

# 第四章 自定义函数

函数 (function) 是所有程序设计语言的核心内容之一，在前面我们已经多次接触过函数。函数的最大优点是增强了代码的重用性和可读性，能提高代码的重复利用率。像 `input()`、`print()` 等这些函数是 Python 工程师已经编写好了，为我们直接使用。但我们也可根据需要自己编写函数，称为自定义函数。本节介绍怎样自己编写函数的相关知识。(注：本章有的地方涉及列表、字典、元组等数据类型概念，理解有困难时可参考第五章数据结构相关内容)

## 4.1 函数的创建与调用

### 4.1.1 自定义函数的语法

```
def functionname(parameterlist):  
    '''函数说明'''  
    [函数体]
```

`functionname`: 函数名，任何有效的 Python 标识符；

`parameterlist`: 参数列表，如果有多个参数，参数之间用“,” 隔开，也可以没有参数；

函数说明: 对函数的描述，可以不写；

函数体: 函数被调用时执行的功能代码，由一条或多条个语句组成。如果你只希望定义一个空函数，可以使用 `pass` 语句占位。

**注意:** 函数说明与函数体需要有一定的缩进量，且具有相同的缩进量。否则会引发“invalid syntax”异常。

例子: 定义一个对变量加 4 的函数。

```
01 def add(x):  
02     '''这个函数的功能是对变量 x 加 4'''  
03     x=x+4  
04     return x
```

在这里，`def` 是定义函数的关键字，凡是在定义函数的地方都需要书写，`add` 是函数的名称，`x` 是参数，语句 `return x` 的作用是返回 `x` 的值。但是，这段代码是不会被计算机执行的，因为这里只是定义而已，相当于我们只是制造了一个具有适当功能的工具，这个工具是否发挥作用，还需要像前面调用内建函数一样进行调用，函数才会被执行。

### 4.1.2 函数的调用

函数的调用很简单，使用: `functionname(parameterlist)` 即可。`parameterlist` 要与定义函数时一致。如果函数有返回值，需要使用形如: `value = functionname(parameterlist)`，接收函数的返回值。比如我们调用上面创建的函数，就可以使用下面的代码调用:

```
y = add(5)
```

当计算机执行完这条语句后，`y` 的值为 9。

例子: 创建一个函数，函数的功能为: 判断两个数是否具有倍数关系。

```
01 def isMultiple(x, y):  
                                #定义函数  
02     if x % y ==0 or y % x ==0:  
03         print('yes')
```

```

04 else:
05     print('no')
06 isMultiple(3,6)           #调用函数
07 isMultiple(2,5)         #调用函数

```

注意：在函数被调用之前，必须已定义。如果把 06、07 行放到 01 行的前面就会引发 “NameError: name 'isMultiple' is not defined r” 异常。表示 isMultiple 没有被定义。

## 4.2 函数参数

### 4.2.1 形式参数与实际参数

在函数定义时，小括号里面的参数叫做形式参数。在调用函数时，小括号里面的参数叫做实际参数。形式参数只是一个标记，没有具体的值，只有当函数被调用时，通过实际参数赋值给形式参数，这时形式参数才有具体的值。

实际参数与形式参数之间是怎样传递值的呢？Python 中有两种传递值的方式。请先测试下面的例子。

例 1:

```

01 def add(x,y):
02     x += 1
03     y += 2
04     print(x,y)           #改变 x,y 的值后，显示 x,y 的值
05 x = 2
06 y = 3
07 print(x,y)             #显示 x,y 的值
08 add(x,y)               #调用函数，
09 print(x,y)             #显示 x,y 的值

```

输出:

```

2 3
3 5
2 3

```

可以看出，调用函数后，尽管在函数中对形式参数 x, y 的值进行了修改，但是函数外面的 x, y 的值并没有被改变。这个例子说明了：尽管形式参数的名称与全局变量的名称相同，但他们并不是同一个对象。

例 2:

```

01 def add(a,b):
02     a[0] += 1
03     b[0] += 2
04     print(a[0],b[0])
05 x = [2]           #x 为列表，元素为 2，关于列表的知识参见第五章
06 y = [3]
07 print(x[0],y[0]) #显示 x,y 的第一个元素的值
08 add(x,y)         #调用函数
09 print(x[0],y[0]) #显示 x,y 的第一个元素的值

```

输出:

```
2 3
3 5
3 5
```

可以看出，调用函数后，在函数中对形式参数的值进行修改的同时也修改了实际参数的值。像例 1 中那样，修改形式参数的值并不影响实际参数的值，这种传递值的方式称为**值传递**。像例 2 中那样，修改形式参数的值也影响实际参数的值，这种传递值的方式称为**引用传递**。那么，什么情况进行值传递，什么情况进行引用传递呢？答案是：由 Python 根据实际参数的类型自动确定，我们是不能改变这种传递方式的。Python 规定：把像数字、字符串、元组等**不可变类型数据**作为**实际参数时按照值传递方式进行**，把像列表、字典等**可变类型数据**作为**实际参数时按照引用传递方式进行**。

