

Chap06–Function

College of Computer Science
Nankai University

Outline

- ▶ 6.1 函数概述
 - 6.1.1 函数定义与调用
 - 6.1.2 函数嵌套定义
 - 6.1.3 函数递归调用
- ▶ 6.2 函数参数
 - 6.2.1 位置参数
 - 6.2.2 默认值参数
 - 6.2.3 关键字参数
 - 6.2.4 可变长度参数
- ▶ 6.3 变量作用域
- ▶ 6.4 lambda函数

函数定义与调用

- ▶ 在完成一项较复杂的任务时，我们通常会将任务分解成若干个子任务，通过完成这些子任务逐步实现任务的整体目标。
- ▶ 实际上，这里采用的就是结构化程序设计方法中模块化的思想。
- ▶ 在利用计算机解决实际问题时，也通常是将原始问题分解成若干个子问题，对每个子问题分别求解后再根据各子问题的解求得原始问题的解。

函数定义与调用

- ▶ Python语言中使用函数分为两个步骤：定义函数和调用函数。
 - 定义函数，即根据函数的输入、输出和数据处理完成函数代码的编写。定义函数只是规定了函数会执行什么操作，但并不会真正去执行。
 - 调用函数，即真正去执行函数中的代码，是指根据传入的数据完成特定的运算，并将运算结果返回到函数调用位置的过程。

函数定义与调用

```
1      def CalCircleArea():
2          s=3.14*3*3
3          print('半径为3的圆的面积为：%.2f'%s)
4      CalCircleArea()
```

- ▶ 函数形参不需要声明类型，也不需要指定函数返回值类型
- ▶ 即使函数不需要接收任何参数，也必须保留一对空的圆括号
- ▶ 括号后面的冒号必不可少
- ▶ 函数体相对于def关键字必须保持一定的空格缩进
- ▶ Python允许嵌套定义函数

函数定义与调用

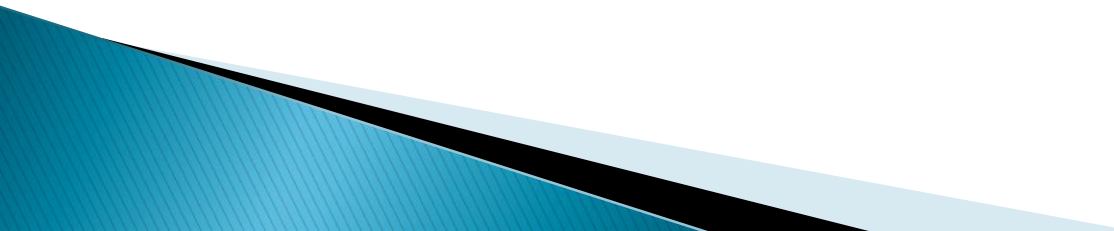
- ▶ 通过函数的参数列表，可以为函数传入待处理的数据，从而使得一个函数更加通用。
- ▶ 例如，对于计算圆面积的函数CalCircleArea，可以将半径 r 作为参数，这样每次调用CalCircleArea函数时只要传入不同的半径值，函数就可以自动计算出传入半径所对应的圆的面积。

函数定义与调用-形参

- ▶ 形参的全称是形式参数，即定义函数时函数名后面的一对小括号中给出的参数列表。
- ▶ 形参只能在函数中使用，其作用是接收函数调用时传入的参数值（即后面要介绍的实参），并在函数中参与运算。
- ▶ 例：圆面积函数和长方形面积函数的定义。

函数定义与调用-形参

```
1    def CalCircleArea(r): #定义名字为CalCircleArea的函数
2        s=3.14*r*r #计算半径为r的圆的面积
3        print('半径为%.2f的圆的面积为：%.2f'%(r,s)) #将计算结果输出
4    def CalRectArea(a,b): #定义名字为CalRectArea的函数
5        s=a*b #计算边长分别为a和b的长方形的面积
6        print('边长为%.2f和%.2f的长方形的面积为：%.2f'%(a,b,s)) #
    将计算结果输出
```



