# 1 Python 函数

## 定义一个函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

* 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号**()**。
* 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
* 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
* 函数内容以冒号起始，并且缩进。
* **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

## 按值传递参数和按引用传递参数

所有参数（自变量）在Python里都是按引用传递。如果你在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。例如：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 可写函数说明def changeme( mylist ):

"修改传入的列表"

mylist.append([1,2,3,4]);

print "函数内取值: ", mylist

return

# 调用changeme函数

mylist = [10,20,30];

changeme( mylist );print "函数外取值: ", mylist

传入函数的和在末尾添加新内容的对象用的是同一个引用。故输出结果如下：

函数内取值: [10, 20, 30, [1, 2, 3, 4]]函数外取值: [10, 20, 30, [1, 2, 3, 4]]

## 参数

以下是调用函数时可使用的正式参数类型：

* 必备参数
* 关键字参数
* 默认参数
* 不定长参数

### **关键字参数**

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。

使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为 Python 解释器能够用参数名匹配参数值。

以下实例在函数 printme() 调用时使用参数名：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

#可写函数说明def printinfo( name, age ):

"打印任何传入的字符串"

print "Name: ", name;

print "Age ", age;

return;

#调用printinfo函数

printinfo( age=50, name="miki" );

以上实例输出结果：

Name: mikiAge 50

### **缺省参数**

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。下例会打印默认的age，如果age没有被传入：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

#可写函数说明def printinfo( name, age = 35 ):

"打印任何传入的字符串"

print "Name: ", name;

print "Age ", age;

return;

#调用printinfo函数

printinfo( age=50, name="miki" );

printinfo( name="miki" );

以上实例输出结果：

Name: mikiAge 50Name: mikiAge 35

### **不定长参数**

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述2种参数不同，声明时不会命名。基本语法如下：

def functionname([formal\_args,] \*var\_args\_tuple ):

"函数\_文档字符串"

function\_suite

return [expression]

加了星号（\*）的变量名会存放所有未命名的变量参数。选择不多传参数也可。如下实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 可写函数说明def printinfo( arg1, \*vartuple ):

"打印任何传入的参数"

print "输出: "

print arg1

for var in vartuple:

print var

return;

# 调用printinfo 函数

printinfo( 10 );

printinfo( 70, 60, 50 );

以上实例输出结果：

输出:10输出:706050

**两星的为字典：**

**def** show(\*\*arg):  
 **for** name **in** arg:  
 **print** name  
**if** \_\_name\_\_ ==**'\_\_main\_\_'**:  
 show(name=**'gao'**,xing=**'jian'**,ni=**'ni'**,wo=**'gao'**)  
 dict1={**'dt1'**:**'gao'**,**'dt2'**:123}   
 show(\*\*dict1) #字典传值必须加两个星

### **Lambda语法**

lambda函数的语法只包含一个语句，如下：

lambda [arg1 [,arg2,.....argn]]:expression

如下实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 可写函数说明

sum = lambda arg1, arg2: arg1 + arg2;

# 调用sum函数print "相加后的值为 : ", sum( 10, 20 )print "相加后的值为 : ", sum( 20, 20 )

以上实例输出结果：

相加后的值为 : 30相加后的值为 : 40

map(**lambda** x:x\*\*x,range(10))

结果：[1, 1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]

吧 range的值进行自乘

## return 语句

return语句[表达式]退出函数，选择性地向调用方返回一个表达式。不带参数值的return语句返回None。之前的例子都没有示范如何返回数值，下例便告诉你怎么做：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 可写函数说明def sum( arg1, arg2 ):

# 返回2个参数的和."

total = arg1 + arg2

print "函数内 : ", total

return total;

# 调用sum函数

total = sum( 10, 20 );

以上实例输出结果：

函数内 : 30

## 三元表达式

result = **'gt' if** 1>3 **else 'lt'**

如过表达式为真 返回gt 否则返回 lt

## 变量作用域

一个程序的所有的变量并不是在哪个位置都可以访问的。访问权限决定于这个变量是在哪里赋值的。

变量的作用域决定了在哪一部分程序你可以访问哪个特定的变量名称。两种最基本的变量作用域如下：

* 全局变量
* 局部变量

## 全局变量和局部变量

定义在函数内部的变量拥有一个局部作用域，定义在函数外的拥有全局作用域。

局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入到作用域中。如下实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

total = 0; # 这是一个全局变量# 可写函数说明def sum( arg1, arg2 ):

#返回2个参数的和."

total = arg1 + arg2; # total在这里是局部变量.

print "函数内是局部变量 : ", total

return total;

