1. 类变量，即类的静态变量。

在类的声明后面，直接声明的变量为类的静态变量。  
关键一：如何在成员方法中访问全局的类变量。  
从上面的代码可以看到。通过  
类名打点的方式：Person.population 或者  
self.\_\_class\_\_ 访问到类，再访问到变量。  
原本应该是直接 Person.population 就可以了。  
但是在 \_\_del\_\_() 方法中，直接Person.population 访问会报错。所以，只能通过 self.\_\_class\_\_.population 来访问。

二、类的静态方法。  
类的静态方法，可以不生成实例，直接通过类名调用。并且为所有的实例所共享。  
通过在方法上面加一个 @staticmethod 来实例。定义为静态方法，也不需要传递self参数了。

Global声明的是全局变量，不属于“类.变量”

编码问题：

低级编辑器有默认编码格式（比如editplus默认编码选项为ANSI;这些使用 1 至 4 个[字节](http://www.bing.com/knows/%E5%AD%97%E8%8A%82" \t "http://www.bing.com/knows/_blank)来代表一个[字符](http://www.bing.com/knows/%E5%AD%97%E7%AC%A6" \t "http://www.bing.com/knows/_blank)的各种[汉字](http://www.bing.com/knows/%E6%B1%89%E5%AD%97" \t "http://www.bing.com/knows/_blank)延伸[编码方式](http://www.bing.com/knows/%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F" \t "http://www.bing.com/knows/_blank)，称为 ANSI 编码。在简体中文[Windows操作系统](http://www.bing.com/knows/Windows%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \t "http://www.bing.com/knows/_blank)中，ANSI 编码代表 [GBK](http://www.bing.com/knows/GBK%E5%AD%97%E5%BA%93" \t "http://www.bing.com/knows/_blank) 编码；在日文[Windows操作系统](http://www.bing.com/knows/Windows%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \t "http://www.bing.com/knows/_blank)中，ANSI 编码代表 Shift\_JIS 编码。 不同 ANSI 编码之间互不兼容）,PYTHON开通的#coding：声明是说当前的内容是以什么编码格式写的，但是与实际的编码格式不一定一样，在u” ”，或().encode(“utf-8”)方法把时间编码格式转换时可能不能转成功；一些比较文本匹配的方法时，也设置两比较内容的编码格式匹配问题。

import A 是指使用A中所有的属性和方法（即变量和函数）

From A Import \* 是A中所有属性和方法的一个备份 ，如果用方法初始化A中的变量(备份后再进行的初始化），并不会初始化备份过来的变量，但可以用return返回A中已初始化的变量。

三、正则相关：

举例：xx = re.compile(*'^(.\*)#([^#]\*)$'*)

Url=*'python-reg-expressions.html#zhu#sifen#abc123'*

ma = xx.match(Url)

ma.group(1, 2) : (python-reg-expressions.html#zhu#sifen', 'abc123')

ma.group() : 'python-reg-expressions.html#zhu#sifen#abc123'

ma.group(1) : 'python-reg-expressions.html#zhu#sifen'

ma.group(2) : 'abc123'

xx.group() 等同于 xx.group(0)，表示匹配到的完整文本字符

xx.group(1) 得到第一组匹配结果，也就是(.\*)匹配到的

xx.group(2) 得到第二组匹配结果，也就是(.\*?)匹配到的

好像是有括号’()’里的才形成一个group

1. HTTP发送：

代码一：登录

# encoding: utf-8

import urllib2

import json

def **send\_req**(url):

# header={'Host':host}

headers = { # 灰色的是不需要的

# 'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.108 Safari/537.36',

# 'Accept':'application/json',

# 'Content-Type':'application/json; charset=UTF-8'

# 'Connection': 'keep-alive',

*'Content-Type'*: *'application/json;charset=UTF-8'*,

*'Accept'*: *'application/json'*,

# 'Content-Length': '50',

# 'Host':'10.253.5.178:8088'

# 'User-Agent': 'Apache-HttpClient/4.5.3 (Java/1.8.0\_144)'

}

*''' data = urllib.urlencode({"username":"zhusifen", "password":"abc123"}) #这不行 '''*

data = *"{'username':'zhusifen', 'password':'abc123'}"* #这行

*'''* data，应该是跟报文头的'Content-Type'值有关 *'''*

req=urllib2.Request(url, headers=headers) # data放在requerest里或urlopen都可以

f = urllib2.urlopen(req, data, timeout=500)

the\_page = f.read()

print f

print the\_page

data = json.loads(the\_page) # str转成dict格式

f.close()

url=*'http://10.253.5.178:8088/login'*

if \_\_name\_\_==*"\_\_main\_\_"*:

send\_req(url)

