#Python 中的字符串处理——2009-12-5

Python 中的清屏指令：

```
import os
os.system('cls')
```

1.字符串的对齐方式:

①:center(int[,str])

```
>>> string = 'Fishhat'
>>> string.center(55)
'                                Fishhat                                '
>>> string.center(55,'*')
'************************Fishhat************************'
```

②:ljust(int[,str])

```
>>> string.ljust(55)
'Fishhat                                                 '
>>> string.ljust(55,'*')
'Fishhat*************************************************'
```

③:rjust(int[,str])

```
>>> string.rjust(55)
'                                                 Fishhat'
>>> string.rjust(55,'*')
'*************************************************Fishhat'
```

④:%(int)s

```
>>> print '% 55s' % string
                                                 Fishhat
```

2.字符串中的转义字符:

\\	斜杠
\'	单引号
\"	双引号
\a	系统喇叭
\b	退格符
\n	换行符
\t	横向制表符
\v	纵向制表符
\r	回车符
\f	换页符
\o	八进制数代表的字符 #还搞不懂...
\x	十六进制数代表的字符 #还搞不懂...

\000 终止符,忽略\000 后的全部字符

3.字符串中的转义符过滤函数:
①:strip()--过滤字符串中所有的转义符
②:lstrip()--过滤字符串中第一个转义符
③:rstrip()--过滤字符串中最后的转义符
后记:在使用的时候返回的结果不是预想中的正确结果.还待查询相关具体资料

4.字符串的连接
①:Python 中可以使用'+'、'+='、连接若干个字符串,如果是以下情况,Python 则自动连接:

```
>>> str = 'fish' 'hat' #若干个字符串之间如果只有空格,Python 会自动连接
>>> print str
fishhat
>>> str += ' learning Python!' #使用运算符追加字符串
>>> print str
fishhat learning Python
```

②:使用 str2.join(str)函数进行连接,其中 str 为需要连接的字符串序列或者一个字符串,str2 为连接中填补的字符:

```
>>> string = ('apple','banana','china')
>>> print '-'.join(string)        #向 string 这个元组中的多个字符串元素之间加上'-'然后输出
apple-banana-china
>>> print ''.join(string)        #加入的字符也可以是空的
applebananachina
>>> print '-'.join('fishhat')     #直接使用
f-i-s-h-h-a-t                     #自动在每个子字符中加入'-'
```

5.用 split(str[,int])函数截取字符串

```
>>> string = 'f-i-s-h-h-a-t'
>>> string.split('-')            #按'-'作为截取子字符串的符号,也就是说遇到一个'-'则截取一个子字符
['f','i','s','h','h','a','t']    #最后返回一个截取得到的子字符列表
>>> string.split('-',2)          #在按'-'截取子字符的时候,当截取到第 2 个'-'的时候就停止截取,后面的字符全部按一个子字符返回(也就是说不管有没有'-'Python 都不会理了,它认为你只指定了 2 个'-'有效)
['f','i','s-h-h-a-t']             #实际上返回了三个子字符,因为在 Python 找到第一个'-'之前已经有一个'f'就把'f'做子字符
```

后记:这里很容易混淆,后面的整数参数(不提供系统默认截取所有的)其实就相当于你要在字符串中画多少条界,相当于切一条蛋糕,你切两刀是不是得三块?至于每块蛋糕的长短,就取决于你在蛋糕上做好的标记,也就相当于刚才的'-'

6.startswith()函数和 endswith()函数
①:startswith(substring[,start[,end]])--用于判断字符串是否以

substring 字符开始的,start 和 end 需要对比的字符区间,默认是整个字符串,成功返回 True 失败返回 False

```
>>> str = 'fishhat'
>>> str.startswith('fi')
True
>>> str.startswith('sh',2,4)
True
>>> str.startswith('sh',)

False
```

②:endswith(substring[,start[,end]])--用于判断字符串是否以 substring 字符结束的,start 和 end 是需要对比的字符区间,默认是整个字符串,成功返回 True 失败返回 False

```
>>> str.endswith('hat')
True
>>> str.endswith('ha',4,6)
True
```