#调用sum函数

sum( 10, 20 );print "函数外是全局变量 : ", total

以上实例输出结果：

函数内是局部变量 : 30函数外是全局变量 : 0

# Python 模块

模块让你能够有逻辑地组织你的Python代码段。

把相关的代码分配到一个 模块里能让你的代码更好用，更易懂。

模块也是Python对象，具有随机的名字属性用来绑定或引用。

简单地说，模块就是一个保存了Python代码的文件。模块能定义函数，类和变量。模块里也能包含可执行的代码。

## 定位模块

当你导入一个模块，Python解析器对模块位置的搜索顺序是：

* 当前目录
* 如果不在当前目录，Python 则搜索在 shell 变量 PYTHONPATH 下的每个目录。
* 如果都找不到，Python会察看默认路径。UNIX下，默认路径一般为/usr/local/lib/python/。

模块搜索路径存储在system模块的sys.path变量中。变量里包含当前目录，PYTHONPATH和由安装过程决定的默认目录。

## PYTHONPATH变量

作为环境变量，PYTHONPATH由装在一个列表里的许多目录组成。PYTHONPATH的语法和shell变量PATH的一样。

在Windows系统，典型的PYTHONPATH如下：

set PYTHONPATH=c:\python20\lib;

在UNIX系统，典型的PYTHONPATH如下：

set PYTHONPATH=/usr/local/lib/python

## 命名空间和作用域

变量是拥有匹配对象的名字（标识符）。命名空间是一个包含了变量名称们（键）和它们各自相应的对象们（值）的字典。

一个Python表达式可以访问局部命名空间和全局命名空间里的变量。如果一个局部变量和一个全局变量重名，则局部变量会覆盖全局变量。

每个函数都有自己的命名空间。类的方法的作用域规则和通常函数的一样。

Python会智能地猜测一个变量是局部的还是全局的，它假设任何在函数内赋值的变量都是局部的。

因此，如果要给全局变量在一个函数里赋值，必须使用global语句。

global VarName的表达式会告诉Python， VarName是一个全局变量，这样Python就不会在局部命名空间里寻找这个变量了。

例如，我们在全局命名空间里定义一个变量money。我们再在函数内给变量money赋值，然后Python会假定money是一个局部变量。然而，我们并没有在访问前声明一个局部变量money，结果就是会出现一个UnboundLocalError的错误。取消global语句的注释就能解决这个问题。

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

Money = 2000def AddMoney():

# 想改正代码就取消以下注释:

# global Money

Money = Money + 1

print MoneyAddMoney()print Money

## dir()函数

dir()函数一个排好序的字符串列表，内容是一个模块里定义过的名字。

返回的列表容纳了在一个模块里定义的所有模块，变量和函数。如下一个简单的实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 导入内置math模块import math

content = dir(math)

print content;

以上实例输出结果：

['\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log','log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']

在这里，特殊字符串变量\_\_name\_\_指向模块的名字，\_\_file\_\_指向该模块的导入文件名。

## globals()和locals()函数

根据调用地方的不同，globals()和locals()函数可被用来返回全局和局部命名空间里的名字。

如果在函数内部调用locals()，返回的是所有能在该函数里访问的命名。

如果在函数内部调用globals()，返回的是所有在该函数里能访问的全局名字。

两个函数的返回类型都是字典。所以名字们能用keys()函数摘取。

## reload()函数

当一个模块被导入到一个脚本，模块顶层部分的代码只会被执行一次。

因此，如果你想重新执行模块里顶层部分的代码，可以用reload()函数。该函数会重新导入之前导入过的模块。语法如下：

reload(module\_name)

在这里，module\_name要直接放模块的名字，而不是一个字符串形式。比如想重载hello模块，如下：

reload(hello)

## Python中的包

包是一个分层次的文件目录结构，它定义了一个由模块及子包，和子包下的子包等组成的Python的应用环境。

考虑一个在Phone目录下的pots.py文件。这个文件有如下源代码：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

def Pots():

print "I'm Pots Phone"

同样地，我们有另外两个保存了不同函数的文件：

* Phone/Isdn.py 含有函数Isdn()
* Phone/G3.py 含有函数G3()

现在，在Phone目录下创建file \_\_init\_\_.py：

* Phone/\_\_init\_\_.py

当你导入Phone时，为了能够使用所有函数，你需要在\_\_init\_\_.py里使用显式的导入语句，如下：

from Pots import Potsfrom Isdn import Isdnfrom G3 import G3

当你把这些代码添加到\_\_init\_\_.py之后，导入Phone包的时候这些类就全都是可用的了。

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 导入 Phone 包import Phone

Phone.Pots()Phone.Isdn()Phone.G3()

以上实例输出结果：

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

# 3 模块和模块的常用方法

至关重要的\_\_init\_\_.py

1、是否为主文件：print \_\_name\_\_

如果是主文件则输出\_\_main\_\_, 如果不是main,则输出 包名.文件名

用法：**if** \_\_name\_\_==**'\_\_main\_\_'**:  
 **print "gao"  
 else**:  
 **print "滚之！"**

