large_num(list)就是调用函数,其中,list 是实参,调用时传给函数的形参 data_list,函数的返回值为一个字典。

5.1.3 函数返回语句

在定义函数时,如果想获取函数的返回结果,可以用 return 语句把结果返回。函数在执行过程中只要遇到 return 语句,就会停止函数的运行并返回结果。如果在函数中没有指定 return,此时函数的返回值为 None。return 可返回多个对象,解释器会把这多个对象组装成一个元组返回。

5.2 函数参数

函数是把具有独立功能的代码块组织成为一个小模块，在函数中定义参数是为了提高函数的通用性。在定义函数时，括号中的参数被称为形式参数，简称形参。定义函数时使用形参是为了在调用函数时接收实参，形参在函数内部作为变量使用。在调用函数时，括号中的参数被称为实际参数，简称实参。实参是用来把数据传递给函数的。调用函数时，实参可以是常量、变量、表达式以及函数。形参与实参的区别：形参是虚拟的，不占内存空间，只有在被调用时才分配存储空间；而实参是一个变量，占用内存空间。

注意：形参与实参之间传输数据是单向的，只能实参传给形参。

5.2.1 参数分类

在 Python 中，函数参数分为必备参数、关键字参数、默认参数与可变参数。

1. 必备参数

必备参数须以正确的顺序传输给函数，调用时的数量必须和声明一致。必备参数也被称为位置参数。

如调用 5.1.1 节中的 info_output 函数时，调用函数必须传入 name、age、sex 3 个参数，且也必须按照这个顺序，否则会报错，见下面程序中函数调用问题。

```
def info_output(name,age,sex):
    '输出个人信息函数'
    print('name:%s'%name)
    print('age:%d'%age)
    print('sex:%s'%sex)
    return
#上述为函数部分
info_output("lili",20,"male")       #正确调用方法
info_output("lili",20)              #错误调用方法,数量不一致
info_output(20,"lili","male")       #错误调用方法,顺序错误
```

2. 关键字参数

在 Python 中，为了提高函数调用的灵活性，引入了关键字参数。关键字参数是指在调用函数时，在圆括号内以形参为关键字、实参为关键字值的方式来传值，这样就不要求实参与形参之间的顺序完全一一对应。见如下示例：

```
def info_output(name,age,sex):
    '输出个人信息函数'
```

```
    print('name:%s'%name)
    print('age:%d'%age)
    print('sex:%s'%sex)
    return 0
#上述为函数部分
info_output(name="lili",sex="male",age=20)      #正确调用方法
info_output("lili",sex="male",age=20)           #正确调用方法
```

3. 默认参数

默认参数是指在定义函数时为参数指定默认值，如果调用函数时默认参数不变，可以不为该参数传值，即使用默认值，参数需要改变才传值。见下面示例：

```
def info_output(name,age,sex="male",city="beijing"):
    '输出个人信息函数'
    print('name:%s'%name)
    print('age:%d'%age)
    print('sex:%s'%sex)
    print('city:%s'%city)
    return
#上述为函数部分
info_output(name="lili",age=20)                  #正确调用方法
info_output("liming",age=22)                     #正确调用方法
info_output(name="liuhua",sex="female",age=18)  #正确调用方法
info_output("liuhua",18,city="changsha")         #正确调用方法
info_output(name="liuhua",18,city="changsha")   #参数报错
```

由上可见，默认参数降低了函数调用的难度，而且一旦需要更复杂的调用时，又可以传递更多的参数来实现。无论是简单调用还是复杂调用，函数只需要定义一个。

调用有多个默认参数的函数时，既可以按顺序提供默认参数，比如调用 info_output("liuhua",18,"female","changsha")，也可以调用 info_output("liuhua",18,city="changsha")，意思是除了 name、age，最后一个参数应用在参数 city 上，sex 参数由于调用时没有提供，仍然使用默认值。在调用有默认值的函数时，也可以不按顺序提供部分默认参数。当不按顺序提供部分默认参数时，应指定参数。从上面的例子可以看出，默认参数可以简化函数的调用。

设置默认参数时，需要注意如下情况。

(1) 如果必备参数和默认参数都存在，则必须将必备参数放在默认参数之前。

(2) 如果有关键字参数和必备参数，在函数定义时，则需要把必备参数放在前面，调用时，必备参数也必须放在前面。

默认参数很有用，但如果使用不当，也会出现一些意想不到的问题，见如下示例：

```
def add_end(l=[ ]):
    l.append('END')
    return l
#上述为函数部分
#下面为函数调用
list1=add_end([1, 2, 3])
print(list1)          #输出为[1, 2, 3, 'END']
list1=add_end(['x', 'y', 'z'])
print(list1)          #输出为['x', 'y', 'z', 'END']
list2=add_end()
print(list2)          #输出为['END']
list2=add_end()
print(list2)          #输出为['END', 'END']
list2=add_end()
print(list2)          #输出为['END', 'END', 'END']
```

