第6章 函数与模块

CONTENTS

内容提要

1.函数

- 函数定义

语法格式：

```
def 函数名（[参数表]）：
    函数体
   [return 返回值]
```

例：

```
def fun():
    print("This is a test function!")
    return
```

1.函数

- 函数调用

调用格式：

函数名（[实际参数表]）

例：

```
def ave(a,b,c):
    return (a+b+c)/3.0
x,y,z=eval(input("请输入三个数：  "))
print("三个数平均值=%f"%(ave(x,y,z)))
# ave(x,y,z)为函数调用
```

运行结果：

```
请输入三个数：1,2,3
三个数平均值=2.000000
```

1.函数

- 函数参数

定义时的参数称为形式参数（**形参**），调用时的参数称为实际参数（**实参**），函数调用通常有值传递和址传递两种参数传递方式。

(1)值传递方式

函数调用时，会为形参分配与实参不同的内存单元，并将实参的值复制给形参。在函数体内形参的改变对实参没有任何影响，传值的实参数据一般是数字、字符串或元组等。

1.函数

值传递方式示例

```
def swap(x,y):
    x,y=y,x
    print("x=",x,"y=",y)
a,b=eval(input("请输入两个数：  "))
swap(a,b)
print("a=",a,"b=",b)
```

运行结果：

请输入两个数：1,2

x= 2 y= 1

a= 1 b= 2

1.函数

- 函数参数

(2)址传递方式

函数调用时，将实参的**地址**传递给形参，形参和实参占用**同一段内存单元**，在函数体内形参的改变也就意味着对实参的改变。显然，这种方式要求实参是可变对象，故传址的实参数据一般是列表或字典等。

1.函数

址传递方式示例

```
def change(List):
    List.append(4)  #在函数体内增加了一个元素
    print("函数内：",List)
mylist=[1,2,3]
change(mylist)
print("函数外：",mylist)
```

运行结果：

函数内： [1, 2, 3, 4]

函数外： [1, 2, 3, 4]

1.函数

- 函数的默认参数

若函数调用时未传递实参，则按定义时的默认参数进行计算。用法参考示例：

```
def printInfo(name,age=20):
    print("姓名:",name,"年龄:",age)
printInfo(name="Zhang")      #此时按默认年龄参数输出
printInfo(name="Fang",age=22)
printInfo(age=18,name="Wang")  #按参数名传入参数时，参数顺序可以任意
```

运行结果：

```
姓名: Zhang 年龄: 20
姓名: Fang 年龄: 22
姓名: Wang 年龄: 18
```

1.函数

- 函数的不定长参数

Python程序中，用户可能需要一个函数能处理比最初声明时更多的参数，这些参数叫不定长参数，也称可变参数。这些可变参数被包装进一个元组或字典。

语法格式如下：

```
def 函数名（[普通参数][，*可变参数]）
    函数体
    [return 返回值]
```

1.函数

- 函数的不定长参数示例

```
def printInfo(arg1,*vartuple):
    print("输出:")
    print(arg1,end=' ')
    for var in vartuple:
        print(var,end=' ')
    print()
printInfo(10)        #10传递给普通参数arg1
printInfo(20,30,40)  #20传递给普通参数arg1，30和40传递给可变参数vartuple
```

运行结果：

```
输出:
10
输出:
20 30 40
```

1.函数

- 函数的返回值

return语句用来结束函数并将函数的运算结果返回给主调函数，一般返回一个值，当需要return返回多个值时，这些值就形成了一个元组，由圆括号括起、逗号分隔，如return (a,b,c)。

1.函数

- 变量作用域

变量作用域是指变量的作用范围。

(1) 局部变量

局部变量是指定义在函数内部的变量，只能在声明它的函数内部使用，当函数运行结束时，局部变量将不再存在。函数内部定义的变量，无需特别说明即为局部变量。

(2) 全局变量

全局变量是指定义在所有函数外部的变量，因此在程序执行全程有效。当函数内有未特别声明的同名局部变量时，局部变量有效。全局变量声明的语法形式：

global〈全局变量〉

1.函数

- 变量作用域

➢ 全局变量和局部变量示例

```
#exp6-9.py
def fun():
    global str    #声明全局变量
    str="Internal"
    print("函数内:",str)
str="external"
print("函数外:",str)
fun()
print("函数外:",str)
```

