内置函数：

len()、min()、max()、range()、sorted()、sum()

进制转换函数

bin()转换为二进制的函数

oct()转换为八进制的函数

hex()转换为十六进制的函数

ord()将字符转换成对应的ASCII码值

chr()将ASCII码值转换成对应的字符

函数的基本语法：

def functionname(arg):

Suite

Return [expression]

1. def 定义函数时的关键字
2. functionname 函数名
3. arg函数的参数名，传入不同的参数，返回值也不同，通过参数的传递使函数的功能得以实现。
4. Suite是实现函数功能的代码段
5. return 后面接的表达式就是函数的返回。

functionname函数名用来描述这个函数作用的名称，写文档对函数所实现功能的简单描述

return 后面的表达式是函数的返回值，后面不接表达式时返回的是None

函数的调用：函数名(参数)

函数的参数：

必备参数：传入的参数必须与函数定义时的个数相等且按顺序传入

关键字参数：在函数调用时可以改变传入参数的顺序

默认参数：在定义函数是将参数赋一个初始值，调用时参数可缺省，也称为可缺省参数

不定长参数：\*参数名：元组，参数不接受关键字参数；

\*\*参数名：字典，调用时使用关键字参数；

必须元组在前面，字典在后面，函数调用时要把直接传的值放在关键字参数的前面。

必备参数（例）：

>>> def func1(a,b,c):

return a,b,c

>>> func1(1,2,3)

(1, 2, 3)

>>> func1('x','y','z')

['x', 'y', 'z']

关键字参数（例）：

>>> def func1(a,b,c):

return a,b,c

>>> func1(c=100,b=10,a=1)

(1, 10, 100)

默认参数（例）：

>>> def func1(a=1,b=1,c=1):

return a,b,c

>>> func1()

(1, 1, 1)

不定长参数(例)：

例1：

>>> def fun3(\*arg):

s = 0

for i in arg:

s +=i

return s

>>> fun3(1,2,3,4)

10

例2：

>>> def fun4(\*\*arg):

for i in arg.items():

print i

>>>fun4(name='jane',age=20,sex='female')

('age', 20)

('name', 'jane')

('sex', 'female')

所有参数（例）：

>>> def fun1(a,b,c=1,\*d,\*\*e):

print (a,b,c)

print (d)

print (e)

>>> fun1(1,2,3,4,5,6,7,x=1,y=2,z=3)

1 2 3 #必备和默认参数

(4, 5, 6, 7) #元组

{'x': 1, 'y': 2, 'z': 3} #字典

全局变量和局部变量：

全局变量：定义在函数外的拥有全局作用域的变量，可以在整个程序范围内访问

全局变量可以在函数内被访问但不可以在函数内被修改

局部变量：定义在函数内部的拥有一个局部作用域的变量，只能在其被声明的函

数内部访问。

局部找不到找全剧。

例1：

>>> x = 2 #这里的X是全局变量

>>> def funct1():

x = 3 #这里的X是局部变量

x = x+1

return x

>>> funct1()

4

>>> x

2

变量的作用域：

变量的在哪里被赋值的就决定了这个变量作用的区域。

global关键字：

global语句用来声明x是全局变量，当我们在函数内给x赋值时，它的改变映射到我们在主块中使用的x的值

例：

>>> x = 2

>>> def funct3():

global x

x +=5

return x

>>> x

2

>>> funct3()

7

>>> x

7

nonlocal关键字：

用来声明使用外层(非全局)变量。

x = 12

def fun1():

m = 1

def fun2():

nonlocal m

m+=1

return m+x

return fun2()

内嵌函数和闭包

函数里面再嵌套一个函数，外面的函数的的返回值return里面的函数，这时里面的函数可以访问到外面这个函数的的变量，在函数外部我们不能直接调用里面的函数，但能通过外面函数的返回值来调用里面的函数，我们称这种现象为闭包。

例（1）

>>> def fun1():

print ('fun1被调用了.....')

def fun2():

print ('fun2被调用了.....')

fun2() #调用def fun2()

>>> fun1()

fun1被调用了.....

fun2被调用了.....

例（2）

>>> def func1(x):

def func2(y):

return x\*y

return func2 #没有加括号是函数对象

>>> func1(5)(6)

30

>>> b = func1(4)

>>> b(6)

24

lambda表达式：

关键字lambda用来创建简单的匿名函数。它既不能包含控制结构也没有return语句，返回的值就仅仅是表达式计算后得到的值。

使用lambda可以省下函数定义的过程，可以使得代码更加精简。对于有些只需要使用一两次的函数，使用lambda也就不需要考虑函数命名的问题。

例（1）

>>> def fun(x):

return x\*x

>>> fun(5)

25

>>> a = lambda x:x\*x

>>> a(5)

25

lambda与内置函数 filter，map

一、filter()函数相当于过滤器

第一个参数可以是一个函数或者是None,第二个参数为序列

例：

>>> list(filter(None,[1,2,0,False,True])) #过滤掉0和False

[1, 2, True]

>>> list(filter(lambda x:x%3==0,range(10))) #过滤掉不能被3整除的数

[0, 3, 6, 9]

二、map()函数相当于一次性完成多次的调用

>>> a = lambda x:x\*x

>>> list(map(a,range(5))) #range5的平方多次调用

[0,1,4,9,16]

>>> a = lambda x:x\*\*x

>>> list(map(a,range(5))) #range5的次方多次调用

[1, 1, 4, 27, 256]

递归

如果一个函数在内部调用自身本身，这个函数就是递归函数。递归效率很低有时候要等很久。

1！=1

2！=1\*2=1!\*2

3！=1\*2\*3=2!\*3

4！=1\*2\*3\*4=3!\*4

n！=1\*2\*3\*……\*(n-1)\*n=(n-1)!\*n

例(1)阶乘:

>>> def fun22(n):

if n==1 or n==0:

return 1

else:

return fun22(n-1)\*n

>>> fun22(4)

24

#fun22(4) = fun22(3)\*4=fun22(2)\*3\*4=fun22(1)\*2\*3\*4=1\*2\*3\*4=24

例(2)求和:

>>> def fun(n):

if n==1:

return 1

else:

return fun(n-1)+n

>>> fun(4)

10

#fun(4) =fun(3)+4=fun(2)+3+4=fun(1)+2+3+4

例(2)阶减

'A说B比他大2岁，B说C比他大2岁，C说D比他大2岁，D说我20岁，问A多少岁？’

>>> def jian2(age,rank):

#当前是第rank个人，年龄是age

#我们希望通过这个函数，求出第一个的岁数

#jian2(20,4) == jian2(18,3) == jian2(16,2) == jian2(14,1) == 14

if rank == 1:

return age

else:

return jian2(age-2,rank-1)

>>> jian2(20,4)

14

例(2)斐波南希 [1,1,2,3,5,8,13,21,34,55]

def fun1(n):

if n == 1:

return [1]

if n == 2:

return [1, 1]

funs = [1, 1]

for i in range(2,n):

funs.append(funs[-1] + funs[-2])

return funs

def fun2(n):

if n==1 or n==2:

return 1

return fun2(n-1)+fun2(n-2)

def fun3(n):

a,b = 1,1

for i in range(n-1):

a,b = b,a+b

return (a)

回调函数：

一般写程序是你调用系统的API，如果把关系反过来，你写一个函数，让系统调用你的函数，那就是回调了，那个被系统调用的函数就是回调函数。

tkinter模块