[**Python零碎知识(1):strip lstrip rstrip使用方法**](http://www.cnblogs.com/BeginMan/archive/2013/03/14/2958985.html)

一、原理介绍：

[复制代码](javascript:void(0);)

Python中的strip用于去除字符串的首尾字符，同理，lstrip用于去除左边的字符，rstrip用于去除右边的字符。

这三个函数都可传入一个参数，指定要去除的首尾字符。

需要注意的是，传入的是一个字符数组，编译器去除两端所有相应的字符，直到没有匹配的字符，比如：

theString = 'saaaay yes no yaaaass'

print theString.strip('say')

theString依次被去除首尾在['s'，'a'，'y']数组内的字符，直到字符在不数组内。所以，输出的结果为：

yes no

比较简单吧，lstrip和rstrip原理是一样的。

注意：当没有传入参数时，是默认去除首尾空格的。

theString = 'saaaay yes no yaaaass'

print theString.strip('say')

print theString.strip('say ') #say后面有空格

print theString.lstrip('say')

print theString.rstrip('say')

运行结果：

yes no

es no

yes no yaaaass

saaaay yes no  
注：这段解释来自：（pylemon's notebook）：http://www.cnblogs.com/pylemon/archive/2011/05/18/2050179.html

[复制代码](javascript:void(0);)

 二、实际应用

这里举一个demo，用于在pythonIED中控制输出菜单，选择相应的选项，操作对应的操作。如下：  
假设程序中已经存在newUser和oldUser两个函数，这里我只贴出了主要部分

[复制代码](javascript:void(0);)

1 #菜单控制界面

2 def showmenu():

3 prompt='''

4 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5 Welcome to Python System!

6 -------------------------

7 |(N)ew User Login |

8 |(L)ogin your system |

9 |(Q)uit |

10 -------------------------

11 Enter Choice:

12 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13 '''

14 done=False

15 while not done:

16 chosen=False

17 while not chosen:

18 try:

19 choice=raw\_input(prompt).strip()[0].lower() #取输入的字符串第一个字符

20 except(EOFError,KeyboardInterrupt):

21 choice='q' #抛出异常

22 print '\n You picked:[%s]' %choice

23 if choice not in 'nlq': #判断输出字符串中首字符是否属于'nlq'

24 print 'Invalid option,tyr again'

25 else:

26 chosen=True #一切正常则跳出循环

27 #根据输入信息进行控制

28 if choice=='q':done=True #如果输入'q'则跳出整个循环，不再执行

29 if choice=='n':newUser() #如果输入'n'则调用用户注册函数

30 if choice=='l':oldUser() #如果输入'l'则调用用户登陆函数

31

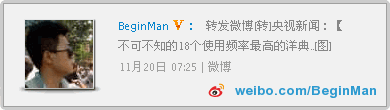
32 #主程序

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

34 showmenu()

[复制代码](javascript:void(0);)

注：请注意红色部分代码

[](http://weibo.com/u/2530590633?s=6uyXnP)

[**Python零碎知识(2):强大的zip**](http://www.cnblogs.com/BeginMan/archive/2013/03/14/2959447.html)

一、代码引导

首先看这一段代码：

[复制代码](javascript:void(0);)

1 >>> name=('jack','beginman','sony','pcky')

2 >>> age=(2001,2003,2005,2000)

3 >>> for a,n in zip(name,age):

4 print a,n

5

6 输出：

7 jack 2001

8 beginman 2003

9 sony 2005

10 pcky 2000

[复制代码](javascript:void(0);)

再看这一段代码：

[复制代码](javascript:void(0);)

1 all={"jack":2001,"beginman":2003,"sony":2005,"pcky":2000}

2 for i in all.keys():

3 print i,all[i]

4

5 输出：

6 sony 2005

7 pcky 2000

8 jack 2001

9 beginman 2003

[复制代码](javascript:void(0);)

发现它们之间的区别么？

最显而易见的是：第一种简洁、灵活、而且能顺序输入。

二、zip()函数

它是Python的内建函数，(与序列有关的内建函数有：sorted()、reversed()、enumerate()、zip()),其中 sorted()和zip()返回一个序列(列表)对象，reversed()、enumerate()返回一个迭代器(类似序列)

[复制代码](javascript:void(0);)

1 >>> type(sorted(s))

2 <type 'list'>

3 >>> type(zip(s))

4 <type 'list'>

5 >>> type(reversed(s))

6 <type 'listreverseiterator'>

7 >>> type(enumerate(s))

8 <type 'enumerate'>

[复制代码](javascript:void(0);)

那么什么是zip()函数 呢？

我们help(zip)看看：

[复制代码](javascript:void(0);)

1 >>> help(zip)

2 Help on built-in function zip in module \_\_builtin\_\_:

3

4 zip(...)

5 zip(seq1 [, seq2 [...]]) -> [(seq1[0], seq2[0] ...), (...)]

6

7 Return a list of tuples, where each tuple contains the i-th element

8 from each of the argument sequences. The returned list is truncated

9 in length to the length of the shortest argument sequence.