打印结果：*<addinfourl at 46763912L whose fp = <socket.\_fileobject object at 0x0000000002BA5408>>*

*{ "fail":false,*

*"processing":false,*

*"remoteRespNo":"5aa63226b8ab4be4a7197fefd061e279",*

*"resultCode":"SUCCESS",*

*"resultMessage":"成功",*

*"resultStatus":"SUCCESS",*

*"success":true,*

*"tenantId":null,*

*"token":"14846695490001ef96b35d5dab107e461514947885916"}*

代码二：

# coding: utf-8

import urllib

import urllib2

import json

class **port\_http**():

def **post\_F101**(*self*, mmlx, oldmm, newmm):

data = urllib.urlencode({*"Mmlx"*: mmlx, *"Zjzh"*: *self*.user\_id,

*"Oldmm"*: oldmm, *"Newmm"*: newmm,

*"Type"*: *"101"*, *"User\_id"*: *self*.user\_id})

# urlencode结果的格式是 ’username=zhusifen&password=abc123’

url = *self*.url + *'/api/Customer/F101'*

send\_headers = {*"em\_tg\_session"*: *self*.session} # self.session是通过一个登录接口返回获得的

req = urllib2.Request(url, headers=send\_headers)

f = urllib2.urlopen(req, data, timeout=10) # data的格式可能跟服务器有关

the\_page = f.read()

f.close()

data = json.loads(the\_page) # str转成dict格式

return data

代码三：

Urllib

-- GET

f=urllib.urlopen("http://m.cnblogs.com/") s=f.read()

print s

带参数

params = urllib.urlencode({'id': 8, 'name': 'jack', 'age': 25}) # http get的方式用到urlencode

f = urllib.urlopen("http://localhost:18797/MailClient/test.aspx?%s" % params)

print f.read()

-- POST

params = urllib.urlencode({'id': 8, 'name': 'jack', 'age': 25})

f = urllib.urlopen("http://localhost:18797/MailClient/test.aspx",params)

print f.read()

函数原型

urllib.urlopen(url[, data[, proxies]])

urllib2：

可以详细的定义header参数

Httplib：

最底层更灵活更强大的http请求和响应

1. Python连接Mysql操作（使用MySQLdb模块）

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

import MySQLdb

conn= MySQLdb.connect(

host=*'10.253.109.205'*,

port = 3306,

user=*'root'*,

passwd=*'root123!@#'*,

db =*'wb\_xdy\_config\_00'*,

)

# 使用cursor()方法获取操作游标

cursor = conn.cursor()

# 一次插入一条记录

sqli=*"insert into cfg\_date values(%s,%s,%s,%s,%s,%s,%s,%s)"*

cursor.execute (sqli,(*'1'*,*'2017-08-21'*,*'HOLIDAY'*,*'ZSF'*,*'2017-08-23'*,*'ZSF'*,*'2017-08-23'*,*'Y'*))

cursor.execute (*"insert into cfg\_date values('1','2','3','4')"*)

# 一次插入多条记录

sqli=*"insert into cfg\_date values(%s,%s,%s,%s)"*

cur.executemany(sqli,[

(*'3'*,*'Tom'*,*'1 year 1 class'*,*'6'*),

(*'3'*,*'Jack'*,*'2 year 1 class'*,*'7'*),

(*'3'*,*'Yaheng'*,*'2 year 2 class'*,*'7'*),

])

# 搜索数据

n = cursor.execute (*"SELECT \* FROM xdy2\_usr\_00.usr\_account where id < 1122"*) # 获得表中有多少条数据

print cursor.fetchone() # 从当前游标开始往后取一条，同时游标向下移动

cursor.scroll(0,*'absolute'*) # 游标调到第一位置

info = cursor.fetchmany(n) # 从当前游标开始往后取多条，没有那么多条就取到完

for ii in info:

print ii

# 关闭数据库连接

cursor.close()

conn.commit() # 方法在提交事物，在向数据库插入一条数据时必须要有这个方法，否则数据不会被真正的插入

conn.close()

代码四：

--------------------

req = urllib2.Request(url, **data=data**, **headers=headers**)

f = urllib2.urlopen(req, timeout=50)

