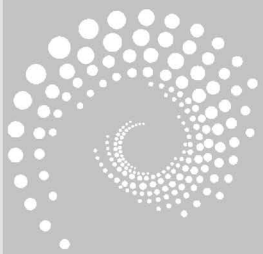
第5章

Python文件的使用

CHAPTER 5



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，如硬盘、光盘或U盘。不易失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，要学习使用Python在磁盘上创建、读写以及关闭文件。本章只讲述基本的文件操作函数，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: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,不包括诸如字体、字号、颜色等信息。它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,如 Word 文档、PDF、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。

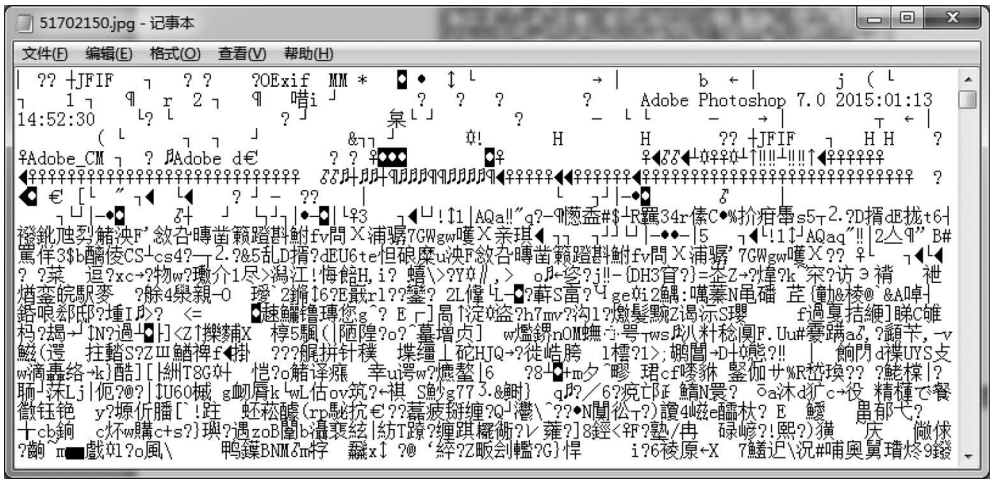


图 5-1 文本编辑器 Notepad 打开 JPG 文件的显示效果

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。使用文件与人们平时生活中使用记事本很相似。使用记事本时,需要先打开本子,使用后要合上它。打开记事本后,既可以读取信息,也可以向本子里写入信息。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,也可以直接跳转到需要的地方。

在 Python 中,对文件的操作通常按照以下三个步骤进行。

(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/写的操作。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(建立)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,执行文件上所有后续操作。文件对象有时也称为文件描述符或文件流。

当建立了 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。

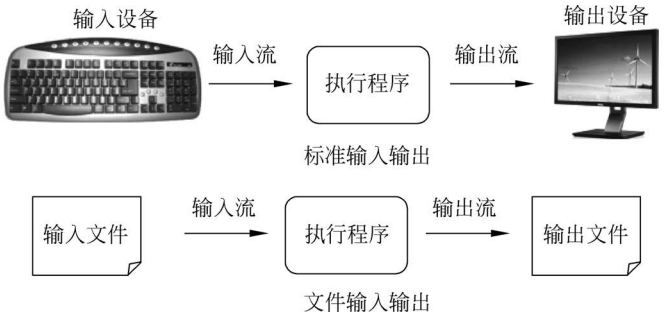


图 5-2　输入输出流

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。语法如下。

```
fileobj = open(filename[,mode[,buffering]])
```

其中,fileobj 是 open()函数返回的文件对象;参数 filename 文件名是必选参数,它既可以是绝对路径,也可以是相对路径;mode(模式)和 buffering(缓冲)是可选参数。

mode 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(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 函数的第三个参数 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,使程序运行速度更快,只有使用 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。

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在文件夹 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 D 盘 Python 文件夹下的 hello 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'>
```

打印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 系统里第 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 file 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,性能根据文件大小而变化,如 1GB 的文件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 文件中的内容。

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

输出结果：

```
Hello!
Henan  Zhengzhou
```

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】 设置参数一次读取三个字符读取文件。

