## python基本语法：

python2.0默认不支持中文，需要声明编码 #！-\*-coding:utf-8-\*- 来设置编码

python3.0默认支持中文

单行注释：#

多行注释 ’’’ ‘’’

用户输入：input eg:input(“your name?”)，但是input输入的都是字符串

ifelse语法：

if \*\*\* :

\*\*\*#必须有缩进，缩进的都是在if条件后执行的，缩进必须一致，推荐4个

else:

\*\*\*#必须有缩进，这儿缩进来的都是else需要执行的，相当于java的大括弧

if a=2:

print(“a=2”)

elif a<2:

print(“a<2”)

else:

pring(“a>2”)

if elif else只进一个，只要有一个匹配，进入一个执行完后直接返回

//整除

/除

\*\*指数

print(“aaa”,end=””)表示不换行输出，因为结尾是一个空字符，平常中用的还有end=”\t”,\t是制表符

多行的字符串只能用’’’ ‘’’（三个单引号）将其引住

单行的变量里面的时候，可以用单引号和双引号，他们两个的意义是一样的。

单行里面包含双引号的时候，可以用单引号将其包住，’hello ,It”s me’

单行里面包含单引号的时候，可以用双引号将其包住 ，“hello,It’s me”

单引号里面既包含单引号，又包含双引号的时候，用三引号将其包住，’’’ hello ,”It’s me”?‘’’

# 集成开发环境

IDE（Intergrated Development Environment）集成开发环境

VIM #经典的Linux下的文本编辑器

Emacs #Linux文本编辑器，比vim更容易使用

Eclips # java IDE,支持Python、Java、c、c++

Visual Studio # 微软开发的IDE，Python、c、java、c++

Notepad++

Sublime # Python开发的

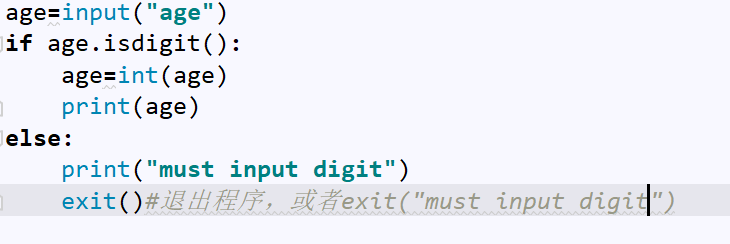
Pycharm，主要用于Python开发的IDE

# 格式化的输出

占位符：%s #s是string的缩写

在字符串的最后边用%()中间写上占位符需要的变量

Eg:msg=’’’Myname:%s,Myage:%s’’’ %(name,age)

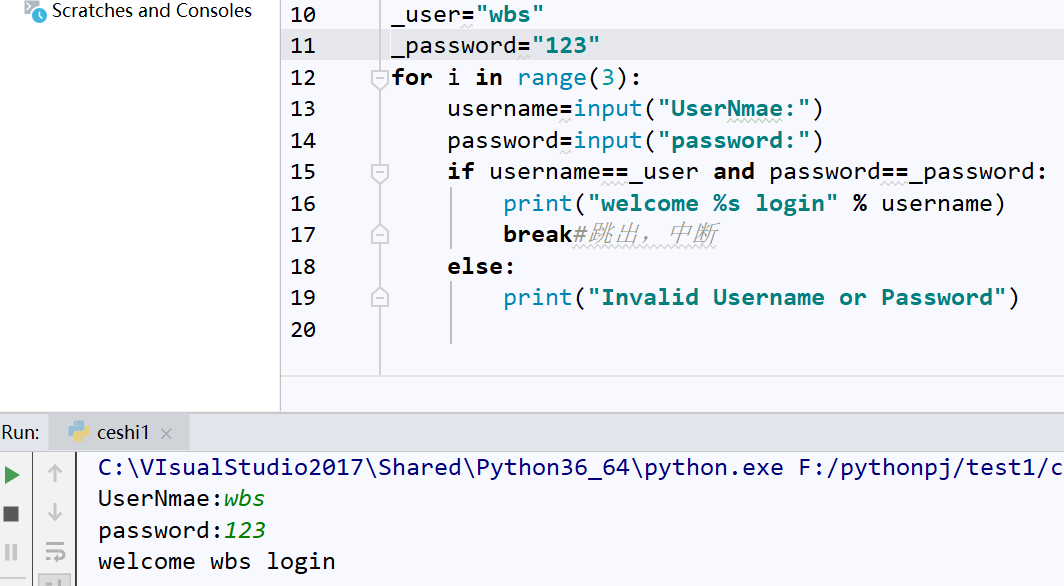


判断是不是一个数字的时候用isdigit();

%d #只能输入数字，d是digit

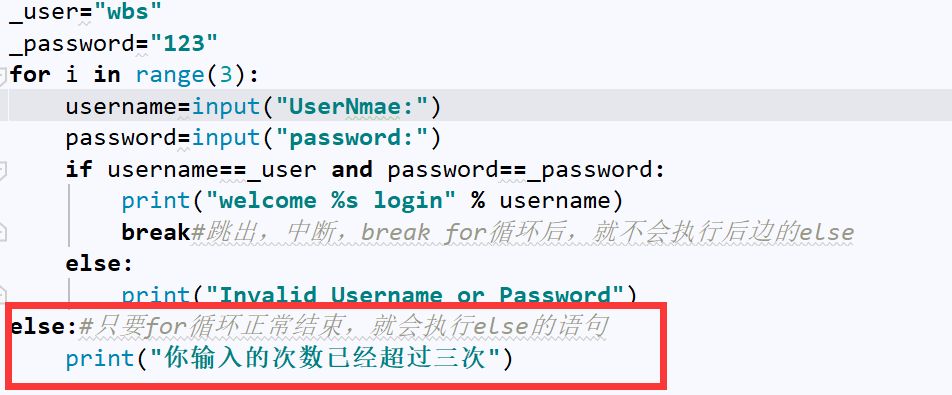
%f #f代表浮点数，约等于小数

简易的登录验证：



用的是for循环，判断三次，如果三次登录出错，就跳出；

for i in range(10):#表示的是0~9，而不是1~10

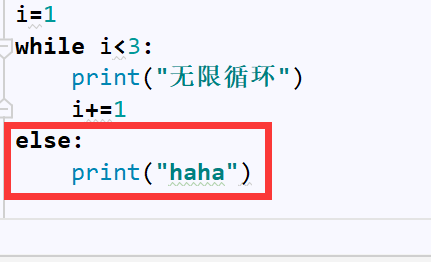


上边的程序，相当于是下边的程序的精简版：



python的for循环之后可以加else，但是不能加elif

# while循环



程序顺利执行完了之后，会执行else后边的代码。

continue #跳出本次的循环，继续进行下次的循环

break #跳出整个当前的循环

# 列表、切片（增删改查）

列表的创建可以用list()