当使用默认参数调用时，第一次调用结果是对的，但是，再次调用 add_end()时，结果就不对了，出现了['END', 'END']。很多初学者很疑惑，默认参数是[]，但是函数似乎每次都记住了上一次添加了'END'后的列表。主要原因：Python 函数在定义的时候，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(引用)，它指向对象[]调用该函数，如果改变了 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。所以，定义默认参数要牢记一点，默认参数必须指向不变对象。

如果不让程序出现这种问题，可修改上面的函数，用 None 这个不变对象来实现，这样修改后，无论调用多少次，都不会出现前面的问题。见如下函数：

```
def add_end(l=None):
    if l is None:
        l=[ ]
        l.append('END')
    return l
```

为什么要设计 None 这样的不变对象呢？因为不变对象一旦创建，对象内部的数据就不能修改，这样就减少了由于修改数据导致的错误。初学者在编写程序的时候，如果可以设计一个不变对象，那就尽量设计成不变对象。

4. 可变参数

在 Python 的函数中，还可以定义可变参数。顾名思义，可变参数就是传给函数的参数的个数是可变的，可以是 1 个、2 个到任意个，还可以是 0 个。在此，以求数值之和为例说明可变参数的使用方法。

需求：给定一组数字 a，b，c…，请计算 a+b+c+…

通过前面的介绍可知，要定义一个实现上述功能的函数，必须要确定函数输入参数

的个数。由于本需求的参数个数不确定，首先会想到把 a，b，c…作为一个列表或元组传给函数，这样，函数可以定义如下：

```
def calc(numbers):
    '求数值元组之和'
    sum=0
    for n in numbers:
        sum=sum +n
    return sum
#上述为函数部分
#下面为函数调用
sum1=calc((1,2,3,4,6,7,8))     #向 numbers 传替一个元组
print(sum1)
sum2=calc([1,2,3,4])           #向 numbers 传替一个列表
print(sum2)
```

这种调用函数的方法过于苛刻，在 Python 中，利用可变参数定义函数能解决上述问题，且能使用传统方法调用定义的函数。利用可变参数定义函数如下：

```
def calc(*args):
    '求数值元组之和'
    print(args)              #输出的是元组
    sum=0
    for n in args:
        sum=sum +n
    return sum
#上述为函数部分
#下面为函数调用
sum1=calc(1,2,3,4,6,7,8)   #把 1,2,3,4,6,7,8 变为元组传替给参数 args
print(sum1)                #输出为 31
sum2=calc(1,2,3,4)         #向 args 传替一个(1,2,3,4)元组
print(sum2)                #输出为 10
```

在该函数中的 *args 是个参数，表示参数的个数是可变的。当调用这个函数时，会创建一个 args 元组，传替过来的数据元素作为元组的元素。

注意：如果传替的实参本身是列表或元组时必须在参数前加 * 号。

如：sum1＝calc(*(1,2,3,4,6,7,8))　　#向 args 直接传替一个元组

如：sum1＝calc(*[1,2,3,4,6,7,8])　　#向 args 直接传替一个列表

如：sum1＝calc(*(1,2,3),*(4,6,7,8))　　#两个元组合并后传给 args

如：sum1＝calc(*[1,2,3],*(4,6,7,8))　　#将一个列表与一个元组合并成
#元组后传给 args

*args 用来传替无名参数。可变参数也可实现有名参数的传替，只是此时定义函数

时用**来修饰可变参数。见如下示例：

```
def print_info(* * kwargs):
    print(kwargs)
#上述为函数部分
#下面为函数调用
print_info(name="lili",sex="female",age=20,city="beijing")
#函数的输出结果为
{'age': 20, 'name': 'lili', 'city': 'beijing', 'sex': 'female'}
```

从输出可以看出，该函数是用一个字典来存储调用函数时的多个参数，由于传入的参数是键/值(key/value)对的格式，该格式与字典相对应，因此，用字典来存储是最好的解决方案。

在 Python 中，字典使用频率是非常高的，调用上述 print_info 函数，实参能不能是字典呢？见下面示例：

```
def print_info(**kwargs):
    print(kwargs)
#上述为函数部分
#下面为函数调用
print_info(* * {"name":"lili","age":20,"city":"changsha"})
#输出结果为
{'age': 20, 'name': 'lili', 'city': 'changsha'}
```

如果传替两个字典，该函数的输出是什么呢？

如果调用方式为

```
print_info(* * {"name":"lili","age":20,"city":"changsha"}, * * {"sex":"
female"})
```