## 4.2.2 形式参数的类型

在函数定义时，参数列表中的参数可能有以下几种形式：

`functionname(parameter1, parameter2=' default', *args, parameter3, **kwargs)`，为方便后面的说明，把这个式子称为“**范式**”。

### (1) 位置参数

位置参数也叫必选参数，是指在调用函数时必须传值的参数。这种参数在函数定义时只有名字且位置在其它参数类型的前面，如“范式”中的 `parameter1`。

### (2) 默认参数

默认参数是指，在函数定义时设置了默认值的参数，在调用函数时可以传值也可以不传值。如“范式”中的 `parameter2`。位置必须在位置参数的后面，关键字参数的前面。建议默认值的类型为不可变类型。

### (3) 可变参数

可变参数是用于接收那些除位置参数、默认参数以外的无名字的实际参数的值，个数可以是 0 个或多个，定义时需在参数名前加上\*号，如“范式”中的 `*args`。如果有位置参数或默认参数的话，可变参数必须写在他们的后面。

### (4) 命名关键字参数

命名关键字参数是指在调用函数时，实际参数必须以“`参数名=值`”的形式进行调用的参数。在函数定义时有两种方式可以指明参数是命名关键字参数。

①在\*后面的非关键字参数；如 `add(a, b, *, c, d)` 中的 `c, d`。（这里的\*不是参数，只表示后面的 `c, d` 是命名关键字参数）；

②在可变参数后面的非关键字参数。如“范式”中的 `parameter3`。

### (5) 关键字参数

关键字参数与可变参数类似，是用于接收那些除命名关键字参数以外的“`键=值`”形式的实际参数的值。个数可以是 0 个或多个，定义时需在参数名称前用\*\*号，如“范式”中的 `**kwargs`。关键字参数必须写在其它所有类型参数的后面。

**注意：**在函数定义时，这 5 种参数可以部分或全部使用，但顺序必须是：**位置参数>默认参数>可变参数>命名关键字参数>关键字参数**。

为了更好地理解这些参数类型的意义，下面以实例解释实际参数与形式参数的匹配顺序。

**说明：**

①调用函数时，实际参数只有两种形式，一是具体的值，二是“`变量=值`”的形式（如果实际参数中有\*和\*\*时，\*会首先被“解包”为具体的值，\*\*会首先被“解包”为“`变量=值`”的形式）。例如：

```
01 tup = (2, 3, 4)
02 dict = {'k2':2, 'k3':3}
```

```
03 fun(1,*tuple)           #等价于 fun(1, 2, 3, 4)
04 fun(k1=1,**dict)        #等价于 fun(k1=1,k2=2,k3=3)
```

②调用函数时，实际参数必须的顺序是：“变量=值”形式的参数在后面。例子：

```
01 def test(arg1, arg2=3, *args, arg3, **kwargs):
02     print('位置参数的值:', arg1)
03     print('默认参数的值:', arg2)
04     print('命名关键参数的值:', arg3)
05     print('可变参数的值:', args)
06     print('关键字参数的值:', kwargs)
07 test(1, 2, 3, 4, 5, 6, 7, k1=1, k2=2, arg3=8)
```

**输出：**

```
位置参数的值: 1
默认参数的值: 2
命名关键参数的值: 8
可变参数的值: (3, 4, 5, 6, 7)
关键字参数的值: {k1 : 1, k2 : 2}
```

**解释：**首先按由前向后的顺序把具体值匹配给位置参数和默认参数，1 与 arg1 匹配，2 与 arg2 匹配，3, 4, 5, 6, 7 打包成元组匹配给 args；然后匹配命名关键字参数，8 匹配给 arg3；最后把其余的“键=值”k1=2、k2=2 打包成字典匹配给关键字参数 kwargs。

```
test(1, k1=1, k2=2, arg3=8)
```

**输出：**

```
位置参数的值: 1
默认参数的值: 3
命名关键参数的值: 8
可变参数的值: ()
关键字参数的值: {k1 : 1, k2 : 2}
```

**解释：**首先按由前向后的顺序，把具体值匹配给位置参数和默认参数，1 与 arg1 匹配，由于具体值只有 1 个，所以 arg2 使用默认值 3，args 为空；然后匹配命名关键字参数，8 匹配给 arg3；最后把其余的“键=值”k1=2、k2=2 打包成字典匹配给关键字参数 kwargs。

