第5章 Python文件与路径的基本操作

为了长期保存数据，也为了能和其他人分享或使用相关数据，就需要对文件进行操作。在实际工程应用中，经常会出现读取某个文件中的数据并在经过某些处理后再把结果存到其他文件的情况，这时就要用到对文件的操作了。本章提到的文件主要包括**文本文件**（没有特定格式的纯文本文件，如 txt 文件等，内容由各行组成，行末有回车换行符，文本文件中的内容无须特定的解析软件即可直接阅读）、csv 文件等简单类型。另外，经常遇到的文件格式还有各种**二进制文件**（内容以字节串的形式进行存储，如 jpg 图片、mp4 文档、数据库文件、PDF 文档等；通常情况下，若无专门的软件进行解码，其中的内容是无法直接编辑、阅读的）等。由于对文件的操作需要在相应的文件夹下进行，因此掌握路径的操作也是必要的。限于本书的科普性质，本章只学习针对 txt 文件、csv 文件等常见文本文件的操作以及相应的基本路径的操作方法。

学习本章内容时，要求：

- 掌握 Python 常用的文件操作方法，包括常用内置方法和部分标准方法、外部引用包的携带方法等；
- 掌握 txt 文件、csv 文件等常见文本文件的打开、读取数据、写入信息的基本操作方法；
- 掌握文件路径的基本操作方法。

5.1 读写文本文件

为了长期保存、传输、修改数据，很多时候需要用文件（包括数据库）存储和管理相关的数据。当程序运行时，变量虽然可用来保存和传递数据，但变量以及对象中存储的数据是暂时的，程序结束后就会丢失。如果希望永久保留数据，就需要将数据保存到文件中，这时就要用到文件操作了。Python 提供了内置的文件对象，以及对文件、目录进行操作的内置模块。可以说，掌握对文件及其路径的操作是十分重要的。

文件一般可以分为纯文本文件（如 txt、ini 文件等）、特定的表格文件（如 csv 文件等），以及非文本

文件(如 pdf、docx、jpg、mp4 等)。对文件的操作一般包括打开文件(open)、读取数据(read)、写入数据(write)、关闭文件(close)等。对于其他非纯文本的二进制文件(如 JPEG、数据库文件、PDF 文档等),需要了解二进制文件的读取方法,这里可能要用到序列化的概念,它可以理解为将内存中的数据在不丢失其类型信息的情况下转换成对象的二进制形式的过程,Python 中常用的序列化模块有 struct、pickle 模块等。限于篇幅及本书的科普定位,这里不再详细介绍二进制文件的读写方式,且不再区分纯文本文件和特定的表格文件。本章将简介 txt、csv(以逗号等符号作为分隔符的特殊文本文件)等类型的文件的读写与数据使用方法。

5.1.1 打开和关闭文件的基本操作

Python 内置了不少和文件操作有关的方法,如通过 open()方法可按指定的模式(mode)“读”或“写”或“追加”数据。通过内置方法 open()可以打开路径下的某个文件;如成功,则返回一个可迭代的文件对象;如果欲打开的文件不存在或打开失败,则系统会给出异常,其语法如下(注:对中学生来说,只需要了解 file、mode、encoding 参数的使用方法即可;若需要深入了解其他相关参数的使用方法,可参考相关手册,这里不再全面介绍):

```
open(file, mode, buffering, encoding, ...)
```

- file 参数:指定欲打开的文件名称,这是必须提供的参数。
- mode 参数:指定打开文件后的处理方式,部分常用 mode 参数及功能如下。

r:以只读方式打开文件,文件的指针将会放在文件的开头。

w:写入信息。如果该文件已经存在,则先清空原有内容;如果该文件不存在,则创建新文件。

a:追加新内容,但不覆盖原有文件中的内容。

b:二进制,可以与其他模式组合使用。

t:文本模式。

+:可与其他模式混合使用。

rb:以二进制格式打开只读文件,文件指针在文件开头。

wb:以二进制格式打开文件并写入信息,文件指针在文件开头。

- buffering 参数:指定读、写文件的缓存模式,0 表示不缓存,1 表示缓存,大于 1 表示缓冲区的大小,默认是缓存模式。
- encoding 参数:指定对文本文件进行编码和解码的方式,如 GBK、UTF-8、CP936 等,当不给出 encoding 参数时,默认编码格式是 UTF-8,在 Windows 平台下自动采用 GBK(CP936)编码。

