```
#encoding=utf-8

##############    文件和输入输出     ######################


#***********     Part 1: 文件对象  ****************
#文件对象不仅可以用来访问普通的磁盘文件, 而且也可以访问任何其它类型抽象层面上的
"文件".

#随着你使用 Python 经验的增长. 您会遇到很多处理"类文件"对象的情况. 有很多这样的
例子,
#例如实时地"打开一个 URL"来读取 Web 页面，在另一个独立的进程中执行一个命令进行
通讯, 就好
#像是两个同时打开的文件, 一个用于读取, 另个用于写入.
#
#内建函数 open() 返回一个文件对象, 对该文件进行后继相关的操作都要用到
#它. 还有大量的函数也会返回文件对象或是类文件( file-like )对象. 进行这
种抽象处理的主要原
#因是许多的输入/输出数据结构更趋向于使用通用的接口. 这样就可以在程序行为和实现
上保持一
#致性. 甚至像 Unix 这样的操作系统把文件作为通信的底层架构接口. 请记住, 文件只是
连续的字
#节序列. 数据的传输经常会用到字节流, 无论字节流是由单个字节还是大块数据组成.

#***********     Part 2: 文件内建函数[open()和 file()]   *****************

#内建函数 open() [以及 file() ]提供了初始化输入/输出(I/O)
#操作的通用接口. open() 内建函数成功打开文件后时候会返回一个文件对象, 否则引发一
个错误.
#当操作失败, Python 会产生一个 IOError 异常,内建函数 open() 的基本语法是：
#       file_object = open(file_name, access_mode='r', buffering=-1)

#file_name 是包含要打开的文件名字的字符串, 它可以是相对路径或者绝对路径. 可选变
量
#access_mode 也是一个字符串, 代表文件打开的模式. 通常, 文件使用模式 'r', 'w', 或是
'a'
#模式来打开, 分别代表读取, 写入和追加. 还有个 'U' 模式, 代表通用换行符支持
```

#
#使用 'r' 或 'U' 模式打开的文件必须是已经存在的. 使用 'w' 模式打开的文件若存在则首
#先清空, 然后(重新)创建. 以 'a' 模式打开的文件是为追加数据作准备的, 所有写入的数据都将
#追加到文件的末尾. 即使你 seek 到了其它的地方. 如果文件不存在, 将被自动创建, 类似以 'w'模式打开文件.
#如果没有给定 access_mode , 它将自动采用默认值 'r' .

#另外一个可选参数 buffering 用于指示访问文件所采用的缓冲方式. 其中 0 表示不缓冲, 1
#表示只缓冲一行数据, 任何其它大于 1 的值代表使用给定值作为缓冲区大小. 不提供该参数或者
#给定负值代表使用系统默认缓冲机制, 既对任何类电报机( tty )设备使用行缓冲, 其它设备使用正
#常缓冲. 一般情况下使用系统默认方式即可.

#文件对象的访问模式
#文件模式 操作
#r 以读方式打开
#rU 或 Ua 以读方式打开, 同时提供通用换行符支持 (PEP 278)
#w 以写方式打开 (必要时清空)
#a 以追加模式打开 (从 EOF 开始, 必要时创建新文件)
#r+ 以读写模式打开
#w+ 以读写模式打开 (参见 w )
#a+ 以读写模式打开 (参见 a )
#rb 以二进制读模式打开
#wb 以二进制写模式打开 (参见 w )
#ab 以二进制追加模式打开 (参见 a )
#rb+ 以二进制读写模式打开 (参见 r+ )
#wb+ 以二进制读写模式打开 (参见 w+ )
#ab+ 以二进制读写模式打开 (参见 a+ )

#这里是一些打开文件的例子:
#fp = open('/etc/motd') # 以读方式打开
#fp = open('test', 'w') # 以写方式打开
#fp = open('data', 'r+') # 以读写方式打开
#fp = open(r'c:\io.sys', 'rb') # 以二进制读模式打开

################ 工厂函数 file()
#open() 和 file() 函数具有相同的功能, 可以任意替换.任何使用 open() 的地方, 都可以使用 file() 替换它.
#一般说来, 建议使用 open() 来读写文件, 在处理文件对象时使用 file() , 例如 if instance(f, file) .

############### 通用换行符支持(UNS)
#Python 2.3 引入了 UNS. 当你使用 'U' 标志打开文件的时候, 所有的行分割符(或行结束符, 无论它原来是什么)通过 Python 的输入方法(例
#如 read*() )返回时都会被替换为换行符 NEWLINE(\n). ('rU' 模式也支持 'rb' 选项) . 这个特性
#还支持包含不同类型行结束符的文件. 文件对象的 newlines 属性会记录它曾“看到的”文件的行结束符.