函数定义与调用-实参

- ▶ 实参的全称是实际参数，即在调用函数时函数名后面的一对小括号中给出的参数列表。
- ▶ 当调用函数时，会将实参的值传递给对应的形参，函数中再利用形参做运算、得到结果。
- ▶ 例：圆面积函数和长方形面积函数的调用。

```
1  a=eval(input('请输入圆的半径: '))  
2  CalCircleArea(a)  
3  x=eval(input('请输入长方形的一条边长: '))  
4  y=eval(input('请输入长方形的另一条边长: '))  
5  CalRectArea(x,y)
```

函数例子

- ▶ 一个返回值

```
def add(a,b):  
    c=a+b  
    return c
```

- ▶ 多个返回值

```
def add_mul(a,b):  
    add=a+b  
    mul=a*b  
    return add,mul
```

- ▶ 无返回值

```
def say_hello(your_name):  
    print("Hello,%s!" %your_name)
```

函数例子

定义函数：

```
def add(a,b):  
    c=a+b  
    return c
```

调用函数：

```
add(1,2)
```

运行结果： 3

函数例子

- ▶ 通过多元赋值语句，同时获取多个返回值

```
def add_mul(a,b):                                #定义函数
    add=a+b
    mul=a*b
    return add,mul

x,y=add_mul(1,2)                                  #调用函数
print("add:",x,";mul:",y)
```

运行结果：

```
add: 3 ;mul: 2
```

函数例子

▶ 无形参函数

```
def say_hello():           #定义函数
    print("Hello! ")

say_hello()                #调用函数
```

调用函数时，函数名后面必须有小括号，
即使没有参数，这个小括号也不能省略。

运行结果：

```
Hello!
```

函数嵌套定义

- ▶ Python允许函数的嵌套定义，在函数内部可以再定义另外一个函数。

```
>>> def abc(it, op, va):  
    if op not in '+-*/':  
        return 'Error'  
    def nested(item):  
        return eval(str(item)+op+str(va))  
    return map(nested, it)
```

```
>>> list(abc(range(5), '+', 5))
```

#调用外部函数，不需要

关心其内部实现

```
[5, 6, 7, 8, 9]
```

```
>>> list(abc(range(5), '-', 5))
```

```
[-5, -4, -3, -2, -1]
```

函数递归调用

- ▶ 函数的递归调用是函数调用的一种特殊情况，函数调用自己，自己再调用自己，自己再调用自己，...，当某个条件得到满足的时候就不再调用了，然后再一层一层地返回直到该函数第一次调用的位置。

```
def factorial (n):  
    if n==1:  
        return 1;  
    else:  
        return n*factorial(n-1)
```

```
factorial(5)
```

Outline

- ▶ 6.1 函数概述
 - 6.1.1 函数定义与调用
 - 6.1.2 函数嵌套定义
 - 6.1.3 函数递归调用
- ▶ 6.2 函数参数
 - 6.2.1 位置参数
 - 6.2.2 默认值参数
 - 6.2.3 关键字参数
 - 6.2.4 可变长度参数
- ▶ 6.3 变量作用域
- ▶ 6.4 lambda函数

函数参数

- ▶ 函数定义时圆括弧内是使用逗号分隔开的形参列表（parameters），函数可以有多个参数，也可以没有参数，但定义和调用时一对圆括弧必须要有，表示这是一个函数并且不接收参数。
- ▶ 调用函数时向其传递实参（arguments），根据不同的参数类型，将实参的引用传递给形参。
- ▶ 定义函数时不需要声明参数类型，解释器会根据实参的类型自动推断形参类型。

函数参数

- 函数的参数如果按值传递，形参和实参分别存储，相互独立。在内部函数改变形参的值时，实参的值不会随之改变。

```
def func(num):  
    num+=1  
    print("num:",num)  
  
a=10  
func(a)  
print("a:",a)
```