一般来说,对文件的操作包括①打开文件:使用 open()语句打开文件,它会返回一个文件对象;②对这个文件对象执行读、写等操作:读操作可使用 read()、readline()、readlines()等方法实现,写操作可使用 write()、writeline()、writelines()等方法完成。

还有一种打开和关闭文件的方式是使用上下文管理语句with open(文件名, mode, encoding) as f:。with 语句块可以自动管理资源,不论什么原因跳出 with 代码块,它都能保证文件被正确关闭,并且可以在代码块执行完毕后自动还原进入该代码块时的上下文,此时无须再使用 close()语句关闭文件,常用于文件操作、数据库连接、网络连接、多线程与多进程同步时的锁对象管理等场合。当然,对文件执行什么读、写操作,还要程序设计者自己设计,with 语句块并不管要读还是要写文件,它只负责执行 open()语句打开文件并在结束后自动关闭文件。例如,向打开的指定文件中追加字符串信息,下框中

右侧的代码一般比左侧的代码更实用。

```
s = 'hello'
f = open('文件名.txt', 'a+')
f.write(s)
f.close()
```

```
s = 'hello'
with open('文件名.txt', 'a+') as f:
    f.write(s)
```

5.1.2 读写文本文件的基本操作

对文件的操作有很多种类，常见操作包括创建文件、删除文件、修改文件读写权限、读取文件中的内容、将相关内容写入文件等。由于本书的科普性质，因此这里只介绍写入、读取文件操作。

读操作的常用方法有以下几种。

- read([size])：一次性读取整个文件内容，可选参数 size 指定读取块的大小。
- readline([size])：每次读取一行内容(包括\n 字符)，可选参数 size 指定读取块的大小。
- readlines()：一次性读取整个文件内容并按行返回到列表，以便进行遍历操作。

例 5.1 使用 open()方法的写模式创建一个文本文件，并向其中写入自定义的一组字符串。之后使用 open()的读模式打开，并使用 read()、readline()、readlines()方法读取内容，分析读到内容的区别；再使用 with 语句块打开文件，并分别使用 read()方法以及 for 循环遍历打开文件的方法，分别读取其中的内容。

【提示与说明】 首先完成文件创建及写内容的操作。在 open()方法中可定义打开文件的名称，操作模式 w 是写操作模式，若该文件不存在，就创建它并把文件句柄赋值给某个变量，之后通过写文件内容的 write()方法向这个文件写入内容。代码如图 5.1 所示。第 7 行开始的代码是使用 open()方法并以读的模式打开上述文件，记得要用 close()方法关闭文件。第 13～20 行是采用 with 语句块管理文件，注意 read()、readline()、read lines()方法执行后结果的区别，因为它是可迭代对象，因此可以使用 for 循环显示文件内容(如第 24 行代码所示)。

例 5.2 向一个文本文件中写入指定次数的一个字符串后，显示写入的内容。

【提示与说明】 在打开文件时指明写模式，写入用 write()方法，读取用 read()方法。既然是指定次数，那么就要用到循环操作。代码实现如图 5.2 所示。

例 5.3 当前路径下的文本文件 english.txt 中有 3 行英文文本，如图 5.3 所示。定义 3 个函数，分别使用 read()、readline()、readlines()方法读取内容并显示，要求使用 read()和 realine()方法时指定读取字符块的大小。

【提示与说明】 定义 3 个函数，如图 5.4 所示。直接通过 open()方法打开文件，执行完毕后，记得使用 close()方法关闭。read()、readline()方法可读取指定字符数的文本块，但 readlines()方法无法指定读取字符数的大小。