```
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)
```

当读到文件结尾之后，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，每个字符串就是文件中的一行。

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

```
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)
```

当读取到文件结尾之后，readline()方法同样返回空字符串，使得 line＝＝""成立退出循环。

3．readlines()方法

readlines()方法返回一个字符串列表，其中的每一项是文件中每一行的字符串。

【例 5-4】 使用 readlines()方法读取文件内容。

```
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。

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,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。“w”写模式打开已有文件时,会覆盖文件原有内容,从头开始,就像用一个新值覆写一个变量的值。

例如:

```
>>> helloFile = open("d:\\python\\hello.txt","w")   #"w"写模式打开已有文件时会覆盖文件原有内容
>>> 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“w”写模式打开已有文件,文件原有内容会被清空,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。

```
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:

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

以写模式打开文件 hello.txt 时,文件原有内容被覆盖。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.”被添加到了文件末尾。最终以读模式打开文件后读取到的内容共有三行字符串。

注意，write()方法不能自动在字符串末尾添加换行符，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。

```
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。

```
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，整数序列会产生错误。

5.2.4　文件内移动

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/写都从文件的开始位置进行。Python 提供了控制文件读写起始位置的方法，使得我们可以改变文件读/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，其中的每一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，文件当前位置移动到下一个回车处。

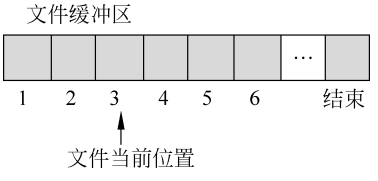


图 5-3　文件当前位置

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

```
>>> 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()
4
>>> exampleFile.close()
```

这里 exampleFile.tell()函数返回的是一个整数 4,表示文件当前位置和开始位置之间有 4B 偏移量。因为已经从文件中读取 4 个字符了,所以有 4B 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 有以下三个取值。

- 文件开始处为 0,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。
- 当前文件位置为 1,则是使用当前位置作为基准位置。此时偏移量可以取负值。
- 文件结尾处为 2,则该文件的末尾将被作为基准位置。

【例 5-7】 用 seek()函数在指定位置写文件。

```
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()
```

运行结果是:

```
'012ZUT6789'
```

注意,在追加模式下打开文件,不能使用 seek()函数进行定位追加。

5.2.5 文件的关闭

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,内存缓冲区的所有内容将写入磁盘,因此必须在使用文件后关闭文件确保信息不会丢失。

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。

```
helloFile = open("d:\\python\\hello.txt","w")
try :
    helloFile.write("Hello,Sunny Day!")
finally:
    helloFile.close()
```

也可以使用 with 语句自动关闭文件。

```
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。


视频讲解

5.3　文件夹的操作

文件有两个关键属性：路径和文件名。路径指明了文件在磁盘上的位置。例如，编者的 Python 安装在路径 D:\Python35，在这个文件夹下可以找到 python.exe 文件，运行可以打开 Python 的交互界面。文件名句点的后面部分称为扩展名（或后缀），它指明了文件的类型。

路径中的 D:\称为“根文件夹”，它包含本分区内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 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()
```

运行结果为

```
'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 rmdir()函数删除目录:

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

这时出现错误:

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

因为 rmdir()函数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 模块包含许多与文件名及文件路径相关的函数。上面的例子里使用了 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。

```
>>> 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']
```

在刚创建 python1 文件夹时,这是个空文件夹,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,列表里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/文件夹
glob.glob("[abcd]*")                      #查找以 abcd 中任意字符开头的文件或文件夹
glob.glob("[!abd]*")                      #查找不以 abd 中任意字符开头的文件或文件夹
```

5.3.3　文件操作

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一个文件夹的名字,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我们判断路径是否存在。

- os.path.exists(path)：判断参数 path 的文件或文件夹是否存在。存在返回 true,否

则返回 false。

- os.path.isfile(path)：判断参数 path 是否存在且是一个文件，是则返回 true，否则返回 false。
- os.path.isdir(path)：判断参数 path 是否存在且是一个文件夹，是则返回 true，否则返回 false。

