第 5 章 Python 文件的使用

在程序运行时，数据保存在内存的变量里。内存中的数据在程序结束或关机后就会消失。如果想要在下次开机运行程序时还使用同样的数据，就需要把数据存储在不易失的存储介质中，例如硬盘、光盘或 U 盘里。不易失的存储介质上的数据保存在以存储路径命名的文件中。通过读/写文件，程序就可以在运行时保存数据。在本章中主要学习使用 Python 在磁盘上创建、读/写及关闭文件。本章只讲述基本的文件操作函数，对于更多函数请参考 Python 标准文档。

视频讲解

5.1 文　　件

简单地说，文件是由字节组成的信息，在逻辑上具有完整的意义，通常在磁盘上永久保存。Windows 系统的数据文件按照编码方式分为两大类，即文本文件和二进制文件。文本文件可以处理各种语言所需的字符，只包含基本文本字符，不包括字体、字号、颜色等信息。文本文件可以在文本编辑器和浏览器中显示，即在任何情况下文本文件都是可读的。

使用其他编码方式的文件即二进制文件，例如 Word 文档、PDF 文件、图像和可执行程序等。如果用文本编辑器打开一个 JPG 文件或 Word 文档，会看到一堆乱码，如图 5-1 所示。也就是说，每种二进制文件都需要自己的处理程序才能打开并操作。

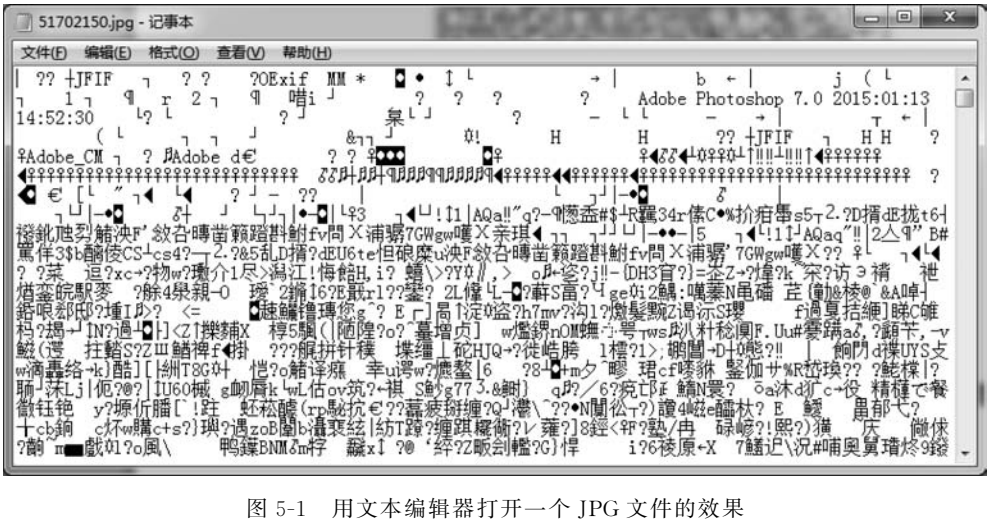

图 5-1　用文本编辑器打开一个 JPG 文件的效果

在本章中重点学习文本文件的操作。当然二进制文件也可以使用 Python 提供的模块进行处理。


视频讲解

5.2 文件的访问

对文件的访问是指对文件进行读/写操作。使用文件跟平时生活中使用记事本很相似。在使用记事本时，需要先打开本子，在使用后要合上它。打开记事本后，既可以读取信息，也可以向本子里写内容。不管哪种情况，都需要知道在哪里进行读/写。在记事本中既可以一页页从头到尾地读，也可以直接跳转到所需要的地方。使用文件也是一样。

在 Python 中对文件的操作通常按照以下 3 个步骤进行。

① 使用 open()函数打开(或建立)文件，返回一个 file 对象。

② 使用 file 对象的读/写方法对文件进行读/写操作。其中，将数据从外存传输到内存的过程称为读操作，将数据从内存传输到外存的过程称为写操作。

③ 使用 file 对象的 close()方法关闭文件。

5.2.1 打开文件

在 Python 中要访问文件，必须打开 Python Shell 与磁盘上文件之间的连接。当使用 open()函数打开或建立文件时，会建立文件和使用它的程序之间的连接，并返回代表连接的文件对象。通过文件对象，就可以在文件所在磁盘和程序之间传递文件内容，执行文件上所有的后续操作。文件对象有时也称为文件描述符或文件流。

在建立了 Python 程序和文件之间的连接后，就创建了“流”数据，如图 5-2 所示。通常程序使用输入流读出数据，使用输出流写入数据，就好像数据流入程序并从程序中流出。打开文件后才能读或写(或读并且写)文件内容。

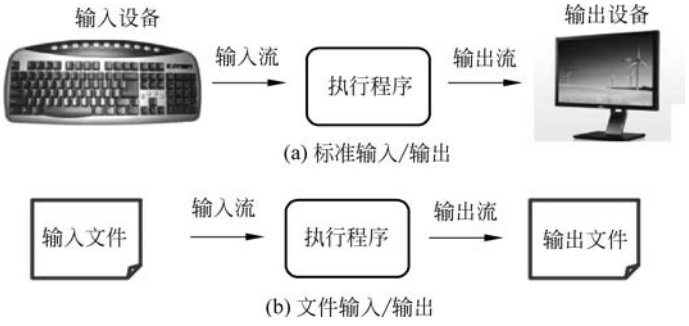

图 5-2 输入/输出流

open()函数用来打开文件。open()函数需要一个字符串路径，表明希望打开文件，并返回一个文件对象。open() 函数的常用形式是接收两个参数——文件名(file)和模式(mode)。代码如下：

```
fileobj = open(file, mode = 'r')
```

完整的语法格式为：

```
fileobj = open(file, mode = 'r', buffering = -1, encoding = None, errors = None, newline = None,
closefd = True, opener = None)
```

其中，fileobj 是 open()函数返回的文件对象；参数 file(文件名)是必写参数，它既可以是绝对路径，也可以是相对路径；mode(模式)是指明文件类型和操作的字符串，可以使用的值如表 5-1 所示。

表 5-1　open()函数中 mode 参数的常用值

值	描　　述
'r'	读模式。如果文件不存在，则发生异常
'w '	写模式。如果文件不存在，则创建文件再打开；如果文件存在，则清空文件内容再打开
'a'	追加模式。如果文件不存在，则创建文件再打开；如果文件存在，则打开文件后将新内容追加至原内容之后
'b'	二进制模式。可添加到其他模式中使用
'+'	读/写模式。可添加到其他模式中使用

