# 第五章 Python函数编程

本节所讲内容：

5.1 函数定义知识点讲解

5.2 函数的调用

5.3 函数划分

5.4 匿名函数lambda

5.5 函数作用域

5.6 实战：利用函数式编程，爬取百度图片

## 5.1 函数定义知识点讲解

### 5.1.1 Python 函数的定义

函数（function [ˈfʌŋkʃn]:功能）：是一个可以被重复调用的带有一个入口和一个出口的固定的程序段.

函数的定义：(function [ˈfʌŋkʃn]) 代码块。计算机的函数，是一个固定的一个程序段，或称其为一个子程序，它在可以实现固定运算功能的同时，还带有一个入口和一个出口，所谓的入口，就是函数所带的各个参数，我们可以通过这个入口，把函数的参数值代入子程序，供计算机处理；所谓出口，就是指函数的函数值，在计算机求得之后，由此口带回给调用它的程序。

简单来说就是将我们要执行的代码进行结构的整合，形成可被重复调用的代码块

要想程序打得好，就要学会函数，函数在手天下我有，日出东方，唯我不败（要有自信哦）



函数的优点：

1、减少冗余代码

2、代码结构清晰

3、保持代码的一致性

函数编写：

语法：

def 函数名(参数):

‘函数参数’

代码块

return

1、关键字: def

2、函数名：

命名规范：

1、字母开头

2、不允许有关键字

3、不允许有特殊符号

4、不允许莫名其妙的函数名 a,b

5、参数：参数是定义在参数括号里，由调用时传入，作用在函数内部的变量

6、如果有参数，写在参照括号里

7、如果没有，写空括号

## 5.2 函数的调用

### 5.2.1 函数调用

函数在没有调用之前不会执行

函数名加括号，并且进行对应的传参的形式

在定义函数的参照括号内定义的参数 我们称之为 形参

在调用函数的时候我们传递值 我们称之为 实参

def func(a):  
 *'这是函数的练习'* print('for is cool ')  
 return a + 1   
 *# print('bai hu zi ')*func(1)  
print(func(1))

## 5.3 函数的划分

按照参数类型划分：位置参数，关键字参数，默认参数，参数组

### 5.3.1 位置参数

位置传参是我们在传参的时候，实参传递的顺序按照形参定义的顺序进行传递的传参方式。

def func(name,city):  
 print('I am %s,I am from %s'%(name,city))  
func('for','xuegod')  
func('xuegod','for')

输出结果：

i am xuegod,i am from for

i am for ,i am from xuegod

### 5.3.2 关键字参数

关键字传参是我们在传参的时候，以形参等于实参的形式忽略形参定义的顺序进行传参的传参方式

因为 Python 解释器能够用参数名匹配参数值。  
def func(name,city):  
 print('I am %s,I am from %s'%(name,city))  
func(name='for',city='xuegod')  
func(city='xuegod',name='for')

输出结果：

i am for ,i am from xuegod

### 5.3.3 默认值参数、缺省参数（面试题）

默认值传参是在我们定义参数时候，我们给形参一个默认值，在我们调用函数的时候，如果不给有默认值的形参传参，会自动采用默认值。  
def func(name,city='beijing'):  
 print('I am %s,I am from %s'%(name,city))  
  
func(name='for')  
func(for', 'xuegod')

输出结果：

i am for ,i am from beijing

i am for ,i am from xuegod

注意：默认值参数必须写在正常参数的后面，

错误示范：

def func(city='beijing',name):  
 print('I am %s,I am from %s'%(name,city))

输出结果：

def func(city='beijing',name):

^

SyntaxError: non-default argument follows default argument

### 5.3.4 参数组（不定长参数 面试）

处理比当初声明时更多的参数，会将传入的参数变成元组（\*args）或者字典（\*\*kwargs），声明的时候不用命名

1. 元组参数组

通过给形参前面添加\*使参数变成一个元组，所有传递的参数变成元组的元素

def say\_hello(\*args):  
 print(args)  
say\_hello()  
say\_hello(1)  
say\_hello(1,2,3,4,5)

运行结果如下:

()  
(1,)

(1,2,3,4,5)

1. 字典参数组

通过给形参前面添加\*\*使参数变成一个字典，所有传递的参数变成字典的键值对，这里传参要求键等于值的形式。

def say\_hello(\*\*kwargs):  
 print(kwargs)  
say\_hello()  
say\_hello(a =1)  
say\_hello(a =1,b = 2,c = 3)  
*# say\_hello()*