#如果文件刚被打开, 程序还没有遇到行结束符, 那么文件的 newlines 为 None .在第一行被读
#取后, 它被设置为第一行的结束符. 如果遇到其它类型的行结束符, 文件的 newlines 会成为一个
#包含每种格式的元组. 注意 UNS 只用于读取文本文件. 没有对应的处理文件输出的方法.

#在编译 Python 的时候，UNS 默认是打开的. 如果你不需要这个特性, 在运行 configure 脚本
#时，你可以使用 --without-universal-newlines 开关关闭它. 如果你非要自己处理行结束符, 请
#查阅核心笔记，使用 os 模块的相关属性.

#********** Part 3: 文件内建方法 *****************
#open() 成功执行并返回一个文件对象之后, 所有对该文件的后续操作都将通过这个"句柄"进
#行. 文件方法可以分为四类: 输入, 输出, 文件内移动, 以及杂项操作.

########### 输入
#read() 方法用来直接读取字节到字符串中, 最多读取给定数目个字节. 如果没有给定 size
#参数(默认值为 -1)或者 size 值为负, 文件将被读取直至末尾. 未来的某个版本可能会删除此方法.

#readline() 方法读取打开文件的一行(读取下个行结束符之前的所有字节). 然后整行，包括行
#结束符，作为字符串返回. 和 read() 相同, 它也有一个可选的 size 参数, 默认为 -1, 代表读至
#行结束符. 如果提供了该参数, 那么在超过 size 个字节后会返回不完整的行.

#readlines() 方法并不像其它两个输入方法一样返回一个字符串. 它会读取所有(剩余的)行然
#后把它们作为一个字符串列表返回. 它的可选参数 sizhint 代表返回的最大字节大小. 如果它大
#于 0 , 那么返回的所有行应该大约有 sizhint 字节(可能稍微大于这个数字, 因为需要凑齐

缓冲区大小).

########## 输出
#write() 内建方法功能与 read() 和 readline() 相反. 它把含有文本数据或二进制数据块的
#字符串写入到文件中去.
#
#和 readlines() 一样，writelines() 方法是针对列表的操作，它接受一个字符串列表作为参
#数，将它们写入文件. 行结束符并不会被自动加入，所以如果需要的话，你必须在调用
#writelines()前给每行结尾加上行结束符.

#注意这里并没有 "writeline()" 方法，因为它等价于使用以行结束符结尾的单行字符串调用
write() 方法.

#核心笔记：保留行分隔符
#当使用输入方法如 read() 或者 readlines() 从文件中读取行时, Python 并不会删除行结束
#符. 这个操作被留给了程序员. 例如这样的代码在 Python 程序中很常见:
#f = open('myFile', 'r')
#data = [line.strip() for line in f.readlines()]
#f.close()
#类似地，输出方法 write() 或 writelines() 也不会自动加入行结束符. 你应该在向文件写入
数据前自己完成。

########## 文件内移动
#seek() 方法(类似 C 中的 fseek() 函数)可以在文件中移动文件指针到不同的位置. offset
#字节代表相对于某个位置偏移量. 位置的默认值为 0，代表从文件开头算起(即绝对偏移
量), 1 代表从当前位置算起, 2 代表从文件末尾算起.
#text() 方法是对 seek() 的补充; 它告诉你当前文件指针在文件中的位置 - 从文件起始算
起,单位为字节.

########## 文件迭代
#一行一行访问文件很简单:
#for eachLine in f:
#
#在这个循环里, eachLine 代表文本文件的一行(包括末尾的行结束符)，你可以使用它做任何
想做的事情.

#文件迭代更为高效，而且写（和读）这样的 Python 代码更容易.

########## 其它

#close() 通过关闭文件来结束对它的访问. Python 垃圾收集机制也会在文件对象的引用计数
降

#至零的时候自动关闭文件. 这在文件只有一个引用时发生, 例如 fp = open(...), 然后 fp 在原文
#件显式地关闭前被赋了另一个文件对象. 良好的编程习惯要求在重新赋另个文件对象前关闭这个文
#件. 如果你不显式地关闭文件, 那么你可能丢失输出缓冲区的数据.