说明：

① 当 mode 参数省略时，可以获得能读取文件内容的文件对象，即'r'是 mode 参数的默认值。

② '+'参数指明读和写都是允许的，可以用到其他任何模式中。例如' r+'可以打开一个文本文件并读/写。

③ 'b'参数改变处理文件的方法。通常，Python 处理的是文本文件。当处理二进制文件时(例如声音文件或图像文件)，应该在模式参数中增加'b'。例如可以用'rb'来读取一个二进制文件。

open()函数的第 3 个参数 buffering 控制缓冲。当参数取 0 或 False 时，输入/输出(I/O)是无缓冲的，所有读/写操作直接针对硬盘。当参数取 1 或 True 时，I/O 有缓冲，此时 Python 使用内存代替硬盘，使程序的运行速度更快，只有在使用 flush 或 close 时才会将数据写入硬盘。当参数大于 1 时，表示缓冲区的大小，以字节为单位；负数表示使用默认缓冲区大小。第 4 个参数 encoding 指明文件编码形式，例如 UTF-8。

下面举例说明 open()函数的使用。

先用记事本创建一个文本文件，取名为 hello.txt。输入以下内容并保存到 D:\Python 中：

```
Hello!
Henan  Zhengzhou
```

在交互式环境中输入以下代码：

```
>>> helloFile = open("D:\\Python\\hello.txt")
```

这条命令将以读取文本文件的方式打开放在 D 盘 Python 文件夹下的 hello.txt 文件。读模式是 Python 打开文件的默认模式。当文件以读模式打开时，只能从文件中读取数据，

而不能向文件写入或修改数据。

当调用 open()函数时将返回一个文件对象,在本例中文件对象保存在 helloFile 变量中。

```
>>> print(helloFile)
<_io.TextIOWrapper name='D:\\Python\\hello.txt', mode='r' encoding='cp936'>
```

在打开文件对象时可以看到文件名、读/写模式和编码格式。cp936 就是指 Windows 系统中的第 936 号编码格式,即 GB2312 的编码。接下来就可以调用 helloFile 文件对象的方法读取文件中的数据了。

5.2.2 读取文本文件

可以调用文件对象的多种方法读取文件内容。

1. read()方法

不设置参数的 read()方法将整个文件的内容读取为一个字符串。read()方法一次读取文件的全部内容,性能根据文件的大小而变化,例如 1 GB 的文件在读取时需要使用同样大小的内存。

【例 5-1】 调用 read()方法读取 hello.txt 文件中的内容。

代码如下:

```
helloFile = open("D:\\Python\\hello.txt")
fileContent = helloFile.read()
helloFile.close()
print(fileContent)
```

输出结果如下:

```
Hello!
Henan Zhengzhou
```

另外也可以设置最大读入字符数来限制 read()函数一次返回的大小。

【例 5-2】 设置参数,一次从文件中读取 3 个字符。

代码如下:

```
helloFile = open("D:\\Python\\hello.txt")
fileContent = ""
while True:
    fragment = helloFile.read(3)
    if fragment == "":      #或者 if not fragment
        break
    fileContent += fragment
helloFile.close()
print(fileContent)
```

当读到文件末尾后,read()方法会返回空字符串,此时 fragment==""成立,退出循环。

2. readline()方法

readline()方法从文件中获取一个字符串,这个字符串就是文件中的一行。

【例 5-3】 调用 readline()方法读取 hello.txt 文件的内容。

代码如下:

```
helloFile = open("D:\\Python\\hello.txt")
fileContent = ""
while True:
  line = helloFile.readline()
  if line == "":        #或者 if not line
    break
  fileContent += line
helloFile.close()
print(fileContent)
```

当读取到文件末尾后,readline()方法同样返回空字符串,使得 line==""成立,跳出循环。

3. readlines()方法

readlines()方法返回一个字符串列表,其中的每项是文件中每行的字符串。

【例 5-4】 使用 readlines()方法读取文件的内容。

代码如下:

```
helloFile = open("D:\\Python\\hello.txt")
fileContent = helloFile.readlines()
helloFile.close()
print(fileContent)
for line in fileContent:      #输出列表
   print(line)
```

readlines()方法也可以设置参数,指定一次读取的字符数。

视频讲解

5.2.3 写文本文件

写文件与读文件相似,都需要先创建文件对象连接。所不同的是,在打开文件时是以写模式或添加模式打开。如果文件不存在,则创建该文件。

与读文件时不能添加或修改数据类似,写文件时也不允许读取数据。在用写模式打开已有文件时会覆盖文件的原有内容,从头开始,就像用一个新值覆盖一个变量的值。例如:

```
>>> helloFile = open("D:\\Python\\hello.txt","w")
                                          #用写模式打开已有文件时会覆盖文件的原有内容
>>> fileContent = helloFile.read()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    fileContent = helloFile.read()
IOError: File not open for reading
>>> helloFile.close()
>>> helloFile = open("D:\\Python\\hello.txt")
>>> fileContent = helloFile.read()
```

```
>>> len(fileContent)
0
>>> helloFile.close()
```

由于以写模式打开已有文件，文件的原有内容会被清空，所以再次读取内容时长度为 0。

1. write()方法

write()方法将字符串参数写入文件。

【例 5-5】 用 write()方法写文件。

代码如下：

```
helloFile = open("D:\\Python\\hello.txt","w")
helloFile.write("First line.\nSecond line.\n")
helloFile.close()
helloFile = open("D:\\Python\\hello.txt","a")
helloFile.write("third line. ")
helloFile.close()
helloFile = open("D:\\Python\\hello.txt")
fileContent = helloFile.read()
helloFile.close()
print(fileContent)
```

运行结果如下：

```
First line.
Second line.
third line.
```

当以写模式打开文件 hello.txt 时，文件的原有内容被覆盖。调用 write()方法将字符串参数写入文件，这里“\n”代表换行符。关闭文件之后再次以添加模式打开文件 hello.txt，调用 write()方法写入的字符串"third line."被添加到文件的末尾。最终以读模式打开文件后读取到的内容共有 3 行字符串。