#定义函数
#函数内部改变形参的值
#打印形参

#调用函数，a为实参
#输出变量a

运行结果：

```
num:11  
a:10
```

函数参数

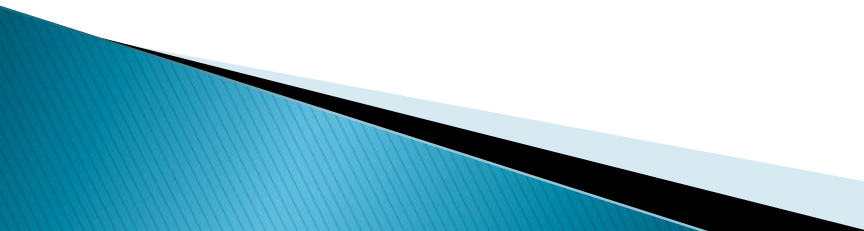
- 函数的参数如果传递的实参是可变序列，并且在函数内部使用下标或可变序列自身的方法增加、删除元素或修改元素时，实参也得到相应的修改。

```
>>> def change(v):  
    v[0] += 1  
>>> a = [2]  
>>> change(a)  
>>> a  
[3]
```

```
>>> def change(v, item):  
    v.append(item)  
>>> a = [2]  
>>> change(a,3)  
>>> a  
[2, 3]
```

函数参数

```
>>> def modify(d):  
    d['age'] = 38  
>>> a = {'name':'Dong', 'age':37, 'sex':'Male'}  
>>> a  
{'age': 37, 'name': 'Dong', 'sex': 'Male'}  
>>> modify(a)  
>>> a  
{'age': 38, 'name': 'Dong', 'sex': 'Male'}
```



位置参数

- ▶ 位置参数（positional arguments）是比较常用的形式，调用函数时实参和形参的顺序必须严格一致，并且实参和形参的数量必须相同。

```
>>> def demo(a, b, c):  
    print(a, b, c)
```

```
>>> demo(3, 4, 5)
```

```
3 4 5
```

```
>>> demo(3, 5, 4)
```

```
3 5 4
```

默认值参数

- ▶ 函数的默认值参数就是缺省值参数，即当调用函数时，如果没有为某些形参传递对应的实参，则这些形参会自动使用默认参数值。
- ▶ 注意
 - 对于没有默认值参数的函数，在调用函数时必须为其指定实参，否则运行程序会报错。
 - 定义带有默认值参数的函数时，任何一个默认值参数右边都不能再出现没有默认值的普通位置参数，否则会提示语法错误。

默认值参数-例子

```
def add(a,b=2):  
    return a+b
```

```
print(add(1))  
print(add(2,3))
```

运行结果

```
3  
5
```

```
def add (a, b=1, c=2):  
    return a+b+c
```

关键字参数

- 在调用函数时，除了前面那种通过位置来体现实参和形参的对应关系的方法（即位置参数），还有一种使用关键字参数的方法，其形式为“形参=实参”。实参顺序可以和形参顺序不一致，但不影响参数值的传递结果。

```
>>> def demo(a, b, c=5):  
    print(a, b, c)  
>>> demo(3, 7)  
3 7 5  
>>> demo(a=7, b=3, c=6)  
7 3 6  
>>> demo(c=8, a=9, b=0)  
9 0 8
```

位置参数和关键字参数可以混合使用，但位置参数必须在前、关键字参数在后

可变长度参数

- ▶ 可变长度参数，即在调用函数时可以接收任意数量的实参，这些实参在传递给函数时会被封装成元组（位置参数）或字典（关键字参数）形式。

def 函数名([普通形参列表,] *可变长度参数 [普通形参列表]):

函数体

或

def 函数名([普通形参列表,] **可变长度参数):

函数体

提示：“*可变长度参数”表示参数接受的是一组位置参数(放在元组中)；

“**可变长度参数”表示参数接受是一组关键字参数(放在字典中)。

可变长度参数

- ▶ 如果一个函数所需要的参数已经存储在了列表、元组或字典中，则可以直接从列表、元组或字典中拆分出来函数所需要的这些参数。
- ▶ 其中列表、元组拆分出来的结果作为位置参数，而字典拆分出来的结果作为关键字参数。

```
1 def SumVal(*args):  
2     sum=0  
3     for i in args:  
4         sum+=i  
5     print('求和结果为:',sum)  
6 ls=[3,5.2,7,1]  
7 SumVal(*ls)
```