字符串查找替换

Python 截取字符串使用 变量[头下标:尾下标]，就可以截取相应的字符串，其中下标是从 0 开始算起，可以是正数或负数，下标可以为空表示取到头或尾。

```
# 例 1：字符串截取
str = '12345678'
print str[0:1]
>> 1    # 输出 str 位置 0 开始到位置 1 以前的字符
print str[1:6]
>> 23456    # 输出 str 位置 1 开始到位置 6 以前的字符
str=' 0000'
str += str(num) # 合并字符串
print str[-5:]  # 输出字符串右 5 位
>> 00018
```

Python 替换字符串使用 变量.replace("被替换的内容","替换后的内容"[,次数])，替换次数可以为空，即表示替换所有。要注意的是使用 replace 替换字符串后仅为临时变量，需重新赋值才能保存。

```
# 例 2：字符串替换
str = 'akakak'
str = str.replace('k',' 8') # 将字符串里的 k 全部替换为 8
print str
>> 'a8a8a8'  # 输出结果
```

Python 查找字符串使用 变量.find("要查找的内容"[，开始位置，结束位置])，开始位置和结束位置，表示要查找的范围，为空则表示查找所有。查找到后会返回位置，位置从 0 开始算，如果每找到则返回-1。

```
# 例 3：字符串查找
str = 'a,hello'
print str.find('hello') # 在字符串 str 里查找字符串 hello
>> 2   # 输出结果
```

Python 分割字符串使用 变量.split("分割标示符号"[分割次数]),分割次数表示分割最大次数，为空则分割所有。

```
例 4：字符分割
str = 'a,b,c,d'
strlist = str.split(',') # 用逗号分割 str 字符串，并保存到列表
for value in strlist: # 循环输出列表值
    print value
>> a   # 输出结果
>> b
>> c
>> d
```

识别一个字符串

```
>>> import urllib
>>> urlread = lambda url: urllib.urlopen(url).read()
>>> import chardet
>>> chardet.detect(urlread("http://google.cn/"))
{'encoding': 'GB2312', 'confidence': 0.99}

>>> chardet.detect(urlread("http://yahoo.co.jp/"))
{'encoding': 'EUC-JP', 'confidence': 0.99}

>>> chardet.detect(urlread("http://amazon.co.jp/"))
{'encoding': 'SHIFT_JIS', 'confidence': 1}

>>> chardet.detect(urlread("http://pravda.ru/"))
{'encoding': 'windows-1251', 'confidence': 0.9355}

>>> chardet.detect(urlread("http://auction.co.kr/"))
{'encoding': 'EUC-KR', 'confidence': 0.99}

>>> chardet.detect(urlread("http://haaretz.co.il/"))
{'encoding': 'windows-1255', 'confidence': 0.99}

>>> chardet.detect(urlread("http://www.nectec.or.th/tindex.html"))
```

{'encoding': 'TIS-620', 'confidence': 0.7675}

>>> chardet.detect(urlread("http://feedparser.org/docs/"))
{'encoding': 'utf-8', 'confidence': 0.99}

python 字符串处理函数

在 python 有各种各样的 string 操作函数。在历史上 string 类在 python 中经历了一段轮回的历史。在最开始的时候，python 有一个专门的 string 的 module，要使用 string 的方法要先 import，但后来由于众多的 python 使用者的建议，从 python2.0 开始， string 方法改为用 S.method()的形式调用，只要 S 是一个字符串对象就可以这样使用，而不用 import。同时为了保持向后兼容，现在的 python 中仍然保留了一个 string 的 module，其中定义的方法与 S.method()是相同的,这些方法都最后都指向了用 S.method ()调用的函数。要注意,S.method()能调用的方法比 string 的 module 中的多，比如 isdigit()、istitle()等就只能用 S.method()的方式调用。

对一个字符串对象，首先想到的操作可能就是计算它有多少个字符组成，很容易想到用 S.len()，但这是错的，应该是 len(S)。因为 len()是内置函数，包括在__builtin__模块中。python 不把 len()包含在 string 类型中，乍看起来好像有点不可理解，其实一切有其合理的逻辑在里头。len()不仅可以计算字符串中的字符数，还可以计算 list 的成员数，tuple 的成员数等等，因此单单把 len()算在 string 里是不合适，因此一是可以把 len()作为通用函数，用重载实现对不同类型的操作,还有就是可以在每种有 len()运算的类型中都要包含一个 len()函数。 python 选择的是第一种解决办法。类似的还有 str(arg)函数，它把 arg 用 string 类型表示出来。