此时输出就为

```
{'age': 20, 'name': 'lili', 'city': 'changsha', 'sex': 'female'}
```

从输出可以看出，传替两个字典时就把两个字典合并为一个字典了。

如果传替的参数既有无名参数也有有名参数，Python 中如何处理呢？

见如下例子：

```
def print_info(*args,**kwargs):
"既有无名参数也有有名参数"
    print(args)
    print(kwargs)
#上述为函数部分
#下面为函数调用
```

```
print_info("lili","female",age=20,city="beijing")
#函数的输出结果为
('lili', 'female')
{'city': 'beijing', 'age': 20}
```

从输出可以看出,前面两个参数是无名参数,用一个元组来存储传替的参数,后面两个参数是有名参数,用一个字典来保存传替过来的参数。这样就可以解决无名参数与有名参数的传替。那如何把字典中的键打印出来呢? 见下面示例:

```
def print_info(**kwargs):
    for i in kwargs:
        print(i)                              #i 为键
        print('%s:%s\n'%(i,kwargs[i]))        #kwargs[i]为值
#上述为函数部分
#下面为函数调用
print_info(name="lili",sex="female",age=20,city="beijing")
```

在参数定义与传替时,要注意如下 5 个问题。

(1) 如果定义一个既可接收可变无名参数与可变有名参数的函数时,* args 放在** kwargs 之前。

(2) 在 Python 中定义函数,可以用必备参数、关键字参数、默认参数和可变参数,这 4 种参数都可以一起使用,或者只用其中一些,但参数定义的顺序必须是必备参数、默认参数、可变参数和关键字参数。

(3) Python 的函数具有非常灵活的参数形态,既可以实现简单的调用,又可以传入非常复杂的参数。

(4) 默认参数一定要用不可变对象,如果是用可变对象,运行会有逻辑错误。

(5) 使用 * args 和**kw 作为参数名是 Python 的习惯用法,可以用其他参数名,但最好采用习惯用法。

5.2.2 参数传替

在 Python 中要认识函数参数的传替,必须清楚变量与对象的关系。Python 中一切皆为对象,数字是对象,列表是对象,函数也是对象。而变量则是对象的一个引用(又称为名字或者标签),对象的操作都是通过引用来实现的。

例如: a=[],[]是一个空列表对象,变量 a 是该对象的一个引用,赋值操作就是把 a 绑定到一个空列表对象上。在 Python 中,变量更准确的叫法是名字,该名称就像给对象添加一个标签。见如下示例:

```
a=[ ]    #把标签 a 绑定到一个空列表对象上
b=2      #把标签 b 绑定到数字对象 2 上
b=3      #相当于把原来整数 2 身上的 b 标签撕掉,贴到整数 3 身上。
c=b      #相当于在对象 3 上贴了 b、c 两个标签,通过 b 与 c 可以对对象 3 进行操作
```

在 Python 函数中，参数的传递本质上是一种赋值操作，而赋值操作是名字到对象的绑定过程。见如下代码，理解为什么是这样的输出。

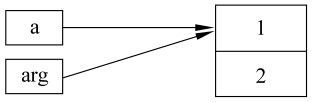
```
def func(arg):
    arg=2
    print(arg)

a=1
func(a)     #输出为 2
print(a)    #输出为 1
```

在上述代码段中，变量 a 绑定了 1，调用函数 func(a)时，相当于给参数 arg 赋值 arg=1，这时两个变量都绑定了 1。在函数中 arg 重新赋值为 2 之后，相当于把 1 上的 arg 标签撕掉，贴到 2 身上，而 1 上的另一个标签 a 一直存在，因此 print(a)还是 1。变量与对象的引用变化过程如图 5.1 所示。

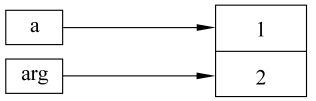
在此，以下面代码来说明函数参数的传替方式：

```
def func(args):
    args.append(1)
b=[ ]
print(b)         #输出为[ ]
print(id(b))     #输出为 2241948639880
func(b)
print(b)         #输出为[1]
print(id(b))     #输出为 2241948639880
```

在上述代码中对象与变量的关系如图 5.2 所示。

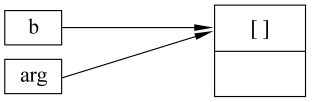


(a) 执行arg=2前

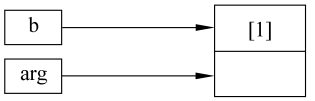


(b) 执行arg=2后

图 5.1　对象的引用变化



(a) 执行append前



(b) 执行append后

图 5.2　对象与变量的关系图

执行 append 方法前 b 和 arg 都指向(绑定)同一个对象[]，执行 append 方法时，并没有重新赋值操作，也就没有新的绑定过程，append 方法只是对列表对象插入一个元素，对象还是那个对象，只是对象里面的内容变了。因为 b 和 arg 都是绑定在同一个对象上，执行 b.append 或者 arg.append 方法本质上都是对同一个对象进行操作，因此 b 的内容在