```
#*************** 列表解析 ********************
#列表解析( List comprehensions，或缩略为 list comps ) 来自函数式编程语言 Haskell . 它
#是一个非常有用，简单，而且灵活的工具，可以用来动态地创建列表.
#
#列表解析的语法:
#[expr for iter_var in iterable]
#这个语句的核心是 for 循环，它迭代 iterable 对象的所有条目. 前边的 expr 应用于序列
的每个成员，最后的结果值是该表达式产生的列表.

#print([x ** 2 for x in range(6)])
#-->
#[0, 1, 4, 9, 16, 25]

#
#结合 if 语句，列表解析还提供了一个扩展版本的语法:
#[expr for iter_var in iterable if cond_expr]
#这个语法在迭代时会过滤/捕获满足条件表达式 cond_expr 的序列成员.

#seq = [11, 10, 9, 9, 10, 10, 9, 8, 23, 9, 7, 18, 12, 11, 12]
#print([x for x in seq if x % 2])
#-->
#[11, 9, 9, 9, 23, 9, 7, 11]

#===矩阵样例===
#print([(x+1,y+1) for x in range(3) for y in range(5)])
#-->
#[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5)]



#*************** Part 12 : 生成器表达式 ********************
#生成器表达式是列表解析的一个扩展.生成器是特定的函数，允许你返回一个值，然后"暂
停"代码的执行，稍后恢复.
#
#列表解析的一个不足就是必要生成所有的数据，用以创建整个列表. 这可能对有大量数据
的迭
#代器有负面效应. 生成器表达式通过结合列表解析和生成器解决了这个问题.
```

```
#
#生成器表达式在 Python 2.4 被引入，它与列表解析非常相似，而且它们的基本语法基本相同;
#不过它并不真正创建数字列表，而是返回一个生成器，这个生成器在每次计算出一个条目后，把这
#个条目“产生”(yield)出来. 生成器表达式使用了"延迟计算"(lazy evaluation)，所以它在使用内存上更有效.

#列表解析:
#       [expr for iter_var in iterable if cond_expr]
#生成器表达式:
#       (expr for iter_var in iterable if cond_expr)

#生成器并不会让列表解析废弃，它只是一个内存使用更友好的结构

#=== 交叉配对例子 ===

#rows = [1, 2, 3]
#def cols(): # example of simple generator
#       yield 2
#       yield 1
#
##不需要创建新的列表，直接就可以创建配对. 我们可以使用下面的生成器表达式:
#
#x_product_pairs = ((i, j) for i in rows for j in cols())
#
#for pair in x_product_pairs:
#       print(pair)
#-->
#(1, 2)
#(1, 1)
#(2, 2)
#(2, 1)
#(3, 2)
#(3, 1)



#=== 重构样例 ===
#我们通过一个寻找文件最长的行的例子来看看如何改进代码. 在以前，我们这样读取文件

#f = open('Dave.txt', 'r')
#longest = 0
#allLines = f.readlines()
```

```
#f.close()
#
#for line in allLines:
#       linelen = len(line.strip())
#
#       if linelen > longest:
#               longest = linelen
#
#print(longest)
#-->23

#列表解析允许我们稍微简化我们代码，而且我们可以在得到行的集合前做一定的处理.
#f = open('Dave.txt', 'r')
#longest = 0
#allLines = [x.strip() for x in f.readlines()]
#f.close()
#for line in allLines:
#       linelen = len(line)
#       if linelen > longest:
#               longest = linelen
#print(longest)
#-->23

#用了迭代器，文件本身就成为了它自己的迭代器，不需要调用 readlines() 函数.使用 max() 内建函数得到最长的字符串长度:
#f = open('Dave.txt', 'r')
#longest = max(len(x.strip()) for x in f)
#f.close()
#print(longest)

#去掉文件打开模式(默认为读取)，然后让 Python 去处理打开的文件.
#print(max(len(x.strip()) for x in open('Dave.txt')))
```