注意：write()方法不能自动在字符串的末尾添加换行符，需要自己添加“\n”。

【例 5-6】 完成一个自定义函数 copy_file()，实现文件的复制功能。

copy_file()函数需要两个参数，指定需要复制的文件 oldfile 和文件的备份 newfile。分别以读模式和写模式打开两个文件，从 oldfile 一次读入 50 个字符并写入 newfile。当读到文件末尾时 fileContent==""成立，退出循环并关闭两个文件。

代码如下：

```
def copy_file(oldfile,newfile):
    oldFile = open(oldfile,"r")
    newFile = open(newfile,"w")
    while True:
        fileContent = oldFile.read(50)
        if fileContent == "":          #读到文件末尾时
            break
        newFile.write(fileContent)
```

```
        oldFile.close()
        newFile.close()
        return
    copy_file("D:\\Python\\hello.txt","D:\\Python\\hello2.txt")
```

2. writelines()方法

writelines(sequence)方法向文件写入一个序列字符串列表,如果需要换行则要自己加入每行的换行符。例如:

```
obj = open("log.py","w")
list02 = ["11","test","hello","44","55"]
obj.writelines(list02)
obj.close()
```

运行结果是生成一个 log.py 文件,内容是"11testhello4455",可见没有换行。另外注意 writelines()方法写入的序列必须是字符串序列,若是整数序列,则会产生错误。

5.2.4 文件内的移动

无论是读或写文件,Python 都会跟踪文件中的读/写位置。在默认情况下,文件的读/写都是从文件的开始位置进行的。Python 提供了控制文件读/写起始位置的方法,使得用户可以改变文件读/写操作发生的位置。

当使用 open()函数打开文件时,open()函数在内存中创建缓冲区,将磁盘上的文件内容复制到缓冲区。文件内容复制到文件对象缓冲区后,文件对象将缓冲区视为一个大的列表,其中的每个元素都有自己的索引,文件对象按字节对缓冲区索引计数。同时,文件对象对文件的当前位置,即当前读/写操作发生的位置进行维护,如图 5-3 所示。许多方法隐式使用当前位置。例如调用 readline()方法后,文件的当前位置移到下一个回车处。

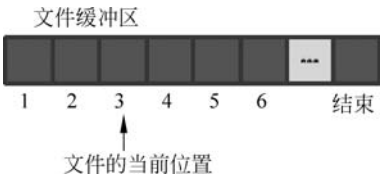

图 5-3 文件的当前位置

Python 使用一些函数跟踪文件的当前位置。tell()函数可以计算文件的当前位置和开始位置之间的字节偏移量。

例如:

```
>>> exampleFile = open("D:\\Python\\example.txt","w")
>>> exampleFile.write("0123456789")
>>> exampleFile.close()
>>> exampleFile = open("D:\\Python\\example.txt")
>>> exampleFile.read(2)
'01'
>>> exampleFile.read(2)
'23'
>>> exampleFile.tell()
4L
>>> exampleFile.close()
```

这里 exampleFile.tell()函数返回的是一个整数 4,表示文件的当前位置和开始位置之间有 4 字节的偏移量。因为已经从文件中读取 4 个字符了,所以有 4 字节的偏移量。

seek()函数设置新的文件当前位置,允许在文件中跳转,实现对文件的随机访问。

seek()函数有两个参数:第一个参数是字节数;第二个参数是引用点。seek()函数将文件的当前指针由引用点移动指定的字节数到指定的位置。语法如下:

```
seek(offset[,whence])
```

说明:offset 是一个字节数,表示偏移量。引用点 whence 有以下 3 个取值。

① 文件开始处为 0,它也是默认取值,意味着使用该文件的开始处作为基准位置,此时字节偏移量必须非负。

② 当前文件位置为 1,则使用当前位置作为基准位置,此时偏移量可以取负值。

③ 文件结尾处为 2,则该文件的末尾将被作为基准位置。

注意:当文件以文本文件方式打开时,只能默认从文件头计算偏移量,即 whence 参数为 1 或 2 时 offset 参数只能取 0,Python 解释器不接受非零偏移量。当文件以二进制方式打开时,可以使用上述参数值进行定位。

【例 5-7】 用 seek()函数在指定位置写文件。

代码如下:

```
exampleFile = open("D:\\Python\\example.txt","w")
exampleFile.write("0123456789")
exampleFile.seek(3)
exampleFile.write("ZUT")
exampleFile.close()
exampleFile = open("D:\\Python\\example.txt")
s = exampleFile.read()
print(s)
exampleFile.close()
```

运行结果如下:

```
'012ZUT6789'
```

注意:在追加模式"a"下打开文件,不能使用 seek()函数进行定位追加。改用"a+"模式打开文件,即可使用 seek()函数进行定位。

5.2.5 文件的关闭

读者应该牢记使用 close()方法关闭文件。关闭文件是取消程序和文件之间连接的过程,内存缓冲区中的所有内容将写入磁盘,因此必须在使用文件后关闭文件以确保信息不会丢失。

要确保文件关闭,可以使用 try/finally 语句,在 finally 子句中调用 close()方法,代码如下:

```
helloFile = open("D:\\Python\\hello.txt","w")
try:
```

```
    helloFile.write("Hello,Sunny Day!")
finally:
    helloFile.close()
```

另外也可以使用 with 语句自动关闭文件,代码如下:

```
with open("D:\\Python\\hello.txt") as helloFile:
    s = helloFile.read()
print(s)
```

with 语句可以打开文件并赋值给文件对象,之后就可以对文件进行操作。文件会在语句结束后自动关闭,即使是由于异常引起的结束也是如此。

5.2.6 二进制文件的读/写

Python 没有二进制类型,但是可以用 string(字符串)类型来存储二进制类型的数据,因为 string 是以字节为单位的。

1. 把数据转换成字节串

pack()方法可以把数据转换成字节串(以字节为单位的字符串)。

格式:pack(格式化字符串,数据)

格式化字符串中可用的格式字符见表 5-2 中的格式字符。例如:

```
import struct
a = 20
bytes = struct.pack('i',a)        #将 a 变为 string 字符串
print(bytes)
```

运行结果如下:

```
b'\x14\x00\x00\x00'
```

此时 bytes 就是一个字符串,字符串按字节与 a 的二进制存储内容相同。在结果中\x 是十六进制的意思,20 的十六进制是 14。

如果字符串是由多个数据构成的,示例代码如下:

```
a = 'hello'
b = 'world!'
c = 2
d = 45.123
bytes = struct.pack('5s6sif',a.encode('utf-8'),b.encode('utf-8'),c,d)
```