请你再编写一些测试用例自行测试。

## 4.3 return 语句

在函数体中使用 return 语句，可以使函数向调用语句返回值。语法格式如下：

```
return value
```

value: 返回值，可以是 1 个或多个用逗号 (,) 隔开的值。

例子：

```
def func_test(a,b):
    c=a+b
    d=a*b
    e=a/b
    f=a-b
    return c, d, e, f
```

#### 说明:

- (1) 在函数体中可以有多多个 return 语句;
- (2) 无论 return 语句在哪个位置, 只要当程序执行 return 语句后, 将结束函数体中后面代码的执行, 释放函数中定义的所有变量, 退出函数;
- (3) 当返回值为多个用逗号 (,) 隔开的值时, Python 会把这些值先包装成元组再返回, 本质上还是返回一个值;
- (4) 当函数体中没有 return 语句时, 默认返回 None。

## 4.4 递归函数

如果一个函数在内部又调用函数自己, 这个函数就叫做递归函数。例子:

```
01 def func(n):
02     if n==1:
03         return 1
04     else:
05         return n*func(n-1)
```

这个函数的功能是求  $n!$  ( $n$  的阶乘)。我们以计算  $4!$  为例分析程序的执行逻辑。

```
func(4)                # 第 1 次调用自己
4 * func(3)            # 第 2 次调用自己
4* (3 * unc(2))        # 第 3 次调用自己
4 * (3 * (2 * func(1))) # 第 4 次调用自己, 由于 n 等于 1, 结束递归
```

从规模上看, 每递归一次相比上次递归都应有所减少, 直到满足某个条件时就结束递归调用。在本例中,  $n$  等于 1 就是结束递归的条件。如果一个递归函数中没有结束递归的条件, 递归过程将一直继续下去, 类似于死循环。

递归函数的优点是定义简单, 逻辑清晰, 非常适合解决具有递推关系的问题, 缺点是占用资源较多, 且 Python 默认递归的最大深度为 979, 如果一个问题可能需要不止 979 层才能完成的话, 那就可以考虑使用循环实现。理论上, 所有的递归函数都可以写成循环的方式, 但循环的逻辑不如递归清晰。例子, 把计算  $n!$  改用循环实现。

```
01 m =1
02 i = 1
03 while i<=n:
04     m *= i
05     i += 1
```

## 4.5 匿名函数

匿名函数就是没有名字的函数, 使用关键字 lambda 定义, 语法格式如下:

```
result = lambda arg1, arg2, …: expression
```

arg1, arg2, … 叫做参数，都可以省略，expression，实现函数具体功能的表达式，且只有一个表达式，不能省略，表达式的值就是匿名函数的返回值。

调用方法：result(arg1, arg2, …)

例子：

```
>>>h=lambda a,b,c:a*b*c           #定义匿名函数
```

```
>>>h(2,3,4)                       #调用匿名函数
```

尽管匿名函数具有代码简捷易读的优点，但是毕竟只有一个表达式，只能实现有限的功能。其实，匿名函数的主要用途是作为回调函数使用，已超出本书范围，不举例说明。

## 4.6 变量的作用域

Python 中，所有的变量名都会按照定义时的位置被保存在不同的区域中，这些区域叫做命名空间或者作用域。作用域分为四种类型：局部作用域 (Local)、嵌套作用域 (Enclosing)、全局作用域 (Global)、内置作用域 (Built-in)。这四种作用域简称 LEGB。本节我们重点介绍局部作用域和全局作用域。

在运行函数时，函数体中定义的所有变量将构成一个局部作用域，模块（一个 py 文件）中已经执行的赋值语句，将构成一个全局作用域。

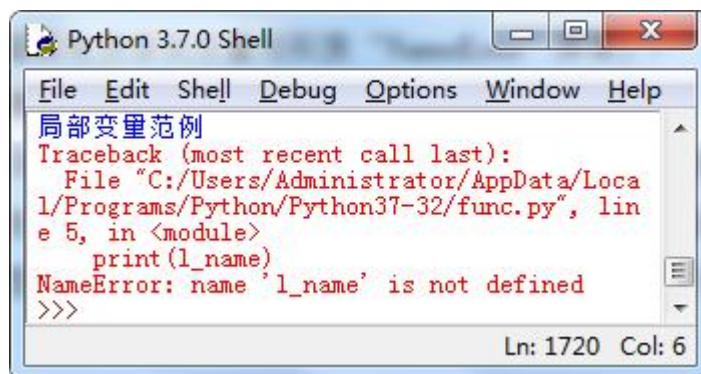
在访问变量时，搜索路径遵循 LEGB 顺序。这里的 LEGB 顺序是指从当前所处区域开始依次搜索当前区域、上一级区域、……，级别的优先级从高到低依次为 L、E、G、B。比如当前级别为 L，则搜索顺序为 L、E、G、B；如果当前级别为 E，则搜索顺序为 E、G、B。一旦在某个作用域搜索到了变量则停止搜索，如果一直没有搜索到变量则引发 “NameError” 异常。

