第5章 文件读写

文件是计算机存储数据的重要形式，用文件组织和表达数据更加有效和灵活。文件有不同编码和存储形式，如文本文件、图像文件、音频和视频文件、数据库文件、特定格式文件等。每个文件都有各自的文件名和属性，对文件进行操作是 Python 的重要功能。

5.1 文本文件读写

5.1.1 读取文件全部内容

文本文件的读取通常有 3 个基本步骤：打开文件、读取文件和关闭文件。

1. 打开文件

文件访问前必须先打开文件，并指定文件将做什么操作。Python 通过内置标准函数 open()打开或创建文件，函数返回文件句柄。打开文件语法如下。

```
1 文件句柄 = open('文件名', '操作模式', '编码')
2 with open('文件名', '操作模式' , '编码') as 文件句柄:
3 with open(file_name1) as file1, open(file_name2) as file2, open(file_name3) as file3:
```

(1) 文件句柄。文件读取是一个非常复杂的过程，它涉及设备(如硬盘)、通道(数据传输方式)、路径(文件存放位置)、进程(读写操作)、文件缓存(文件在内存中的存放)等复杂问题。文件句柄是函数 open()的返回值，它隐藏了设备、通道、路径、进程、缓存等复杂操作，帮助程序员关注正在处理的文件。**文件句柄用变量名表示**(如 file、file_data、f 等)，它通过函数 open()与文件连接。文件打开或创建后，文件句柄就代表了打开的文件对象，这简化了程序设计，程序员可以方便地读取文件中的数据。

(2) 文件函数。函数 open()可以打开一个文本文件或者新创建一个文本文件。函数 open()参数为'文件名'、'操作模式'、编码，参数用引号括起来。

(3) 文件名。文件名表示文件的存储位置，包括文件路径和文件名。文件可以采用相对路径，如'江南春.txt'；也可以采用绝对路径，如'd:\\test\\05\\江南春.txt'。

(4) 编码。编码指源文件的编码模式，如 encoding='utf8'、encoding='gbk'等。

(5) 操作模式。操作模式即文件的读写模式，操作模式用引号括起来，如'r'、'rb'、'w'、'a'等。常用操作模式如表 5-1 所示。

表 5-1　文件打开常用操作模式

操作模式	参数说明
r	仅读。待打开的文件必须存在；若文件不存在则会返回异常 FileNotFoundError
w	仅写，不可读。若文件已存在则内容将先被清空；若文件不存在则创建文件
a	仅写。若文件已存在则在文件最后追加新内容；如果文件不存在则创建文件
r+	可读，可写，可追加。待打开的文件必须存在(+参数说明允许读和写)
a+	读写。若文件已存在，则内容不会清空
w+	读写，若文件已存在，则内容将先被清空
rb	仅读，读二进制文件。待打开的文件必须存在(b 参数说明读二进制文件)
wb	仅写，写二进制文件。若文件已存在，则内容将先被清空
ab	仅写，写二进制文件。若文件已存在，则内容不会清空

2. 读取文件

Python 提供了 read()、readlines()和 readline()三个文件读取函数。这 3 种方法都会把文件每行末尾的'\n'(换行符)也读进来，可以用 splitlines()等函数删除换行符。

(1) **函数 read()一次读取文件的全部内容，返回值是一个大字符串**。读取文件时会包含行末尾的换行符(\n)。它常用于文本按字符串处理，如统计文本中字符的个数。

【例 5-1】 用函数 read()读取“琴诗.txt”文件中全部内容。

```
>>> f = open('d:\\test\\05\\琴诗.txt', 'r', encoding = 'gbk')    # 打开文件，f 为文件句柄
>>> s = f.read()                                                # 读取文件全部内容
>>> s                                                           # 输出内容为字符串
'[宋] 苏轼《琴诗》\n若言琴上有琴声，放在匣中何不鸣？\n若言声    # 参数\n 为换行符
在指头上，何不于君指上听？\n'
```

【例 5-2】 读取“琴诗.txt”文件中全部内容，并且删除文件中的换行符。

```
>>> file = open('d:\\test\\05\\琴诗.txt', 'r')                  # 打开文件，file 为文件句柄
>>> s = file.read()                                            # 读取文件全部内容
>>> s = s.splitlines()                                         # 删除字符串中的换行符
>>> s                                                          # 输出内容已转换为列表
['[宋] 苏轼《琴诗》', '若言琴上有琴声，放在匣中何不鸣？', '若言  # 输出已经删除换行符
声在指头上，何不于君指上听？']
```

(2) **函数 readlines()一次读取文件的全部内容，返回值是以行为元素的字符串列表**，该列表可以用 for 循环进行处理。注意，行可能是文本中的一个短行，也可能是一个大的段落。它常用于文本按行处理，如统计文本中字符串的行数(参见 5.2.5 节)。