a**=**[**'jinxing'**,**'sanpang'**,**'xiaohong'**,**'xiaomi'**,**'daming'**,**'ligang'**]  
print(a[1**:**2]) **#只取到了下标为1的**print(a[1**:**4])**#去到了下表为1，2，3的**print(a[1**:**])**#取到最后一个**print(a[1**:-**1])**#除了最后一个都能取到**print(a[1**:-**1**:**2])**#最后一个1代表步长，从左到右一个一个去取，取到倒数第二**print(a[1**::**2])**#最后一个1代表步长，从左到右一个一个去取，取到最后**print(a[6**::-**2])**#从右往左取，步长为2，**print(a[**-**3**::**])**#从左往右，从到书第三个开始取，一直取到最后**print(a[**-**3**::-**1])**#从右往左，从到书第三个开始取，一直取到最后**a.append(**'zhangqiang'**)**#只能插入到最后的位置**print(a)  
a.insert(1,**'aaa'**)**#在第一个后边插入aaa,可以将数据插入到任意位置insert(index,'内容')**print(a)  
a[1]**='bbb'#修改a[1]为bbb**print(a)  
a[1**:**3]**=**[**'a'**,**'b'**]**#同时修改1，2位置为a,b**print(a)  
a.remove(**'a'**)**#或者a.remove(a[1])也可以,remove删除的时候是作为一个整体，比如a.remove(['a','b'])是错误的删除方法**print(a)  
a.pop(1)**#pop可以根据索引删除元素，而且可以得到删除元素的值，不指定元素的时候删除最后一个元素**aa**=**a.pop(1)**#将删除的值赋给aa变量**print(a)  
**del** a[0]**#直接删除**print(a)  
**del** a**#也可以删除整个对象**t**=**[**'l'**,**'p'**,**'o'**,**'i'**,**'u'**,**'y'**,**'t'**,**'r'**,**'e'**,**'w'**,**'q'**,**'q'**].count(**'q'**)**#计算所包含元素的个数**print(t)  
a**=**[1,2,3]  
b**=**[4,5,6]  
a.extend(b)  
print(a)**#将b的元素添加到a**print(b)  
print(a.index(5))**#可以取出来某一个元素的索引值**a.reverse()**#将a元素倒转,无返回值**print(a)  
a.sort()**#将排序,无返回值**a.sort(reverse**=True**)**#排序完成后将其倒转**

b**=**sorted(a)**#sorted也是排序**print(a)  
print(5 **in** a)**#判断5是否在a里面**print(type(a) **is** list)**#判断a是否是一个列表**

a**=**[1,2,3]  
b**=**[4,5,6]  
c**=**a**+**b**#可以对连个列表进行相加，就是元素相加，而原来的a和b的元素不发生变化**print(c)

## 字符格式化输出

占位符 %s s = string

%d d = digit 整数

%f f = float 浮点数，约等于小数

## 数据运算

数据类型出初识

数字

整数 int(integer)

整型

长整型

in py3 已经不区分整型与长整型，统一都叫整型

in C int age 22 , long age

布尔 只有2种状态，分别是

真 True

假 False

字符串

salary.isdigit()

计算机中， 一切皆为对象

世界万物，皆为对象，一切对象皆可分类

## 循环loop

有限循环 ，次数限制

无限循环=死循环

continue 结束本次循环，继续下一次循环

break 跳出整个当前的循环

for

while

break , continue

## 数据类型

整数

字符串

列表，元组

查

索引(下标) ，都是从0开始

切片

.count 查某个元素的出现次数

.index 根据内容找其对应的位置

"haidilao ge" in a

增加

a.append() 追加

a.insert(index, "内容")

a.extend 扩展

修改

a[index] = "新的值"

a[start:end] = [a,b,c]

删除

remove("内容")

pop(index)

del a, del a[index]，默认不写时候删除最后一个元素

a.clear() 清空

排序

sort ()

reverse()

身份判断

>>> type(a) is list

True

>>>

## 购物车程序

salary = 5000

1. iphone6s 5800

2. mac book 9000

3. coffee 32

4. python book 80

5. bicyle 1500

>>>:1

余额不足，-3000

>>>:5

已加入bicyle 到你的购物车， 当前余额:3500

>>>:quit

您已购买一下商品

bicyle 1500

coffee 30

您的余额为:2970

欢迎下次光临

**列表里面可以存放各种类型的元素**

# 元组

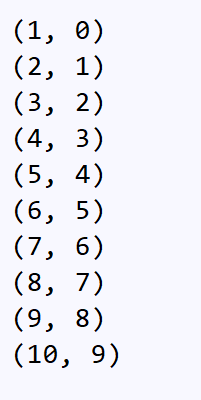
元组和列表差不多，值是将[]换成了()

元组只可以将其中的值拿出来，但是不能修改，是只读的列表

元组用的时候在最后一个加一个逗号。

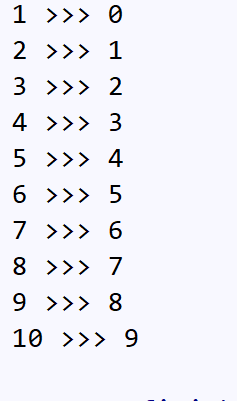
a**=**range(0,10)  
**for** i **in** enumerate(a,1)**:#enumerate可以自动加一个序号，默认从0开始，可以加一个参数enumerate(a,1)使得其编号从1开始** print(i)

结果如下：



**for** i,j **in** enumerate(a, 1)**: #前边加两个参数的时候，将第一个变量赋给i,将第二个变量赋给j,这样就可以不输出一个元组** print(i,**'>>>'**,j)

结果如下：



a,b**=**[3,4]**#将第一个元素给a，将第二个元素给b**print(a,**'----'**,b)

结果：

len(a)#可以计算出列表a的长度，

# Dictionary（字典）

字典是用{}包住

id(a)#可以算出a的地址。

字典是无序的，是可修改的类型。字典中的键必须是不可修改的类型，值可以是修改的类型

**#python的字典**dic1**=**{1**:'a'**,2**:'b'**}**#常用的创建字典的方式**print(dic1)  
dic2**=**dict(((**'name'**,**'wbs'**),))**#另外的一种创建字典的方式**print(dic2)  
dic3**=**dict([[**'name'**,**'wbs'**],])**#中括弧和小括弧的效果是一样的**print(dic3)

## #python字典的增加

dic1**=**{**'name':'wbs'**}  
dic1[**'age'**]**=**20**#加入一个元素**dic1[**'name'**]**='bo'#已经有了键的时候是修改**print(dic1)  
dic1.setdefault(**'age'**,18)**#如果键已经存在，不做任何修改，返回键对应的值，如果没有，加入字典，并返回相应的值**print(dic1)

## 字典的查找

a**=**list([**'a'**,**'d'**,4])**#列表的创建**print(a)  
  
dic4**=**{**'age':**18,**'name':'wbs'**,**'hobby':'girl'**}  
print(list(dic4.keys()))**#dic4.keys()返回的是一个dict\_keys类型的，可以将其转换成list**print(list(dic4.values()))  
print(list(dic4.items()))