[复制代码](javascript:void(0);)

提示：不懂的一定多help

定义：zip([seql, ...])接受一系列可迭代对象作为参数，将对象中对应的元素打包成一个个tuple（元组），然后返回由这些tuples组成的list（列表）。若传入参数的长度不等，则返回list的长度和参数中长度最短的对象相同。

[复制代码](javascript:void(0);)

1 >>> z1=[1,2,3]

2 >>> z2=[4,5,6]

3 >>> result=zip(z1,z2)

4 >>> result

5 [(1, 4), (2, 5), (3, 6)]

6 >>> z3=[4,5,6,7]

7 >>> result=zip(z1,z3)

8 >>> result

9 [(1, 4), (2, 5), (3, 6)]

10 >>>

[复制代码](javascript:void(0);)

zip()配合\*号操作符,可以将已经zip过的列表对象解压

1 >>> zip(\*result)

2 [(1, 2, 3), (4, 5, 6)]

更近一层的了解：  
内容来源：http://www.cnblogs.com/diyunpeng/archive/2011/09/15/2177028.html   (博客园人才真多！)

[复制代码](javascript:void(0);)

\* 二维矩阵变换（矩阵的行列互换）

比如我们有一个由列表描述的二维矩阵

a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

通过python列表推导的方法，我们也能轻易完成这个任务

print [ [row[col] for row in a] for col in range(len(a[0]))]

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

另外一种让人困惑的方法就是利用zip函数：

>>> a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> zip(\*a)

[(1, 4, 7), (2, 5, 8), (3, 6, 9)]

>>> map(list,zip(\*a))

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

zip函数接受任意多个序列作为参数，将所有序列按相同的索引组合成一个元素是各个序列合并成的tuple的新序列，新的序列的长度以参数中最短的序列为准。另外(\*)操作符与zip函数配合可以实现与zip相反的功能，即将合并的序列拆成多个tuple。

①tuple的新序列

>>>>x=[1,2,3],y=['a','b','c']

>>>zip(x,y)

[(1,'a'),(2,'b'),(3,'c')]

②新的序列的长度以参数中最短的序列为准.

>>>>x=[1,2],y=['a','b','c']

>>>zip(x,y)

[(1,'a'),(2,'b')]

③(\*)操作符与zip函数配合可以实现与zip相反的功能,即将合并的序列拆成多个tuple。

>>>>x=[1,2,3],y=['a','b','c']

>>>>zip(\*zip(x,y))

[(1,2,3),('a','b','c')]

[复制代码](javascript:void(0);)

**其他高级应用：**

[复制代码](javascript:void(0);)

1.zip打包解包列表和倍数

>>> a = [1, 2, 3]

>>> b = ['a', 'b', 'c']

>>> z = zip(a, b)

>>> z

[(1, 'a'), (2, 'b'), (3, 'c')]

>>> zip(\*z)

[(1, 2, 3), ('a', 'b', 'c')]

2. 使用zip合并相邻的列表项

>>> a = [1, 2, 3, 4, 5, 6]

>>> zip(\*([iter(a)] \* 2))

[(1, 2), (3, 4), (5, 6)]

>>> group\_adjacent = lambda a, k: zip(\*([iter(a)] \* k))

>>> group\_adjacent(a, 3)

[(1, 2, 3), (4, 5, 6)]

>>> group\_adjacent(a, 2)

[(1, 2), (3, 4), (5, 6)]

>>> group\_adjacent(a, 1)

[(1,), (2,), (3,), (4,), (5,), (6,)]

>>> zip(a[::2], a[1::2])

[(1, 2), (3, 4), (5, 6)]

>>> zip(a[::3], a[1::3], a[2::3])

[(1, 2, 3), (4, 5, 6)]

>>> group\_adjacent = lambda a, k: zip(\*(a[i::k] for i in range(k)))

>>> group\_adjacent(a, 3)

[(1, 2, 3), (4, 5, 6)]

>>> group\_adjacent(a, 2)

[(1, 2), (3, 4), (5, 6)]

>>> group\_adjacent(a, 1)

[(1,), (2,), (3,), (4,), (5,), (6,)]

3.使用zip和iterators生成滑动窗口 (n -grams)

>>> from itertools import islice

>>> def n\_grams(a, n):

... z = (islice(a, i, None) for i in range(n))

... return zip(\*z)

...

>>> a = [1, 2, 3, 4, 5, 6]

>>> n\_grams(a, 3)

[(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]

>>> n\_grams(a, 2)

[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]

>>> n\_grams(a, 4)

[(1, 2, 3, 4), (2, 3, 4, 5), (3, 4, 5, 6)]

4.使用zip反转字典

>>> m = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

>>> m.items()

[('a', 1), ('c', 3), ('b', 2), ('d', 4)]

>>> zip(m.values(), m.keys())

[(1, 'a'), (3, 'c'), (2, 'b'), (4, 'd')]

>>> mi = dict(zip(m.values(), m.keys()))

>>> mi

{1: 'a', 2: 'b', 3: 'c', 4: 'd'}

[复制代码](javascript:void(0);)

# [python零碎知识(4):UserDict 类](http://www.cnblogs.com/BeginMan/archive/2013/03/08/2949321.html)

一、UserDict概述  
UserDict 模块中的 UserDict 类是在python中经常使用的类，保存在Python 安装目录的 lib 目录下UserDict.py。其高仿字典(Dictionary)打开之后可见如下：

[复制代码](javascript:void(0);)

1 ""A more or less complete user-defined wrapper around dictionary objects."""