字符串中字符大小写的变换：

S.lower() #小写
S.upper() #大写
S.swapcase() #大小写互换
S.capitalize() #首字母大写
String.capwords(S)
#这是模块中的方法。它把 S 用 split()函数分开，然后用 capitalize()把首字母变成大写，最后用 join()合并到一起
S.title() #只有首字母大写，其余为小写，模块中没有这个方法

字符串在输出时的对齐：

S.ljust(width,[fillchar])
#输出 width 个字符，S 左对齐，不足部分用 fillchar 填充，默认的为空格。
S.rjust(width,[fillchar]) #右对齐

S.center(width, [fillchar]) #中间对齐
S.zfill(width) #把 S 变成 width 长，并在右对齐，不足部分用 0 补足

字符串中的搜索和替换：

S.find(substr, [start, [end]])
#返回 S 中出现 substr 的第一个字母的标号，如果 S 中没有 substr 则返回-1。start 和 end 作用就相当于在 S[start:end]中搜索
S.index(substr, [start, [end]])
#与 find()相同，只是在 S 中没有 substr 时，会返回一个运行时错误
S.rfind(substr, [start, [end]])
#返回 S 中最后出现的 substr 的第一个字母的标号，如果 S 中没有 substr 则返回-1，也就是说从右边算起的第一次出现的 substr 的首字母标号
S.rindex(substr, [start, [end]])
S.count(substr, [start, [end]]) #计算 substr 在 S 中出现的次数
S.replace(oldstr, newstr, [count])
#把 S 中的 oldstar 替换为 newstr，count 为替换次数。这是替换的通用形式，还有一些函数进行特殊字符的替换
S.strip([chars])
#把 S 中前后 chars 中有的字符全部去掉,可以理解为把 S 前后 chars 替换为 None
S.lstrip([chars])
S.rstrip([chars])
S.expandtabs([tabsize])
#把 S 中的 tab 字符替换没空格，每个 tab 替换为 tabsize 个空格，默认是 8 个
字符串的分割和组合：

S.split([sep, [maxsplit]])
#以 sep 为分隔符，把 S 分成一个 list。maxsplit 表示分割的次数。默认的分割符为空白字符
S.rsplit([sep, [maxsplit]])
S.splitlines([keepends])
#把 S 按照行分割符分为一个 list，keepends 是一个 bool 值，如果为真每行后而会保留行分割符。
S.join(seq) #把 seq 代表的序列——字符串序列，用 S 连接起来

字符串的 mapping，这一功能包含两个函数：

String.maketrans(from, to)
#返回一个 256 个字符组成的翻译表，其中 from 中的字符被一一对应地转换成 to，所以 from 和 to 必须是等长的。
S.translate(table[,deletechars])
# 使用上面的函数产后的翻译表，把 S 进行翻译，并把 deletechars 中有的字符删掉。需要注意的是，如果 S 为 unicode 字符串，那么就不支持 deletechars 参数，可以使用把某个字符翻译为 None 的方式实现相同的功能。此外还可以使

用 codecs 模块的功能来创建更加功能强大的翻译表。
字符串还有一对编码和解码的函数：

S.encode([encoding,[errors]])
# 其中 encoding 可以有多种值，比如 gb2312 gbk gb18030 bz2 zlib big5 bzse64 等都支持。errors 默认值为"strict"，意思是 UnicodeError。可能的值还有 'ignore', 'replace', 'xmlcharrefreplace', 'backslashreplace' 和所有的通过 codecs.register_error 注册的值。这一部分内容涉及 codecs 模块，不是特明白

S.decode([encoding,[errors]])
字符串的测试函数,这一类函数在 string 模块中没有,这些函数返回的都是 bool 值：

S.startwith(prefix[,start[,end]])
#是否以 prefix 开头
S.endwith(suffix[,start[,end]])
#以 suffix 结尾
S.isalnum()
#是否全是字母和数字，并至少有一个字符
S.isalpha() #是否全是字母，并至少有一个字符
S.isdigit() #是否全是数字，并至少有一个字符
S.isspace() #是否全是空白字符，并至少有一个字符
S.islower() #S 中的字母是否全是小写
S.isupper() #S 中的字母是否便是大写
S.istitle() #S 是否是首字母大写的

