Python语言

第六章　函数

6.1.1 函数的定义

一个程序可以按不同的功能实现拆分成不同的模块，而函数就是能实现某一部分功能的代码块。

在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号(:)，然后在缩进块中编写函数体，函数的返回值用return语句返回。

注意：Python是靠缩进块来标明函数的作用域范围的，缩进块内是函数体，这和其它高级编程语言是有区别的，比如：C/C++/java/R语言大括号{ }内的是函数体。

我们以自定义一个求正方形面积的函数area_of_square为例，示例代码如下：

```
def area_of_square(x):
    s = x * x
    return s
```

Python不但能非常灵活地定义函数，而且本身内置了很多有用的函数，可以直接调用。

6.1.2 全局变量

在函数外面定义的变量称为全局变量。全局变量的作用域在整个代码段（文件、模块），在整个程序代码中都能被访问到。在函数内部可以去访问全局变量。如下所示代码：

```
def foodsprice(per_price, number):
    sum_price = per_price * number
    print('全局变量PER_PRICE_1的值：', PER_PRICE_1)
    return sum_price


PER_PRICE_1 = float(input('请输入单价：'))
NUMBER_1 = float(input('请输入斤数：'))
SUM_PRICE_1 = foodsprice(PER_PRICE_1, NUMBER_1)
print('蔬菜的价格是：', SUM_PRICE_1)
```

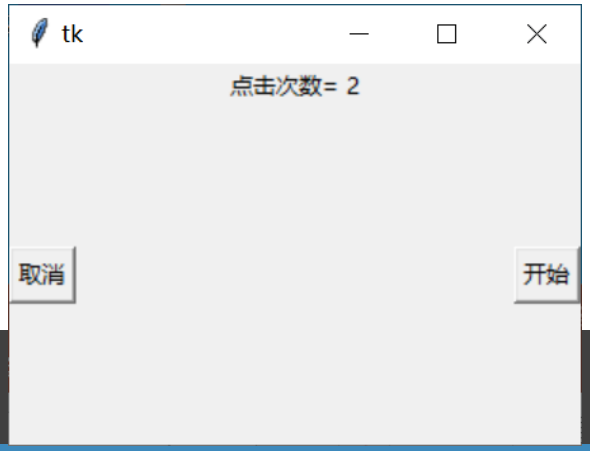
```
请输入单价：50
请输入斤数：3.5
全局变量PER_PRICE_1的值： 50.0
蔬菜的价格是： 175.0
```

在函数内部可以去访问全局变量，但不要去修改全局变量，否则会得不到想要的结果。这是因为在函数内部试图去修改一个全局变量时，系统会自动创建一个新的同名的局部变量去代替全局变量，采用屏蔽(Shadowing)的方式，当函数调用结束后函数的栈空间会被释放，数据也会随之释放。

如果要在函数内部去修改全局变量的值，并使之在整个程序生效，采用关键字global即可。

例子

```
from tkinter import *

root = Tk()
root.geometry('300x200+150+150')

lab_text = Label(root, text = '点击次数= 0')
lab_text.pack()    #独立.pack出来
count = 0

def start():
    #要在函数内部去修改全局变量的值 改lab_text中text的内容，就要global ！！！
    global lab_text,count
    count += 1
    lab_text.configure(text='点击次数= %d'%count)

but_OK = Button(root,text = '开始', command=start)
but_OK.pack(side= 'right')

but_Cancel = Button(root,text = '取消', command=start)
but_Cancel.pack(side = 'left')


root.mainloop()
```



6.1.3 局部变量

在函数内部定义的参数和变量称为局部变量，超出了这个函数的作用域局部变量是无效的，它的作用域仅在函数内部。如下所示代码：

```
def foodsprice(per_price,number):
    sum_price = per_price * number
    return sum_price
PER_PRICE_1 = float(input('请输入单价：'))
NUMBER_1 = float(input('请输入斤数：'))
SUM_PRICE_1 = foodsprice(PER_PRICE_1,NUMBER_1)
print('蔬菜的价格是：',SUM_PRICE_1)
print('局部变量sum_price的值：',sum_price)
```

代码运行结果如下：

请输入单价：12

请输入斤数：1.56

蔬菜的价格是： 18.72

Traceback (most recent call last):