2

3 class UserDict:　　　　　　　　　　　　　　　　　　　　[1]

4 def \_\_init\_\_(self, dict=None, \*\*kwargs):　　　[2]

5 self.data = {}　　　　　　　　　　　　　　　　[3]

6 if dict is not None:　　　　　　　　　　　　 [4]

7 self.update(dict)　　　　　　　　　　　　[5]

8 if len(kwargs):

9 self.update(kwargs)

10 def \_\_repr\_\_(self): return repr(self.data)

11 def \_\_cmp\_\_(self, dict):

12 if isinstance(dict, UserDict):

13 return cmp(self.data, dict.data)

14 else:

15 return cmp(self.data, dict)

16 def \_\_len\_\_(self): return len(self.data)

..................省略

[复制代码](javascript:void(0);)

 注意：  
[1]. UserDict 是一个基类，不是从任何其他类继承而来  
[2].通过在 dict 参数中传入一个字典来定义初始值  
[3].Python 支持数据属性，如上的 data （在C#、C++叫数据成员，java中叫实例变量）。它是由某个特定的类实例所拥有的数据，在本例中，每个 UserDict 实例将拥有一个 data 数据属性。  
　　引用：(1).要从类外的代码引用这个属性，需要用实例的名字限定它，*instance*.data，限定的方法与你用模块的名字来限定函数一样  
　　　　　(2).要在类的内部引用一个数据属性，我们使用 self 作为限定符

习惯上，所有的数据属性都在 \_\_init\_\_ 方法中初始化为有意义的值。然而，这并不是必须的，因为数据属性，象局部变量一样，当你首次赋给它值的时候[突然产生](mk:@MSITStore:C:%5CDocuments%20and%20Settings%5CAdministrator%5C%E6%A1%8C%E9%9D%A2%5C%E5%B7%A5%E4%BD%9C%E6%96%87%E4%BB%B6%5CPython%E7%A0%94%E7%A9%B6.chm::/native_data_types/declaring_variables.html)。  
[4].if ....is...语法  
[5].update 方法是一个字典复制器：它把一个字典中的键和值全部拷贝到另一个字典。 这个操作 *并不* 事先清空目标字典，如果一些键在目标字典中已经存在，则它们将被覆盖，那些键名在目标字典中不存在的则不改变。应该把 update 看作是合并函数，而不是复制函数。

二、UserDict 常规方法

[复制代码](javascript:void(0);)

1 def clear(self): #[1]

2 self.data.clear() #[2]

3 def copy(self): #[3]

4 if self.\_\_class\_\_ is UserDict: 　　#[4]

5 return UserDict(self.data)

6 import copy

7 return copy.copy(self) 　　#[5]

8 def keys(self):

9 self.data.keys()

10 def items(self):

11 self.data.items()

12 def values(self):

13 self.data.values()

[复制代码](javascript:void(0);)

注意：封装类的基本技术：  
将一个真正的字典 (data) 作为数据属性保存起来，定义所有真正字典所拥有的方法，并且将每个类方法重定向到真正字典上的相应方法。（如上的clear，对应dictionary中的clear()方法）

[复制代码](javascript:void(0);)

1 从 dictionary 中删除元素

2 >>> d

3 {'server': 'mpilgrim', 'uid': 'sa', 'database': 'master',42: 'douglas', 'retrycount': 3}

4 >>> del d[42]

5 >>> d

6 {'server': 'mpilgrim', 'uid': 'sa', 'database': 'master', 'retrycount': 3}

7 >>> d.clear()

8 >>> d

9

10 del 允许您使用 key 从一个 dictionary 中删除独立的元素。

11 clear 从一个 dictionary 中清除所有元素。注意空的大括号集合表示一个没有元素的 dictionary。

[复制代码](javascript:void(0);)

[1].clear 是一个普通的类方法，可以在任何时候被任何人公开调用。注意，clear 象所有的类方法一样（常规的或专用的），使用 self 作为它的第一个参数。（记住，当你调用方法时，不用包括 self；这件事是 Python 替你做的。）

[2].真正字典的 copy 方法会返回一个新的字典，它是原始字典的原样的复制（所有的键-值对都相同）。但是 UserDict 不能简单地重定向到 self.data.copy，因为那个方法返回一个真正的字典，而我们想要的是返回同一个类的一个新的实例，就象是 self。

[3].我们使用 \_\_class\_\_ 属性来查看是否 self 是一个 UserDict，如果是，太好了，因为我们知道如何拷贝一个 UserDict：只要创建一个新的 UserDict ，并传给它真正的字典，这个字典已经存放在 self.data 中了。 然后你立即返回这个新的 UserDict，你甚至于不需要再下面一行中使用 import copy。

[4].如果 self.\_\_class\_\_ 不是 UserDict，那么 self 一定是 UserDict 的某个子类（如可能为 FileInfo），生活总是存在意外。 UserDict 不知道如何生成它的子类的一个原样的拷贝，例如，有可能在子类中定义了其它的数据属性，所以我们只能完全复制它们，确定拷贝了它们的全部内容。幸运的是，Python 带了一个模块可以正确地完成这件事，它叫做 copy。能够拷贝任何 Python 对象.