--------------------

req = urllib2.Request(url, **headers=headers**)

f = urllib2.urlopen(req, **data=data**, timeout=50)

--------------------

**# data 放在urllib2.Request 和 放在urllib2.urlopen一样，但是“headers=headers”** **keyword arg入参的方式必须加上**，否则headers的值会入参到Request的data的位置，虽然urlopen处会把值用真正的data替换（看源代码）回来才往下走，但还是报错，原因看源代码没看出原因！

1. python默认编码问题

临时设置默认编码：

import sys

sys.setdefaultencoding("utf-8") # 文件里此次运行临时设置默认编码是 sys.getdefaultencoding() # get默认编码是什么

==================================

如果你在python中进行编码和解码的时候，不指定编码方式，那么python就会使用defaultencoding。

比如上一节例子中将str编码为另一种格式，就会使用defaultencoding。

等价于 u = unicode("人生苦短",defaultencoding)

==================================  
如何永久地将默认编码设置为utf-8呢?  有2种方法:   
==================================  
第一个方法<不推荐>: 编辑site.py, 修改setencoding()函数, 强制设置为 utf-8   
第二个方法<推荐>: 增加一个名为 sitecustomize.py, 推荐存放的路径为 site-packages 目录下  
sitecustomize.py 是在 site.py 被import 执行的, 因为 sys.setdefaultencoding() 是在 site.py 的最后删除的, 所以, 可以在 sitecustomize.py 使用 sys.setdefaultencoding().

#file name:  sitecustomize.py

import sys

sys.setdefaultencoding('utf-8')

既然 sitecustomize.py 能被自动加载,  所以除了设置编码外, 也可以设置一些其他的东西

==================================

1. 字典和字符串转换问题

json.dumps是将dict转化成str格式，json.loads是将str转化成dict格式。

Json.dump和dumps.load入参加一个类似于文件指针

Json.dumps（有个ensure\_ascii参数设置成False） 比str格式化的**编码**更更专业，

（最好用json库来互转，不然容易有坑）

传http请求没问题

# encoding: utf-8

import json

t = {*u'regSuccess'*: True, *u'success'*: True, *u'processing'*: False, *u'resultMessage'*: *u'\u6210\u529f'*}

print json.dumps(t)

data = {*"productCode"*:*"productCodex2"*,*"productName"*:*"产品名x"*}

print json.dumps(data)

print json.dumps(data,ensure\_ascii=false)

print str(data)

data = {*"productCode"*:*"productCodex2"*,*"productName"*:*u"产品名x"*}

print json.dumps(data)

print json.dumps(data,ensure\_ascii=false)

print str(data)

打印：

{"processing": false, "regSuccess": true, "success": true, "resultMessage": "\u6210\u529f"}

{"productCode": "productCodex2", "productName": "\u4ea7\u54c1\u540dx"}

{"productCode": "productCodex2", "productName": "\u4ea7\u54c1\u540dx"}

{'productCode': 'productCodex2', 'productName': '\xe4\xba\xa7\xe5\x93\x81\xe5\x90\x8dx'}

{"productCode": "productCodex2", "productName": "\u4ea7\u54c1\u540dx"}

{"productCode": "productCodex2", "productName": "\u4ea7\u54c1\u540dx"}

{'productCode': 'productCodex2', 'productName': u'\u4ea7\u54c1\u540dx'}

json.dumps() 是将一个Python数据结构转换为一个JSON编码的字符串

json.loads() 是将一个JSON编码的字符串转换为一个Python数据结构

如下：

>>> import json

>>> a={'name':'xiaoming'}

>>> json.dumps(a)

'{"name": "xiaoming"}'

>>> b='{"name": "xiaoming"}'

>>> json.loads(b)

{u'name': u'xiaoming'}

注意：我们原先定义a的元素都是用单引号，而dumps转化后都变成了双引号，而在loads后变成python变量时，元素都变成了单引号，并且字符串前加多了个u。

一般要求当要字符串通过loads转为python数据类型时，得外层用单引号，里面元素key和value用双引号

x = {*u'accountNo'*: *u'abcd'*}

print x[*"accountNo"*].encode(*"utf-8"*)

y = {*"yy"*:x[*"accountNo"*]}

print y

y = {*"yy"*:x[*"accountNo"*].encode(*"utf-8"*)}

print y

print y[*"yy"*]

y = {*"yy"*:x[*"accountNo"*].encode(*"gbk"*)}

print y

print y[*"yy"*]