File "G:/6_1_3.py", line 9, in <module> print('局部变量sum_price的值：',sum_price) NameError: name 'sum_price' is not defined

在上例中，我们试图在函数作用域外访问函数内的局部变量sum_price，程序运行到此处时报出了NameError的异常，提示变量sum_price没有定义。

第六章　函数

函数的参数就是使得函数个性化的一个实例。代码如下所示：

```
def MyFirstFunction(name_city):
    print(‘我喜欢的城市:’ + name_city)
```

运行结果如下：

```
>>> MyFirstFunction('南京')
    我喜欢的城市:南京
>>> MyFirstFunction('上海')
    我喜欢的城市:上海
```

在上例中，我们对函数MyFirstFunction的形参name_city赋予不同的实参“南京”、“上海”后，函数就输出不同的结果。

函数有了参数之后，函数的输出结果变得可变了，如果需要多个参数，函数用逗号“,” 隔开即可。

在Python中对函数参数的数量没有限制，但是定义函数参数的个数不宜太多，一般2~3个即可。在定义函数时，一般要把函数参数的意义注释清楚，便于阅读程序。

那什么是形参和实参呢?

函数定义时的小括号“()”内的参数叫形参。

而实参则是指函数在调用过程中传递进来的参数。

6.2.1 参数传递的方式

在Python中，将函数参数分为三类：位置参数、可变参数、关键字参数。

(1)位置参数

直接传入参数数据即可，如果有多个参数，位置先后顺序不能改变。

(2)可变参数

有2种传递方式：1是直接传入参数值；2是先封装成列表(list)或元组(tuple)，再在封装后的列表或元组前面添加一个星号“*”传入。

(3)关键字参数

有2种传递方式：1是直接传入参数值；2是可以先将参数封装成字典(dict)，再在封装后的字典前添加2个星号“**”传入。

6.2.2 位置参数和关键字参数

(1)位置参数

我们调用函数时，传入参数值按照位置顺序依次赋给参数，这样的参数称为位置参数。如下所示代码：

```
def Sub(x,y):
    return x-y
```

运行结果如下：

```
>>> Sub(100,30)
70
```

6.2.2 位置参数和关键字参数

上例中，Sub(x,y)函数有两个参数：x和y，这两个参数都是位置参数，调用函数时，传入的两个值按照位置顺序依次赋给参数x和y，得到的两数相减的结果是70。

如果交换了参数的位置，就会得到不同的结果，如上例中交换参数后的运行结果如下：

```
>>> Sub(30,100)

-70
```

从上面的运行结果可以看出，交换了参数顺序后的运行结果是-70，而不是我们期望的结果70。

6.2.2 位置参数和关键字参数

（2）关键字参数

关键字参数就是在函数调用的时候，通过参数名指定需要赋值的参数。通常我们在调用一个函数的时候，如果参数有多个，我们常常会混淆一个参数的顺序，达不到我们希望的效果。在Python中引入关键字参数就可解决这个潜在的问题。如下所示代码：

```
>>> def Subtraction(num_1,num_2):
        return (num_1 - num_2)
```

6.2.2 位置参数和关键字参数

如下所示代码：

```
def Subtraction(num_1,num_2):
        return (num_1 - num_2)
```

运行结果如下：

```
>>> Subtraction(34,11)
23
>>> Subtraction(11,34)
-23
>>> Subtraction(num_2=11,num_1=34)
23
```

6.2.2 位置参数和关键字参数

在上例中，我们调用函数Subtraction时：第1次调用函数Subtraction时，给2个参数顺序赋值34、11时得到的结果是23；第2次调用该函数时，交换了2个赋值参数的顺序得到的结果是-23，这不是所期望的结果；第3次调用该函数时，引用了关键字参数并对其分别赋值，虽然改变了顺序，但仍然得到了所期望的结果23。

6.2.3 默认值参数

我们在定义函数时给参数赋了一个初值，这样的参数称为默认值参数。应用默认值参数的意义在于，当在函数调用的时候忘记了给函数参数赋值的时候，函数就会自动去找它的初值，使用默认值来代替，而使函数调用不会出现错误。如下所示代码：

```
>>> def Subtraction(num_1=99,num_2=45):
        return (num_1 - num_2)
```

运行结果如下：

```
>>> Subtraction()
54
>>> Subtraction(46)
1
>>> Subtraction(46,12)
34
```