[5].其余的方法是直截了当的重定向到 self.data 的内置函数上。

三、 直接继承自内建数据类型 dict

注意：dict是内建的数据类型，如同内建函数一般，可以在任何地方使用。在python中，可以直接继承自内建数据类型dict，如下：  
1.继承自UserDict.UserDict

[复制代码](javascript:void(0);)

1 #继承自 UserDict.UserDict

2 #import UserDict 这个错误，一定要导入UserDict模块的UserDict类

3 from UserDict import UserDict

4 class FileInfo(UserDict):

5 "store file metadata"

6 def \_\_init\_\_(self, filename=None):

7 UserDict.\_\_init\_\_(self)

8 self["name"] = filename

[复制代码](javascript:void(0);)

2.直接继承自内建数据类型 dict

1 #直接继承自内建数据类型 dict

2 class demo(dict):

3 def \_\_init\_\_(self,test=None):

4 self["name"]=test

我们发现第二种精简了许多。它们之间的区别如下3点：

（1）不需要导入 UserDict 模块，因为 dict 是已经可以使用的内建数据类型  
（2）继承方式不同，一个是继承自 UserDict.UserDict ，另一个直接继承自 dict  
（3）UserDict 内部的工作方式要求你手工地调用它的 \_\_init\_\_ 方法去正确初始化它的内部数据结构。 dict 并不这样工作，它不是一个封装所以不需要明确的初始化。

四、python类的深入[1]

1.关于重载  
在各编程语言中大致分为两种重载方式:  
　　(1).通过参数列表的重载，一个类可以有同名的多个方法，但这些方法它们参数个数不同，或参数的类型不同，如java  
　　(2).支持通过参数名的重载，一个类可以有同名的多个方法，这些方法有相同类型，相同个数的参数，但参数名不同如（PL/SQL）

不过注意：Python 两种都不支持！  
总之是没有任何形式的函数重载。一个 \_\_init\_\_ 方法就是一个 \_\_init\_\_ 方法，不管它有什么样的参数。每个类只能有一个 \_\_init\_\_ 方法，并且如果一个子类拥有一个 \_\_init\_\_ 方法，它总是覆盖父类的 \_\_init\_\_ 方法，甚至子类可以用不同的参数列表来定义它。子类可以覆盖父类中的方法。

2.关于数据属性  
应该总是在 \_\_init\_\_ 方法中给一个实例的所有数据属性赋予一个初始值。这样做将会节省你在后面调试的时间，不必为捕捉因使用未初始化（也就是不存在）的属性而导致的 AttributeError 异常费时费力。

在 Java 中，静态变量（在 Python 中叫类属性）和实例变量（在 Python 中叫数据属性）两者是紧跟在类定义之后定义的（一个有 static 关键字，一个没有）。在 Python 中，只有类属性可以定义在这里，数据属性定义在 \_\_init\_\_ 方法中。

参考 Dive Into Python

[**python零碎知识(5):有关正则表达式**](http://www.cnblogs.com/BeginMan/archive/2013/03/08/2950313.html)

一、前言

推荐几篇相关博文  
 1.<http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html>  
 2.<http://www.cnblogs.com/deerchao/archive/2006/08/24/zhengzhe30fengzhongjiaocheng.html>  
可以学习参考

二、我对python正则表达式的认识regular expression (RegExp)

[复制代码](javascript:void(0);)

1 ^:匹配字符串的开始.

2 $:匹配字符串的结尾。

3 \b:匹配一个单词的边界

4 \d:匹配任意数字

5 \D:匹配任意非数字字符

6 x?:匹配一个可选的x字符（换句话说，它匹配1次或者0次x 字符）。

7 x\*:匹配0次或者多次x字符

8 x+:匹配1次或者多次x字符。

9 x{n,m}: 匹配x字符，至少n次，至多m次

10 {a|b|c}:要么匹配a，要么匹配b，要么匹配c。

11 (x):一般情况下表示一个记忆组(remembered group).

你可以利用re.search函数返回对象的groups()函数获取它的值。

[复制代码](javascript:void(0);)

三、罗马字母

在罗马数字中，利用7个不同字母进行重复或者组合来表达各式各样的数字。

[复制代码](javascript:void(0);)

1 I = 1

2 V = 5

3 X = 10

4 L = 50

5 C = 100

6 D = 500

7 M = 1000

[复制代码](javascript:void(0);)

构造罗马数字的一些规则：  
1.字符是叠加的。  
I表示1, II表示2, 而III表示3. VI 表示 6 (字面上为逐字符相加, “5 加 1”), VII 表示 7, VIII 表示 8.

2.能够被10整除的字符(I, X, C, 和 M)至多可以重复三次.  
 对于4, 你则需要利用下一个最大的能够被5整除的字符进行减操作得到，你不能把4 表示成 IIII（因为能够被10整除的字符（I,X,C,M）最多能重复三次）;而应表示为 IV (比“5小 1”)。数字40写成XL (比50小10),41 写成 XLI，42 写成 XLII, 43 写成 XLIII；而 44 写成 XLIV (比50 小10, 然后比5小1).  
类似的，对于数字 9,你必须利用下一个能够被10整除的字符进行减操作得到: 8 表示为 VIII, 而 9 则表示为 IX (比10 小1), 而不是 VIIII (因为字符I 不能连续重复四次)。数字90 表示为 XC, 900 表示为 CM.

3.被5整除的字符不能重复（V,L,D)  
数字10 常表示为X, 而从来不用VV来表示。数字100常表示为C, 也从来不表示为 LL.