'5s6sif'就是格式化字符串,由数字加字符构成。5s 表示占 5 个字符宽度的字符串,2i 表示两个整数等。表 5-2 所示是可用的格式字符及对应 C 语言、Python 中的类型。

表 5-2　可用的格式字符及对应 C 语言、Python 中的类型

格式字符	C 语言中的类型	Python 中的类型	字节数
c	char	string of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer or long	4
l	long	integer	4
L	unsigned long	long	4
q	long long	long	8
Q	unsigned long long	long	8
f	float	float	4
d	double	float	8
s	char[]	string	1
p	char[]	string	1
P	void *	long	与 OS 有关

此时的 bytes 就是二进制形式的数据了，可以直接写入文件。例如：

```
binfile = open("D:\\Python\\hellobin.txt","wb")
binfile.write(bytes)
binfile.close()
```

2. 把字节串(以字节为单位的字符串)还原成数据

unpack()方法可以把相应数据的字节串还原成数据。例如先将 20 转变为字节串：

```
bytes = struct.pack('i',20)          #将 20 变为字节串(以字节为单位的 string 字符串)
```

再进行反操作，将现有的字节串(其实就是二进制数据 bytes)转换成 Python 的数据类型，代码如下：

```
a, = struct.unpack('i',bytes)
```

注意：unpack()返回的是元组。所以如果只有一个变量：

```
bytes = struct.pack('i',a)
```

那么在解码的时候需要：

```
a, = struct.unpack('i',bytes)
```

或者

```
(a,) = struct.unpack('i',bytes)
```

如果直接用 a=struct.unpack('i',bytes),那么 a=(20,)是一个元组而不是原来的整数。

例如,把“D:\\Python\\hellobin.txt”文件中的数据读取并显示,代码如下：

```
import struct
binfile = open("D:\\Python\\hellobin.txt","rb")
bytes = binfile.read()
(a,b,c,d) = struct.unpack('5s6sif',bytes)   #通过 struct.unpack()解码成 Python 变量
t = struct.unpack('5s6sif',bytes)           #通过 struct.unpack()解码成元组
print(t)
```

读取结果如下：

```
(b'hello', b'world!', 2, 45.12300109863281)
```

视频讲解

5.3 文件夹的操作

文件有两个关键属性,即路径和文件名。路径指明了文件在磁盘上的位置。例如,Python 安装在路径 D:\Python35 下,在这个文件夹下可以找到 python.exe 文件,运行该文件可以打开 Python 的交互界面。文件名中圆点后面的部分称为扩展名(或后缀),它指明了文件的类型。

路径中的 D:\称为根文件夹,它包含了本分区内的所有其他文件和文件夹。文件夹可以包含文件和其他子文件夹。Python35 是 D 盘下的一个子文件夹,它包含了 python.exe 文件。

5.3.1 当前工作目录

每个运行在计算机上的程序都有一个当前工作目录。所有没有从根文件夹开始的文件名或路径都假定工作在当前工作目录下。在交互式环境中输入以下代码：

```
>>> import os
>>> os.getcwd()
```

运行结果如下：

```
'D:\\Python35'
```

在 Python 的 GUI 环境中运行时，当前工作目录是 D:\Python35。路径中多出的一个反斜杠是 Python 的转义字符。

5.3.2 目录的操作

在大多数操作系统中，文件被存储在多级目录（文件夹）中。这些文件和目录（文件夹）被称为文件系统。Python 的标准 os 模块可以处理它们。

1. 创建新目录

程序可以用 os.makedirs()函数创建新目录。在交互式环境中输入以下代码：

```
>>> import os
>>> os.makedirs("E:\\python1\\ch5files")
```

os.makedirs()在 E 盘下分别创建了 python1 文件夹及其子文件夹 ch5files，也就是说，路径中所有必需的文件夹都会被创建。

2. 删除目录

当目录不再使用时可以将它删除。使用 os.rmdir()函数删除目录，代码如下：

```
>>> import os
>>> os.rmdir("E:\\python1")
```

这时出现如下错误：

```
WindowsError: [Error 145] : 'E:\\python1'
```

这是因为 os.rmdir()函数在删除文件夹时要保证文件夹内不包含文件及子文件夹。也就是说，os.rmdir()函数只能删除空文件夹。

```
>>> os.rmdir("E:\\python1\\ch5files")
>>> os.rmdir("E:\\python1")
>>> os.path.exists("E:\\python1")       #运行结果为 False
```

Python 的 os.path 模块中包含了许多与文件名及文件路径相关的函数。在上面的例子中使用了 os.path.exists()函数判断文件夹是否存在。os.path 是 os 模块中的模块，所以只要执行 import os 就可以导入它。

3. 列出目录内容

使用 os.listdir()函数可以返回给出路径中文件名及文件夹名的字符串列表，代码如下：

```
>>> os.mkdir("E:\\python1")
>>> os.listdir("E:\\python1")
```

```
[]
>>> os.mkdir("E:\\python1\\ch5files")
>>> os.listdir("E:\\python1")
['ch5files']
>>> dataFile = open("E:\\python1\\data1.txt","w")
>>> for n in range(26):
    dataFile.write(chr(n + 65))
>>> dataFile.close()
>>> os.listdir("E:\\python1")
['ch5files', 'data1.txt']
```

刚创建的 python1 文件夹是一个空文件夹,所以返回的是一个空列表。后续在文件夹下分别创建了一个子文件夹 ch5files 和一个文件 data1.txt,列表中返回的是子文件夹名和文件名。

4. 修改当前目录

使用 os.chdir()函数可以更改当前工作目录:

```
>>> os.chdir("E:\\python1")
>>> os.listdir(".")     #.代表当前工作目录
['ch5files', 'data1.txt']
```

5. 查找匹配文件或文件夹

使用 glob()函数可以查找匹配文件或文件夹(目录)。glob()函数使用 UNIX Shell 的规则来查找。

- *:匹配任意个任意字符。
- ?:匹配单个任意字符。
- [字符列表]:匹配字符列表中的任意一个字符。
- [!字符列表]:匹配除列表外的其他字符。

```
import glob
glob.glob("d*")          #查找以 d 开头的文件或文件夹
glob.glob("d????")       #查找以 d 开头并且全长为 5 个字符的文件或文件夹
glob.glob("[abcd]*")     #查找以 abcd 中的任一字符开头的文件或文件夹
glob.glob("[!abd]*")     #查找不以 abd 中的任一字符开头的文件或文件夹
```

