博客地址：<http://www.cnblogs.com/znyyy/p/8109949.html>

函数也叫方法，比如你用手机打电话就是一个函数，你只管用就好，里面的功能别人已经帮你研发好了，你只需要输入手机号，按下拨打就好了，这个概念用在代码里尤为重要，比如重复某个功能，比如用户输入，你要判断用户的输入不能为空吧，那么这个你可以定义一个函数，就是用来判断用户的输入是否为空，每次调用一下就可以，假如现在校验用户输入的需求变化了，那么你直接维护这个函数就好，没必要再去所有的代码里找这个校验，一个个的去改，所以函数有以下两点显而易见的优点：

　　1.重复代码精简，简化复用性代码的编写，将重复性代码写成函数，要用的时候调用即可；  
　　2.方便以后维护，你改一个函数，所有调用这个函数的地方的功能都能及时更新，避免修改代码遗漏

　　函数在前面的字符串方法已经接触过了，像strip()这些，就是python内置的一些函数，当然还有很多，这些函数python已经帮我们写好，直接用就好了，这样是不是很方便，什么时候需要什么时候就调用。显而易见，函数要调用的时候才会执行，调用函数就是在函数名后加()就可以调用了，像strip()这些。

**1.函数初识**

　　定义函数的时候，写的入参叫做形参；调用函数的时候，传的值叫做实参：

def sayName(name,age,sex): #name,age,sex就是形式参数

# name,age,sex函数的参数类型是位置参数、必填参数

print('姓名： %s'%name)

print('年龄： %s'%age)

print('性别： %s'%sex)

# 上面这个就是函数体

sayName('星星',18,'男') #'星星',18,'男'这三个就是实际参数，调用的时候传进来的参数

#打印结果：

# 姓名： 星星

# 年龄： 18

# 性别： 男

**2.函数return**

　　函数如果需要返回值，用return即可，有两点需要注意：

　　1.函数里，只要遇到return就结束，不会再往下运行  
　　2.如果函数没有指定return返回值，那么函数默认返回的是None

def cal1(a,b):

return a+b

return a\*b

def cal2(a,b):

print('函数内a,b运行结果：',a+b)

print('cal1函数return结果：',cal1(1,2))

print('cal2函数return结果',cal2(1,2))

#运行结果

# cal1函数return结果： 3 #因为函数里ruturn了a+b的结果，所以这里的结果是3

# 函数内a,b运行结果： 3 #这个是函数体内的运行情况

# cal2函数return结果 None #因为函数里没有ruturn，所以这里的结果是None

**3.函数即变量**

　　从零单排0中介绍到的变量，在内存里开放一个地方存变量，函数也是一样，也是变量，存在内存里的摸个地方，调用直接就是函数名+()，就调用了，看一个而有趣的调用方法，先定义一个字典，value存函数名，根据key来调用相应的函数，详情见：<http://www.cnblogs.com/znyyy/p/7670081.html>

def play():

print('出去浪')

def study():

print('好好学习')

def\_method={'1':play,'2':study}

while True:

choice=input('请输入你的选择：1代表出去浪；2代表好好学习\n')

if choice=='1' or choice=='2':

def\_method[choice]()

break

else:

print('请输入正确的选择！')

continue

复制代码

**4.入参类型**

　　函数的入参有很多类型，上面已经了解过位置参数，还有默认参数，可变参数，关键字参数。

　　注意：参数有顺序，如果上述四种参数你都有用，那么参数类型顺序必须是：位置参数->默认参数->可变参数->关键字参数，否则报错。

　　1.**位置参数**，这个参数就和格式化输出里的占位一样，形参先把位置站好，然后就等着用户调用的时候输入实参，排队填坑，如果实参和形参个数不一致，多了或者少了都会报错，如下：

def add(a,b):

print('%s+%s=%s'%(a,b,a+b))

return a\*b

add(1,2) #打印结果：1+2=3

add(6,2) #打印结果：6+2=8

add(1,2,3) #打印结果：报错，参数多了

add(2) #打印结果：报错，参数少了

　　2.**默认参数**，这个参数就是程序会先给一个默认的，不传不会报错，会直接用默认的这个参数值，如果传了就用你传的这个值；等于是定义函数的时候先定义了一个参数变量，调用时如果没传就用之前定义好的，如果传了就是覆盖之前的值，用传入的参数值：