## 字典的修改

**#对list的修改**li**=**[1,2,3,4]  
li[2]**=**10**#对列表的修改直接赋值就行**print(li[2])  
**#字典的修改**dic5**=**{**'age':**18,**'name':'wbs'**,**'hobby':'girl'**}  
dic5[**'age'**]**=**20**#将键取出来，直接修改**print(dic5)

## 字典的增加

**#字典的增加**dic6**=**{**'1':**1,**'2':**2}  
dic7**=**{**'3':**3,**'4':**4}  
dic6.update(dic7)**#将dic7加入到dic6中，dic7不做任何改变，如果有相同的，则更新，没有则加入dic6**print(dic6)

## 字典的删除

**#字典的删除  
del** dic6[**'1'**]**#将键值对一起删除**print(dic6)  
dic6.pop(**'2'**)**#根据索引删除，但是pop返回值是键对应的值**print(dic6)  
dic6.popitem()**#随机删除键值对，并且以元组的方式返回，用处不大**print(dic6)  
dic6.clear()**#直接清空dic6**print(dic6)

**del** dic6**#删除整个字典**

dic7**=**dict.fromkeys([**'host1'**,**'host2'**,**'hsot3'**],**'test'**)**#将三个key的值都赋值成test,一般做初始化**

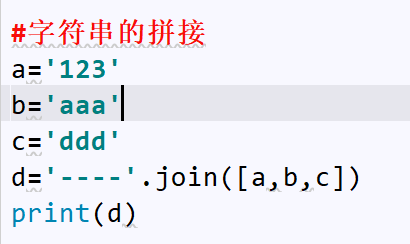
## 字典的排序

**#字典的排序**dic8**=**{3**:'666'**,7**:'555'**,2**:'444'**}  
print(sorted(dic8))**#默认按照键排序，结果是键的集合**print(sorted(dic8.values()))**#按照值排序，结果是值的集合**print(sorted(dic8.items()))**#按照键排序，结果是整个集合**

字典的遍历dic9**=**{**'age':**18,**'name':'wbs'**,**'hobby':'girl'**}  
**for** i **in** dic9**:** print(i)**#默认打印键  
for** i **in** dic9**:** print(i,dic9[i])**#打印键值对  
for** i **in** dic9.items()**:** print(i)**#打印键值**

**for** i,j **in** dic9.items()**:** print(i,j)**#打印键值**

# 字符串



结果如右：

#string的内置对象str**='hello wbs'**print(str.count(**'l'**))  
print(str.capitalize())**#将字符串首字母转换成大写**print(str.center(50,**'-'**))**#居中：--------------------hello wbs---------------------**print(str.endswith(**'bs'**))**#判断是否是以某个字符结尾**print(str.startswith(**'h'**))**#判断是否以某字符开头**

str1**='hellowo\trld'**print(str1.expandtabs(tabsize**=**10))**#在某地方加入10个空格**str.find(**'l'**)**#产找到第一个元素，并将索引值返回**

## 格式化字符串

str2**='hello {name} id {age}'**print(str2.format(name**='wbs'**,age**='20'**))**#格式化字符串方法1**print(str2.format\_map({**'name':'bo'**,**'age':**20}))**#格式化字符串方法2**

print(str2.index(**'ll'**))**#index和find一样的，但是找不到的时候会报错**print(**'abc456'**.isalnum())**#判断是否是一个字符或数字**

print(**'3'**.isidentifier())**#判断是否是一个整数**print(**'qq'**.isalpha())**#判断是否是一个字母**print(**'34abc'**.isidentifier())**#检验是否是一个非法字符**print(**'AAb'**.islower())**#判断是否全部是小写，isupper()正好相反**print(**' '**.isspace())**#判断是否是一个空格**

print(**'My Title'**.istitle())**#判断是否是一个标题，标题必须是首字母大写**print(**'My title'**.lower())**#所有的字符变成小写**print(**'My Title'**.upper())**#所有的字符编程大学**print(**'My Title'**.swapcase())**#所有的字符反转大小写**print(**'aaa'**.ljust(50,**'\*'**))**#在aaa右边跟50个\***print(**'aaa'**.rjust(50,**'\*'**))**#在aaa左边跟50个\***print(**' MMM MMM '**.strip())**#将文本前后的空格都去掉，lstrip,rstrip效果是去掉一左一右的空格**print(**'My Title'**.replace(**'Title'**,**'wbs'**))**#替换，可以加第三个参数，来控制替换的次数**print(**'My Title Title'**.replace(**'Title'**,**'bo'**,1))**#只替换一次**print(**'My title title'**.split(**' '**))**#将一个字符串分割成一个列表**print(**'My title'**.title())**#将字符串写成title的类型**

## 重点的方法：

切片、format、split、strip、replace、find、center、lower、upper、startwith、count、

# Python2中解码与编码的过程

**Unicode**

encode

decode

encode

decode

**GBK**

**utf-8**

GBK需要转换成utf-8格式流程如上：

1. 首先通过编码【decode】转换成Unicode编码
2. 然后通过解码【encode】转换成utf-8编码

# Python3转码过程

python3中默认编码是unicode，支持中文，不用转换成别的格式

注意：python3在encode时候，会转成bytes类型，解码的时候会把bytes转换成字符串

s**='王博盛'**unicode\_to\_gbk**=**s.encode(**'gbk'**)**#python中encode的时候会自动转换成一个bytes类型**print(s)**#python3是以Unicode编码，支持中文**print(unicode\_to\_gbk)**#转换成字节数组的时候是看不懂的 b'\xcd\xf5\xb2\xa9\xca\xa2'**print(unicode\_to\_gbk.decode(**'gbk'**))**#只有继续将其转换成Unicode时候，才可以是看到原来的样子，**

# 文件操作

**#读操作**f**=**open(**'wenben.txt'**,**'r'**,encoding**='utf-8'**)**#打开文件**data**=**f.read()**#读文件,read()可以加数字，比如read(5)表示读5个字符**print(data)  
f.close()  
  
  
**#写操作**f**=**open(**'wenben1.txt'**,**'w'**,encoding**='utf-8'**)**#打开文件，并清空文件，当文件不存在的时候，创建文件并清空文件的内容**f.write(**'hello world\n'**)**#写入多少个字符，指针跟着移动多少位，下次写入的时候接着从上边的地方写入**f.write(**'hello 王博盛'**)**#上边的写与这一行的写入文件都是从一行的末尾直接写入，不会自动换行**f.close()**#打开一个，一定要关闭一个文件  
  
#追加模式 a-->append**f**=**open(**'wenben1.txt'**,**'a'**,encoding**='utf-8'**)  
f.write(**'aaa'**)**#不会清空之前文本的内容**time.sleep(10)**#10s后执行下边的语句**f.close()**#缓冲区的内容写入文件中**

f**=**open(**'wenben1.txt'**,**'r'**,encoding**='utf-8'**)  
  