打印：

abcd

{'yy': u'abcd'}

{'yy': 'abcd'}

abcd

{'yy': 'abcd'}

abcd

x = {*u'accountNo'*: *u'中文'*}

print x[*"accountNo"*].encode(*"utf-8"*)

y = {*"yy"*:x[*"accountNo"*]}

print y

y = {*"yy"*:x[*"accountNo"*].encode(*"utf-8"*)}

print y

print y[*"yy"*]

y = {*"yy"*:x[*"accountNo"*].encode(*"gbk"*)}

print y

print y[*"yy"*]

打印：

中文

{'yy': u'\u4e2d\u6587'}

{'yy': '\xe4\xb8\xad\xe6\x96\x87'}

中文

{'yy': '\xd6\xd0\xce\xc4'}

����

1. import和from import命名空间

首先，在python，

5文件,共享变量x存在glovar.py里面，需要在任意模块都能更改，更改后任意模块获取到的值都相同，这就是多模块文件共享一个变量

#5个文件的路径

c:\test\main.py

c:\test\show.py

c:\test\test.py

c:\test\common\\_\_init\_\_.py #空文件

c:\test\common\glovar.py

c:\test\common\calc.py

#glovar.py

x = 1

#show.pyfrom comon import glovarclass Show:

def showchange(self):

if glovar.x!=1:

print 'show x change'

else:

print 'show x=1'

#test.pyfrom comon.glovar import xclass Test:

def testchange(self):

if x!=1:

print 'test x change'

else:

print 'test x=1'

#calc.pyimport glovarclass Calc:

def add(self):

glovar.x+=1

#main.pyfrom comon import glovar

c = Calc()

c.add()

s = Show()

s.showchange()

t = Test()

t.testchange()

运行main.py

#输出

show x changetest x=1

上面这些文件，除了test.py失败，其他文件都能共享变量能成功,是因为python的命名空间相同，靠的就是import用正确

来看看这三种不同

import glovar

from comon import glovar

from common.glovar import x

Python import 包的机制是,import进来的和默认的系统的module了,都放在sys.module这个字典里面.

多个py文件再次import的时候,会先去sys.module里面检查是否已经import了,如果已经import了,就不再重复import,否则就import进来

===

import glovar和from comon import glovar的命名空间是一样的，key都是glovar

from common.glovar import x 则不一样，test.py中这样from import，就相当于在test.py文件中写了一行代码x = 1,

此时x就是test自己命名空间中的变量。所以x只在test.py中有效，无聊自己如何对x修改，都无法影响glovar 中的x

#换种说法from glovar import x#等同于

improt glovar

x= glovar.x#你调用x的时候，就是自己的命名空间中的x#当你执行x=2时，完全影响不到glovar.x

下面是解释

#引入当前glovar文件，此时namespace是glovarimport glovar

#从common目录中引入glovar，此时namespace是glovarfrom comon import glovar

#从common目录中的glovar文件中引入x=1，此时namespace就是引用者，也就是上面的test.pyfrom common.glovar import x

所以，如果需要共享变量，就不要使用from file import x这种形式，而是使用import file,然后就可以通过file.x来使用，然后file.x='abc'可以进行修改。这样都这样处理全局性的变量就可以共享的。也就是保持一个独立的namespace，这样python不会再次导入，从而实现共享

1. 装饰器真谛领悟

*遇到@符号（@糖语句），\*会立即执行一次\*@右边的装饰函数，把@下行起遇到的第一个对象A的引用（要求遇到‘函数对象的引用/类对象的引用/@糖语句’；定义函数或类是定义一个变量指向一个引用；@糖语句返回的也是一个引用）作为入参，执行后装饰函数返回的结果的地址替换刚才那个对象A的引用。*

*@糖语句上有入参,则作为@右侧函数的第一批入参，之后@下的对象引用作为第一批返回结果的第二批入参*

1. MYSqlDb操作

（安装见python环境搭建文档）

import MySQLdb

①先数据库连接

db = MySQLdb.connect("localhost","username","pwd","schemel",charset="utf8" )

②使用cursor()方法获取操作游标

cursor = db.cursor()

③使用execute方法执行SQL语句

*a例:*

cursor.execute("SELECT VERSION()")