6.2.3 默认值参数

在上例中，函数Subtraction的功能为:返回两个数相减的结果，在定义函数时分别给2个参数num_1，num_2赋了初值99和45，分别做了3次调用：第1次调用时没有赋值，程序就引用了2个参数的默认值99，45，返回的结果是54；第2次调用时，给第1个参数赋值为46，程序就引用了第2个参数的默认值45，返回的结果是1；第3次调用时，给2个参数分别赋值为46和12，程序就没有引用函数定义的默认值，返回的结果是34。

6.2.4 可变参数

当在定义函数参数的时候，我们不知道究竟需要多少个参数的时候，只要在参数前面加上星号“*”即可，这样的参数称为可变参数。如下所示代码：

```
>>> def val_par(*param):
        print('第三个参数是：',param[2]);
        print('可变参数的长度是：',len(param));
```

6.2.4 可变参数

```
>>> def val_par(*param):
        print('第三个参数是：',param[2]);
        print('可变参数的长度是：',len(param));
```

运行结果如下：

```
>>> val_par('南京云创科技股份',345,9,9.8,2.37,'Python')
第三个参数是： 9
可变参数的长度是： 6
```

在上例中，我们定义函数val_par的参数param为可变参数，我们在调用该函数的时候就可以根据实际的应用来输入不同长度、不同类型的参数值。

6.2.4 可变参数

可变参数又称收集参数，是将一个元组赋值给可变参数。如果可变参数后面还有**其它参数**，在参数传递时要把可变参数后的参数作为**关键字参数**来赋值，或者在定义函数参数时要给它赋默认值，否则会出错。

如下所示代码：

```
>>> def val_par(*param,str1):
        print('第三个参数是：',param[2]);
        print('可变参数的长度是：',len(param));
```

6.2.4 可变参数

运行结果如下：

```
>>>  val_par('南京云创科技股份',345,9,9.8,2.37,'Python','函数')
SyntaxError: unexpected indent
>>> val_par('南京云创科技股份',345,9,9.8,2.37,'Python',str1='函数')
第三个参数是： 9
可变参数的长度是： 6
```

6.2.4 可变参数

在上例中，在定义函数val_par()时分别定义了1个可变参数param，1个普通参数str1，在第1次调用该函数的时候由于没有将可变参数后面的普通参数作为关键字参数来传值，导致程序运行时报错。在第二次调用该函数时将可变参数后的普通参数作为关键字参数传值(str1='函数')后，程序运行正常。

如下所示代码：

```
>>> def val_par(*param,str1='可变函数'):
        print('可变参数后的参数是：',str1);
        print('可变参数的长度是：',len(param));
```

6.2.4 可变参数

运行结果如下：

```
>>> val_par('南京云创科技股份',345,9,9.8,2.37,'Python')
可变参数后的参数是：可变函数
可变参数的长度是： 6
```

在上例中，在定义函数val_par()时分别定义了1个可变参数param，1个普通参数str1，并给参数str1赋了初值“可变函数”，在调用该函数的时候没有将可变参数后面的普通参数值作为关键字参数来传值，程序运行仍然正常，程序引用了函数的默认值参数。

6.2.5 函数的返回值

有些时候，需要函数返回一些数据来报告函数实现的结果。在函数中用关键字“return”返回指定的值。如下所示代码：

```
>>> def Subtraction(num_1,num_2):
        return (num_1 - num_2)
```

运行结果如下：

```
>>> print(Subtraction(65,23))
42
>>> Subtraction(34,11)
23
```

6.2.5 函数的返回值

函数中如果没有用关键字return指定返回值，则返回一个“None”对象。

如下所示代码：

```
>>> def test_return():
        print('Hello First1')
```

6.2.5 函数的返回值

运行结果如下：

```
>>> tempt = test_return()
Hello First1
>>> tempt
>>> print(tempt)
None
>>> type(tempt)
<class 'NoneType'>
```