4.罗马数字经常从高位到低位书写，从左到右阅读，因此不同顺序的字符意义大不相同  
DC 表示 600; 而CD 是一个完全不同的数字(为400, 也就是比500 小100). CI 表示 101; 而IC 甚至不是一个合法的罗马字母（因为你不能直接从数字100减去1; 比需要写成XCIX, 意思是 比100 小10, 然后加上数字9，也就是比 10小1的数字).

四、利用罗马字母来校验

1.目标：校验任意一个字符串是否为一个有效的罗马数字  
2.思想：  
　　(1).一定要熟练正则表达式，才能对于下面的校验熟练掌握  
　　(2).根据罗马字母的匹配规则，由于罗马数字经常是从高位到低位书写，我们从高位开始：千位。对于大于、等于1000的数字，千位有一系列的字符 M 表示。  
3.流程：  
Python通过re模块提供对正则表达式的支持。使用re的一般步骤是先将正则表达式的字符串形式编译为Pattern实例，然后使用Pattern实例处理文本并获得匹配结果（一个Match实例），最后使用Match实例获得信息，进行其他的操作。  
4.用途：很广泛、很强大

还是引用 Dive Into Python上的实例吧，加深下理解。谁叫我还是菜鸟呢！

NO1：校验千位

[复制代码](javascript:void(0);)

1 import re

2 pattern='^M?M?M?$'

3 #语法：re.search(pattern, string, flags)

4 print re.search(pattern, 'M')

5 print re.search(pattern,'MM')

6 print re.search(pattern,'MMM')

7 print re.search(pattern,'MMMM')

8 print re.search(pattern,'')

9 输出如下：

10 <\_sre.SRE\_Match object at 0x00AEDC60>

11 <\_sre.SRE\_Match object at 0x00AEDC60>

12 <\_sre.SRE\_Match object at 0x00AEDC60>

13 None

14 <\_sre.SRE\_Match object at 0x00AEDC60>

[复制代码](javascript:void(0);)

分析：  
1.模式分析：

（1）：^表示仅仅在一个字符串的开始匹配其后的字符串内容。如果没有这个字符，这个模式将匹配出现在字符串任意位置上的 M，而这并不是你想要的。你想确认的是：字符串中是否出现字符M，如果出现，则必须是在字符串的开始。

（2）：M? 可选的匹配单个字符M，由于他重复出现三次，你可以在一行中匹配0次到3次字符M。

（3）：$ 字符限制模式只能够在一个字符串的结尾匹配。当和模式开头的字符^结合使用时，这意味着模式必须匹配整个串，并且在在字符M的前后都不能够出现其他的任意字符。

2.功能分析：

（1）：re 模块的本质是一个search 函数，该函数有两个参数，一个是正则表达式(pattern)，一个是字符串 ('M')，函数试图匹配正则表达式。如果发现一个匹配，search 函数返回一个拥有多种方法可以描述这个匹配的对象，如果没有发现匹配，search 函数返回一个None, 一个Python 空值（null value）。你此刻关注的唯一事情，就是模式是否匹配上，可以利用 search函数的返回值弄清这个事实。字符串'M' 匹配上这个正则表达式，因为第一个可选的M匹配上，而第二个和第三个M 被忽略掉了。

（2）：'MM' 匹配上是因为第一和第二个可选的M匹配上，而忽略掉第三个M。

（3）：'MMM' 匹配上因为三个M 都匹配上了

（4）：'MMMM' 没有匹配上。因为所有的三个M都匹配上，但是正则表达式还有字符串尾部的限制 (由于字符 $), 然而字符串没有结束(因为还有第四个M字符)， 因此 search 函数返回一个None.

（5）：有趣的是，一个空字符串也能够匹配这个正则表达式，因为所有的字符 M 都是可选的。

NO2.检验百位数

……头疼，正在思考中

[**Python零碎知识(6):split 和 join**](http://www.cnblogs.com/BeginMan/archive/2013/03/21/2972857.html)

一、关于split 和 join 方法

1只针对字符串进行处理。split:拆分字符串、join连接字符串  
2.string.join(sep):　　以string作为分割符，将sep中所有的元素(字符串表示)合并成一个新的字符串  
3.string.split(str=' ',num=string.count(str)):　　以str为分隔，符切片string，如果num有指定值，则仅分隔num个子字符串。

4.对导入os模块进行os.path.splie()/os.path.join() 貌似是处理机制不一样，但是功能上一样。

二、split()方法

help后的信息如下：

[复制代码](javascript:void(0);)

split(…)

S.split([sep [,maxsplit]]) -> list of strings

Return a list of the words in the string S, using sep as the

delimiter string. If maxsplit is given, at most maxsplit

splits are done. If sep is not specified or is None, any

whitespace string is a separator.