打开一个文本文件并向其中写入信息的方法，与打开一个文件并从其中读出信息的方法是类似的，都需要通过 open()方法做好准备，如图 5.5 所示，只不过在 open()方法中要设定文件操作模式为写模式(模式含义参见上文)，使用 write()方法可以写入信息。

```
#建立文件并向其中写入字符串
s = 'Hello world\nHello Romania\nHello Suceava\n'
with open('sample.txt', 'w') as myfp:    #若文件不存在就创建它
    myfp.write(s)  #写入信息之后关闭文件

#下面是直接打开文件并读取内容的方法
f = open('sample.txt','r')
str1 = f.read()
print("直接打开并读取文件：",str1,"文件字符长度是：", len(str1))
f.close()

#下面是使用with块打开并读取文件内容的方法
with open('sample.txt') as fp:
    print("使用with打开并read方法:",fp.read())

with open('sample.txt') as fp:
    print("使用with打开并readline方法:",fp.readline())

with open('sample.txt') as fp:
    print("使用with打开并readlines方法:",fp.readlines())

print("另一种通过遍历可迭代对象而显示内容的方法")
with open('sample.txt') as fpp:
    for myline in fpp:
        print(myline, end = "")
```

```
直接打开并读取文件： Hello world
Hello Romania
Hello Suceava
 文件字符长度是： 40
使用with打开并read方法: Hello world
Hello Romania
Hello Suceava

使用with打开并readline方法: Hello world

使用with打开并readlines方法: ['Hello world\n', 'Hello Romania\n', 'Hello Suceava\n']
另一种通过遍历可迭代对象而显示内容的方法
Hello world
Hello Romania
Hello Suceava
```

图 5.1　直接打开文件和使用 with 语句块打开文件

```
file = open('demo.txt', 'w')
for count in range(1,4):
    file.write(str(count)+":"+"重要的话说三遍\n")
with open('demo.txt') as fp:
    print(fp.read())

```

```
1:重要的话说三遍
2:重要的话说三遍
3:重要的话说三遍
```

图 5.2　写入指定次数的字符串

From the moment that Asia-Pacific leaders arrived in Bangkok for the first in-person APEC Economic Leaders' Week in four years, the focus has been on how members in the grouping can facilitate free and open trade and investment, reconnect the region, and promote sustainable economic growth.

图 5.3　文本内容

```
#使用read()方法读取文件中指定字符数
def file_read(fname):
    txt = open(fname)  #打开文件
    print("1. read()读取指定字符数：",txt.read(20)) #读取指定大小块的文件
    txt.close()

#使用readline()每次读取一行内容
def file_readline(fname):
    txt2 =open(fname)
    print("2. 使用readline()结果：",txt2.readline(20))  #也可以指定size
    txt2.close()

#使用readlines()读取文件内容并按行返回list
def file_readlines(fname):
    txt3 = open(fname)
    for i in txt3.readlines(): #readlines()size无效
        print("3. 使用realines()结果：",i, end = "")
    txt3.close()

file_read("english.txt")
file_readline("english.txt")
file_readlines("english.txt")
```

```
1. read()读取指定字符数： From the moment that
2. 使用readline()结果： From the moment that
3. 使用realines()结果： From the moment that Asia-Pacific leaders arrived in Bangkok fo
r the first in-person APEC Economic Leaders' Week in four years,
3. 使用realines()结果： the focus has been on how members in the grouping can facilitat
e free and open trade and investment,
3. 使用realines()结果： reconnect the region, and promote sustainable economic growth.
```

图 5.4 read()、readline()、readlines()方法使用示例

```
s = "Hello World\n文本文件的读取方法\n算法与数据结构\n"
with open("sample_1.txt", "w") as fp:
    fp.write(s)
with open("sample_1.txt", "r") as fp:
    print(fp.read())
```

```
Hello World
文本文件的读取方法
算法与数据结构
```

图 5.5 向文件中写信息

要掌握对文件对象的读写操作。

- 以读模式打开文件并设定文件对象 f1：　f1＝ open(“文件名”, ‘r’)
- 以写模式打开文件并设定文件对象 f1：　f1 ＝ open(“文件名”, ‘w’)
- 从文件对象 f1 中读取信息并赋值给变量：　str ＝f1. read()
- 关闭由 f1 对象指向的文件：　f1.close()