### 4.6.1 局部变量

局部作用域中的变量叫做局部变量，在函数内部定义的变量都是局部变量，只在函数内部有效，在函数运行之前或在函数运行之后，这些变量都是不存在的。每个函数具有自己的作用域，因此，即使两个函数中存在名字相同的变量，也不是同一个变量。

例子：访问函数内部的变量。

```
01 def fun_test():
02     l_name = '局部变量范例'
03     print(l_name)
04 fun_test()
05 print(l_name)
```



### 4.6.2 全局变量

全局作用域中的变量叫做全局变量。在函数外定义的变量是全局变量，在函数内也可以访问全局变量。

例子：访问全局变量。

```
01 g_name = '全局变量范例'  
02 def fun_test():  
03     print(g_name)  
04 fun_test()  
05 print(g_name)
```

输出：

全局变量范例

全局变量范例

下面列举几个需要注意的例子，并总结经验。

(1) 局部变量与全局变量同名

```
01 book_name = 'Python 入门与实战'  
02 def fun_test():  
03     book_name = 'Python 教学系列'  
04     print(book_name)  
05 fun_test()  
06 print(book_name)
```

输出：

Python 教学系列

Python 入门与实战

结论：当局部变量与全局变量同名时，按照 LEGB 规则，在函数内首先搜索到局部作用域里的变量，在函数外部首先搜索到全局作用域里的变量。

(2) 在函数中修改全局变量

```
01 book_name = 'Python 入门与实战'  
02 def fun_test():  
03     global book_name                #声明 book_name 为全局变量  
04     book_name = 'Python 教学系列'  
05     print(book_name)  
06 fun_test()  
07 print(book_name)
```

输出：

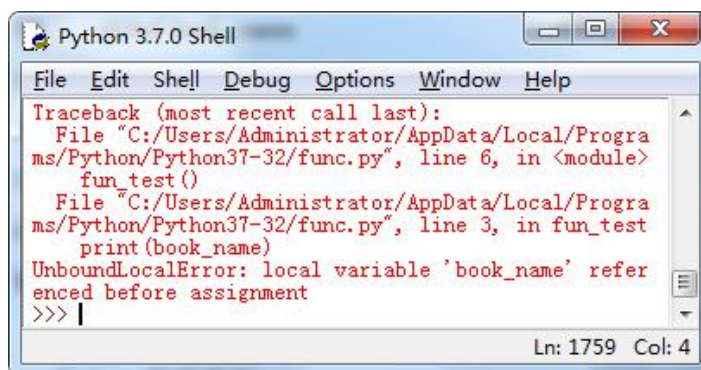
Python 教学系列

Python 教学系列

结论：如果需要在函数中修改全局变量，须使用 global 关键字声明。

(3) 先访问局部变量，后定义局部变量

```
01 book_name = 'Python 入门与实战'
02 def fun_test():
03     print(book_name)
04     book_name = 'Python 教学系列'
05 fun_test()
    运行错误:
```



为什么不是输出“Python 入门与实战”或“NameError”异常，而是“UnboundLocalError”异常？这是因为模块中的代码在执行之前，并不会经过预编译，但是函数体中的代码在运行前会经过预编译。换句话说，就是在函数被执行之前，Python 解释器就已经知道了函数中有哪些局部变量。所以在执行 03 语句时，Python 解释器知道 book\_name 是局部变量，但并未赋值，就引发了该异常（在引用前需要先赋值）。

## 本章小结

本章全面介绍了 Python 自定义函数的基础知识，包括函数的创建、调用、参数的类型、返回值、变量的作用域等。这些技术涉及到许多细节，如值传递和引用传递、传递参数的顺序，不注意这些细节，尽管可以编写出没有语法错误的程序，但是会出现得不到预期结果的逻辑错误，建议在编写函数时自己多设计一些测试用例，测试函数的健壮性，逐步养成良好的编程习惯。对于递归函数，由于 Python 对递归函数的一些限制，重点是理解它的思想，在具体使用时一定要对规模进行预测评估。匿名函数在入门阶段了解即可。

## 练习题

1. 自定义一个无参函数，打印 ‘I am a student’。
2. 自定义一个有参函数，计算  $x^2+y^2$ 。
3. 编写递归函数，计算  $1+2+3+4+\dots+n$ 。
4. 编写一个函数，计算 n 个数的最大值。
5. 自定义一个函数，已知圆的半径求圆的面积。
6. 水仙花数是指一个三位数，其各位数字立方和等于该数本身的数，请编写程序打印出 [100, 999] 之间的所有水仙花数。如 153 是一个水仙花数， $153=1^3+5^3+3^3$