5.3.3 文件的操作

os.path 模块主要用于文件的属性获取,在编程中经常用到。

1. 获取路径和文件名

- os.path.dirname(path):返回 path 参数中的路径名称字符串。
- os.path.basename(path):返回 path 参数中的文件名。
- os.path.split(path):返回参数的路径名称和文件名组成的字符串元组。

```
>>> helloFilePath = "E:\\Python\\ch5files\\hello.txt"
>>> os.path.dirname(helloFilePath)
'E:\\Python\\ch5files'
>>> os.path.basename(helloFilePath)
'hello.txt'
>>> os.path.split(helloFilePath)
('E:\\Python\\ch5files', 'hello.txt')
>>> helloFilePath.split(os.path.sep)
['E:', 'Python', 'ch5files', 'hello.txt']
```

如果想要得到路径中每个文件夹的名字，可以使用字符串方法 split()，通过 os.path.sep 对路径进行正确的分隔。

2. 检查路径的有效性

如果提供的路径不存在，许多 Python 函数就会崩溃报错。os.path 模块提供了一些函数帮助用户判断路径是否存在。

- os.path.exists(path)：判断参数 path 的文件或文件夹是否存在，若存在，则返回 True，否则返回 False。
- os.path.isfile(path)：参数 path 若存在且是一个文件，则返回 True，否则返回 False。
- os.path.isdir(path)：参数 path 若存在且是一个文件夹，则返回 True，否则返回 False。

3. 查看文件的大小

os.path 模块中的 os.path.getsize()函数可以查看文件的大小。此函数与前面介绍的 os.path.listdir()函数配合可以统计文件夹的大小。

【例 5-8】 统计 D:\\Python 文件夹下所有文件的大小。

代码如下：

```
import os
totalSize = 0
os.chdir("D:\\Python")
for fileName in os.listdir(os.getcwd()):
    totalSize += os.path.getsize(fileName)
print(totalSize)
```

4. 重命名文件

os.rename()函数可以帮助用户重命名文件。例如：

```
os.rename("D:\\Python\\hello.txt","D:\\Python\\helloworld.txt")
```

5. 复制文件和文件夹

shutil 模块中提供了一些函数，可以帮助用户复制、移动、改名和删除文件夹，还可以实现文件的备份。

- shutil.copy(source,destination)：复制文件。

➢ shutil.copytree(source,destination)：复制整个文件夹，包括其中的文件及子文件夹。

例如，将 E:\\Python 文件夹复制为新的 E:\\python-backup 文件夹，代码如下：

```
import shutil
shutil.copytree("E:\\Python","E:\\python-backup")
for fileName in os.listdir("E:\\python-backup"):
    print(fileName)
```

在使用这些函数前先导入 shutil 模块。shutil.copytree()函数复制包括子文件夹在内的所有文件夹内容。例如：

```
shutil.copy("E:\\python1\\data1.txt","E:\\python-backup")
shutil.copy("E:\\python1\\data1.txt","E:\\python-backup\\data-backup.txt")
```

shutil.copy()函数的第二个参数 destination 可以是文件夹，表示将文件复制到新文件夹里；也可以是包含新文件名的路径，表示在复制的同时将文件重命名。

6. 文件和文件夹的移动和改名

shutil.move(source,destination)与 shutil.copy()函数的用法相似，参数 destination 既可以是一个包含新文件名的路径，也可以仅包含文件夹。例如：

```
shutil.move("E:\\python1\\data1.txt","E:\\python1\\ch5files")
shutil.move("E:\\python1\\data1.txt","E:\\python1\\ch5files\\data2.txt")
```

需要注意的是，不管是 shutil.copy()函数还是 shutil.move()函数，函数参数中的路径必须存在，否则 Python 会报错。

如果参数 destination 中指定的新文件名与文件夹中已有的文件重名，则文件夹中已有的文件会被覆盖。因此，在使用 shutil.move()函数时应当小心。

7. 删除文件和文件夹

os 模块和 shutil 模块都有函数可以删除文件或文件夹。

➢ os.remove(path)/os.unlink(path)：删除参数 path 指定的文件。例如：

```
os.remove("E:\\python-backup\\data-backup.txt")
os.path.exists("E:\\python-backup\\data-backup.txt")     #False
```

➢ os.rmdir(path)：如前所述，os.rmdir()函数只能删除空文件夹。

➢ shutil.rmtree(path)：删除整个文件夹，包含所有文件及子文件夹。

```
shutil.rmtree("E:\\python1")
os.path.exists("E:\\python1")     #False
```

这些函数都是从硬盘中彻底删除文件或文件夹，不可恢复，因此在使用时应特别谨慎。

8. 遍历目录树

想要处理文件夹中包括子文件夹内的所有文件(即遍历目录树),可以使用 os.walk()函数。os.walk()函数将返回该路径下的所有文件及子目录信息元组。

【例 5-9】 显示 H:\\档案科技表格文件夹下的所有文件及子目录。

代码如下:

```
import os
list_dirs = os.walk("H:\\档案科技表格")        #返回一个元组
print(list(list_dirs))
for folderName,subFolders,fileNames in list_dirs:
    print("当前目录: " + folderName)
    for subFolder in subFolders:
        print(folderName +"的子目录" + " 是 --" + subFolder)
        for fileName in fileNames:
            print(subFolder +"的文件 " + " 是 --" + fileName)
```

5.4 常用格式文件的操作

视频讲解

5.4.1 操作 CSV 格式文件

CSV(Comma-Separated Values)文件以纯文本形式存储表格数据。CSV 文件由任意数量的记录组成,记录间以换行符分隔;每条记录由字段组成,字段间常见的分隔符是逗号或制表符。例如:

```
序号,姓名,年龄
3,张海峰,25
4,李伟,38
5,赵大强,36
……
```

而 Excel 是电子表格,包含文本、数值、公式和格式。当不需要公式和格式时表格可用 CSV 格式保存,当需要时则需保存为 Excel 格式。

1. 直接读/写 CSV 文件

CSV 文件是一种特殊的文本文件,且格式简单,可以根据文本文件的读/写方法操作 CSV 文件。

【例 5-10】 直接读取存储人员信息的 CSV 格式文件(test2.csv)到二维列表。

代码如下:

```
myfile = open('test2.csv', 'r')
ls = []
for line in myfile:
    line = line.replace('\n','')        #将换行符去掉
    ls.append(line.split(','))          #将一行中以','分隔的多个数据转换成数据元素的列表
```

```
print(ls)
myfile.close()                    # close 文件
```