如果需要读取某个源文件中的内容并将其写入其他文件，则可在一条 with 语句中同时定义要读入信息的源文件和要写入信息的目标文件，如图 5.6 所示，其中 src 和 dsc 分别代表源文件和目标文件，注意二者的 open()方法中的模式是不一样的；src.read()是返回读取的源文件内容并放置在缓存中，dst.write()则是将存于缓存的已读取的信息写入目标文件。

```
1 with open("sample_1.txt","r") as src,open("sample_2.txt", "w") as dst:
2     dst.write(src.read())
```

图 5.6　同时读取源文件中的内容并写入目标文件

下面通过一个例子看看在 open()方法中用来控制字符编码方式的 encoding 参数的用法。

图 5.7 所示是通过定义函数的方式，将一个以 CP936 编码的文本文件中的内容复制到另一个使用 UTF-8 编码的文本文件中的操作。此例通过两个嵌套的 with 语句分别以读(第 2 行代码)和写(第 3 行代码)的方式打开两个文件，并分别赋值给两个文件变量。在通过 read()方法读出源文件内容后，将其作为参数传递给 write()方法，完成对目标文件的写入。注意第 2 行和第 3 行代码分别在 open()方法中设置了文件编码方式，其参数来源于函数的形参。在最后调用该函数时，指明了源文件、目标文件、源文件编码方式、目标文件编码方式等。

```
1 def fileCopy(src, dst, srcEncoding, dstEncoding):
2     with open(src, 'r', encoding=srcEncoding) as srcfp:
3         with open(dst, 'w', encoding=dstEncoding) as dstfp:
4             dstfp.write(srcfp.read())
5 fileCopy('sample.txt', 'sample_new.txt', 'cp936', 'utf8')
```

图 5.7　复制文件时对文件编码的转换

从图 5.7 函数定义中，要再一次理解形式参数和实际参数的区别。

由于本书的科普性质，因此这里仅给出有关文件的常用操作方法(如表 5.1 所示，这里假设文件变量为 f)，有关文件操作的完整内容，可参阅相关 Python 编程手册。

表 5.1　对文件的常用基本操作方法

方　法	说　明
f.open()	在 open()方法中指定打开某文件及其模式，并赋值给文件变量 f
f.close()	把缓冲区中的内容写入 f 文件，同时关闭 f 文件并释放文件对象
f.flush()	把缓冲区中的内容写入 f 文件(但不关闭文件)
f.write(aString)	将 aString 表示的内容写入打开的文件 f 中，返回写入的字符数
f.writeline(aString)	把内容列表写入文件 f 中
f.writelines(aString)	把内容列表写入文件 f，不添加换行符
f.read([size])	读取 f 文件内容并以字符串的形式返回，size 可选参数为字符数，未指定时读取所有内容
f.readline(size)	从文件 f 中读取一行内容作为结果返回，size 指定读取块的大小
f.readlines([sizehint])	一次性读取整个文件内容并按行返回到列表
f.seek(offset [,from])	改变文件 f 的当前指针位置；offset 是相对于 from 的位置：from 是默认值 0，表示开头；1 表示当前位置；2 表示文件结尾。示例如例 5.4、例 5.5 所示

下面通过几个例题说明表 5.1 中部分方法的使用。

例 5.4　考察 seek()方法的使用。假设有一个文本文件 sample.txt，现在要以写模式打开它，并在

特定位置插入新的给定字符串，插入位置和插入内容可在代码中自行定义。

【提示与说明】 这道题目考察 seek()方法的使用。通过 with 语句块以写方式打开文件，采用 seek()方法定位到需要的位置上，再基于 write()方法写入信息，如图 5.8 所示。

同理，也可以通过 seek()方法定位到特定位置后读取相应长度的内容，如图 5.9 所示。

```
1  with open("sample.txt", 'w') as fp:
2      fp.seek(12,0)  #定位
3      x =fp.write("众志成城")
4      print("写入的字符数是：", x)
```

```
写入的字符数是： 4
```