*b例:*

sql = """CREATE TABLE EMPLOYEE (

FIRST\_NAME CHAR(20) NOT NULL,

LAST\_NAME CHAR(20),

AGE INT,

SEX CHAR(1),

INCOME FLOAT )"""

或sql = "INSERT INTO EMPLOYEE(FIRST\_NAME, \

LAST\_NAME, AGE, SEX, INCOME) \

VALUES ('%s', '%s', '%d', '%c', '%d' )" % \

('Mac', 'Mohan', 20, 'M', 2000)

cursor.execute(sql) # 执行增删改查语句

*c例：*

sqli="insert into cfg\_date values(%s,%s,%s,%s,%s,%s,%s,%s)"

cursor.execute (sqli,('1','2017-08-21','HOLIDAY','ZSF','2017-08-23','ZSF','2017-08-23','Y'))

cursor.execute ("insert into cfg\_date values('1','2','3','4')")

④try:

# 执行sql语句

cursor.execute(sql)

# 提交到数据库执行

db.commit() # 涉及到改动操作的要提交才行

except:

# 发生错误时回滚

db.rollback()

⑤查询用到的语句（

data = cursor.fetchone()

cursor.fetchall() # 接收全部的返回结果行(查询结果是数组包数组的结构）

cursor.fetchone() # 从当前游标开始往后取一条，同时游标向下移动(查询结果是一个数组的结构）

cursor.scroll(0,'absolute') # 游标调到第一位置

info = cursor.fetchmany(n) # 从当前游标开始往后取多条，没有那么多条就取到完.

rowcount: 这是一个只读属性，并返回执行execute()方法后影响的行数

⑥关闭数据库连接

db.close()

其他

1.Python要立即执行的函数，必须在该代码位置的前面位置就import或者def了，如果不立即执行只是用了其他函数，其他函数也要在立即执行部分的前面。

2.**os.path.join()**是在拼接路径的时候用的。举个例子，  
 os.path.join(“home”, "me", "mywork")  
 在Linux系统上会返回: “home/me/mywork"  
 在Windows系统上会返回:"home\me\mywork"  
 好处是可以根据系统自动选择正确的路径分隔符"/"或"\"

3. 列表式推导式

{/[exp表达式 **for** item **in** collection **if** condition ]/}

Item可用在表达式和条件中

列表式推导式不能用于( )，不能生成元祖‘tuple’，而只是迭代式‘generator’

4.#any(x)判断x对象是否为空对象，如果都为空、0、false，则返回false，如果不都为空、0、false，则返回true

#all(x)如果all(x)参数x对象的所有元素不为0、''、False或者x为空对象，则返回True，否则返回False

>>> any('123')

True

>>> any([0,1])

True

>>> any([0,'0',''])

True

>>> any([0,''])

False

>>> any([0,'','false'])

True

>>> any([0,'',bool('false')])

True

>>> any([0,'',False])

False

>>> any(('a','b','c'))

True

>>> any(('a','b',''))

True

>>> any((0,False,''))

False

>>> any([])

False

>>> any(())

False

>>> all(['a', 'b', 'c', 'd']) #列表list，

True

>>> all(['a', 'b', 'c', 'd']) #列表list，元素都不为空或0

True

>>> all(['a', 'b', '', 'd']) #列表list，存在一个为空的元素

False

>>> all([0, 1,2, 3]) #列表list，存在一个为0的元素

False

>>> all(('a', 'b', 'c', 'd')) #元组tuple，元素都不为空或0

True

>>> all(('a', 'b', '', 'd')) #元组tuple，存在一个为空的元素

False

>>> all((0, 1,2, 3)) #元组tuple，存在一个为0的元素

False

>>> all([]) # 空列表

True

>>> all(()) # 空元组

True

>>> #注意：空元组、空列表返回值为True，这里要特别注意

>>> all(('', '', '', '')) #元组tuple，全部为空的元素

False

>>> all('')

True

>>> #如果all(x)参数x对象的所有元素不为0、''、False或者x为空对象，则返回True，否则返回False

>>>

-------------

要把数组传给函数，要用copy()方法，不然原值会被改变！

axle = [[1, 's', 1], [2, 's', 3], [1, 'e', 4], [0, 's', 6], [2, 'e', 6], [0, 'e', 8]]

axleCopy = axle.copy()  
**for** i **in** range(len(axleCopy)):  
 axleCopy[i] = axleCopy[i].copy()

-------

python逻辑运算符是 and or not。

&，|，^,~,<<,>>是位运算符。