字符串类型转换函数，这几个函数只在 string 模块中有：

string.atoi(s[,base])
#base 默认为 10，如果为 0,那么 s 就可以是 012 或 0x23 这种形式的字符串，如果是 16 那么 s 就只能是 0x23 或 0X12 这种形式的字符串
string.atol(s[,base]) #转成 long
string.atof(s[,base]) #转成 float

这里再强调一次，字符串对象是不可改变的，也就是说在 python 创建一个字符串后，你不能把这个字符中的某一部分改变。任何上面的函数改变了字符串后，都会返回一个新的字符串，原字串并没有变。其实这也是有变通的办法的，可以用 S=list(S)这个函数把 S 变为由单个字符为成员的 list,这样的话就可以使用 S[3]='a'的方式改变值，然后再使用 S="".join(S)还原成字符串

- python 字符串处理以灵活为最大优点

python 字符串如何进行相关的知识学习，我们在使用的时候有不少的问题阻碍着我们的使用。下面我们就详细的看看如何才能更好的使用相关的 python 字符串，希望对大家有所帮助。

- Python 列表与 C#语言的相似度介绍
- Python 数据类型在常见集合中的语法
- 对 python 开源技术开发的相关了解
- Python 数组实践中具体问题分析
- Python socket 服务如何进行配置详解

看了大家用 C++、C#、Java 等语言的实现，总感觉牛刀杀鸡太麻烦，有兴趣的朋友可以自己写写看或者直接看原文的网友回复。我最近一段时间 Python 写的比较多，读到这些题目时候，就有一种跃跃欲试的冲动。因为我知道用 Perl，Python,Ruby 等动态语言来做这类题目，会是非常理想的。后来我做了这两道题目，结果也令人满意，代码之简洁保持在所有答案的前列。

先看第一题 Python 解答：

```
6.            dic = {}
7.    for s in "abcdefgabc":
8.    dic[s] = 1 if s not in dic else (dic[s]+1)
9.    print '\n'.join('%s,%s' % (k, v) for k, v in dic.items())
10.   输出结果：
11.   a,2
12.   c,2
13.   b,2
14.   e,1
15.   d,1
16.   g,1
17.   f,1
```

Python 的四行代码分别做了 dictionary 的声明，赋值，字符串的遍历，以及高效拼接。

如果还没有看出它的简洁和强大的话，请看第二题的解法：

```
18.           def main(offset=6):
19.   string = u'静夜思 李白床前明月光，疑似地上霜。举头望明月，低头思故乡。090131'
20.   a = [[' ']*offset for row in xrange(offset)]
21.   for i in xrange(offset):
22.   for j in xrange(offset):
23.   a[i][j] = string[j + i*offset]
24.   b = [[r[col] for r in a[::-1]] for col in xrange(len(a[0]))]
25.   print '\n'.join([u'┆'.join(unicode(c) for c in row)for row in b])
```

输出结果：

```
26.           0┆低┆举┆疑┆床┆静 9┆头┆头┆似┆前┆夜 0┆思┆望┆地┆明┆思 1┆故┆明┆上┆月┆  3┆乡┆月┆霜┆光┆李 1┆。┆，┆。┆，┆白
```

这题如果用 C#等实现，代码要在 20 行以上。下面我简单介绍一下这几行代码：

```
27.           第 3 行，在 Python 里面二维"数组"通过嵌套 list 来实现，这里初始化一个 6 行 6 列的二维数组；
28.   第 7 行，我们把"矩阵"顺时针旋转了 90 度（行列置换，并且对置换后的行首尾对调-这里的::-1 就是用来置换一个 list 的 trick）；
```

最后一行，我们把数组里的每行中元素，每行之间分别用两个不同字符拼接起来。join 方法以及 for..in..语句在 python 字符串中是相当常见的用法。

通过这两题，我们看到 Python 在处理字符串时候的十分灵活方便，虽然一种语言的好坏不应完全靠是否简洁来衡量，但对于我个人而言，Python 是目前我用过的最好的语言。而且对于趣味题来说，这不就是我们解题的趣味所在吗？