图 5.8 seek()方法定位并写入信息

```
1  f = open("sample.txt","r")
2  f.seek(6)
3  str1 = f.read(50)
4  print(str1,len(str1))
```

```
众志成城 10
```

图 5.9 seek()方法定位并读取信息

例 5.5 假设一个文本文件 data.txt 中有若干用英文逗号分隔的多行数据，这些数据可能是数量不等的整数或小数(但无字母和特殊符号)。这些行的数据头、尾可能有数量不等的空格。请先清洗数据，删除头、尾空格，之后将这些整数按升序排序后，再用分号分隔，写入另一个文本文件 data_asc.txt 中。

【提示与说明】 首先要打开文件，可通过 readlines()方法读取所有行。数据清洗时，可以用 strip()方法移除每行头、尾的指定字符，默认为移除空格或换行符，但该方法只能删除开头或结尾的字符，不能删除中间部分的字符。为了便于统一处理这些位于多行的数，可以将它们以逗号为分隔符拼接起来，形成一个“大串”。之后，用 split()方法分隔得到一组数字字符串序列，通过类型转换为整数后，放入一个列表中，对该列表采用 sort()方法排序，最后在这些有序数据之间插入指定的分号分隔符，将排好序的、中间添加了分隔符的这串数据写到新的文件中。伪代码如下。

例 5.5：

Input：文本文件
Output：文本文件
Steps：
1. 打开文件读取所有内容
2. 清洗数据
   2.1 利用读取内容的可迭代特征遍历它并分别进行处理
3. 将干净的数按“特定字符”拼接在一起，以便于集中处理
4. 按“特定字符”进行分隔，得到纯数内容的字符串
5. 类型转换为整数并存于列表中
6. 对列表中的纯数据进行排序
7. 在各个已经排好序的数据之间插入指定的分号作为分隔符
8. 写文件

代码实现如图 5.10 所示。第 3 行代码完成无关字符清理；第 4 行代码是以逗号为分隔而合并所有行，但执行完毕后会在最后增加一个逗号进行数值类型转换，因此第 5 行代码的作用是将最后那个多余的逗号删除，且由于字符串函数 rstrip()执行完毕后不影响原字符串 data 的内容，故这里将其赋值给一个新的变量 newdata。后续完成类型转换、排序、写文件等操作，详见图 5.10 中的代码注释，这里不再赘述。

例 5.6 打开含有多行英文句子的一个文本文件，如图 5.11 所示。统计其最长行的英文字符数，并显示该行内容。

```
with open('data.txt', 'r') as fp:
    data = fp.readlines()                    #读取所有行
data = [line.strip() for line in data]       #删除每行两侧的空白字符
data = ','.join(data)                        #合并所有行
newdata = data.rstrip(",")
newdata = newdata.split(',')                      #分隔得到所有数字字符串
newdata = [float(item) for item in newdata]         #将字符转换为数字
newdata.sort()                                 #升序排序
newdata = ';'.join(map(str,newdata))               #将结果再转换为字符串
with open('data_asc.txt', 'w') as fp:        #将结果写入文件
    fp.write(newdata)
```

图 5.10　综合示例(1)