3. 查看文件大小

os.path 模块中的 os.path.getsize()函数可以查看文件大小。此函数与前面介绍的 os.path.listdir()函数配合可以帮助统计文件夹大小。

【例 5-8】 统计 d:\\python 文件夹下所有文件的大小。

```
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()函数可以帮助重命名文件。

```
os.rename("d:\\python\\hello.txt","d:\\python\\helloworld.txt")
```

5. 复制文件和文件夹

shutil 模块中提供一些函数，帮助复制、移动、改名和删除文件夹，也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：

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

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。

```
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)：shutil.move()函数与 shutil.copy()函数用法相似，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。

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

但要注意的是，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文件重名，则文件夹中的已有文件会被覆盖。因此使用 shutil.move()函数应当小心。

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。

```
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)：shutil.rmtree()函数删除整个文件夹，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所有文件及子目录信息元组。

【例 5-9】 显示“d:\档案科技表格”文件夹下所有文件及子目录。

```
import os
list_dirs = os.walk("d:\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　常用格式文件操作

5.4.1　操作 CSV 格式文件

CSV(Comma-Separated Values,逗号分隔值)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,字段间的分隔符常见的是逗号或制表符。例如,CSV 格式文件 test2.CSV:

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

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实现操作 CSV 文件。

【例 5-10】 直接读取 CSV 格式文件到二维列表。

```
myfile = open('test2.csv', 'r')
ls = []
for line in myfile:
    line = line.replace('\n','')      #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
```

以上代码将一行转换成一个列表,多行数据就组成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。

```
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间隔
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间隔。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，也就是逗号(,)分隔。另外，csv 模块也支持 excel-tab 风格，也就是制表符(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 属性表示当前行数；reader 对象还提供一些 dialect、next()方法。

写入 CSV 文件使用 writer 对象，格式如下。

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

参数的意义同上，不再赘述。这个对象有两个函数 writerow()和 writerows()实现写入 CSV 文件。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。

```
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
```

这个程序涉及下面的函数：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 不默认包含这两个模块。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]                    #通过索引获取 Sheet
table = data.sheet_by_index(0)              #通过索引获取 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
```

例如，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】 读取如图 5-4 所示的 Excel 文件 test.xls 示例。

```
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 提供的接口就没有 xlrd 那么多了，主要有以下几个。

Workbook()是构造函数，返回一个工作簿的对象。

Workbook.add_sheet(name)添加了一个名为 name 的表，类型为 Worksheet。

Workbook.get_sheet(index)可以根据索引返回 Worksheet。

Worksheet.write(r, c, vlaue)是将 vlaue 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。

通过对 Row.height 或 Column.width 赋值可以改变行或列默认的高度或宽度（单位：0.05pt，即 1/20pt）。

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。

```
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 1/20 pt
book.save('test.xls')
```

程序运行生成如图 5-4 所示 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

5.4.3 操作 JSON 格式文件

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式,比 XML 更小、更快、更易解析,易于读写且占用带宽小,网络传输速度快,适用于数据量大、不要求保留原有类型的情况。它是 JavaScript 的子集,易于人阅读和编写。

前端和后端进行数据交互,其实往往就是通过 JSON 进行的。因为 JSON 易于被识别的特性,常被作为网络请求的返回数据格式。在爬取动态网页时,会经常遇到 JSON 格式的数据,Python 中可以使用 json 模块来对 JSON 数据进行解析。

1. JSON 的结构

常见形式为“名称/值”对的集合。

例如:

```
{"firstName": "Brett", "lastName": "McLaughlin"}
```

JSON 允许使用数组,采用方括号[]实现。

例如,用 JSON 表示中国部分省市数据如下,其中省份采用的是数组。

```
{
    "name": "中国",
    "province": [{
        "name": "黑龙江",
        "cities": {
            "city": ["哈尔滨", "大庆"]
```