以上就是对 python 字符串的相关信息介绍。

字符串处理和正则表达式

python 提供了 ord 函数,它取得一个字符做为参数，返回字符的字符代码

字符串方法

capitalize()　返回原始字符串的首字母大写版本,将其他任何大写字母都转换为小写

center(width) 返回宽度为 width 的一个字符串，并让原始字符串在其中居中(两边补空格)

count(substring [,start [,end]]) 返回 substring 在原始字符串中出现的次数.如果指定了 start 参数,就从这个索引位开始搜索,如果还指定了 end 参数，就从 start 开始搜索，在 end 停止

decode([encoding [,errors])

encode([encoding [,errors]) 返回一个编码的字符串. Python 的默认编码方式(encoding)是标准 ASCII. error 参数定义要适用的错误处理类型,默认为"strict"

expandtabs([tabsize]) 返回一个新字符串,其中所有制表符都被替换成空格.可选的 tabsize 参数指定了用于替代一个制表符的空格字符数,默认为 8

endswith(substring [,start [,end]]) 如果以字符串 substring 结束,就返回 1;否则返回 0，如果指定了 start 参数,就从那个索引位置开始搜索,如果还指定了 end 参数,方法就在 start , end 这个"分片"中搜索

find(substring [,start [,end]]) 返回 substring 在字符串中出现的最低索引位置;如果字符串不包括该 substring 就返回-1,start end 用法同上

index(substring [,start [,end]]) 搜索与 find 相同；但假如在字符串中没有发现 substring;就引发一个 ValueError 异常

isalnum() 如果字符串只包括 字母/数字字符，就返回 1；否则返回 0

isalpha() 如果字符串只包含字母,就返回 1；否则返回 0

isdigit() 判断只包含数字

islower() 判断小写

isspace() 如果字符串只包含空白字符,就返回 1;否则返回 0

istitle() 如果字符串中各单词的第一个字符是该单词中惟一的大写字母，就返回 1;否则返回 0

isupper() 判断大写

join(sequence) 返回一个字符串,它链接了 sequence(序列) 中的所有字符串,并将原始字符串做为各个被链接字符串的分隔符适用

ljust(width) 返回一个新字符串,原始字符串在宽度为 width 的一个空白字符串中左对齐

lower() 返回一个新字符串,将原始字符串中的所有字符都转变成小写形式

lstrip() 返回一个新字符串,删除开头的所有空白字符

partition(seq) Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings. New in version 2.5.

```
>>> a = "abcdefg"
>>> a.partition("cd")
('ab', 'cd', 'efg')
```

rpartition(seq) Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself. New in version 2.5.

replace(old, new [, maximum]) 返回一个新字符串,将原始字符串中出现的所有 old 都替换成 new, 可选的 maximum 参数指定最多要执行几次替换;默认全部替换

rfind(substring [,start [,end]]) 返回 substring 在字符串中出现的最高索引位置: 如果字符串不包括该 substring. 就返回-1.start , end 同样用法

rindex(substring [,start [,end]]) 执行方法与 rfind 相同;只是找不到的情况下引发一个 ValueError 异常

rjust(width [,fillchar]) 返回一个新字符串,原始字符串在宽度为 width 中右对齐

rstrip() 返回一个新字符串,删除末尾的所有空白字符

rsplit

split([separator [,maximum]]) 返回一个由于字符串构成的列表,它在每个 separator 处对原始字符串进行分解.如果省略可选的 separator 参数,或者设为 None,会在任何空白字符序列处对原始字符串进行分解 -----这相当于返回一个单词列表. maximum 参数规定最多进行多少次分解

splitline([keepbreaks]) 返回一个由字符串构成的列表,它在每个换行符处分解原始字符串.如果可选 keepbreaks 参数为 1,在返回的列表中,子字符串会保留换行符

startswith(substring [, start [, end]]) 如果字符串以 substring 开头,就返回 1,否则返回 0

strip() 返回一个新字符串,其中删除了原始字符串开头和结尾的所有空白字符

swapcase() 返回一个新字符串,将原始字符串的大写字母换为小写; 小写换为大写

title() 返回一个新字符串,使每个单词的首字母大写,单词中的其他字母小写

translate(table [,delete]) 将原始字符串转换成一个新字符串.首先删除可选参数 delete 中的所有字符,然后将原始字符串的每个字符 c 替换成 table[ord(c)]值