6.2.5 函数的返回值

Python是动态的确定变量类型，Python没有变量，只有名字。Python可以返回多个类型的值。

如下所示代码：

```
>>> def back_test():
        return ['南京云创科技',3.67,567]
```

运行结果如下：

```
>>> back_test()
['南京云创科技', 3.67, 567]
```

6.2.5 函数的返回值

在上例中，Python返回多个值是列表数据。

如下所示代码：

```
>>> def back_test():
        return '南京云创科技',3.67,567
```

运行结果如下：

```
>>> back_test()
('南京云创科技', 3.67, 567)
```

在上例中，Python返回多个值是元组数据。

第六章　函数

6.3.1 函数的调用方法

要调用一个函数，需要知道函数的名称和参数。

函数分为自定义函数和内置函数。

自定义函数需要先定义再调用，内置函数直接调用，有的内置函数是在特定的模块下，这时需要用import命令导入模块后再调用。

我们可以在交互式命令行通过help(函数名)查看函数的帮助信息。

调用函数的时候，如果传入的参数数量不对，会报TypeError的错误，同时Python会明确地告诉你参数的个数。如果传入的参数数量是对的，但参数类型不能被函数所接受，也会报TypeError的错误，同时给出错误信息。

函数名其实就是指向一个函数对象的引用，可以把函数名赋给一个变量。

6.3.2 嵌套调用

允许在函数内部创建另一个函数，这种函数叫内嵌函数或者内部函数。内嵌函数的作用域在其内部，如果内嵌函数的作用域超出了这个范围就不起作用。如下所示代码：

```
>>> def function_1():
        print('正在调用function_1()...')
        def function_2():
            print('正在调用function_2()...')
        function_2()
```

6.3.2 嵌套调用

```
def function_1():
    print('正在调用function_1()...')
    def function_2():
        print('正在调用function_2()...')
    function_2()
```

运行结果如下：

```
>>> function_1()
正在调用function_1()...
正在调用function_2()...
>>> function_2()
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    function_2()
NameError: name 'function_2' is not defined
```

6.3.4 递归调用

递归是算法的范畴，从本质上讲不是Python的语法范围。

函数调用自身的行为是递归。

递归的条件

1、可以把要解决的问题转化为一个新问题，而这个新的问题的解决方法仍与原来的解决方法相同，只是所处理的对象有规律地递增或递减。

2、可以应用这个转化过程使问题得到解决。

3、必定要有一个明确的结束递归的条件。

Python默认递归深度100层（Python限制）。设置递归的深度的系统函数是：sys.setrecursionlimit(stepcount)。 参数：stepcount设置递归的深度。

递归有危险性：消耗时间和空间，因为递归是基于弹栈和出栈操作。递归忘掉返回使程序崩溃，消耗掉所有内存。

6.3.4 递归调用

递归实例

程序分析 斐波那契数列（Fibonacci sequence）

在数学上，

斐波纳契数列以如下被以递归的方法定义：

F（0）=0，

F（1）=1，

F（n）=F(n-1)+F(n-2)（n≥2，n∈N*）

从0,1开始，后面每一项等于前面两项之和。

对照递归条件，考虑 F（n）=F(n+1)+F(n+2)

```
#斐波那契数列（Fibonacci）
def Fib(n):
    '''
    斐波那契数列
    '''
    if (n<2):
        if (n == 0):
            return 0
        else :
            return 1
    else :
        return Fib(n-1)+Fib(n-2)
    #return 1 if n<=2 else Fib(n-1)+Fib(n-2)

# 输出前 10 项
for i in range (10):
    print(Fib(i), end = '  ')
print ()
help (Fib)
```

```
0  1  1  2  3  5  8  13  21  34
Help on function Fib in module __main__:

Fib(n)
    斐波那契数列
```

第六章　函数

6.4 实验

6.4.1 声明和调用函数

6.4.2 在调试窗口中查看变量的值

6.4.3 使用函数参数和返回值

6.4.4 使用闭包和递归函数

6.4.5 使用python的内置函数

第六章　函数

义函数时，需要确定函数名和参数个数；函数体内用return返回函数结果；函数没有return语句或return语句后面为空时，自动返回None。函数可以同时返回多个值。

默认值参数一定要用不可变对象，如果是可变对象，程序运行时会有逻辑错误。

命名的关键字参数是为了限制调用者可以传入的参数名，同时可以提供默认值。

使用递归函数的优点是逻辑简单清晰，缺点是过深的调用会导致栈溢出。

第六章　函数

习题：

1．在函数内部可以通过什么关键字来定义全局变量？

2．如果函数中没有return语句或者return语句不带任何返回值，那么该函数的返回值是什么？

感谢聆听