## Python常用操作代码

（省的以后重复百度）

1. 文件按行读写

**1. 最基本的读文件方法：**

|  |
| --- |
| # File: readline-example-1.py    file = open("sample.txt")    while 1:      line = file.readline()      if not line:          break      pass # do something |

　　一行一行得从文件读数据，显然比较慢；不过很省内存。

　　在我的机器上读10M的sample.txt文件，每秒大约读32000行

**2. 用fileinput模块**

|  |
| --- |
| # File: readline-example-2.py    import fileinput    for line in fileinput.input("sample.txt"):      pass |

　　写法简单一些，不过测试以后发现每秒只能读13000行数据，效率比上一种方法慢了两倍多……

**3. 带缓存的文件读取**

|  |
| --- |
| # File: readline-example-3.py    file = open("sample.txt")    while 1:      lines = file.readlines(100000)      if not lines:          break      for line in lines:          pass # do something |

　　这个方法真的更好吗？事实证明，用同样的数据测试，它每秒可以读96900行数据！效率是第一种方法的3倍，第二种方法的7倍！

————————————————————————————————————————————————————————————

　　在Python 2.2以后，我们可以直接对一个file对象使用for循环读每行数据：

|  |
| --- |
| # File: readline-example-5.py    file = open("sample.txt")    for line in file:      pass # do something |

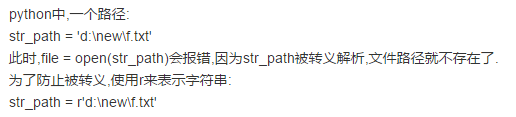
1. 读取文件后对每行的处理（因为当时直接存储的python列表格式）

**for** line **in** fi.readlines():  
 line = line.strip('\n') # 去掉换行符  
 line = line[1:]  
 li = line.replace("]", '').replace("'", '').split(", ")

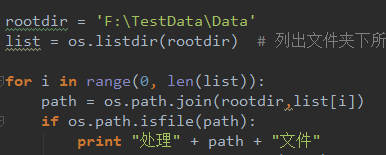
2.写文件：截取读取文件绝对路径的文件数字

**def getFileID**(filename):  
 fileID = filter(str.isdigit, filename)  
 **return** fileID  
  
**def writeFile**(allfea, filename, vocabulary):  
 fileID = getFileID(filename)  
 newpath = "F:\TestData\Feature2\User" + fileID  
 fo = open(newpath, 'w') # r只读，w可写，a追加  
 fo.write(str(vocabulary) + '\n')  
 **for** i **in** range(len(allfea)):  
 fo.write(str(allfea[i]) + '\n')  
 fo.close()  
 **return**

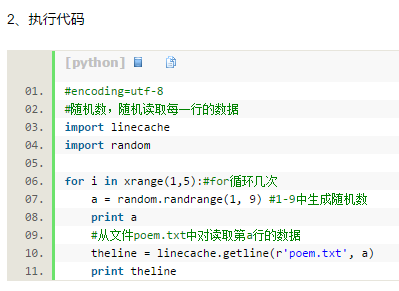
3.路径转义字符去除



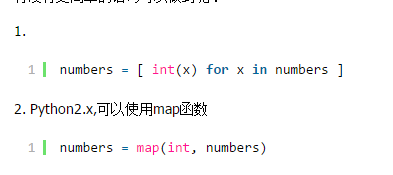
1. 遍历一个目录下的所有文件



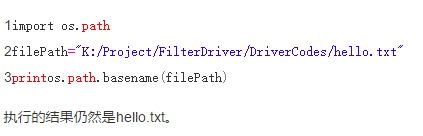
1. [python随机读取文件中的某一行内容](http://blog.csdn.net/baidu_24617085/article/details/49616491)



1. Python中列表元素转为数字

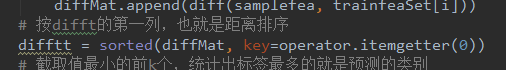


1. [Python截取路径中的文件名](http://blog.csdn.net/willhuo/article/details/49333557)



1. 按照列表的某一列排序

第一种：



第二种



1. 正则表达式匹配

两个基本的操作函数：search()和match()。这两个基本的函数都是匹配正则表达式的字符串，但它们之间去匹配的模式是有一些区别的，re.match是给出的字符串的头部开始匹配，匹配成功就结束了；而re.search是从给出的字符串头部开始，只要任何位置匹配成功都找出来

E1F8400E-0550-AD80-C8DA-BD2BFD4BC9B1

E1F377A8-384D-E97A-69DA-D69A79982A2C