**for** i **in** f**:** print(i.strip())**#用for循环取出来其中的值的时候，不会全部加载到内存，是因为for循环做成了迭代器，开始打印第二行的时候，第一行的内容已经不存在了，以后读取文件就用这样的  
 #而readlines一下子会把所有的内容加载到内存中，尽量不要用Readlines**f.close()  
f**=**open(**'wenben1.txt'**,**'r'**,encoding**='utf-8'**)  
print(f.tell())**#打印当前光标的位置**print(f.read(4))  
print(f.tell())**#打印当前光标的位置，utf-8格式一个中文占三个字符，read与tell在中文的时候是对应的，但是时中文的时候是不对应的**f.seek(0)**#将光标移动到头**f.flush()**#将内存中的内容写到磁盘上  
  
import** sys,time  
**for** i **in** range(30)**:** sys.stdout.write(**'\*'**)  
 sys.stdout.flush()**#刷新将缓存区的写到磁盘或者屏幕,如果没有这句话，直接是等到所有的字符都写到缓冲区了，再直接打印到屏幕，而不是一个一个打印  
 #上边两行代码可以写成 print('\*',end='',flush=True)** time.sleep(0.2)  
  
f**=**open(**'wenben1.txt'**,**'a'**,encoding**='utf-8'**)  
f.truncate(5)**#从第五个字符后边的字符全部删掉，不加参数表示全部删掉，此方法必须要以a的方式打开文件**

# r+ w+ a+模式

**#\_\_author:bo  
#date:2018/9/1  
  
#w+ r+ a+**f**=**open(**'wenben1.txt'**,**'r+'**,encoding**='utf-8'**)**#r+模式读和正常的一样，但是写的时候是在最后写的**f**=**open(**'wenben1.txt'**,**'w+'**,encoding**='utf-8'**)**#w+模式是先清空，再去读**print(f.readline())  
f.write(**'wbs'**)**#光标跑到文本最后，再写入wbs**print(f.readline())**#此时是打印不到任何东西的，因为写的时候是将光标跑到最后，再读文件的时候是读不到东西的**f**=**open(**'wenben1.txt'**,**'a+'**,encoding**='utf-8'**)**#a+模式打开的时候其实的位置是在文本的最后边，  
#注意：r+ w+ 模式写入文本的时候永远是在文本的末尾写入的，不管光标在哪个位置，写的时候永远是在末尾写**

# 字符串与字典类型的转换

**#字典类型转换成字符串类型**s**="{'name':{'age':18}}"#此时是一个文本类型**s**=**str(s)**#此时是一个字符串类型**print(type(s))**#此时将字典类型完全转换成了一个字符串类型的对象**print(s)  
s**=**eval(s)**#此时又转换成了一个字典类型**print(s[**'name'**])

# #with语句 with open('wenben1.txt','r') as f: f.readline() f.read() print('hello')#退出with代码块的时候，with会自动关闭文件，而不用我们手动去关闭，推荐用with #可以同时打开多个文件 with open('wenben1.txt','r') as f,open('wenben1.txt','r') as f2: print('文件操作')

# 运算符

身份运算符：is 、 is not 、

成员运算符：in 、not in

a**=**1  
id(a)**#id用来查看内存地址**b**=**5  
print(b**//**2)**#/代表是正常的除，//是整除**c**=**2  
print(c**\*\***8)**#\*\*是次方  
  
  
#！/usr/bin/env python放在有些代码的第一行，起到解释器的作用，  
# 如果执行的时候使用python a.py的方式执行代码，name这个第一行的#！/usr/bin/env python会没用的**

**input不管输入的是什么，都会当做字符串来解释**

aa**=**3333333  
print(id(aa))

字符串的表示形式：三种

**'aa'  
"aaa"**s**=""""aaaaaa"""**print(s)

# 深浅拷贝

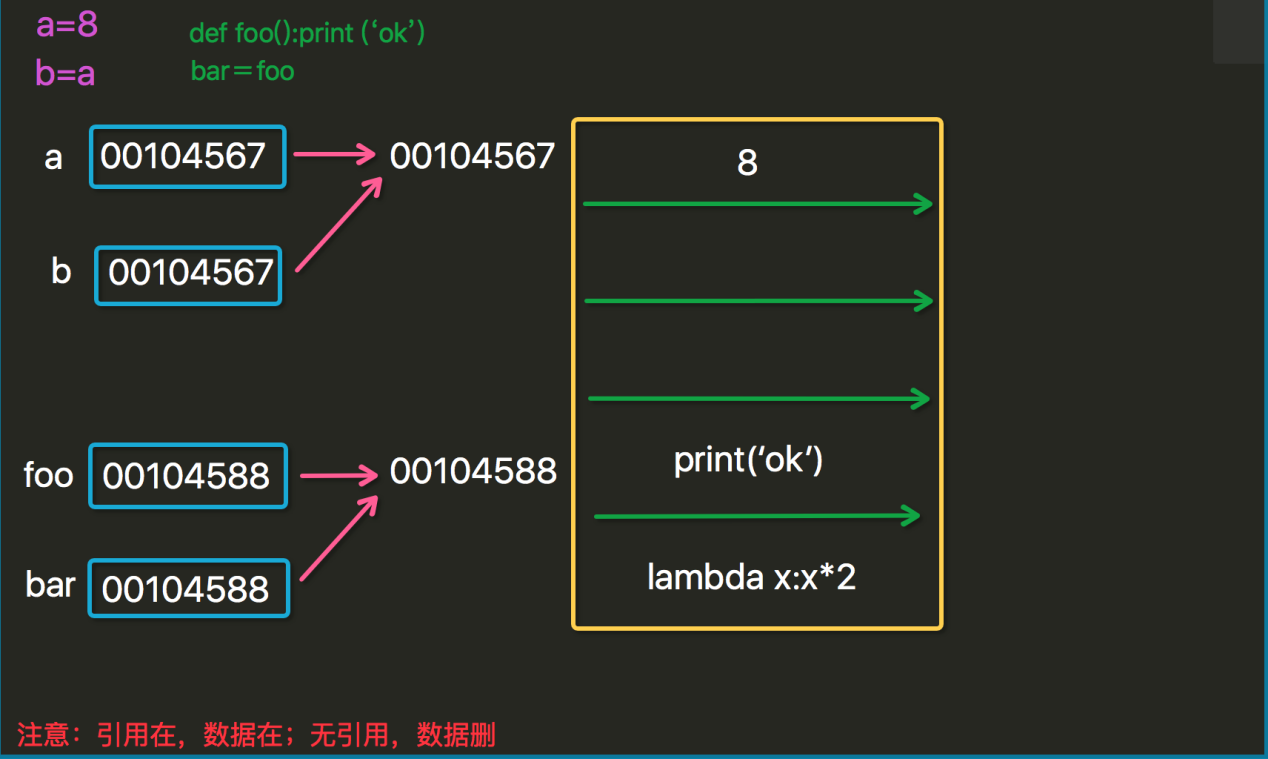
**import** copy  
s**=**[1,**'wbs'**,**'wbs2'**]  
s1**=**s.copy()**#将s列表完全拷贝一份**s1[0]**=**3**#修改拷贝后的，不会影响之前的元素**print(s)  
print(s1)**#修改字符串的时候，s不会跟着变化的**s2**=**[[1,2],**'wbs'**,**'wbs1'**]  
s3**=**s2.copy()  
s3[0][1]**=**5**#修改s3**print(s3)  
print(s2)**#s2也跟着变化，因为在修改的时候直接修改了第一个元组总第二个元素的指针**print(**'============================'**)  
a**=**[[1,2],3,4]  
b**=**a**#这种拷贝，改变一个元素，拷贝前后的元素都是保持一致的**print(a)  
print(b)  
c**=**a.copy()**#这种拷贝，普通类型的修改不会改变原来的值，叫做浅拷贝，浅拷贝意味着有些没有真正的拷贝，而是共享**c[1]**=**55555  
print(a)  
print(c)  
**#浅拷贝是拷贝第一层，深拷贝是拷贝全部**print(**"============="**)  
a**=**[[1,2],3,4]  
b**=**copy.deepcopy(a)**#这个是深拷贝**b[0][1]**=**5555**#只修改了深拷贝后的内容**print(a)  
print(b)