```
        }
    }, {
        "name": "广东",
        "cities": {
            "city": ["广州", "深圳", "珠海"]
        }
    } ]
}
```

2. json 模块中常用的方法

在使用 json 这个模块前，首先要导入 json 库：import json。

它主要提供了 4 个方法 dumps、dump、loads、load，如表 5-2 所示。

表 5-2　json 模块中常用的方法

方　　法	功 能 描 述
json.dumps()	将 Python 对象转换成 JSON 字符串
json.loads()	将 JSON 字符串转换成 Python 对象
json.dump()	将 Python 类型数据序列化为 JSON 对象后写入文件
json.load()	读取文件中 JSON 形式的字符串并转换为 Python 类型数据

下面通过例子说明这 4 个方法的使用。

1）json.dumps()

其作用是将 Python 对象转换成 JSON 字符串。

```
import json
data = {'name':'nanbei','age':18}
s = json.dumps(data)                          #将 Python 对象编码成 JSON 字符串
print(s)
```

运行结果：

```
{"name": "nanbei", "age": 18}
```

JSON 注意事项：

- 名称必须用双引号(即"name")来包括。
- 值可以是字符串、数字、true、false、null、数组或子对象。

从运行结果可见，原先的'name','age'单引号已经变成双引号"name", "age"。

2）json.loads()

其作用是将 JSON 字符串转换成 Python 对象。

```
import json
data = "{'name':'nanbei','age':18}"
a = json.dumps(data)
print(json.loads(a))                          #将 JSON 字符串编码成 Python 对象——dict 字典
```

运行结果：

```
{'name': 'nanbei', 'age': 18}
```

如果是一个 JSON 文件则要先读文件,然后才能转换成 Python 对象。

```
import json
f = open('stus.json',encoding = 'utf - 8')     # 'stus.json'是一个 JSON 文件
content = f.read()                             # 使用 loads()方法,需要先读文件成字符串
user_dic = json.loads(content)                 # 转换成 Python 的字典对象
print(user_dic)
```

3) json.load()方法

该方法的作用是读取文件中 JSON 形式的字符串并转换为 Python 类型数据。

```
import json
f = open('stus.json',encoding = 'utf - 8')
user_dic = json.load(f)                        # f 是文件对象
print(user_dic)
```

可见,loads()传入的是字符串,而 load()传入的是文件对象。使用 loads()时需要先读文件成字符串再使用,而 load()则不用先读文件成字符串而是直接传入文件对象。

4) json.dump()

该方法的作用是将 Python 类型数据序列转换为 JSON 对象后写入文件。

```
stus = { 'xiaojun':88,  'xiaohei':90,  'lrx':100}
f = open('stus2.json','w',encoding = 'utf - 8')     # 以写方式打开 stus2.json 文件
json.dump(stus,f)                                   # 写入 stus2.json 文件
f.close()                                           # 文件关闭
```

实验五　文件操作

一、实验目的

通过本实验,掌握 Python 语言文件的读写方法以及打开和关闭等基本操作,理解并应用文件操作相关知识。

二、实验要求

(1) 掌握文件的打开(或建立)和关闭方法。

(2) 掌握文件的读写方法。

(3) 掌握文件和文件夹(目录)的操作方法。

三、实验内容与步骤

(1) 编程实现文件读写操作,完成以下功能。

① 随机生成 100 个 100～500 的整数,每 10 个整数占一行,写入 test.txt 文件。

```
import random
f = open("test.txt", 'w')
for i in range(1,101):
    if i % 10 == 0:
```

```
            f.write(str(random.randrange(100,500)) + '\n')
        else:
            f.write(str(random.randrange(100,500)) + ' ')
    f.close()
```

② 读取 test.txt 文件，显示每行数据并计算出每行最大值。

```
f2 = open("test.txt", 'r')
s = f2.readlines()                          #读取文件
f2.close()
for m in s:
    m = m.strip()
    ls = m.split(' ')
    print(ls)
    print(max(int(c) for c in ls))
```

③ 统计偶数的个数并写入 test.txt 文件中。

```
f3 = open("test.txt", 'r + ')
s = f3.readlines()                          #读取文件
n = 0
for m in s:
    m = m.strip()
    ls = m.split(' ')
    ls2 = [int(c) for c in ls]
    for x in ls2:
        if x % 2 == 0:
            n = n + 1
f3.write("偶数的个数是: " + str(n))
f3.close()
```

(2) 假设有一个英文文本文件 demo.txt，要求完成如下操作。

① 读取 demo.txt 文件内容并显示。

② 编写程序对读取内容进行加密，并把加密后内容保存到另一个文件 cipher.txt 中。按照以下加密规则对其加密后输出其密文形式(加密规则：字母 a→z，b→y，c→x，…，x→c，y→b，z→a，其他字符保持不变)。

③ 编程实现对密文的解密。

分析：根据加密规则，很容易得到如下转换公式。