以上代码将一行数据转换成一个列表,多行数据就组成了一个每个元素都是列表的列表,即二维列表。

运行结果类似如下:

```
[['序号', '姓名', '年龄'], ['3', '张海峰', '25'], ['4', '李伟', '38'], ['5', '赵大强', '36']]
```

【例 5-11】 直接将二维列表写入 CSV 格式文件。

代码如下:

```
myfile = open('test2new.csv', 'w')          #新建 CSV 文件并以写模式打开
ls = [['序号', '姓名', '年龄'], ['3', '张海峰', '25'], ['4', '李伟', '38'],
    ['5', '赵大强', '36'], ['6', '程海鹏', '28']]
for line in ls:
    myfile.write(','.join(line) + "\n")    #在同一行的数据元素之间加上逗号分隔
myfile.close()
```

在写入过程中,与读取数据时的处理相反,需要借助字符串的 join()方法,在同一行的数据元素之间加上逗号分隔。同时行尾加上换行符"\n"。

2. 使用 csv 模块读/写 CSV 文件

Python 自带的 csv 模块可以处理 CSV 文件,与读/写 Excel 文件相比,CSV 文件的读/写是相当方便的。

读取 CSV 文件使用 reader 对象。格式如下:

```
reader(csvfile[, dialect = 'excel'][, fmtparam])
```

- csvfile:通常的文件(file)对象,或者列表(list)对象都是适用的。
- dialect:编码风格,默认为 excel 方式,也就是以逗号(,)分隔。另外 csv 模块也支持 excel-tab 风格,也就是以制表符(tab)分隔。
- fmtparam:格式化参数,用来覆盖之前 dialect 对象指定的编码风格。

reader 对象是可以迭代的,line_num 参数表示当前行数。reader 对象还提供了一些 dialect、next()方法。

写入 CSV 文件使用 writer 对象。格式如下:

```
writer(csvfile, dialect = 'excel', fmtparams)
```

其参数的意义同上,这里不再赘述。这个对象有 writerow()和 writerows()两个函数实现 CSV 文件的写入。例如:

```
with open('1.csv','w',newline = '') as f:
  head = ['标题列 1','标题列 2']
  rows = [ ['张三',80],['李四',90] ]
  writer = csv.writer(f)
  writer.writerow(head)        #写入一行数据
  writer.writerows(rows)       #写入多行数据
```

【例 5-12】 将人员信息写入 CSV 文件并读取出来。

代码如下：

```
import csv
#写入一个文件
myfile = open('test2.csv', 'w', newline = '')                    #'a'追加,'w'写方式
mywriter = csv.writer(myfile)                                    #返回一个 writer 对象
mywriter.writerow(['序号', '姓名', '年龄'])                        #加入标题行
mywriter.writerow([3, '张海峰', 25])                              #加入一行
mywriter.writerow([4, '李伟', 38])
mywriter.writerows([[5, '赵大强', 36],[6, '程海鹏', 28]])        #加入多行
myfile.close()
#读取一个 CSV 文件
myfilepath = 'test2.csv'
#这里用到的 open 都要加上 newline = '',否则会多一个换行符(参见标准库文档)
myfile = open(myfilepath, 'r', newline = '')
myreader = csv.reader(myfile)                                    #返回一个 reader 对象
for row in myreader:
    if myreader.line_num == 1:                                   #line_num 是从 1 开始计数的
        continue                                                 #第一行表头不输出
    for i in row:                                                #row 是一个列表
        print(i, end = ' ')
    print()
myfile.close()                                                   #close 文件
```

这个程序涉及两个函数，writer.writerow(list)是将 list 列表以一行形式添加，writer.writerows(list)可写入多行。

程序的运行结果如下：

```
3 张海峰 25
4 李伟 38
5 赵大强 36
6 程海鹏 28
```

并生成与显示同样内容的 test2.csv 文件，不过还有序号、姓名、年龄这样的标题行信息。

5.4.2 操作 Excel 文档

Excel 是电子表格，包含文本、数值、公式和格式。第三方的 xlrd 和 xlwt 两个模块分别用来读和写 Excel，只支持.xls 和.xlsx 格式，Python 默认不包含这两个模块。这两个模块

之间相互独立,没有依赖关系,也就是说可以根据需要只安装其中一个。xlrd 和 xlwt 模块的安装可以在命令行下使用 pip install 模块名:

```
pip install xlrd
pip install xlwt
```

当看到类似 Successfully 的字样时表明已经安装成功了。

1. 使用 xlrd 模块读取 Excel

xlrd 提供的接口比较多,常用的如下:

open_workbook()打开指定的 Excel 文件,返回一个 Book 工作簿对象。

```
data = xlrd.open_workbook('excelFile.xls')  #打开 Excel 文件
```

1) Book 工作簿对象

通过 Book 工作簿对象可以得到各个 Sheet 工作表对象(一个 Excel 文件可以有多个 Sheet,每个 Sheet 就是一张表格)。Book 工作簿对象的属性和方法如下。

- Book.nsheets():返回 Sheet 的数目。
- Book.sheets():返回所有 Sheet 对象的 list。
- Book.sheet_by_index(index):返回指定索引处的 Sheet,相当于 Book.sheets()[index]。
- Book.sheet_names():返回所有 Sheet 对象名字的 list。
- Book.sheet_by_name(name):根据指定 Sheet 对象名字返回 Sheet。

例如:

```
table = data.sheets()[0]                    #通过索引顺序获取 Sheet
    table = data.sheet_by_index(0)          #通过索引顺序获取 Sheet
    table = data.sheet_by_name('Sheet1')    #通过名称获取 Sheet
```

2) Sheet 工作表对象

通过 Sheet 对象可以获取各个单元格,每个单元格是一个 Cell 对象。Sheet 对象的属性和方法如下:

- Sheet.name():返回表格的名称。
- Sheet.nrows():返回表格的行数。
- Sheet.ncols():返回表格的列数。
- Sheet.row(r):获取指定行,返回 Cell 对象的 list。
- Sheet.row_values(r):获取指定行的值,返回 list。
- Sheet.col(c):获取指定列,返回 Cell 对象的 list。
- Sheet.col_values(c):获取指定列的值,返回 list。
- Sheet.cell(r, c):根据位置获取 Cell 对象。
- Sheet.cell_value(r, c):根据位置获取 Cell 对象的值。