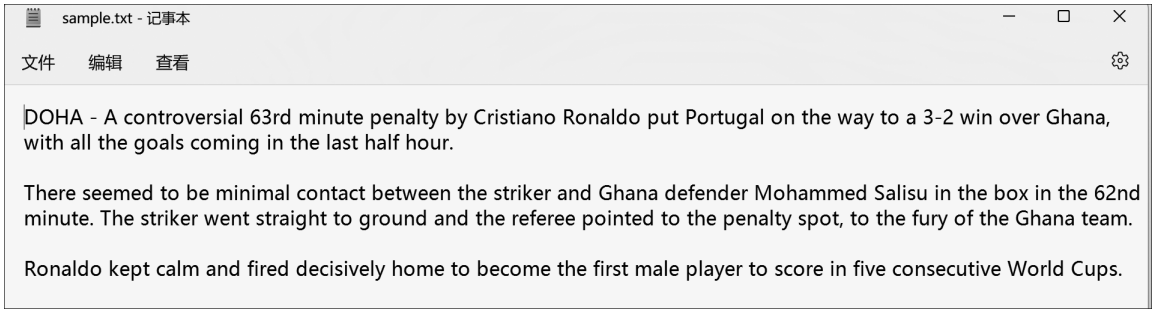

图 5.11　示例文本文件

【提示与说明】　首先打开这个文本文件。因为每一行的字符数长度都不一样，而列表可以存储不同的元素类型，因此可定义一个列表，列表的第一个元素为某行的字符数，第二个元素为该行内容(某个字符串)。采用循环机制，依次顺序读取文件中的每一行，若其长度大于已处理的行的长度，则将该行字符内容与其长度存储在列表中，否则不变。可见，这就是一个找最大值的算法。这样，就能保证这个列表中存储的是最长那行的长度及其对应的字符串内容。最后输出这个列表中的元素，即可得到最长行的字符串以及字符数。伪代码如下，代码实现如图 5.12 所示。

例 5.6：

Input：文本文件
Output：含字符数最多的行的内容及对应的字符数
Steps：
1. 打开文件读取所有内容
2. 定义存储最终结果的列表：其第一个元素为待处理行中的字符数，第二个元素为字符串内容
3. 依次读取各行内容
   3.1 若其长度大于列表中的第一个元素，则存储到列表，否则继续判断下一行的内容

5.1.3　读写 CSV 文件的基本操作

CSV 文件(全称是 Comma-Separated Values，意思是逗号分隔值)可以用 Excel 打开。鉴于本书的科普定位，可以利用 Python 自带的标准 CSV 模块中的 reader 类和 writer 类读写 CSV 文件的数据：使用其中的 csv.reader()函数接收这个 CSV 文件对象，就可以通过迭代对象的方法处理文件中的各行内容；使用其中的 csv.writer()函数可向其中写入信息。下面对这两个方法的主要参数进行简介。

```
with open('sample.txt') as fp:
    result = [0, '']  #第一个参数为字符串（默认为0），第二个参数存该行内容
    for line in fp:
        t = len(line)
        if t > result[0]:
            result = [t, line]
print(result)

```

```
[233, 'There seemed to be minimal contact between the striker and Ghana defender Mohammed Salisu in the box in the 62nd minute. The striker went straight to ground and the referee pointed to the penalty spot, to the fury of the Ghana team.\n']
```

图 5.12　排序各行长度

- csv.reader(csvfile，[dialect='excel']，…)。csvfile 参数用于读取 CSV 文件；可选参数之一的 dialect 是用于不同 CSV 变种的特定参数组。该方法的参数很多，不止列出的这两个，如需了解更多内容，可以查询相关手册，这里不再详述。reader()方法返回一个 reader 对象，该对象将逐行遍历 CSV 文件，CSV 文件的每一行都读取为一个由字符串组成的列表(如图 5.13 输出结果所示)。
- csv.writer(csvfile，[dialect='excel']，…)。csvfile 参数是要写入信息的 CSV 文件(可以是任何具有 write()方法的对象)；可选参数之一的 dialect 是用于不同 CSV 变种的特定参数组。该方法也有其他多个参数，不再赘述。writer()方法返回一个 writer 对象，该对象负责将用户数据在给定的文件类对象上转换为带分隔符的字符串。

例 5.7　在当前路径下，一个名为 student_scores.csv 的文件中存储了多行信息，每行的属性列为 sno、sname、grade(分别代表学号、姓名、考试分数)，每行数据对应于一个学生的信息。请显示该 CSV 文件中各行的信息。