可变长度参数-例子

```
>>> def demo(*p):  
    print(p)
```

```
>>> demo(1,2,3)
```

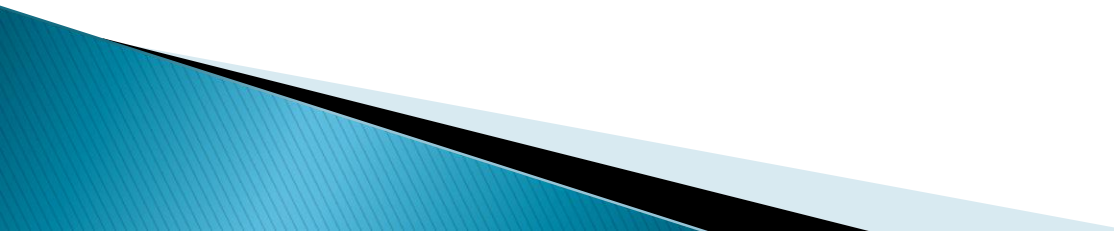
```
(1, 2, 3)
```

```
>>> demo(1,2)
```

```
(1, 2)
```

```
>>> demo(1,2,3,4,5,6,7)
```

```
(1, 2, 3, 4, 5, 6, 7)
```



可变长度参数-例子

```
>>> def demo2(**p):  
    for i in p.items():  
        print(i)
```

```
>>> demo2(x=1,y=2,z=3)  
(  
('x', 1)  
('y', 2)  
('z', 3)
```

Outline

- ▶ 6.1 函数概述
 - 6.1.1 函数定义与调用
 - 6.1.2 函数嵌套定义
 - 6.1.3 函数递归调用
- ▶ 6.2 函数参数
 - 6.2.1 位置参数
 - 6.2.2 默认值参数
 - 6.2.3 关键字参数
 - 6.2.4 可变长度参数
- ▶ 6.3 变量作用域
- ▶ 6.4 lambda函数

变量作用域

- ▶ 变量的**作用域**是指变量的作用范围，即定义一个变量后，在哪些地方可以使用这个变量。
 - 不同作用域内变量名可以相同，互不影响
- ▶ 按照作用域的不同，Python中的变量可分为**局部变量**和**全局变量**。
 - 局部变量(local variable): 在函数中定义的变量，仅在定义它的函数内部有效。
 - 全局变量(global variable): 在函数体之外定义的变量，在定义后的代码中都有效，包括在它之后定义的函数体内。
- ▶ 局部变量的读写比全局变量速度快，应优先考虑使用。

变量作用域

```
def setNumber():  
    a= 9  
    a=a+1  
    print("setNumber:", a)
```

```
setNumber()
```

运行结果：

```
setNumber: 10
```

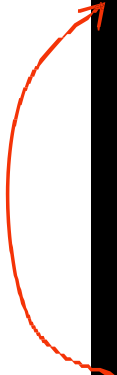
定义局部变量的函数中，**只有局部变量是有效的**

```
>>>print(a)
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-9-5315f3e3adca> in <module>()  
----> 1 print(a)
```

```
NameError: name 'a' is not defined
```

变量作用域



```
a=100                                     #定义全局变量
def setNumber():                          #定义函数
    a= 9                                  #定义局部变量
    a=a+1
    print("setNumber:", a)               #打印局部变量

setNumber()                               #调用函数，打印局部变量
print(a)                                 #打印全局变量
```

运行结果：

```
setNumber: 10
100
```

- 在函数外部定义的变量a是**全局变量**，当它与函数内部定义的局部变量同名时，在函数内部失效。
- 在函数内定义的变量a是**局部变量**，它只在函数体内局部有效，并不影响全局变量a的取值。局部变量会在自己的作用域内隐藏同名的全局变量。

变量作用域-nonlocal

- 在Python中，函数的定义可以嵌套，即在一个函数的函数体中可以包含另一个函数的定义。通过nonlocal关键字，可以使内层的函数直接使用外层函数中定义的变量。

```
1      def outer(): #定义函数outer
2          x=10 #定义局部变量x并赋为10
3          def inner(): #在outer函数中定义嵌套函数inner
4              x=20 #将x赋为20
5              print('inner函数中的x值为：',x)
6          inner() #在outer函数中调用inner函数
7          print('outer函数中的x值为：',x)
8      outer() #调用outer函数
```

运行结果：
inner函数中的x值为： 20
outer函数中的x值为： 10

变量作用域-nonlocal

```
1      def outer(): #定义函数outer
2          x=10 #定义局部变量x并赋为10
3          def inner(): #在outer函数中定义嵌套函数inner
4              nonlocal x #nonlocal声明
5              x=20 #将x赋为20
6              print('inner函数中的x值为：',x)
7          inner() #在outer函数中调用inner函数
8          print('outer函数中的x值为：',x)
9      outer() #调用outer函数
```