# 集合

**#列表**a**=**[1,2,3]  
b**=**list([4,5,6])**#创建列表的两种方式**c**=**(1,2,3)  
d**=**tuple([4,5,6])**#创建元组的两种方式  
#集合的创建方式set()**s**=**set(**'wbs'**)  
print(s)**#运行结果 {'b', 'w', 's'}，无序**s**=**set(**'wbswbs'**)  
print(s)**#运行结果 {'b', 'w', 's'}，无序,会自动去掉重复的内容  
#使用set可以去重功能**a**=**[**'wbs'**,**'xiaoPOooo'**,**'wbs'**]  
b**=**set(a)**#会自动去掉重复的**print(b)**#结果：{'xiaoPOooo', 'wbs'}  
#注意：对某一个元素进行set的时候，每一个元素必须是课Hash的，即每一个元素都是不可变的（可以是元组，不可以是字典和列表）  
#集合的访问，只能通过遍历或者迭代器**a**=**[**'wbs'**,**'xiaoPOooo'**,**'wbs'**]  
b**=**set(a)  
b.add(**'tianjia'**)**#集合的添加，将添加的内容作为一个整体添加到集合中**print(b)  
b.update(**"很好"**)**#update也是一个更新，但是这个元素会被拆分成一个个字符来对待**print(b)  
b.update([12,**"添加2"**])**#此时，12和‘添加2’被当做两个整体来对待**print(b)  
b.remove(12)**#删除元素**print(b)  
b.pop()**#随机删除一个**print(b)  
b.clear()**#清空，结果是：set()，代表一个空集合**print(b)  
**del** b**#直接删除b  
# print(b)#打印会报错**print(set(**'wbswbs'**)**==**set(**'wbs'**))**#结果为true**print(set(**'wbswbs'**)**<**set(**'wbsaaa'**))**#结果为true,判断为是否包含**print(set(**'wbswbs'**)**<**set(**'wbs'**))**#结果为false,相等的关系  
#联合 or**print(set(**'wbs'**) **and** set(**"wbsabc"**))**#and 取出两个的**print(set(**'wbs'**) **or** set(**'1243'**))**#只取一个  
#交叉并**print(**"================"**)  
a**=**set([1,2,3,4,5])  
b**=**set([3,4,5,6,7])  
print(a.intersection(b))**#取交集**print(a **&** b)**#求并集**print(a.union(b))  
print(a**|**b)**#管道符合union是一样的**print(a.difference(b))**#求交集，a中有b中没有的**print(a**-**b)**#减号是求差集**print(a.symmetric\_difference(b))**#对称差集，取两个集合中不是共有的**print(a**^**b)**#是求对称差集**

# 函数

## 作用域：



**#函数  
import** time  
**def F**()**:** print(**"ok"**)  
  
  
**for** i **in** range(10)**:** F()**#打印了10此ok  
#注意：  
#函数名是区分大小写的  
#函数名不能是保留字  
#函数名不能使用标点符号**print(**"======================"**)  
**def sum**(*a*,*b*)**:** print(*a***+***b*)  
 print(*a*)  
 print(*b*)**#实参和形参的顺序是按照参数的名称来分配**sum(5,4)  
print(**"======================"**)  
**def f**(*name*,*age*)**:** print(**'我是 %s,今年%s' %**(*name*,*age*) )  
f(**'wbs'**,**'20'**)  
f(age**=**30,name**='wbs'**)**#可以写上对应的关键字参数，效果和上边的一样  
def f1**(*name*,*sex***='man'**)**:** print(**"%s is %s"%**(*name*,*sex*) )**#可以设置一个默认的性别**f1(**'wbs'**)  
f1(**"alix"**,**'girl'**)**#也可以设定参数**time\_format**='%Y-%m-%d %X'**current\_time**=**time.strftime(time\_format)  
print(current\_time)  
  
**def f2**(*\*args*)**:** sum**=**0  
 **for** i **in** *args***:** sum**+=**i  
 print(sum)  
f2(1)  
f2(1,2,3)**#不定长的参数，输出的是一个元组  
def f3**(*\*args*,*\*\*kwargs*)**:** print(*args*)**#结果：('wbs', 12)，被处理称为一个元组** print(*kwargs*)**#结果：{'name': 'wbs', 'age': 12}，被处理称为一个字典  
 for** i **in** *kwargs***:** print(**'%s:%s'%**(i,*kwargs*[i]))**#打印键值对  
 #注意：无命名参数放在前边，有键值对的参数放在右边**f3(**'wbs'**,12,name**='wbs'**,age**=**12)  
  
**def f4**(sex**='male'**,*\*args*,**\*\***kwargs)**:#有名的参数必须在无名的前边** print(*args*)  
f4()  
f4(**'man'**,1,2,3,4)  
f4(1,2,3,4,**'wbs'**)

**#\_\_author:bo  
#date:2018/9/14  
  
#函数的返回值  
def f**()**:** print(**'function'**)  
 **return 'ok'#结束函数，返回某个对象，后边的程序不会执行，不加return的时候自动添加return None  
#注意：如果return多个对象，那么python会将多个对象封装成一个元组返回**a**=**f()  
print(a)  
  