例如:

```
cell_A1 = table.cell(0,0).value     #获取 A1 单元格的值
cell_C4 = table.cell(2,3).value     #获取 C4 单元格的值
```

以下代码循环输出表数据：

```
nrows = table.nrows     #表格的行数
ncols = table.ncols     #表格的列数
for i in range(nrows):
    print(table.row_values(i) )
```

3）Cell 对象

Cell 对象的 Cell.value 返回单元格的值。

【例 5-13】 读取 Excel 文件 test.xls。

代码如下：

```
import xlrd
wb = xlrd.open_workbook('test.xls')          #打开文件
sheetNames = wb.sheet_names()                #查看包含的工作表
print(sheetNames)                            #输出所有工作表的名称,['sheet_test']
#获得工作表的两种方法
sh = wb.sheet_by_index(0)
sh = wb.sheet_by_name('sheet_test')          #通过名称'sheet_test'获取对应的 Sheet
#单元格的值
cellA1 = sh.cell(0,0)
cellA1Value = cellA1.value
print(cellA1Value)                           #王海
#第一列的值
columnValueList = sh.col_values(0)
print(columnValueList)                       #['王海', '程海鹏']
```

程序的运行结果如下：

```
['sheet_test']
王海
['王海', '程海鹏']
```

2. 使用 xlwt 模块写 Excel

相对来说，xlwt 模块提供的接口没有 xlrd 模块那么多，主要如下。

- Workbook()：构造函数，返回一个工作簿对象。
- Workbook.add_sheet(name)：添加一个名为 name 的表，类型为 Worksheet。
- Workbook.get_sheet(index)：可以根据索引返回 Worksheet。
- Worksheet.write(r, c, value)：将 value 填充到指定位置。
- Worksheet.row(n)：返回指定的行。
- Row.write(c, value)：在某一行的指定列写入 value。
- Worksheet.col(n)：返回指定的列。

- Row.height 或 Column.width：对它们赋值可以改变行或列的默认高度或宽度(单位为 0.05 pt,即 1/20 pt)。
- Workbook.save(filename)：保存文件。

表的单元格默认是不可重复写的,如果有需要,在调用 add_sheet()的时候指定参数 cell_overwrite_ok=True 即可。

【例 5-14】 写入 Excel 示例。

代码如下：

```
import xlwt
book = xlwt.Workbook(encoding = 'utf - 8')
sheet = book.add_sheet('sheet_test', cell_overwrite_ok = True)      #单元格可重复写
sheet.write(0, 0, '王海')
sheet.row(0).write(1, '男')
sheet.write(0, 2, 23)
sheet.write(1, 0, '程海鹏')
sheet.row(1).write(1, '男')
sheet.write(1, 2, 41)
sheet.col(2).width = 4000                                           #单位为 1/20 pt
book.save('test.xls')
```

运行程序生成如图 5-4 所示的 test.xls 文件。

	A	B	C
1	王海	男	23
2	程海鹏	男	41
3			
4			

图 5-4　test.xls 文件

视频讲解

5.5　文件应用案例——游戏地图的存储

在游戏开发中往往需要存储不同关卡的游戏(例如推箱子、连连看等游戏)的地图信息。这里以推箱子游戏的地图存储为例来说明游戏地图信息如何存储到文件中并读取出来。

对于图 5-5 所示的推箱子游戏,可以看成 7×7 的表格,这样如果按行存储到文件中,就可以把这一关游戏的地图存到文件中了。

墙	墙	墙			墙	墙
		墙			墙	墙
						墙
		墙				墙
			墙			墙
			墙	墙		墙
	墙	墙	墙	墙	墙	墙

图 5-5　推箱子游戏

为了表示方便，每个格子的状态值分别用常量 Wall(0)代表墙，Worker(1)代表人，Box(2)代表箱子，Passageway(3)代表路，Destination(4)代表目的地，WorkerInDest(5)代表人在目的地，RedBox(6)代表放到目的地的箱子。文件中存储的原始地图中格子的状态值采用相应的整数形式存放。假如推箱子游戏界面对应的数据如下。

0	0	0	3	3	0	0
3	3	0	3	4	0	0
1	3	3	2	3	3	0
4	2	0	3	3	3	0
3	3	3	0	3	3	0
3	3	3	0	0	3	0
3	0	0	0	0	0	0

5.5.1 将地图写入文件

地图只需要使用 write()方法按行/列(这里按行)存到文件 map1.txt 中即可。例如：

```
import os
#将地图写入文件
helloFile = open("map1.txt","w")
helloFile.write("0,0,0,3,3,0,0\n")
helloFile.write("3,3,0,3,4,0,0\n")
helloFile.write("1,3,3,2,3,3,0\n")
helloFile.write("4,2,0,3,3,3,0\n")
helloFile.write("3,3,3,0,3,3,0\n")
helloFile.write("3,3,3,0,0,3,0\n")
helloFile.write("3,0,0,0,0,0,0\n")
helloFile.close()
```

5.5.2 从地图文件读取信息

只需要按行从文件 map1.txt 中读取即可得到地图信息。在本例中将信息读取到二维列表中存储。

代码如下：

```
#读文件
helloFile = open("map1.txt","r")
myArray1 = []
while True:
    line = helloFile.readline()
    if line == "":                          #或者用 if not line
        break
    line = line.replace("\n","")            #将读取的一行中最后的换行符去掉
    myArray1.append(line.split(","))
helloFile.close()
print(myArray1)
```

运行结果如下：

```
[['0', '0', '0', '3', '3', '0', '0'], ['3', '3', '0', '3', '4', '0', '0'], ['1', '3', '3', '2', '3', '3',
'0'], ['4', '2', '0', '3', '3', '3', '0'], ['3', '3', '3', '0', '3', '3', '0'], ['3', '3', '3', '0','0',
'3', '0'],['3','0','0','0',',0',',0',',0']]
```

在 7.6.4 节的图形化推箱子游戏中，根据数字代号将对应图形显示到界面上，即可完成地图的读取任务。

5.6 文件应用案例——词频统计

对文章内容进行统计，从中找出出现频率高的词语，从而概要分析文章内容，是人们经常遇到的需求；对网络信息进行自动检索及归档，也是同样的需求。这就是“词频统计”问题。

以英文文章为例，将文章作为文件读取其内容，对文章中的每个单词设计计数器，每出现一次其计数加 1，最后得出每个单词出现的次数。这里可以使用字典类型，以单词作为键，以次数作为值，形成(单词，次数)的键值对。英文文章以空格或标点符号进行单词的分隔，因此获得单词并统计数量相对容易。下面对程序进行分析。