[复制代码](javascript:void(0);)

中文翻译：

split(…)

S.split([sep [,maxsplit]]) -> 由字符串分割成的列表

返回一组使用分隔符（sep）分割字符串形成的列表。如果指定最大分割数，则在最大分割时结束。如果分隔符未指定或者为none，则分隔符默认为空格。

实例：

[复制代码](javascript:void(0);)

1 s='a b c'

2 print s.split(' ')

3 st='hello world'

4 print st.split('o')

5 print st.split('o',1)

6

7 --------output---------

8 ['a', 'b', 'c']

9 ['hell', ' w', 'rld']

10 ['hell', ' world']

[复制代码](javascript:void(0);)

**注意：分隔符不能为空，否则会出错如下：**

Traceback (most recent call last):

File "<pyshell#12>", line 1, in <module>

s.split('')

ValueError: empty separator

[复制代码](javascript:void(0);)

#但是可以有不含其中的分隔符

>>> s.split('x')

['a b c']

>>> s.split('xsdfadsf')

['a b c']

[复制代码](javascript:void(0);)

**os.path.split()**  
os.path.split是按照路径将文件名和路径分割开，比如d:\\python\\python.ext，可分割为['d:\\python', 'python.exe']，示例如下：

[复制代码](javascript:void(0);)

1 import os

2 print os.path.split('c:\\Program File\\123.doc')

3 print os.path.split('c:\\Program File\\')

4 -----------------output---------------------

5 ('c:\\Program File', '123.doc')

6 ('c:\\Program File', '')

[复制代码](javascript:void(0);)

二、join()

1 a='abcd'

2 print '.'.join(a)

3 print '|'.join(['a','b','c'])　　#可以把['a','b','c']看做是 a='abcd';下面同理

4 print '.'.join({'a':1,'b':2,'c':3,'d':4})

注意：'.'等做分隔符，将join里的所有元素(字符串)通过分隔符连接成一个新的字符串  
可能有人像我一样咬文嚼字，针对string.join()的定义爱钻牛角尖，硬生生地将['a','b','c']先转换为字符串，然后在join  
如：

1 b=str(['a','b','c'])

2 print '|'.join(b)

我以为这样是正解，但是不然。输出结果是：[|'|a|'|,| |'|b|'|,| |'|c|'|]，而导致与上面不一致的原因就是画蛇添足了，把['a','b','c']转换成了字符串，在Python中，我们发现字符串、元祖、列 表它们是序列类型，有着相同的访问方式，可以以下标来访问其中的元素。

以上可以再敲一遍试试。

输入：

1 输出：

2 a.b.c.d

3 a|b|c

4 a.c.b.d

**os.path.join(path1[,path2[,......]])**

[复制代码](javascript:void(0);)

1 os.path.join(path1[, path2[, ...]])

2

3 将多个路径组合后返回，第一个绝对路径之前的参数将被忽略。

4

5 >>> os.path.join('c:\\', 'csv', 'test.csv')

6

7 'c:\\csv\\test.csv'

8

9 >>> os.path.join('windows\temp', 'c:\\', 'csv', 'test.csv')

10

11 'c:\\csv\\test.csv'

12

13 >>> os.path.join('/home/aa','/home/aa/bb','/home/aa/bb/c')

14

15 '/home/aa/bb/c'

[复制代码](javascript:void(0);)

三、参考推荐

1.http://www.cnblogs.com/luhouxiang/archive/2012/05/22/2514081.html  
2.http://docs.python.org/2/library/os.path.html

# [Python零碎知识(7):硬性出错](http://www.cnblogs.com/BeginMan/archive/2013/03/22/2974957.html)

1.eclipse下的错误方式1：

安装好pydev后新建python项目时提示”Project interpreter not specified“的错误,这是因为没有导入python开发环境所致

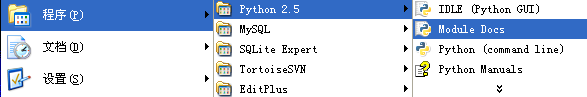
解决方法如下：  
1.找到eclipse---->window----->freference.找到左侧栏边得pydev，展开，点击Interpreter-Python。  
2.在右侧找到New，名字可以不命名，找到python安装路径，我的是c:\python26.然后点击ok，等待导入完成，即可新建python项目

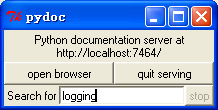
(参考：http://qdjalone.blog.51cto.com/1222376/609857)

2.

[**Python零碎知识(8)：模块的学习|资源利用**](http://www.cnblogs.com/BeginMan/archive/2013/03/27/2984189.html)

一、利用资源：





1.再者可以看看Python/Django官网，查看文档  
2.再者浏览博客园、csdn等相关技术文章。  
不建议在qq群里问，因为很多人都是一蛋疼的人，不讨论技术，却讨论乱七八糟的，而且对于很多人提出的问题置之不理，效率很低。

二、模块学习

**1.python datetime处理时间**

**参考：**  
[(1):http://blog.csdn.net/JGood/archive/2010/04/07/5457284.aspx](http://blog.csdn.net/JGood/archive/2010/04/07/5457284.aspx)  
[(2):http://www.cnblogs.com/lhj588/archive/2012/04/23/2466653.html](http://www.cnblogs.com/lhj588/archive/2012/04/23/2466653.html)

**2.Python标准日志模块logging**

**参考：**

(1):python的logging模块【zz】：

<http://guoxinmiao8.blog.sohu.com/167025590.html>

(2):python模块——logging(日志管理)：

[http://blog.sina.com.cn/s/blog\_4b5039210100f1wv.html](http://www.cnblogs.com/BeginMan/admin/python%E6%A8%A1%E5%9D%97%E2%80%94%E2%80%94logging%28%E6%97%A5%E5%BF%97%E7%AE%A1%E7%90%86%29)