**#函数的作用域  
#built\_in>global>enclosing>local**a**=**5**#global**b**=**int(2.9)**#built\_in  
def outer**()**:** a**=**4**#enclosing  
 def inner**()**:** b**='aa'#local** print(b)**#优先级是local>enclosing>global>built\_in** inner()  
outer()  
**#注意：局部作用域不能修改全局作用域的变量**count**=**10  
**def f**()**:** print(count)  
 **#count+=1#报错，内部引用的时候已经是全局的count，不能对他修改  
 #count=5#这句话会报错，因为是在局部中已经引用全局的变量了，但是局部内不能修改全局的变量  
def f1**()**:** count**=**5  
 print(count)**#这句是可以执行的，因为在局部中重新创建了一个count，与外边的不是一个对象  
#如果在局部的函数中间需要修改全局的变量，只需要定义一下global  
def f3**()**:  
 global** count**#表示函数内部用的count是全局的count** print(count)**#10** count**+=**1  
 print(count)**#11**f3()  
  
  
**def outer1**()**:** a**=**4**#enclosing  
 def inner1**()**:  
 nonlocal** a**#如果在局部需要声明一个enclosing的变量  
 # ，需要用nonlocal来声明，作用和用global声明全局的变量是一样的** print(a)**#4** a**+=**1  
 print(a)**#5** inner1()  
outer1()  
**#global nonlocal 在局部函数中的作用差不多是一样的，都是讲外部的变量声明为可以在函数内部可以修改的**

**def f5**(*\*args*)**:** print(*args*)  
a**=**[1,2,3]  
f5(**\***a)**#将一个元组传入到args,传入的时候带一个\*就可以了  
def f6**(*\*\*kwargs*)**:** print(*kwargs*)  
f6(name**='www'**,age**=**20)**#{'name': 'www', 'age': 20}**f6(**\*\***{**'name':'wbs'**,**'age':'12'**})**#{'name': 'wbs', 'age': '12'}  
#两个\* 号直接可以传入一个字典类型**print(**'================'**)  
**def fun1**(*n*)**:  
 return** *n***\****n***def fun**(*a*,*b*,*f*)**:** print(*f*(*a*)**+***f*(*b*))  
  
fun(3,4,fun1)**#25,可以传入一个函数  
# 函数名可以赋值，  
# 函数名字既可以作为函数的参数，又可以做为函数的返回值  
def foo**()**:  
 def foo1**()**:  
 return** 8  
 **return** foo1  
result**=**foo()  
print(result)**#得到函数foo()的地址<function foo.<locals>.foo1 at 0x000001DE682CE620>**print(result())**#得到函数的结果：8**

**#递归函数  
def fun**(*n*)**:  
 if** *n***==**1**:  
 return** 1  
 **return** *n***\***fun(*n***-**1)**# 可以用递归函数解决的问题都可以用循环解决**print(fun(5))  
**# 递归的效率很低,递归必须要有结束条件  
  
# 斐波那契数列  
# 0 1 1 2 3 5 8 13 21 34 55 。。。。  
def fibo**(*n*)**:  
 if** *n***<=**2**:  
 return** *n***-**1  
  
 **return** fibo(*n***-**1)**+**fibo(*n***-**2)  
print(fibo(8))

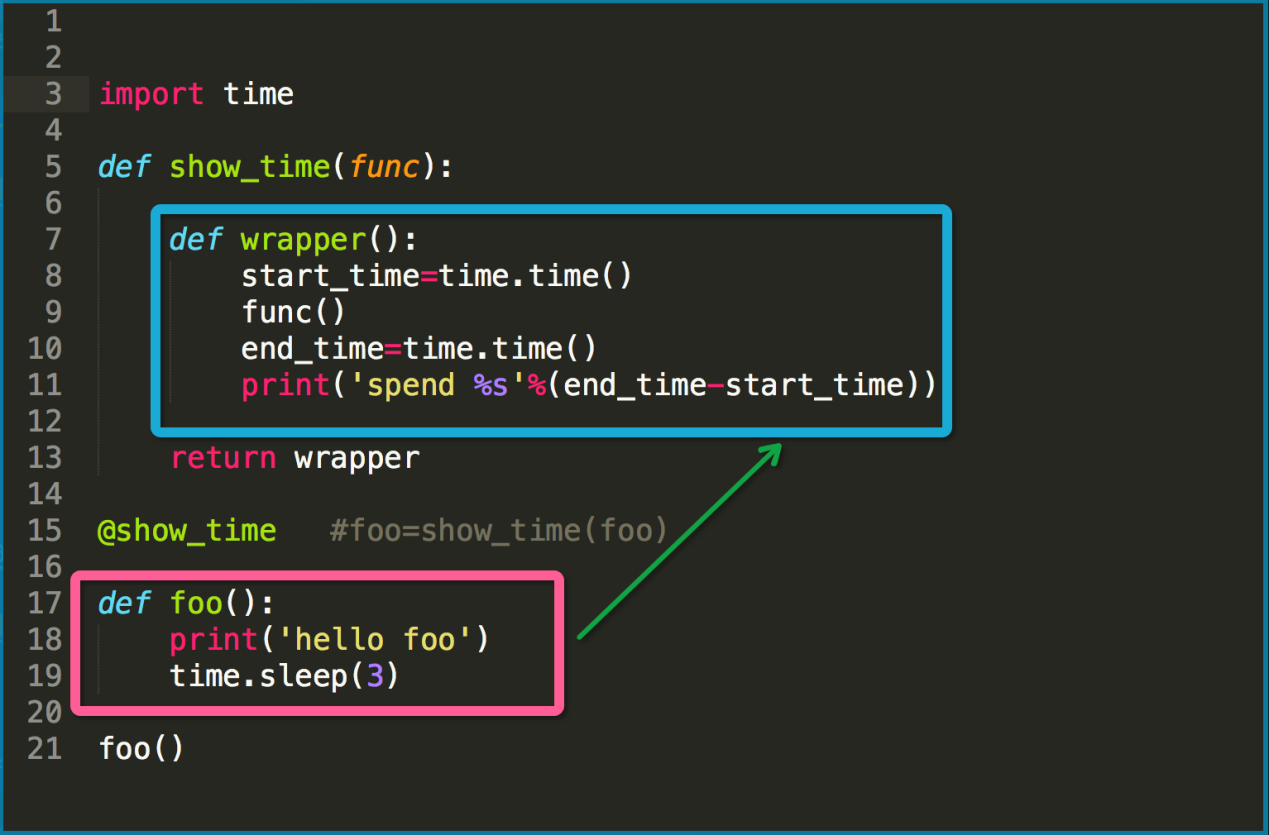
闭包(closure)是函数式编程的重要的语法结构。

定义：如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包(closure).