#fileno() 方法返回打开文件的描述符. 这是一个整数, 可以用在如 os 模块( os.read() )的
#一些底层操作上.
#调用 flush() 方法会直接把内部缓冲区中的数据立刻写入文件, 而不是被动地等待输出缓冲
#区被写入. isatty() 是一个布尔内建函数, 当文件是一个类 tty 设备时返回 True , 否则返回
#False . truncate() 方法将文件截取到当前文件指针位置或者到给定 size , 以字节为单位.

############ 文件方法杂项

#filename = input('Enter file name: ')
#f = open(filename, 'r')
#allLines = f.readlines()
#f.close()
#for eachLine in allLines:
# print(eachLine)

#与大多数标准的文件访问方法相比, 它的不同在于它读完所有的行才开始向屏幕输出数据.
#很明显如果文件很大, 这个方法并不好. 这时最好还是回到最可靠的方法:
#使用文件迭代器, 每次只读取和显示一行:

#filename = input('Enter file name: ')
#f = open(filename, 'r')
#for eachLine in f:
# print(eachLine)
#f.close()

#核心笔记: 行分隔符和其它文件系统的差异
#操作系统间的差异之一是它们所支持的行分隔符不同. 在 POSIX (Unix 系列或 Mac OS X)系统
#上, 行分隔符是 换行符 NEWLINE ( \n ) 字符. 在旧的 MacOS 下是 RETURN ( \r ) , 而 DOS 和
#Wind32 系统下结合使用了两者 ( \r\n ). 检查一下你所使用的操作系统用什么行分隔符。
#另个不同是路径分隔符(POSIX 使用 "/", DOS 和 Windows 使用 "\", 旧版本的 MacOS 使用
#":"), 它用来分隔文件路径名, 标记当前目录和父目录.
#当我们创建要跨这三个平台的应用的时候, 这些差异会让我们感觉非常麻烦（而且支持的平台

#越多越麻烦）。幸运的是 Python 的 os 模块设计者已经帮我们想到了这些问题.os 模块有五个很
#有用的属性.

#有助于跨平台开发的 os 模块属性
#os 模块属性 描述
#linesep 用于在文件中分隔行的字符串
#sep 用来分隔文件路径名的字符串
#pathsep 用于分隔文件路径的字符串
#curdir 当前工作目录的字符串名称
#pardir (当前工作目录的)父目录字符串名称
#不管你使用的是什么平台，只要你导入了 os 模块，这些变量自动会被设置为正确的值，减少了你的麻烦.

#print 语句默认在输出内容末尾后加一个换行符，而在语句后加一个逗号
#就可以避免这个行为. readline() 和 readlines() 函数不对行里的空白字符做任何处理(参见本章
#练习)，所以你有必要加上逗号．如果你省略逗号，那么显示出的文本每行后会有两个换行符，其
#中一个是输入是附带的，另个是 print 语句自动添加的.

#文件对象还有一个 truncate() 方法，它接受一个可选的 size 作为参数．如果给定，那么文
#件将被截取到最多 size 字节处．如果没有传递 size 参数，那么默认将截取到文件的当前位置.
#例如，你刚打开了一个文件，然后立即调用 truncate() 方法，那么你的文件(内容)实际上被删除，
#这时候你是其实是从 0 字节开始截取的( tell() 将会返回这个数值 ).

#######示例一：输出到文件
#每次从用户接收一行输入，然后将文本保存到文件中．由于 input()不会保留用户输入的换行符， 调用 write() 方法时必须加上换行符。
#而且， 在键盘上很难输入一个 EOF(end-of-file)字符，所以，程序使用句号( . )作为文件结束的标志，当用户输入句号后会自动结束输入并关闭文件.

```
#import os
#filename = input('Enter file name: ')
#fobj = open(filename, 'w')
#
#while True:
#       aLine = input("Enter a line ('.' to quit): ")
#       if aLine != ".":
#              fobj.write('%s%s' % (aLine, os.linesep))
#       else :
```

```
#               break
#fobj.close()
#-->
#Enter file name: test.txt
#Enter a line ('.' to quit): Dave come from anqing
#Enter a line ('.' to quit): Dave is good man!
#Enter a line ('.' to quit): Dave love bl
#Enter a line ('.' to quit): .


#####示例二：
#以可读可写模式创建一个新的文件(可能是清空了一个现有的文件). 在向文件写入
#数据后，我们使用 seek() 方法在文件内部移动，使用 tell() 方法展示我们的移动过程.

#f = open('test.txt', 'w+')
#print(f.tell())

#f.write('test line 1\n') # 加入一个长为 12 的字符串 [0-11]
##print(f.tell())
##13
#f.write('test line 2\n') # 加入一个长为 12 的字符串 [12-23]
##print(f.tell()) # 告诉我们当前的位置
##-->13
##print(f.readline())
##-->test line 1
#
#f.seek(0, 0) # 回到最开始
#print(f.readline())
#
##print(f.tell()) # 又回到了第二行
##13
##print(f.readline())
##-->test line 2
##print(f.tell()) # 又到了结尾
##-->26
#f.close() # 关闭文件

#文件对象的内建方法列表
#文件对象的方法              操作
#file.close()               关闭文件
#file.fileno()              返回文件的描述符(file descriptor ,FD, 整数值)
#file.flush()               刷新文件的内部缓冲区
#file.isatty()              判断 file 是否是一个类 tty 设备
#file.nexta()               返回文件的下一行(类似于 file.readline() ), 或在没有其它行时引发
```