# def multi(a,b=2):

# print('%s\*\*%s=%s' % (a, b, a \*\* b))

# multi(2) #不传的时候默认b=2，这里求的是平方

# multi(3) #不传的时候默认b=2，这里求的是平方

# multi(2,3) #传的时候b=3了，这里求的是立方

# multi(2,b=4) #和上面的这个调用时一样的，但是一般这样写，如果默认参数较多时，不会因为位置而出错

# multi(b=4,2) #报错，位置参数必须在前

#运行结果：

# 2\*\*2=4

# 3\*\*2=9

# 2\*\*3=8

# 2\*\*4=16

　　3**.可变参数**，这里一般用的少，可变就是参数是变化的，考虑到了程序的扩展性，定义函数的时候并不确定会有多少个参数，这里就用到了可变参数，既然可变，那么在调用函数的时候也是可以不输入的：

#可变参数

def user(name,passwd,\*args): #一般是用args来命名，当然也可以用其他你想要的名字

print('用户名：%s,密码：%s'%(name,passwd))

print('还有属性如下：\n',args) #这里的args，就是用户输入参数的一个元祖

# user('星星',123456,'男','180cm','60kg')

# 打印结果：

# 用户名：星星,密码：123456

# 还有属性如下：

# ('男', '180cm', '60kg')

　　4.**关键字参数**，也不是必填的，他是一个key-value的字典形式：

#关键字参数

def user(name,passwd,\*\*kwargs): #一般是用args来命名，当然也可以用其他你想要的名字

print('用户名：%s,密码：%s'%(name,passwd))

print('还有属性如下：\n',kwargs) #这里的args，就是用户输入参数的一个字典

user('星星',123456,sex='男',height='180cm',weight='60kg')

# 打印结果：

# 用户名：星星,密码：123456

# 还有属性如下：

# {'height': '180cm', 'sex': '男', 'weight': '60kg'}

**5.变量作用域**

变量作用域有四种：全局作用域，局部作用域，内建作用域，闭包函数外的函数中，后两者用的少，全局作用域和局部作用域概念用的多。全局作用域就是变量在所有的范围都可以用，比如在代码开始定义一个变量，那么不论是在函数外部还是内部都可以用这个变量；局部作用域就是只能在某个区域内才能用，除了这个区域就用不了了，比如函数内定义的变量只能在函数体内用，出了函数就用不了了。(建议不要用全局变量，不安全，所有人都可以修改)

#全局作用域

hys=['屏幕','主机'] #全局变量

def yxg(thing):

hys.append(thing) #函数体内可以用全局变量hys

yxg('麦克')

print(hys) #打印结果：['屏幕', '电脑', '麦克']

#局部作用域

def yxg(thing):

desk = ['显示器', '方便面'] # 局部变量

desk.append(thing) #函数体内修改局部变量

yxg('键盘')

print(desk) #函数体外调用局部变量报错

　　值得一提的是，当全局变量是一个不可编辑的参数时，需要先申明才可以用全局变量(str/int/float等不可变变量函数在调用全局变量时，需要global才可以修改)：

#不可变变量的全局变量引用，如int、float、str

money=500

def chage():

money=1000

print('函数内的money：',money) #打印结果是 1000，因为函数内部作用域中money=1000，不能修改外面的money

chage()

print('函数外的money：',money) #打印结果是 500，因为函数不可以修改外部作用域中的不可变变量的全局变量引用，如int、float、str

def chage():

global money #global后就可以引用了，这就申明了money是一个全局的变量，函数内可以修改

money = 1000

print('函数内的money：', money) #打印结果是 1000

chage()

print('函数外的money：',money) #打印结果是 1000，因为global申明了全局变量，所以可以修改

　　更详细的变量作用域见：<http://www.cnblogs.com/znyyy/p/7670081.html>

**6.函数迭代**

函数迭代，就是自己调用自己，一个死循环，一直调用下去，最多迭代999次，超过999程序自动停止运行(while没有次数限制)。既然这样，那么迭代肯定要有一些特性才可以正常运用：

　　1. 必须有一个明确的结束条件  
　　2. 每次进入更深一层递归时，问题规模相比上次递归都应有所减少  
　　3. 递归效率不高，递归层次过多会导致栈溢出（在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出）

#函数迭代，就是自己调用自己，一个死循环，调用下去：

def dd():

print('大雷好帅')

dd() #函数内，又调用了函数本身，那么就会一直调用下去

dd() #迭代函数最多会调用999次,这里会打印999次“大雷好帅”

　　用迭代函数写斐波拉契数列：[1, 1, 2, 3, 5, 8, 13, 21, 34],数列从第3项开始，每一项都等于前两项之和。

　　先用以前的知识来写：

list = []

def fb(num):

a=1

b=1

while num>0:

list.append(a)

a,b=b,a+b

num-=1

fb(9)

print(list)

　　用迭代来写：

list = []

a = 1

b = 1

def fb(num):

global a,b

list.append(a)

a,b=b,a+b

num-=1

if num>0:

fb(num)

fb(5)

print(list)

　　更多函数剖析详见：<http://www.cnblogs.com/znyyy/p/7670081.html>