**可以有效的防止攻击，别的模块导入本模块不可以执行**

2、当前文件路径：**print** \_\_file\_\_

* 1. 输出：J:/workspace/Pycharm/pystduy/November\_24/\_init\_test.py

1. 当前文件描述：**print** \_\_doc\_\_

输出：文档最上边的备注信息如：”””@author: gaojian”””

## Xrange()

Xrange()和range差不多 ，xrange只有在循环的时候才会产生值，延迟创建

跟xreadline() 差不多

## yield 生成器

上面的xrange就是根据yield生成的，执行一个生成一个

**def** gaoRead(file):  
 **with** open(file) **as** f:  
 seek=0   
 **while** True:  
 data=f.readline()  
 **if** data: #不加if data会一直跑，但不打印  
 seek=f.tell() #返回当前seek值，可有可无  
 **yield** data  
 **else**:  
 **return  
  
for** line **in** gaoRead(**'../November\_3/myFile.txt'**):  
 **print** line

## re模块

### Re.match

会从字符串的开头起匹配，开头开始匹配不到，就返回None

re.search会全部的字符串匹配，找不到返回None

**import** re  
result1=re.match(**"\d+"**,**"w325345fdgsdfggds"**)  
**if** result1:  
 **print** result1.group()  
**else**:  
 **print "nothing"**result2=re.search(**"\d+"**,**"325345fdgsdfggds"**)  
**print** result2.group()

结果：

nothing

325345

### Findall()

方法会查找所有匹配的字符串，返回列表  
result3=re.findall(**"\d+"**,**"34sdgsd3fdg4354ds32235"**)  
**print** result3

如果findall中包含子组，则结果会把子组包装成元祖返回元祖列表

str=**'gdfgdgd192.168.8.103fdg243.192.1.1dsfg'**

*# 匹配ip*result=re.search(**"(([01]?\d?\d|2[0-4]\d|25[0-5])\.){3}([01]\d\d|2[0-4]\d|25[0-5])"**,str)

采用（？： ）的形式可以屏蔽findall自动分组

result=re.findall(**"(?:(?:[01]?\d?\d|2[0-4]\d|25[0-5])\.){3}(?:[01]\d\d|2[0-4]\d|25[0-5])"**,str)

结果：['34', '3', '4354', '32235']

### Compile()提前编译

com=re.compile(**"\d+"**)  
result4=com.findall(**"34sdgsd3fdg4354ds32235"**)  
**print** result4

结果：['34', '3', '4354', '32235']

正则查找ip

ipstr=**"fh.345dfgsdtfsg,fgs.4354d3425,gr.4325ewt,4wre,gg127.324.134.32"**ip=re.findall(**"(?:\d{1,3}\.){3,4}\d{1,3}"**,ipstr)  
**print** ip

# 4 内置函数

## Help()

列出所有的方法

a=[]  
**print** help(a)

count(...)

| L.count(value) -> integer -- return number of occurrences of value

|

| extend(...)

| L.extend(iterable) -- extend list by appending elements from the iterable

|

| index(...)

| L.index(value, [start, [stop]]) -> integer -- return first index of value.

| Raises ValueError if the value is not present.

|

## Dir()

列出对象多有的可用函数

a =[]  
**print** dir(a)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_delslice\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_setslice\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

## Vair()

列出本文件中 所有的属性方法和值

**print** vars()

{'a': [], 'foo': <function foo at 0x029E6FB0>, 'gaoRead': <function gaoRead at 0x029E6EB0>, '\_\_builtins\_\_': <module '\_\_builtin\_\_' (built-in)>, '\_\_file\_\_': 'J:/workspace/Pycharm/pystduy/November\_24/\_init\_test.py', 'detail': <function detail at 0x029E6F30>, '\_\_doc\_\_': '\n@author: gaojian\n', 'go': <function go at 0x029E6F70>, '\_\_name\_\_': '\_\_main\_\_', 'login': <function login at 0x029E6EF0>, '\_\_package\_\_': None, 'str1': <module 'November\_3.str1' from 'J:\workspace\Pycharm\pystduy\November\_3\str1.pyc'>}

## Type()

判断属性的类型

a=[]

a=list()  
**print** type(a)

<type 'list'>

## Import temp

导入 模块，导入的时候就会执行所有代码

## Reload(temp)

重新加载导入的文件

## Id()

a=list()  
**print** id(a)

取地址

## Divmod()

**print** divmod(22,2)

(11, 0)

## All()

参数为iterable，便利所有值，全为真返回True，否则返回false

**print** all([23,435,**'435234'**])

True

## Bool()

判断布尔类型

**print** bool(**""**)  
**print** bool(None)

False

False

## Enumerate（sque，index）

li=[**'七尺'**,**'把持'**,**'劫持钉耙'**]  
**for** item **in** enumerate(li,1):  
 **print** item  
 **print** item[0],item[1]