StopIteration 异常
#file.read(size=-1) 从文件读取 size 个字节，当未给定 size 或给定负值的时候，读取剩余的所有字节，然后作为字符串返回
#file.readintob(buf, size) 从文件读取 size 个字节到 buf 缓冲器(已不支持)
#file.readline(size=-1) 从文件中读取并返回一行(包括行结束符)，或返回最大 size 个字符
#file.readlines(sizhint=0) 读取文件的所有行并作为一个列表返回(包含所有的行结束符)；如果给定 sizhint 且大于 0，
# 那么将返回总和大约为 sizhint 字节的行(大小由缓冲器容量的下一个值决定)（比如说缓冲器的大小只能为 4K 的倍数，如果 sizhint 为 15k，则最后返回的可能是 16k———译者按)
#file.xreadlinesc() 用于迭代，可以替换 readlines() 的一个更高效的方法
#file.seek(off, whence=0) 在文件中移动文件指针，从 whence（0 代表文件其始，1 代表当前位置，2 代表文件末尾)偏移 off 字节
#file.tell() 返回当前在文件中的位置
#file.truncate(size=file.tell()) 截取文件到最大 size 字节，默认为当前文件位置
#file.write(str) 向文件写入字符串
#file.writelines(seq) 向文件写入字符串序列 seq；seq 应该是一个返回字符串的可迭代对象；在 2.2 前，它只是字符串的列表

#********** Part 4: 文件内建属性 **********
#文件对象除了方法之外，还有一些数据属性. 这些属性保存了文件对象相关的附加数据，例如
#文件名(file.name )，文件的打开模式（file.mode )，文件是否已被关闭（file.closed)，以及
#一个标志变量，它可以决定使用 print 语句打印下一行前是否要加入一个空白字符( file.softspace ).

#文件对象的属性 描述
#file.closed True 表示文件已经被关闭，否则为 False
#file.encoding 文件所使用的编码 - 当 Unicode 字符串被写入数据时，它们将自动使用 file.encoding 转换为字节字符串;
# 若 file.encoding 为 None 时使用系统默认编码
#file.mode 文件打开时使用的访问模式
#file.name 文件名
#file.newlinesa 未读取到行分隔符时为 None，只有一种行分隔符时为一个字符串，当文件有多种类型的行结束符时，
# 则为一个包含所有当前所遇到的行结束符的列表
#file.softspace 为 0 表示在输出一数据后，要加上一个空格符，1 表示不加。这个属性一般程序员用不着，由程序内部使用。

#********** Part 5: 标准文件 **********

#一般说来，只要你的程序一执行，那么你就可以访问三个标准文件．它们分别是标准输入(一
#般是键盘)，标准输出(到显示器的缓冲输出)和标准错误(到屏幕的非缓冲输出)．(这里所说的"缓冲
#"和"非缓冲"是指 open() 函数的第三个参数.) 这些文件沿用的是 C 语言中的命名，分别为
#stdin，stdout 和 stderr．我们说"只要你的程序一执行就可以访问这三个标准文件"，意思是这
#些文件已经被预先打开了，只要知道它们的文件句柄就可以随时访问这些文件．

#Python 中可以通过 sys 模块来访问这些文件的句柄．导入 sys 模块以后，就可以使用
#sys.stdin，sys.stdout 和 sys.stderr 访问．print 语句通常是输出到 sys.stdout；而内建
#raw_input() 则通常从 sys.stdin 接受输入．
#记得 sys.* 是文件，所以你必须自己处理好换行符．而 print 语句会自动在要输出的字符串
#后加上换行符。 .