运行结果如下：

{}

{‘a’: 1 }

{‘a’:1,’c’:3,’b’:2}

案例：

def say\_hello(e,f,p = 2,\*args,\*\*kwargs):  
 a = args  
 b = kwargs  
 print(a,b)  
say\_hello(1,2,3,a=1)

运行结果如下：

(1, 2) {'a': 1}

不定长参数在python2和python3 的区别

Python2:

def say\_hello(b=1,\*arg,\*\*kwargs):

print arg

python3：

def say\_hello(\*arg ,a=1,\*\*kwargs):

print(arg)

在python2中参数定义的顺序必须是：必选参数、默认参数、可变参数和关键字参数。而python3中却可以可变参数写在默认参数前面。

## 5.4 匿名函数lambda（面试）

Lambda函数，即Lambda 表达式(lambda expression)，是一个匿名函数（不存在函数名的函数），Lambda表达式基于数学中的λ演算得名，直接对应于其中的lambda抽象(lambda abstraction)。

好处：  
1、lambda 函数比较轻便，即用即删除，很适合需要完成一项功能，但是此功能只在此一处使用，  
连名字都很随意的情况下；  
2、匿名函数，一般用来给 filter， map 这样的函数式编程服务;  
3、作为回调函数，传递给某些应用，比如消息处理

4、不用考虑命名冲突问题

语法：lambda 参数:表达式(block)

### 5.4.1 lambda 参数:表达式(block)

参数：可选，通常以逗号分隔的变量表达式形式，也就是位置参数

表达式：不能包含循环、return，elif ，可以包含if.

L = lambda x:x\*x  
print(L(5))  
print(L)  
def L(x):  
 return x\*x  
print(L(5))

L = lambda x : 'x>10' if x >10 else 'x<10'  
print(L(5))  
*#函数实现*def L(x):  
 if x > 10:  
 return 'x>10'  
 else:  
 return 'x<10'  
print(L(11))

不支持return

>>>L= lambda x: return 'x>10' if x>10 else 'x<10'

错误示范：

>>> File "<input>", line 1

L= lambda x: return 'x>10' if x>10 else 'x<10'

^

SyntaxError: invalid syntax

invalid [ɪnˈvælɪd]：无效

syntax [ˈsɪntæks]：句法

注意：在python2中lambda 不支持print，但在python3中支持print（）

例子：

比如现在要对一个列表中的元素进行平方

那平方之前我们先讲一个map函数

1、map()

map () 是python的内置函数，它接受一个函数和一个可迭代对象，并且通过函数依次作用在可迭代对象的每个元素上，得到一个新的对象。

这个是普通函数的写法

比如现在要对一个列表中的元素进行平方

a = [1,2,3,4]  
b = []  
*#返回平方*def func(x):  
 return x\*x  
*#遍历执行*for i in a:  
 b.append(func(i))*#重复调用*print(b)

#list（a）

输出结果：

[1 4 9 16]

map

def func(x):  
 return x \* x  
a = map(func,[1,2,3,4])  
print(list(a))  
*# print(a)  
# for i in a:  
# print(i,end='')*

*#lambada 如何实现*b = map(lambda x : x\*x,[1,2,3,4])  
print(list(b))

输出结果：

[1 4 9 16]

看用lambda怎么写

b = map(lambda x:x\*x,[1,2,3])  
 for i in b:  
 print(i)

练习：给你两个列表a = [1,2,3], b = [4,5,6] 把两者中每个元素的数值相加在一起c=[5,7,9]，你该如何处理？

a = [1,2,3]  
b = [4,5,6]  
c = map(lambda x,y:x+y,a,b)  
print(list(c))

注意：即使lambda简洁好用，但是不建议大家滥用

1、好多人觉得方便，容易滥用；

2、用多了，程序可读性很差；

3、每个人对抽象层级的接受和忍耐不同；

## 5.5 函数作用域

### 5.5.1 变量起作用的范围

L local局部作用域，本地作用域

E enclosing嵌套作用域

G global全局作用域

B built-in内建作用域

for，if，while这些流程控制不会形成自己的作用域

import sys  
*# 得到所有该模块的方法 内建作用域*print(dir(sys))  
*# 全局*name = 'while'  
def outer():  
 *# name = 'for'#嵌套作用域* def inner():  
 *#本地作用域  
 # name = 'django'* age = 18  
 print(name)  
 print(age)  
 inner()  
outer()