【例 5-3】 用函数 readlines()读取“琴诗.txt”文件中全部内容。

```
>>> file = open('d:\\test\\05\\琴诗.txt', 'r')                  # 打开文件，file 为文件句柄
>>> file.readlines()                                           # 读取文件全部内容
['[宋] 苏轼《琴诗》\n', '若言琴上有琴声，放在匣中何不鸣？    # 输出内容为列表
\n', '若言声在指头上，何不于君指上听？\n']
```

(3) **函数 readline()每次只读取一行，返回值是字符串**。它的读取速度比 readlines()慢得多，只有在没有足够内存一次读取整个文件时，才会使用 readline()函数。

【例 5-4】 用函数 readline()读取“琴诗.txt”文件中一行内容。

```
>>> file = open('d:\\test\\05\\琴诗.txt', 'r')                    # 打开文件,file 为文件句柄
>>> file.readline()                                               # 读取文件一行内容
'[宋] 苏轼《琴诗》\n'                                             # 输出一行字符串
>>> file.close()                                                  # 关闭文件
```

3. 关闭文件

文件使用结束后一定要关闭,这样才能保存文件内容,释放文件句柄占用的内存资源。关闭文件语法如下。

```
文件句柄.close()
```

5.1.2 读取文件指定内容

1. 文件指针

文件打开后,对文件的读写有一个读取指针(元素索引号),从文件中读入内容时,读取指针不断向文件尾部移动,直到文件结束位置。Python 提供了两个读写文件指针位置相关的函数 tell()和 seek()。它们的语法如下。

```
文件对象名.tell()                                                 # 获取当前文件操作指针的位置
文件对象名.seek(偏移量[, 偏移位置])                               # 改变当前文件操作的指针位置
```

函数 seek()中,偏移量表示要移动的字节数,正数表示向文件尾部移动,负数表示向文件头部移动。数字、英文字符、换行符等占 1 字节,GBK 编码下汉字占 2 字节。

函数 seek()中,偏移位置=0 表示文件开头,偏移位置=1 表示当前位置,偏移位置=2 表示文件结尾位置。函数 seek()返回文件行指针位置。

【例 5-5】 获取文件指针位置。

```
>>> file = open('d:\\test\\05\\琴诗.txt')
>>> file.tell()                                                   # 获得当前文件读取指针
0
>>> file.seek(10)                                                 # 将文件指针向后移动 10 字节
10
>>> file.seek(0, 2)                                               # 将文件读取指针移动到文件尾部
85                                                                # 文件大小为 85 字节
```

2. 读取文件指定行

【例 5-6】 文件“成绩 utf8.txt”内容如下所示,读取文件中第 2、3 行。

```
学号,姓名,班级,古文,诗词,平均
1,宝玉,01,70,85,0
2,黛玉,01,85,90,0
3,晴雯,02,40,65,0
4,袭人,02,20,60,0
```

案例分析:用函数 readlines()读出文件到列表,循环对列表指定数据进行切片。

```
file_name = 'd:\\test\\05\\成绩 utf8.txt'                         # 路径赋值(绝对路径)
with open(file_name, 'r', encoding = 'utf8') as file:             # 打开文件(文件编码为 UTF8)
    lines = file.readlines()                                      # 读取全部文件到列表 lines
```

```
4       for i in lines[1:3]:                          # 循环对列表切片,读取文本行
5           print(i.strip())                          # 函数 strip()为删除字符串两端的空格
```

```
>>>                                                   # 程序输出
1,宝玉,01,70,85,0
2,黛玉,01,85,90,0
```

程序说明：程序第 4 行，语句 for i in lines[1:3]中，[1:3]为列表切片索引号位置。其中，0 行是表头，不读取，列表第 3 行不包含，因此语句功能为循环读取列表第 1、2 行。

程序扩展：如果希望对文件数据隔一行读一行时，只需要修改程序第 5 行中列表索引号即可，如“for i in lines[1:4:2]:”。语句表示读取列表第 1～4 行，步长＝2(即读第 1、3 行)。

【例 5-7】 读取“成绩 utf8.txt”文件中第 2 行的内容。

案例分析：只要文件是 UTF8 编码，都可以用标准模块 linecache 中函数 getline()读出文件中的指定行。函数语法为 linecache.getline(文件名，行号)。

```
1  import linecache                                   # 导入标准模块——行读取
2  s = linecache.getline('d:\\test\\05\\成绩 utf8.txt', 2)   # 读取文件第 2 行
3  print('第 2 行:', s)
```

```
>>>第 2 行: 1,宝玉,01,70,85,0                        # 程序输出
```

【例 5-8】 读取“梁山 108 将 gbk.txt”文件，然后随机打印 5 个人。

```
1  import random                                      # 导入标准模块——随机数
2  n = 1                                              # 计数器初始化
3  file = open('d:\\test\\05\\梁山 108 将 gbk.txt', 'r')   # 打开文件
4  members = file.readlines()                         # 按行读取文件
5  while n <= 5:
6      winner = random.choice(members)                # 随机返回列表中的一个元素
7      print(winner)                                  # 打印人物姓名
8      n = n+1                                        # 计数器自增
```

```
>>>段景住    扈三娘    穆春    鲁智深    呼延灼        # 程序输出
```