```
newc = 122 - ord(c) + 97
```

其中，122 是 ord('z')，即'z'字符的 ASCII 码；97 是 ord('a')即'a'字符的 ASCII 码。ord 函数将字符转换为其 ASCII 码，chr 函数则相反，即把一个 ASCII 码整数转换成字符。

同时可以看出加密和解密公式一致。

```
def inv(c):                                 #加密和解密公式实现
   if 'a' <= c <= 'z':
      return chr(122 - ord(c) + 97)         #'z' - c + 'a'
   if 'A' <= c <= 'Z':
      return chr(90 - ord(c) + 65)          #'Z' - c + 'A'
   return c
```

```
def jiami(ls):                              #将字符串转换成加密后的字符串
    ls2 = []
    for i in range(len(ls)):
        c = ls[i]
        if 'a'<= c <= 'z':
            ls2.append(inv(c))
        else:
            ls2.append(c)
    return ''.join(ls2)                     #将列表列连接字符串
```

实际上,以上加密函数如下实现。

```
def jiami(ls):                              #将字符串转换成加密后的字符串
    ls2 = [inv(c) for c in ls]              #列表生成式
    return ''.join(ls2)
```

以下是主程序读取文件 demo.txt 内容,并把加密后的列表 r 写入文件 cipher.txt,最后读取文件 cipher.txt,采用与加密一样的函数进行解密并打印出来。

```
f = open('demo.txt','r')
s = f.readlines()           #读取文件
f.close()
r = [jiami(ls) for ls in s]
#写入文件 cipher.txt
f = open('cipher.txt','w+') #可读写模式
f.writelines(r)
f.seek(0)
result = f.readlines()
print('转换结果为:')
for s in result:
    print(s.strip())        #strip()法用于移除字符串头尾指定的字符(默认为空格或换行符)
f.close()
#解密方法和加密一致
r = [jiami(ls) for ls in result]
print('原文为:')
for s in r:
    print(s.strip())
```

运行结果如下。

```
转换结果为:
zyxwvut
cba
原文为:
abcdefg
xyz
```

四、编程并上机调试

(1) 编写程序,将 100 以内所有素数写入文件 result1.txt 中。

(2) 编写程序,将 100 以内孪生素数写入文件 result2.txt 中。(孪生素数指两个素数相差为 2,例如,3 和 5、5 和 7、11 和 13 等都是孪生素数。)

（3）编写程序，统计一篇英文文章中单词的个数，并输出其中出现最多的前 5 个单词。

（4）编写程序实现，在文件“data.txt”的第一行中写入自己的学号和姓名，然后随机生成 20 个[100,200]的整数，统计出最大值，并将这些整数写入文件“data.txt”中，每行一个数字，以换行符分开。

（5）文件 file.txt 存储的是一篇英文文章，编写程序，将其中所有小写字母转换成大写字母输出。

习题

1. 编写程序，打开任意文本文件，读出其中内容，判断该文件中某些给定关键字（如“中国”）的出现次数。

2. 编写程序，打开任意文本文件，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“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）包含字符最多和最少的行分别在第几行？

4. 统计 test.txt 文件中大写字母、小写字母和数字的出现次数。

5. 编写程序统计调查问卷各评语出现的次数，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，字典最终统计结果追加至 result.txt 文件中。

6. 文件 src.txt 存储的是一篇英文文章，将其中所有大写字母转换为小写字母输出。例如，若 src.txt 中的存储内容为 This is a Book，则输出内容应为 this is a book。

7. 文件“score.txt”中存储了歌手大奖赛中每个歌手的打分，10 名评委的分数在同一行，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

…

从文件中读取数据并存入列表，计算歌手的最终得分，其计算方式是去掉最高分和最低分后求平均分。最终得分保留两位小数，将其输出到屏幕。