什么叫局部变量，全局变量？

局部变量是指在程序中只在特定过程或函数中可以访问的变量。

如果一个变量，既能在一个函数中使用，也能在其他的函数中使用，这样的变量就是全局变量

总结：

局部变量，就是在函数内部定义的变量。

不同的函数，可以定义相同的名字的局部变量，但是各用个的不会产生影响。

局部变量的作用，为了临时保存数据需要在函数中定义变量来进行存储，这就是它的作用。

### 5.5.2 声明变量

global声明全局变量，nonlocal非本地变量

*#作用于全局 全局变量*name = 'while'  
def outer():  
 name = 'for'*#嵌套作用域* def inner():  
 *# global name* nonlocal name  
 name = 'for is cool'  
 age = 18  
 print(name)  
 print(age)  
 inner()  
 print(name)  
outer()

输出结果：

for

for is cool

for is cool

在学习作用域的过程当中我们要认识到变量查找遵循就近原则，由内到外的查找。

注意：如果c是一个列表，字典这些可变的类型的数据。那还要声明global？

解答：如果是可变类型的数据，不用声明global。

c = [1,2,3]  
d = {}  
e = set()  
def test01():  
 c.append('5')  
 d.update({'name':'for'})  
 e.add(1)  
test01()  
print(c,d,e)

运行结果：

[1, 2, 3, '5'] {'name': 'for'} {1}

总结：

在函数中不使用global声明全局变量时不能修改全局变量的,其本质是不能修改全局变量的指向，即不能将全局变量指向新的数据。

对于不可变类型的全局变量来说，因其指向的数据不能修改，所以不使用global时无法修改全局变量。

对于可变类型的全局变量来说，因其指向的数据可以修改，所以不使用global时也可修改全局变量。

拓展：偏导数

应用场景：函数在执行时，要带上所有必要的参数进行调用。然后，有时参数可以在函数被调用之前提前获知。这种情况下，一个函数有一个或多个参数预先就能用上，以便函数能用更少的参数进行调用。

import functools  
def add(a, b):  
 return a + b  
*# add(4, 2)  
#利用偏导数给一个固定值*plus3 = functools.partial(add, 3)  
plus5 = functools.partial(add, 5)  
print(plus3(2))  
print(plus5(3))

运行结果：

5

8

5.6 实战：利用函数式编程，爬取百度图片import time  
import requests *# pip install requests*from urllib import request *#需要这个下载  
#url地址*url = 'https://image.baidu.com/search/acjson?tn=resultjson\_com&ipn=rj&ct=201326592&is=&fp=result&queryWord=%E5%94%AF%E7%BE%8E&cl=2&lm=-1&ie=utf-8&oe=utf-8&adpicid=&st=-1&z=&ic=0&hd=&latest=&copyright=&word=%E5%94%AF%E7%BE%8E&s=&se=&tab=&width=&height=&face=0&istype=2&qc=&nc=1&fr=&expermode=&force=&pn=60&rn=30&gsm=3c&1554171103838='  
*#模拟请求头*headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.75 Safari/537.36'  
}  
*#向服务器发送的数据*data\_str = '''  
 tn: resultjson\_com  
 ipn: rj  
 ct: 201326592  
 is:  
 fp: result  
 queryWord: 唯美  
 cl: 2  
 lm: -1  
 ie: utf-8  
 oe: utf-8  
 adpicid:  
 st:  
 z:  
 ic:  
 hd:  
 latest:  
 copyright:  
 word: 唯美  
 s:  
 se:  
 tab:  
 width:  
 height:  
 face:  
 istype:  
 qc:  
 nc: 1  
 fr:  
 expermode:  
 selected\_tags:  
 pn: 30  
 rn: 60  
 gsm: 1e  
 1545019467831:  
 '''  
*#todo 分析得到图片的所有url*def get\_image\_url():  
 send\_data = {}  
 for line in data\_str.splitlines():*#以行进行分割* line\_data = line.split(': ') *#分割* if len(line\_data) == 2: *#判断* key,value = line\_data *#序列解包赋值* if key and value:  
 send\_data[key] = value *#添加字典* response = requests.get(url,headers = headers,params=send\_data)  
 json\_data = response.json()['data']  
 for src in json\_data:  
 img\_url = src.get('middleURL')  
 if img\_url:  
 down\_image(img\_url)#注意：在内部调用外部的函数  
*#todo 下载图片*def down\_image(url):  
 name = str(time.time()) + '.png'  
 request.urlretrieve(url = url,filename=name)  
 print('{} is download'.format(name))  
 time.sleep(1)  
*#调用*get\_image\_url()

总结：

5.1 函数定义知识点讲解

5.2 函数的调用

5.3 函数划分

5.4 匿名函数lambda

5.5 函数作用域

5.6 实战：利用函数式编程，爬取百度图片

练习作业：

1. A = [‘a’,’b’,’c’] B = [‘d’,’e’,’f’]利用map()，lambda 生成一个新的列表

C =[‘ad’,’be’,’cf’]