运行结果： inner函数中的x值为： 20
 outer函数中的x值为： 20

变量作用域-global

- ▶ Python中定义函数时，若想在函数内部对函数外的变量进行操作，就需要在函数内部声明其为global。

```
>>> x = 1
```

```
>>> def func():
```

```
    x = 2
```

```
>>> func()
```

```
>>> print(x)
```

```
1
```

变量作用域-global

```
>>> x = 1
```

```
>>> def func():  
    global x  
    x = 2
```

```
>>> func()
```

```
>>> print(x)
```

```
2
```

global需要在函数内部声明，若在函数外声明，则函数依然无法操作x。

Outline

- ▶ 6.1 函数概述
 - 6.1.1 函数定义与调用
 - 6.1.2 函数嵌套定义
 - 6.1.3 函数递归调用
- ▶ 6.2 函数参数
 - 6.2.1 位置参数
 - 6.2.2 默认值参数
 - 6.2.3 关键字参数
 - 6.2.4 可变长度参数
- ▶ 6.3 变量作用域
- ▶ 6.4 lambda函数

Lambda函数

- ▶ lambda函数也称为匿名函数，是一种不使用def定义函数的形式，其作用是能快速定义一个简短的函数。
 - 适合需要一个函数作为另一个函数参数的时候
- ▶ lambda函数的函数体只是一个表达式，所以lambda函数通常只能实现比较简单的功能。
 - lambda函数只可以包含一个表达式，该表达式的计算结果可以看作是函数的返回值，不允许包含复合语句，但在表达式中可以调用其他函数。

Lambda函数

- ▶ lambda [参数1[, 参数2, ..., 参数n]]: 表达式
 - 冒号后面的表达式的计算结果即为该lambda函数的返回值。

- ▶ 例子

```
>>> f = lambda x, y, z: x+y+z      #可以给lambda表达式起名字
```

```
>>> f(1,2,3)                      #像函数一样调用
```

```
6
```

```
>>> g = lambda x, y=2, z=3: x+y+z  #参数默认值
```

```
>>> g(1)
```

```
6
```

```
>>> g(2, z=4, y=5)                #关键字参数
```

```
11
```

Lambda函数

```
>>> L = [1,2,3,4,5]
>>> print(list(map(lambda x: x+10, L)))
[11, 12, 13, 14, 15]
>>> L
[1, 2, 3, 4, 5]
>>> def demo(n):
    return n*n
```

```
>>> demo(5)
25
>>> a_list = [1,2,3,4,5]
>>> list(map(lambda x: demo(x), a_list)) #在lambda表达式
中调用函数
[1, 4, 9, 16, 25]
```


Lambda函数

```
>>> data = list(range(20))          #创建列表
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> import random
>>> random.shuffle(data)            #打乱顺序
>>> data
[4, 3, 11, 13, 12, 15, 9, 2, 10, 6, 19, 18, 14, 8, 0, 7, 5, 17, 1, 16]
>>> data.sort(key=lambda x: x)      #和不指定规则效果一样
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```



Lambda函数

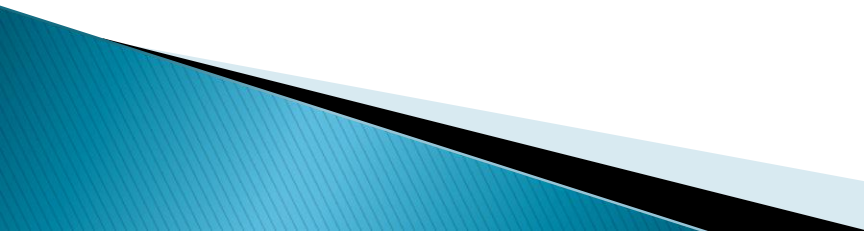
```
>>> import random
```

```
>>> x = [[random.randint(1,10) for j in range(5)] for i in  
range(5)]
```

```
>>> y = sorted(x, key=lambda item: (item[1], item[4]))
```

#按子列表中第2个元素升序、第5个元素升序排序

```
>>> for item in y:  
    print(item)
```



Thanks for listening

College of Computer Science
Nankai University