运行结果：

```
函数外: external
函数内: Internal
函数外: Internal
```

1.函数

- 匿名函数

用关键字lambda来创建，并可以赋给一个变量供调用，没有名字的函数。用于定义简单的、能够在一行内表示的函数。定义语法形式如下：

函数名=lambda params :expr

等价于：

```
def  函数名(params):
        函数体
        return 返回值
```

1.函数

- 匿名函数

示例：

```
sum = lambda a,b=100:a+b    #允许带默认参数
print("变量之和为:",sum(100,200))
print("变量之和为:",sum(100))
```

运行结果：

```
变量之和为: 300
变量之和为: 200
```

1.函数

- 特殊函数

(1)map函数

为Python的内置函数，其作用是将一个单参数函数依次作用到一个序列对象的每个元素上，并返回一个map对象作为结果，其中每个元素是原序列中元素经过该函数处理后的结果，该函数不对原序列对象做任何修改。使用语法格式：

map(func,seq)

示例：

```
def sqr(x):
    return x**2
item1=[1,2,3,4,5]
for i in map(sqr,item1):
    print(i,end=' ')
```

运行结果：

1 4 9 16 25

1.函数

- 特殊函数

(2) reduce函数

为Python的内置函数，其作用是将一个序列对象（列表、元组、字典或字符串等）中的所有数据进行下列操作：用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，以此类推，最后得到一个结果。

使用语法格式：

reduce(function , iterable[, initializer])

initializer为可选参数，用于设置初始值，若有，则第一次运算对象为初始值和第一个元素

1.函数

- reduce函数示例：

```
from functools import reduce #需要导入模板
def add(x,y):
    return x+y
print("累加结果:",end=' ')
print(reduce(add, [1,2,3,4]))
```

运行结果：

累加结果： 10

1.函数

- 特殊函数

(3) filter函数

为Python的内置函数，其作用是将一个单参数函数作用到一个序列上，返回该序列中使得该函数返回值为True的那些元素组成的列表、元组或字符串。使用语法格式：

filter(func, iterable)

其中，func是一个函数；iterable是一个序列对象。

1.函数

filter函数示例：

```
#exp6-15.py
def is_odd(n):
    return n%2==1
newlist1 = filter(is_odd,range(1,11))
 #通过自定义函数构造过滤器
print("奇数序列为:",end=' ')
for i in  newlist1:
    print(i,end=' ')
```

运行结果：

奇数序列为：1 3 5 7 9

1.函数

filter函数示例：

```
#exp6-15.py
newlist2 = filter(lambda x: x%2==0, range(1,11)) #通过lambda
函数构造过滤器
print("偶数序列为:",end=' ')
for i in  newlist2 :
    print(i,end=' ')
```

运行结果：

偶数序列为： 2 4 6 8 10

1.函数

- 递归函数

函数定义中出现直接或间接调用函数自身的方式，该函数就是递归函数。如：阶乘问题：

$$f(\mathrm{n}) = \mathrm{n}! = \begin{cases} 1 & \text{当}\mathrm{n} = 0\text{ 时} \\ n * (n-1)! & \text{当}\mathrm{n} > 0\text{时} \end{cases}$$

```
def fac(n):
    if n==0:
        return 1
    else:
        return fac(n-1)*n #递归调用
n=int(input("输入一个整数:"))
print("%d!=%d"%(n,fac(n)))
```

运行结果：

输入一个整数:5

5!=120

1.函数

- 递归函数示例

【例6-17】用递归函数求解Fabinacci数列。

$$f(n)=\begin{cases}1 & \text{当n}=1\text{时}\\ 1 & \text{当n}=2\text{时}\\ f(n-1)+f(n-2) & \text{当n}>2\text{时}\end{cases}$$

```
#exp6-17.py
def fun(n):
    if n==1 or n==2:
        f=1
    else:
        f= fun(n-1)+fun(n-2)
    return f
n=int(input("输入一个整数(n>1):"))
print("Fabinacci数列为:")
for i in range(1,n+1):
    print("%d"%(fun(i)),end=' ')
```