【提示与说明】　可使用 CSV 包中的 reader()方法读取该 CSV 文件的内容，CSV 文件的每一行被读取为一个由字符串组成的列表，也可将其转换为字符串。代码实现如图 5.13 所示。第 1 行代码首先导入相关的用来处理 CSV 文件的包，第 3 行代码为使用 csv.reader()方法读取打开的 CSV 文件，读取的内容可被迭代处理(第 5～8 行代码的 for 循环)。由于本 CSV 文件的第 1 行为属性信息，因此输出的第 0 行显示的是具体的字段结构信息，而非某个具体同学的信息。

```
import csv
with open('student_scores.csv') as csvfile:
    mystring = csv.reader(csvfile)
    i = 0
    for row in mystring:
        print("第 %d行的列表信息： " %i,row) #列表
        i +=1
        print(', '.join(row))  #字符

```

```
第 0行的列表信息：  ['sno', 'sname', 'grade']
sno, sname, grade
第 1行的列表信息：  ['1001', 'smith', '70']
1001, smith, 70
第 2行的列表信息：  ['1002', 'tom', '50']
1002, tom, 50
第 3行的列表信息：  ['1003', 'alan', '90']
1003, alan, 90
```

图 5.13　读取 CSV 文件中各行的信息

一个数据表是由横着的“行”和纵着的“列”组成的。列是结构信息，即每一行都要有的字段结构，如“姓名　性别　年龄”等；而每一行是一组特定的人或物的各个属性信息，如“晓明　男　12”等。列称为结构，行称为记录。

例 5.8　向上述 CSV 文件中增加一行信息，包含学号、姓名、考试成绩等信息（具体数据值可自行在代码中给定）。

【提示与说明】　在 open()方法中以追加方式打开上述 CSV 文件（参见图 5.14 第 2 行代码中的‘a’模式），并将文件对象 csvfile 作为参数传递给 csv.writer()方法，可以向该文件写入信息。由于各行是以列表形式存储的，因此拟增加的信息也用列表方式表示，如图 5.14 所示。

```
1  import csv
2  with open('student_scores.csv', 'a') as csvfile:
3      mywriter = csv.writer(csvfile)
4      newrow = [1004,"Hao",100]  #拟增加的行中各个属性列的信息
5      mywriter.writerow(newrow)
```

图 5.14　向 CSV 文件中写入一行信息

CSV 文件的 writerow()是单行写入，即将一个列表全部写入 CSV 的同一行；CSV 文件的 writerows()是多行写入，即将一个二维列表的每一个列表写为一行，适合于写入多行信息。因此，可以用 writerow()完成对数据表结构的定义（定义属性字段），用 writerows()完成对多行信息的写入（录入多行数据）。

课堂练习

请使用 CSV 文件的 writerows()方法，仿照图 5.14 中的示例，一次性地在 CSV 文件中插入多行数据。

例 5.9　新建一个 CSV 文件，写入两名同学的基本信息（班级号、姓名、性别、身高、爱好），之后打开该文件，显示刚才写入的信息（首行是属性信息，接下来是这两名同学的记录信息）。

【提示与说明】　向一个新建的 CSV 文件中写入信息时，首先要定义它的结构（属性或字段名）。本题要求的字段结构是班级号、姓名、性别、身高、爱好，可以先用一个列表进行定义，通过 writerow()写入字段的结构信息（参见图 5.15 第 2、3 行代码）。之后，通过 open()方法新建并打开该 CSV 文件后，通过 writerow()写入结构，即属性信息，writerows()一次性写入各个记录的详细信息。完成写入后，可以使用文件的 reader()方法读取并迭代输出每一行信息以完成显示。

```
1  import csv
2  header=['班级','姓名','性别','身高','爱好']
3  rows=[['001','晓明','男',178,'足球'],['002','晓红','女',162,'钢琴']]
4  with open('mynewcsv.csv','w',newline='')as f: #newline=" "是为了避免写入之后有空行
5      fw=csv.writer(f)
6      fw.writerow(header) #字段头
7      fw.writerows(rows)  #写入各记录
8
9  with open('mynewcsv.csv') as f1:
10     mystring = csv.reader(f1)
11     for row in mystring:
12         print(row)
13
14
```

```
['班级', '姓名', '性别', '身高', '爱好']
['001', '晓明', '男', '178', '足球']
```

图 5.15　writerow()和 writerows()的比较