(3):Python标准日志模块logging简介：

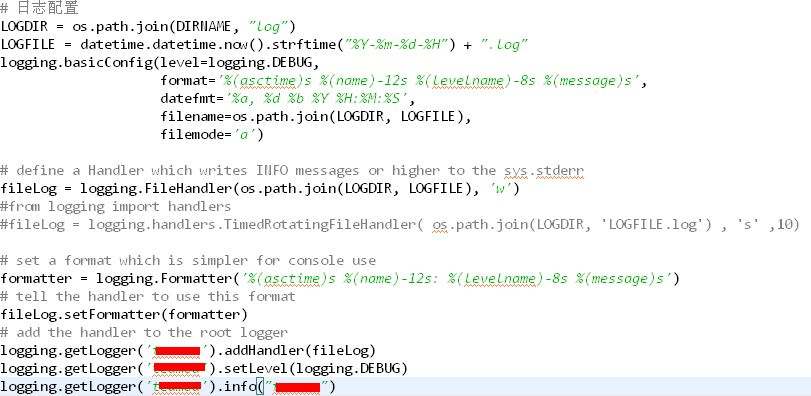
http://www.open-open.com/lib/view/open1348498434510.html

注意上面查看的顺序，要循序渐进地去学习。

在settings.py配置日志如下：

import os, datetime, logging

DIRNAME = os.path.dirname(\_\_file\_\_)



[**django 模板过滤备忘**](http://www.cnblogs.com/lhj588/archive/2012/05/24/2516042.html)

http://www.cnblogs.com/lhj588/archive/2012/05/24/2516042.html

[**Python零碎知识(10):对日期时间的处理**](http://www.cnblogs.com/BeginMan/archive/2013/04/08/3007403.html)

Python提供了time/datetime/calendar等模块来处理日期时间。

**一、time模块常用的函数**

1、 time.time

1970年1月1日以来的秒数，这是一个浮点数

2、 time.sleep

可以通过调用time.sleep来挂起当前的进程。time.sleep接收一个浮点型参数，表示进程挂起的时间。

3、time.clock

在windows操作系统上，time.clock() 返回第一次调用该方法到现在的秒数，其精确度高于1微秒。可以使用该函数来记录程序执行的时间。下面是一个简单的例子：

[复制代码](javascript:void(0);)

1 >>> print time.clock()

2 1.26999365003e-006

3 >>> time.sleep(2)

4 >>> print time.clock()

5 21.4221419643

6 >>> time.sleep(3)

7 >>> print time.clock()

8 44.0470900595

[复制代码](javascript:void(0);)

4、 time.gmtime

该 函数原型为：time.gmtime([sec])，可选的参数sec表示从1970-1-1以来的秒数。其默认值为 time.time()，函数返回time.struct\_time类型的对象。（struct\_time是在time模块中定义的表示时间的对象），下 面是一个简单的例子：

[复制代码](javascript:void(0);)

1 >>> print time.gmtime() #获取当前时间的struct\_time对象

2 (2013, 4, 8, 4, 28, 30, 0, 98, 0)

3 >>> print time.gmtime(time.time())

4 (2013, 4, 8, 4, 28, 50, 0, 98, 0)

5 >>> print time.gmtime(time.time()-24\*60\*60) #获取昨天这个时间的struct\_time对象

6 (2013, 4, 7, 4, 29, 10, 6, 97, 0)

7 >>>

[复制代码](javascript:void(0);)

5、 time.localtime

与time.gmtime非常类似，也返回一个struct\_time对象，可以把它看作是gmtime()的本地化版本。

6、 time.mktime

time.mktime执行与gmtime(), localtime()相反的操作，它接收struct\_time对象作为参数，返回用秒数来表示时间的浮点数。例如:

1 >>> print time.mktime(time.localtime())

2 1365395531.0

7、 time.strftime

time.strftime将日期转换为字符串表示，它的函数原型为：time.strftime(format[, t])。参数format是格式字符串（格式字符串的知识可以参考：[time.strftime](http://docs.python.org/library/time.html#time.strftime) ），可选的参数t是一个struct\_time对象。下面的例子将struct\_time对象转换为字符串表示：

1 >>> print time.strftime('%Y-%m-%d %H:%M:%S',time.gmtime())

2 2013-04-08 04:35:34

[复制代码](javascript:void(0);)

**time.strftime参数:**

strftime(format[, tuple]) -> string

将指定的struct\_time(默认为当前时间)，根据指定的格式化字符串输出

python中时间日期格式化符号：  
  
%y 两位数的年份表示（00-99）

%Y 四位数的年份表示（000-9999）

%m 月份（01-12）

%d 月内中的一天（0-31）

%H 24小时制小时数（0-23）

%I 12小时制小时数（01-12）

%M 分钟数（00=59）

%S 秒（00-59）

%a 本地简化星期名称

%A 本地完整星期名称

%b 本地简化的月份名称

%B 本地完整的月份名称

%c 本地相应的日期表示和时间表示

%j 年内的一天（001-366）

%p 本地A.M.或P.M.的等价符

%U 一年中的星期数（00-53）星期天为星期的开始

%w 星期（0-6），星期天为星期的开始

%W 一年中的星期数（00-53）星期一为星期的开始

%x 本地相应的日期表示

%X 本地相应的时间表示

%Z 当前时区的名称

%% %号本身

[复制代码](javascript:void(0);)