运行结果：

```
输入一个整数(n>1):10
Fabinacci数列为:
1 1 2 3 5 8 13 21 34 55
```

内置函数

Python 3解释器提供了近70个内置库函数（随着版本升级，数量还会增加），这些函数不需要导入相关模块，可直接使用。

内置函数

部分内置函数

函数名	功 能 描 述
abs()	获取绝对值
bin()	将十进制数分别转换为 2 进制
bool()	测试一个对象是 True 还是 False
chr()	查看十进制数对应的 ASCII 字符
cmp()	用于比较两个元组元素
compile()	将字符串编译成 Python 能识别或可以执行的代码，也可以将文字读成字符串再编译
complex()	创建一个值为 real＋imag * j 的复数或者转化一个字符串或数为复数
dict()	创建数据字典
divmod()	分别取商和余数
eval()	将字符串 str 当成有效的表达式来求值并返回计算结果
filter()	过滤器，构造一个序列，等价于[item for item in iterables if function(item)]
float()	将一个字符串或整数转换为浮点数
format()	格式化输出字符串
hash()	用于获取取一个对象（字符串或者数值等）的哈希值
hex()	将十进制数分别转换为 16 进制
input()	获取用户输入内容
iter()	用来生成迭代器
len()	返回对象长度

2.模块

- 概念

在Python中，模块是一个包含变量、函数或类的定义以及各种语句的程序文件。模块可以被其它程序引入，以使用该模块中的函数等功能。

2.模块

- 模块调用示例

```
#mymodule.py
import time          #导入系统标准库模块
def now_time():   #设计显示系统当前时间的函数
    nt=time.localtime()
    s=("%02d:%02d:%02d"%nt[3:6])
    print(s)
    time.sleep(1)
```

```
#test-module.py
import mymodule
print("现在的时间是:",end=' ')
mymodule.now_time()   #通过其它模块的函数，显示系统当前时间
```

运行结果：

现在的时间是: 09:23:07

2.模块

- 模块导入方法

有标准模块、自定义模块和第三方库之分。

(1) import　模块名

解释器会按系统搜索路径将指定模块导入当前程序中，这种方式，在使用被导入模块中的函数时，需用“模块名.函数名”的格式。

(2) from 模块名　import　函数

解释器会将模块中的指定函数单个导入到当前程序中，这种方式，在使用被导入模块中的函数时，前面无需加“模块名.”，直接使用函数名。

(3) from 模块名　import　*

解释器会将模块中的所有函数导入到当前程序中，模块中的所有函数可以在本程序直接使用。

3.命名空间

在编写Python程序的过程中，如果要使用变量和函数，都需要先对变量和函数命名后才能使用。

Python会把命名后的变量和函数分配到不同的命名空间(Namespace)，并通过名称来识别它们。

Python区分不同的命名空间有两个作用：一个作用是不同的命名空间对应不同的作用域；另外一个作用是防止命名冲突。

3.命名空间

- 命名空间的分类：

（1）局部命名空间(Local)：每个函数所拥有的命名空间，记录了函数中定义的所有变量，包括函数的参数、内部定义的局部变量。

（2）全局命名空间(Global)：每个模块加载执行时创建的，记录了模块中定义的变量，包括模块中定义的函数、类、其他导入的模块、模块级的变量与常量。

（3）内建命名空间(Built-in)：是Python自带的，任何模块均可以访问，放着内置的函数和异常。

3.命名空间

- 命名空间的生命周期：

在Python程序中不同时刻创建的命名空间会有不同的生命周期。具体体现在以下几点：

（1）内建命名空间在Python解释器启动时创建，并会一直保留下去。

（2）模块的全局命名空间在导入模块时创建，一直保持到解释器退出。

（3）当调用函数时创建一个局部命名空间，当函数返回结果或抛出异常时，删除局部命名空间。

3.命名空间

- 命名空间示例

```
#module.py
name="module_name"
def mo_fun():
    print("函数mo_fun:")
    print("变量name:",name)
```

```
#exp6-19.py
from module import mo_fun
name="current_name"
def test_fun():
    print("当前模块函数test_fun:")
    print("变量name:",name)
    mo_fun()
test_fun()
```

运行结果：

```
当前模块函数test_fun:
变量name: current_name
函数mo_fun:
变量name: module_name
```

3.命名空间

- 命名空间示例

分析：以下代码没有全局声明global i时会报错，给出以下错误提示：

UnboundLocalError: local variable 'i' referenced before assignment。原因是虽定义了全局变量i，但函数fun()内的变量i是局部变量，因没有初值，故不能加1操作。

```
i=1
def fun():
#   global i
    i=i+1
    print("变量i:",i)
fun()
```

运行结果：

```
变量i: 2
```

小结

- 函数：介绍了函数的定义、参数传递、返回值、变量作用域、匿名函数、递归函数和几个特殊函数；
- 内置函数：列举了部分常用内置函数。
- 命名空间：防止同一个作用域里命名冲突。