输出结果：

(1, '\xe4\xb8\x83\xe5\xb0\xba')

1 七尺

(2, '\xe6\x8a\x8a\xe6\x8c\x81')

2 把持

(3, '\xe5\x8a\xab\xe6\x8c\x81\xe9\x92\x89\xe8\x80\x99')

3 劫持钉耙

## Format（）

s=**'i am {0}'  
print** s.format(**'gao'**)

## Map（）

便利这个列表，并给列表的每个值传入前面的函数中

**print** map(**lambda** x:x+100,[2,3,4,5,7])

结果：[102, 103, 104, 105, 107]

## Filter（function,sque）

过滤

li=[234,4,52,2]

**def** foo(num):  
 **if** num<100:  
 **return** num  
 **else**:  
 **return** None  
  
**print** filter(foo,li)

输出：[4, 52, 2]

## Reduce（function,sque）

遍历序列，并做聚集

li=[234,4,52,2]  
**print** reduce(**lambda** x,y:x+y,li)

292

## Zip（） 拉链咬合

a=[1,2,36,4]  
b=[34,6,8,0]  
c=[3,4,7,5]  
d=[3,5,80,1]  
**print** zip(a,b,c,d)

输出：[(1, 34, 3, 3), (2, 6, 4, 5), (36, 8, 7, 80), (4, 0, 5, 1)]

## \_\_import\_\_() 反射

通过模块的名称导入模块

temp=**"other1" #这里必须是包名.模块名**model=\_\_import\_\_(temp)  
**print** model.count()

## **Getattr(Object,string) 反射获取方法** temp=**"other"** fun=**"count" #这里** model=\_\_import\_\_(temp) func=getattr(model,fun) **print** func()

data=raw\_input(**'请输出地址：'**)  
array=data.split(**"/"**) *#获得输入的元祖*namespace=\_\_import\_\_(**'back.'**+array[0]) *#得到包*mode=getattr(namespace,array[0]) *#得到模块*func=getattr(mode,array[1]) *#得到方法*func()

## Join()

生成验证码：

**import** random  
code=[]  
**for** x **in** range(6):  
 **if** x==random.randint(1,5):  
 code.append(str(random.randint(0,9)))  
 **else**:  
 code.append(chr(random.randint(65,90)))  
**print ''**.join(code)

Join的效率高，遍历一次列表只在内存创建个新的内存，用其他的每遍历一个元素就会创建一次内存

## Md5哈希

**import** hashlib  
hash=hashlib.md5()  
hash.update(**'admin'**)  
**print** hash.hexdigest()

## Pickle.dumps()序列化 反序列化

**import** pickle  
li=[**"gao"**,343,43,**'八戒'**]  
*#序列化 pickle.dumps()*temp=pickle.dumps(li)  
*#反序列化 pickle.loads()***print** pickle.loads(temp)

序列化到文件

**import** pickle  
li=[**"gao"**,343,43,**'八戒'**]  
pickle.dump(li,open(**"../November\_3/myFile.txt"**,**"a"**))  
result=pickle.load(open(**"../November\_3/myFile.txt"**,**"r"**))

## Json的序列化

**import** json  
dict1={**'xing'**:**'gao'**,**"age"**:23}  
jsonstr=json.dumps(dict1)

Print jsonstr  
result=json.loads(jsonstr)   
**print** result

结果：

{"age": 23, "xing": "gao"}

{u'age': 23, u'xing': u'gao'}

# 5装饰器

类似于java的代理模式

## 带参数：

|  |
| --- |
| **def** outer(fun):  **def** wrapper(arg1):  **print 'yanzheng  print 'wo'** fun(arg1)  **print 'tongguo'  return** wrapper @outer **def** fun1(arg1):  **print 'gao'**,arg1  fun1(**'jian'**) |

## 参数为方法：

|  |
| --- |
| **def** Filter(before\_fun,after\_fun):  **def** outer(main\_fun):  **def** wapper(arg1,arg2):  before\_result=before\_fun(arg1,arg2)  **if**(before\_result!=None):  **return** before\_result;  main\_result=main\_fun(arg1,arg2)  **if**(main\_result!=None):  **return** main\_result;  after\_result=after\_fun(arg1,arg2)  **if**(after\_result!=None):  **return** after\_result;  **return** wapper  **return** outer  **def** before(arg1,arg2):  **print 'before'**+arg1+**','**+arg2 **def** after(arg1,arg2):  **print 'after'**+arg1+**','**+arg2  @Filter(before,after) **def** main\_print(arg1,arg2):  **print 'main'**+arg1,arg2  main\_print(**'gao'**,**'jian'**) |