**@show\_time # foo=show\_time(foo)  
def foo**()**:** print(**'hello foo'**)  
 time.sleep(3)  
  
  
**@show\_time # bar=show\_time(bar)  
def bar**()**:** print(**'in the bar'**)  
 time.sleep(2)  
  
  
foo()  
print(**'\*\*\*\*\*\*\*\*\*\*\*'**)  
bar()

如上所示，这样我们就可以省去bar = show\_time(bar)这一句了，直接调用bar()即可得到想要的结果。如果我们有其他的类似函数，我们可以继续调用装饰器来修饰函数，而不用重复修改函数或者增加新的封装。这样，我们就提高了程序的可重复利用性，并增加了程序的可读性。

      这里需要注意的问题：  foo=show\_time(foo)其实是把wrapper引用的对象引用给了foo，而wrapper里的变量func之所以可以用，就是因为wrapper是一个闭包函数。



@show\_time帮我们做的事情就是当我们执行业务逻辑foo()时，执行的代码由粉框部分转到蓝框部分，仅此而已！

### 带参数的被装饰函数

**import** time  
  
  
**def show\_time**(*func*)**:  
 def wrapper**(*a*, *b*)**:** start\_time **=** time.time()  
 func(*a*, *b*)  
 end\_time **=** time.time()  
 print(**'spend %s' %** (end\_time **-** start\_time))  
  
 **return** wrapper  
  
  
**@show\_time # add=show\_time(add)  
def add**(*a*, *b*)**:** time.sleep(1)  
 print(*a* **+** *b*)  
  
  
add(2, 4)

**import** time  
  
**def show\_time**(*func*)**:  
  
 def wrapper**(*a*,*b*)**:** start\_time**=**time.time()  
 ret**=**func(*a*,*b*)  
 end\_time**=**time.time()  
 print(**'spend %s'%**(end\_time**-**start\_time))  
 **return** ret  
  
 **return** wrapper  
  
**@show\_time #add=show\_time(add)  
def add**(*a*,*b*)**:** time.sleep(1)  
 **return** *a***+***b*print(add(2,5))

**#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*不定长参数  
import** time  
  
**def show\_time**(*func*)**:  
  
 def wrapper**(*\*args*,*\*\*kwargs*)**:** start\_time**=**time.time()  
 func(**\****args*,**\*\****kwargs*)  
 end\_time**=**time.time()  
 print(**'spend %s'%**(end\_time**-**start\_time))  
  
 **return** wrapper  
  
**@show\_time #add=show\_time(add)  
def add**(*\*args*,**\*\***kwargs)**:** time.sleep(1)  
 sum**=**0  
 **for** i **in** *args***:** sum**+=**i  
 print(sum)  
  
add(2,4,8,9)

**不定长参数**

**#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*不定长参数  
import** time  
  
**def show\_time**(*func*)**:  
  
 def wrapper**(*\*args*,*\*\*kwargs*)**:** start\_time**=**time.time()  
 func(**\****args*,**\*\****kwargs*)  
 end\_time**=**time.time()  
 print(**'spend %s'%**(end\_time**-**start\_time))  
  
 **return** wrapper  
  
**@show\_time #add=show\_time(add)  
def add**(*\*args*,**\*\***kwargs)**:** time.sleep(1)  
 sum**=**0  
 **for** i **in** *args***:** sum**+=**i  
 print(sum)  
  
add(2,4,8,9)

### 带参数的装饰器

装饰器还有更大的灵活性，例如带参数的装饰器：在上面的装饰器调用中，比如@show\_time，该装饰器唯一的参数就是执行业务的函数。装饰器的语法允许我们在调用时，提供其它参数，比如@decorator(a)。这样，就为装饰器的编写和使用提供了更大的灵活性。

**import** time  
  
  
**def time\_logger**(*flag***=**0)**:  
 def show\_time**(*func*)**:  
 def wrapper**(*\*args*, *\*\*kwargs*)**:** start\_time **=** time.time()  
 func(**\****args*, **\*\****kwargs*)  
 end\_time **=** time.time()  
 print(**'spend %s' %** (end\_time **-** start\_time))  
 **if** flag**:** print(**'将这个操作的时间记录到日志中'**)  
 **return** wrapper  
 **return** show\_time  
  
  
**@time\_logger**(3)  
**def add**(*\*args*, **\*\***kwargs)**:** time.sleep(1)  
 sum **=** 0  
 **for** i **in** *args***:** sum **+=** i  
 print(sum)  
  
  
add(2, 7, 5)

@time\_logger(3) 做了两件事：

    （1）time\_logger(3)：得到闭包函数show\_time，里面保存环境变量flag

    （2）@show\_time   ：add＝show\_time(add)

上面的time\_logger是允许带参数的装饰器。它实际上是对原有装饰器的一个函数封装，并返回一个装饰器(一个含有参数的闭包函数)。当我 们使用@time\_logger(3)调用的时候，Python能够发现这一层的封装，并把参数传递到装饰器的环境中。

**#装饰器加参数  
# 需求：传入的参数为true的时候打印日志，出入false时候不打印日志  
import** time  
**def logger**(*flag***=''**)**:  
 def show\_time**(*f*)**:  
 def inner**(*\*x*, *\*\*y*)**:** start **=** time.time()  
 f(**\****x*, **\*\****y*)  
 time.sleep(1)  
 end **=** time.time()  
 **if** flag**:** print(**'打印日志'**)  
 print(**'spend time is%s' %** (end **-** start))  
 **return** inner  
 **return** show\_time  
**@logger**(**'false'**)**#这个不仅可以在需要的时候打印日志，而且可以计算时间  
def add**(*\*a*,**\*\***b)**:** sums**=**0  
 **for** i **in** *a***:** sums**+=**i  
 print(**'sums is '**,sums)  
add(1,2,3)

# 列表生成器

**#\_\_author:bo  
#date:2018/9/16  
#列表生成器**a**=**[x **for** x **in** range(10)]  
b**=**[x**\***2 **for** x **in** range(10)]**#先取出列表中的元素，再对列表中的元素乘以2，最后一个一个放入列表中**print(a)  
print(b)  
print(**'================================================================'**)  
**def f**(*n*)**:  
 return** *n***\****n*a**=**[f(x) **for** x **in** range(10)]**#这种方式直接将每个元素放入了内存中**print(a)**#也可以在一个列表中传入一个函数  
#简单的赋值方法**t**=**[**'wbs'**,21]  
a,b**=**t**#简单的取出列表中的元素的方法，接受元素的参数个数不能多出列表中的元素的个数  
#上边的语句相当于是**a**=**t[0]  
b**=**t[1]  
print(a,**' '**,b)  
**#a,b,c=t#报错not enough values to unpack (expected 3, got 2)  
#t=['wbs',21,22]# 报错 too many values to unpack (expected 2)，值太多**a,b**=**t  
print(a,**' '**,b)  
  
  
a**=**(x **for** x **in** range(10))**#创建生成器**print(a)**#<generator object <genexpr> at 0x000001F1C3E21C50>  
#print(a.\_\_next\_\_())#\_\_next\_\_()是内部方法，不建议使用**print(next(a))**#等价于上边的方法，建议使用,python2中是a.next()**print(next(a))**# 这次打印的时候是从上边的值的下一个  
  