upper() 返回一个新字符串,将原始字符串中的所有字符都转换成大写形式

zfill(width) 在字符串前面填充足够多的 0

正则表达式

re 模块

re.search()

re.match()

re.search 函数会在字符串的任意部分和表达式匹配时返回一个 SRE_Match 对象

与之不同的是, re.match 函数会在字符串的开头与正则表达式匹配的前提下返回 SRE_Match 对象

原始字符串 也就是在字符串之前加上字符前缀 r 后创建的一个字符串. 通常, 如果字符串中出现\, python

会把它视为转义字符,并试图用正确的转义序列替换元字符\及其后续字符. 但是,如果元字符\出现在一个

原始字符串中,Python 不会把它解释成转义字符,而是将其视为字面意义的反斜杠字符.

正则表达式的模式字符串中常常包含反斜杠字符.适用原始字符串来创建模式,可避免对其中每个反斜杠进行转义

以多个分隔符分隔字符串 可以用 re.split

其中第一个参数 就是 指明分隔符

例如 testString = 1+2x*3-y

re.split(r"\+", testString)

注意 一定要在+号前 添加\;

但是当指定多个分隔符时 用 [] 时 里面倒是不用 转义了

例如

re.split(r"[+\-*/\%], testString)

这演示了正则表达式的一个容易被忽视的特性;类中出现的任何字符(用于求反的字符^和用于

表示范围的字符-除外) 都会被解释成字面意义的字符. 所以 $ + * 等元字符在类中无需转义;

但是对于 - 字符如果需要其字面意义 则必须转义.

大多数元字符处于序列中时会失去它们的特殊意义。为了包含一个字面意义 (literal) 的] ，需要将它放在序列的最前。与此相似，为了包含一个字面意义 (literal) 的 ^ ，需要将它放在除了序列最前之外的其他位置。最后，为了包含一个字面意义 (literal) 的 - ，需要将它放在序列最后。

```
>>> a = "hello, world, people"
>>> import re
>>> re.split( r"\w+",a)
['', ', ', ', ', '']
>>> re.split( r"\W+",a)
['hello', 'world', 'people']
>>> a = "hello, world, peo pl,e"
>>> re.split( r"\W+",a)
['hello', 'world', 'peo', 'pl', 'e']
>>> re.split(r" |,", a)
```

['hello', '', 'world', '', 'peo', 'pl', 'e']

>>>

在指定多个分隔符是 会有 "" 这样的项

'\W+' 没事

关键是那个+

>>> re.split(r"[,]+", a)

['hello', 'world', 'peo', 'pl', 'e']

split(*string* [, *maxsplit* = 0])

Split *string* by the matches of the regular expression. **If capturing parentheses are used in the RE, then their contents will also be returned as part of the resulting list**. If *maxsplit* is nonzero, at most *maxsplit* splits are performed

If *replacement* is a string, any backslash escapes in it are processed. That is, "\n" is converted to a single newline character, "\r" is converted to a carriage return, and so forth. Unknown escapes such as "\j" are left alone. Backreferences, such as "\6", are replaced with the substring matched by the corresponding group in the RE. This lets you incorporate portions of the original text in the resulting replacement string.

This example matches the word "section" followed by a string enclosed in "{", "}", and changes "section" to "subsection":

```
>>> p = re.compile('section{ ( [^}]* ) }', re.VERBOSE)
>>> p.sub(r'subsection{\1}','section{First} section{second}')
'subsection{First} subsection{second}'
```

注意替换字符串用 r 否则想引用原字符串括起来的部分; 得使用 "\\1" 的形式

切记!!!!!

比如我想分割

a = f1(`MAX(b, 0.0, 1.0) + 0.5);

我要把这句指令分成

a f1 `MAX b 0.0 1.0 0.5

c++中使用 tokenizer 类可以很轻松的实现

用 python 如何实现?

127366435	firefox_panda	firefox%5Fpand	Python分割字符	0

10	

最佳答案

import re

```
str="a = f1(`MAX(b, 0.0, 1.0) + 0.5);"
print re.split('[ ()+=;]+',str)
```

这样会多出一个"，可以把最后一个字符串去掉：

```
print re.split('[ ()+=;]+',str)[:-1]
```