8、 time.strptime

按指定格式解析一个表示时间的字符串，返回struct\_time对象。该函数原型为：time.strptime(string, format)，两个参数都是字符串，下面是一个简单的例子，演示将一个字符串解析为一个struct\_time对象：

1 >>> print time.strptime('2013-04-09 12:30:25','%Y-%m-%d %H:%M:%S')

2 (2013, 4, 9, 12, 30, 25, 1, 99, -1)

更多内容参考Python手册

**二、datetime模块**

time模块它提供 的接口与C标准库time.h基本一致。相比于time模块，datetime模块的接口则更直观、更容易调用。

1、两个常量：  
datetime.MINYEAR和datetime.MAXYEAR，分别表示datetime所能表示的最 小、最大年份。其中，MINYEAR = 1，MAXYEAR = 9999。

2、几个重要的类：

2.1、datetime.date：表示日期的类。常用的属性有year, month, day；

year的返回在两个常量之间；  
month的范围是[1, 12]。（月份是从1开始的，不是从0开始的）；  
day依据month来决定。

date类提供了常用的类方法和类属性：  
date.max、date.min：date对象所能表示的最大、最小日期；  
date.today()：返回一个表示当前本地日期的date对象

[复制代码](javascript:void(0);)

1 >>> from datetime import \*

2 >>> import time

3 >>> print date.today()

4 2013-04-08

5 >>> print date.max

6 9999-12-31

7 >>> print date.min

8 0001-01-01

[复制代码](javascript:void(0);)

date提供的实例方法和属性

* date.year、date.month、date.day：年、月、日；
* date.replace(year, month, day)：生成一个新的日期对象，用参数指定的年，月，日代替原有对象中的属性。（原有对象仍保持不变）
* date.timetuple()：返回日期对应的time.struct\_time对象；
* date.weekday()：返回weekday，如果是星期一，返回0；如果是星期2，返回1，以此类推；
* data.isoweekday()：返回weekday，如果是星期一，返回1；如果是星期2，返回2，以此类推；
* date.isocalendar()：返回格式如(year，month，day)的元组；
* date.isoformat()：返回格式如'YYYY-MM-DD’的字符串；
* date.strftime(fmt)：自定义格式化字符串。在下面详细讲解。

[复制代码](javascript:void(0);)

1 >>> now=date(2013,04,8)

2 >>> tomorrow=now.replace(day=9)

3 >>> print 'now:',now

4 now: 2013-04-08

5 >>> print 'tomorrow:',tomorrow

6 tomorrow: 2013-04-09

7 >>> print 'timetuple():',now.timetuple()

8 timetuple(): (2013, 4, 8, 0, 0, 0, 0, 98, -1)

9 >>> print 'weekday():',now.weekday()

10 weekday(): 0

11 >>> print 'isoweekday():',now.isoweekday()

12 isoweekday(): 1

13 >>> print 'isocalendar():',now.isocalendar()

14 isocalendar(): (2013, 15, 1)

15 >>> print 'isoformat():',now.isoformat()

16 isoformat(): 2013-04-08

[复制代码](javascript:void(0);)

date还对某些操作进行了重载，它允许我们对日期进行如下一些操作：

* date2 = date1 + timedelta  # 日期加上一个间隔，返回一个新的日期对象（timedelta将在下面介绍，表示时间间隔）
* date2 = date1 - timedelta   # 日期隔去间隔，返回一个新的日期对象
* timedelta = date1 - date2   # 两个日期相减，返回一个时间间隔对象
* date1 < date2  # 两个日期进行比较

[复制代码](javascript:void(0);)

1 now = date.today()

2 tomorrow = now.replace(day = 7 )

3 delta = tomorrow - now

4 print 'now:' , now, ' tomorrow:' , tomorrow

5 print 'timedelta:' , delta

6 print now + delta

7 print tomorrow > now

8

9 # # ---- 结果 ----

10 # now: 2010-04-06 tomorrow: 2010-04-07

11 # timedelta: 1 day, 0:00:00

12 # 2010-04-07

13 # True

[复制代码](javascript:void(0);)

参考：http://blog.csdn.net/JGood/article/details/5457284

[**Python零碎知识(11):assert用法**](http://www.cnblogs.com/BeginMan/archive/2013/04/28/3048705.html)

参考：

http://www.cnblogs.com/herbert/archive/2013/01/12/2857233.html

assert语句，如果没记错，这个东西在C或者C++里面也有的。属于短小的断言。下面的是来自python help document的说明：

Assert statements are a convenient way to insert debugging assertions into a program:

assert语句是一种插入调试断点到程序的一种便捷的方式。

assert语句的使用格式

assert expression

这个语句是等价于下面的个句式:

if \_\_debug\_\_:

if not expression: raise AssertionError

assert也可以用于多个表达式的断言

assert expression1, expression2

我自己写的一个关于质数判定的assert使用示例

[复制代码](javascript:void(0);)

复制代码

def isPrime(n):

"""This function return a number is a prime or not"""

assert n >= 2

from math import sqrt

for i in range(2, int(sqrt(n))+1):

if n % i == 0:

return False

return True

复制代码

[复制代码](javascript:void(0);)

assert使用起来还是很方便的，可以避免不必要的未知错误。