词频统计问题的 IPO(Input，输入；Processing，处理；Output，输出)描述如下。

(1) 程序输入：选取李某给刚上大学的女儿的一封英文信并保存在 letter.txt 文件中，从 letter.txt 文件中读取文章。

(2) 处理：使用字典类型，分词并统计每个单词出现的次数。

(3) 程序输出：显示统计的结果，每个单词及其出现的次数。

使用字典类型 wdCountDict 进行单词的计数。对于已经出现在字典中的单词，其计数加 1；对于没有出现的单词，添加并将键值设置为 1，新建键值对。对应的代码如下：

```
for word in words:
    if word in wdCountDict:
        wdCountDict[word] = wdCountDict[word] + 1
    else
        wdCountDict[word] = 1
```

另外也可以使用 wdCountDict.get()方法将上述代码中的 if 语句替换为：

```
wdCountDict[word] = wdCountDict.get(word,0) + 1
```

此时将词频统计结果从大到小倒序排序并输出。字典类型是无序的，因此必须先将字典转换为列表，对列表进行排序。为了使用 sort()方法排序，在转换列表时需要将每个字典项(键，值)转换为新元组(值，键)添加到列表中。这是因为 sort()方法对于复合对象比较的是每个元素的第一个值。代码如下：

```
valKeyList = []
for key,val in wdCountDict.items():
    valKeyList.append((val,key))
```

另外也可以使用以下代码进行更简洁的替换：

```
valKeyList = [(val,key) for key,val in wdCountDict.items()]
```

全部代码如下：

```
#letter.py
def getFileText():
    with open("C:\lynn\Python\letter.txt","r") as letterFile:
        filTxt = letterFile.read()
    filTxt = filTxt.lower()
    for ch in '!"#$%&()*+-*/,.:;<=>?@[]\^_{}|~':
        filTxt = filTxt.replace(ch," ")
    return filTxt
letterTxt = getFileText()
words = letterTxt.split()
wdCountDict = {}
for word in words:
    wdCountDict[word] = wdCountDict.get(word,0) + 1
valKeyList = [(val,key) for key,val in wdCountDict.items()]
valKeyList.sort(reverse = True)
print("{0:<10}{1:>5}".format("word","count"))
print("*" * 21)
for val,key in valKeyList:
    print("{0:<10}{1:>5}".format(key,val))
```

注意：在使用 sort()方法排序时，若第一个值相同，它会使用复合对象的其他元素排序。因此，出现次数相同的单词以单词字母的倒序输出。

观察结果可以发现，在列表中会出现很多常见且对文章分析无意义的词，例如 and、you、or、it 等，这些词被称作停用词。停用词列表可以在网上找到，通常可以设置停用词列表，并将它们从字典中排除。代码如下：

```
excludes = {"the","of","you","your","that","will","this","don't"}
for word in excludes:
    del(wdCountDict[word])
```

在这个示例中列表中的单词并不完整，读者可以试着完善列表。

更简单的方法是排除长度小于 3 的单词，并在最终结果中将出现次数少于两次的单词也排除，完善输出结果。完整代码如下：

```
#letter.py
def getFileText():
    with open("C:\lynn\Python\letter.txt","r") as letterFile:
        filTxt = letterFile.read()
    filTxt = filTxt.lower()
    for ch in '!"# $ %&()* + - */,.:;< = >?@[]\^_{}|~':
        filTxt = filTxt.replace(ch," ")
    return filTxt
letterTxt = getFileText()
words = letterTxt.split()      #实现文章分隔成单词的列表 words
wdCountDict = {}
excludes = {"the","of","you","your","that","will","this","don't"}
for word in words:
    wdCountDict[word] = wdCountDict.get(word,0) + 1
for word in excludes:
    del(wdCountDict[word])
items = list(wdCountDict.items())                    #将字典转换为列表
items.sort(key = lambda x:x[1],reverse = True)       #按记录的第 2 列排序
print("{0:<10}{1:>5}".format("word","count"))
print("*"*21)
for key,val in items:
    if len(key)>3 and val>2:
        print("{0:<10}{1:>5}".format(key,val))
```

此后,在最终结果中就可以看到单词出现的次数。

5.7 习　　题

1. 编写程序,打开任意的文本文件,读出其中的内容,判断该文件中某些给定的关键字(例如“中国”)出现的次数。

2. 编写程序,打开任意的文本文件,在指定的位置产生一个相同文件的副本,即实现文件的复制功能。

3. 用 Windows 的记事本创建一个文本文件,其中每行包含一段英文。试读出文件的全部内容,并判断:

(1) 该文本文件共有多少行?

(2) 文件中以大写字母 P 开头的有多少行?

(3) 一行中包含字符最多的和包含字符最少的分别在第几行?

4. 统计 test.txt 文件中大写字母、小写字母和数字出现的次数。

5. 编写程序,统计调查问卷中各评语出现的次数,并将最终统计结果放入字典。

调查问卷结果:

不满意,一般,满意,一般,很满意,满意,一般,一般,不满意,满意,满意,满意,满意,一般,很满意,一般,满意,不满意,一般,不满意,满意,满意,满意,满意,满意,满意,很满意,不满意,满意,不满意,不满意,一般,很满意

要求:问卷调查结果用文本文件 result.txt 保存,编写程序读取该文件后统计各评语出

现的次数，将字典的最终统计结果追加到 result.txt 文件中。

6. src.txt 文件中存储的是一篇英文文章，将其中所有的大写字母转换成小写字母输出。

假如 src.txt 里面存储的内容为“This is a Book”，则输出内容应为“this is a book”。

7. score.txt 文件中存储了歌手大奖赛中 10 名评委给每个歌手打的分，10 个分数在一行，形式如下：

歌手 1，8.92，7.89，8.23，8.93，7.89，8.52，7.99，8.83，8.99，8.89

歌手 2，8.95，8.86，8.24，8.63，7.66，8.53，8.59，8.82，8.93，8.89

……

从文件中读取数据，存入列表中，计算该名歌手的最终得分，最终得分的计算方式是 10 个评分去掉最高分，去掉最低分，然后求平均分。最终得分保留两位小数，输出到屏幕上。