#生成器就是一个可迭代对象  
# 用for循环来遍历一个生成器的时候，不会将生成器的所有元素加载到  
# 内存，它只是用一个加载一个，然后再用垃圾回收机制回收一个  
for** i **in** a**:** print(i)  
  
**#生成器的两种创建方式  
# 1 a=(x for x in range(10))  
# 2 yield方式创建  
def f**()**:#yield方式创建生成器** print(**'ok'**)**#生成器中的代码不会执行  
 yield** 1**#yield相当于是返回一个1，下边的代码不在执行** print(**'ok2'**)  
 **yield** 2**# 返回的是1 ，** print(**'ok3'**)  
 **yield** 3**#相当于返回的是2**g**=**f()**#是一个生成器对象，而不是执行这个函数  
# next(g)  
# next(g)#第二个next是从上一个next接着开始的，而不是从头开始的  
# next(g)#得到的是f()返回的结果  
for** i **in** g**:#每次进入到g=f()生成器的时候，会执行next(f())这段代码，执行完了将其yield 1返回给i，在for循环中打印了出来** print(i)  
**#上边的循环的结果是：  
# ok  
# 1 #这个1是f（）函数返回的那个yield 1，在for循环中将其打印了处理啊  
# ok2 #这个ok2是在for循环中的g=f()生成器自动执行next(f())的时候打印的  
# 2 #这个2是f（）函数返回的那个 yield 2，在for循环中将其打印了处理啊  
# ok3 #这个ok3是在for循环中的g=f()生成器自动执行next(f())的时候打印的  
# 3**a**=**[1,1,2,3]  
a.\_\_iter\_\_()  
b**=**(1,2,3)  
b.\_\_iter\_\_()  
c**=**{**'name':'wbs'**}  
c.\_\_iter\_\_()  
**#for循环后边加的是可迭代对象  
#什么是可迭代对象？  
# 看是否是可迭代对象，只需要看它是否有————next（）————方法或者\_\_iter\_\_()方法  
# 字典、集合、元组都是可迭代对象**print(**'=================='**)  
**#斐波那契数列  
def fib**(*max*)**:** n,before,after**=**0,0,1  
 **while** n**<***max***:** print(before)  
 before,after**=**after,before**+**after  
 n**+=**1  
fib(10)  
  
print(**'==================='**)  
**def fib**(*max*)**:** n,before,after**=**0,0,1  
 **while** n**<***max***:  
 yield** before**#用yield返回一个值，在下边需要的时候再执行，这样就不会直接加载到内存中** before,after**=**after,before**+**after  
 n**+=**1  
g**=**fib(10)  
print(next(g))  
print(next(g))  
print(next(g))  
print(next(g))  
print(next(g))  
print(next(g))  
print(next(g))  
print(next(g))  
  
print(**'==================='**)  
**def f**()**:** print(**'ok'**)  
 count**=yield** 1**#可以给yield加一个接受值的参数** print(count)  
 print(**'ok2'**)  
 **yield** 2  
 print(**'ok3'**)  
 **yield** 3  
b**=**f()**# 将生成器赋给b**s**=**b.send(**None**)**# 此句话相当于是next（b）,第一次send前如果没有next，只能给它传入一个None  
#用send执行的时候，第一次必须传入None这个参数，不然会 报错**print(s)  
s**=**b.send(**'wbs'**)**# 在send中传入值的时候，可以在yield中接收到这个值  
# 注意：第一次没有执行next并且执行send的时候，必须要传入None，因为第一次的时候没有  
# 执行到yield，因此没有变量去接收传入的参数，会报错**print(s)

生成器都是迭代器，但是迭代器不一定是生成器

a**=**[1,2,3,4]  
b**=**iter(a)**# iter()生成一个迭代对象并且返回  
# 迭代器对象的两个条件  
# 有iter()方法 有next()方法  
  
# for循环做的三件事  
 # 1 调用可迭代对象的iter()方法返回一个可迭代对象  
 # 2 不断调用迭代器对象的next方法  
 # 3.处理StopIteration  
  
from** collections **import** Iterator,Iterable  
print(isinstance([1,2,3],list))**# 判断前边的[1,2,3]是不是list**print(isinstance([1,2,3],Iterable))**# 判断是否是一个可迭代对象  
# Iterable是可迭代对象，Iterator是迭代器**

时间  
**#\_\_author:bo  
#date:2018/9/17  
  
import** time  
print(help(time))  
print(time.time())**# 打印时间戳**print(time.clock())**# 计算CPU花费的时间**print(time.gmtime())**# UTC标准时间，结构化时间  
# 结果： time.struct\_time(tm\_year=2018, tm\_mon=9, tm\_mday=17, tm\_hour=14, tm\_min=30, tm\_sec=4, tm\_wday=0, tm\_yday=260, tm\_isdst=0)**print(time.localtime())**# 东八区的时间**print(time.strftime)**# 字符串时间**print(help(time.strftime))**# 字符串时间  
  
# %Y Year with century as a decimal number.  
# %m Month as a decimal number [01,12].  
# %d Day of the month as a decimal number [01,31].  
# %H Hour (24-hour clock) as a decimal number [00,23].  
# %M Minute as a decimal number [00,59].  
# %S Second as a decimal number [00,61].  
# %z Time zone offset from UTC.  
# %a Locale's abbreviated weekday name.  
# %A Locale's full weekday name.  
# %b Locale's abbreviated month name.  
# %B Locale's full month name.  
# %c Locale's appropriate date and time representation.  
# %I Hour (12-hour clock) as a decimal number [01,12].  
# %p Locale's equivalent of either AM or PM.**struct\_time**=**time.localtime()  
print(time.strftime(**'%Y-%m--%d %H:%M:%S'**,struct\_time))**# 可以自己拼接时间**print(time.strftime(**'%Y-%m--%d %H:%M:%S'**))**# 可以自己拼接时间**print(time.strptime(**'2018-09--17 22:47:56'**,**'%Y-%m--%d %H:%M:%S'**))**# 转换成结构化时间  
# 结果：time.struct\_time(tm\_year=2018, tm\_mon=9, tm\_mday=17, tm\_hour=22, tm\_min=47, tm\_sec=56, tm\_wday=0, tm\_yday=260, tm\_isdst=-1)**a**=**time.strptime(**'2018-09--17 22:47:56'**,**'%Y-%m--%d %H:%M:%S'**)  
print(a.tm\_mday)**# 可以将一个时间转换成结构化时间后再取出其中的具体参数的值**print(time.ctime())**# Mon Sep 17 22:52:41 2018，也可以加一个时间戳，来转换成一个可以显示的时间**print(time.ctime(12344))**# 转换成标准时间后为：Thu Jan 1 11:25:44 1970**print(time.mktime(time.localtime()))**# 1537196076.0,结构化方法转换成一个时间戳**print(time.localtime())**# 打印结构化时间  
  
import** datetime  
print(datetime.datetime.now())**# 2018-09-17 22:56:28.235684**