5.1.3 文件内容遍历

文件遍历就是读取文件中每个数据，然后对遍历结果进行某种操作，如输出遍历结果、将遍历结果赋值到某个列表、对遍历结果进行统计(如字符数、段落数等)、对遍历结果进行排序、将遍历结果添加到其他文件等操作。遍历是一个非常重要的操作。

1. 文件遍历方法

用 open()或 with open()都可以实现文件遍历，它们的差别如下。

(1) 用 open()读取文件后，需要用 close()关闭文件；用 with open()读取文件结束后，语句会自动关闭文件，不需要再写 close()。

(2) 用 open()读取文件如果发生异常，则没有任何处理功能；而 with open()会处理好上下文产生的异常。

(3) open()一次只能读一个文件；with open()一次可以读多个文件。

【例 5-9】 文件遍历 1：逐行读取文件内容。

```
1  with open('登鹳雀楼.txt','r') as file:   # 打开文件(相对路径,当前目录在 d:\test\05)
2      content = file.read()               # 循环读取文件中每一行
```

```
3       print(content)                          # 输出行内容
        >>>[唐] 王之涣…                         # 程序输出(略,输出无空行)
```

【例 5-10】 文件遍历 2：一次全部读入文件内容到列表,再逐行遍历列表。

```
1  file = open('d:\\test\\05\\登鹳雀楼.txt', 'r')          # 打开文件(绝对路径),'r'为读操作
2  lst = file.readline()                                  # 将文件内容一次全部读入列表 lst
3  while lst:                                             # 循环输出列表 lst 中的内容
4      print(lst, end = '')                               # 参数 end = ''为不换行输出
5      lst = file.readline()                              # 读取列表中行的内容
6  file.close()                                           # 关闭文件
   >>>[唐] 王之涣…                                       # 程序输出(略,输出无空行)
```

【例 5-11】 文件遍历 3：循环读取文件内容到列表,再输出列表。

```
1  for lst in open('登鹳雀楼.txt','r'):                  # 打开文件,循环输出列表内容(相对路径)
2      print(lst)                                         # 输出列表内容
   >>>[唐] 王之涣…                                       # 程序输出(略,输出有空行)
```

【例 5-12】 文件遍历 4：文件内容一次全部读入列表,再逐行遍历列表内容。

```
1  file = open('d:\\test\\05\\登鹳雀楼.txt', 'r')         # 打开文件(绝对路径),r 为读操作
2  lst = file.readlines()                                 # 读取文件全部内容到列表
3  for s in lst:                                          # 循环读取列表中的行
4      print(s, end = '')                                 # 输出行内容(没有 end = ''参数会多输出一些空行)
5  file.close()                                           # 关闭文件
   >>>[唐] 王之涣…                                       # 程序输出(略,输出无空行)
```

【例 5-13】 文件遍历 5：一次读取 2 个文件全部内容到列表,再输出列表。

```
1  with open('d:\\test\\05\\金庸名言 1.txt', 'r') as file1,    # 读入文件 1
2      open('d:\\test\\05\\金庸名言 2.txt', 'r') as file2:     # 读入文件 2
3      print(file1.read())                                    # 打印第 1 个文件
4      print(file2.read())                                    # 打印第 2 个文件
   >>>                                                        # 程序输出
   侠之大者,为国为民。
   ——金庸《射雕英雄传》
   只要有人的地方就有恩怨,有恩怨就会有江湖,人就是江湖。
   ——金庸《笑傲江湖》
```

【例 5-14】 文件遍历 6：读取“成绩 utf8.txt”文件,打印内容和统计行数。

案例分析：函数 enumerate()的功能是遍历一个序列(参见 3.3.1 节),可以用它显示文件内容,用循环计数的方法统计文件行数。

```
1  file = open('d:\\test\\05\\成绩 utf8.txt', 'r', encoding = 'utf8')   # 打开文件,定义文件编码
2  count = 0                                                            # 计数器初始化
3  for index, value in enumerate(file):                                 # 循环读取文件(元组变量)
4      count += 1                                                       # 行数累加
5      print(f'{index}:{value}')                                       # 打印索引号和行内容
6  file.close()                                                         # 关闭文件
7  print('文件行数为:', count)                                          # 打印文件并统计行数
   >>> 0:学号,姓名,班级,…                                            # 程序输出(略,输出有空行)
```

程序说明：

程序第 1 行，文件必须为 UTF8 编码，而且设置编码参数 encoding＝'utf8'。

程序第 3 行，函数 enumerate()返回的迭代变量是元组。

2. 用 if 语句判断文件结束

【例 5-15】 用 if 语句判断文件是否结束。

```
1  file_path = 'd:\\test\\05\\登鹳雀楼.txt'    # 路径变量赋值
2  file = open(file_path, 'r')                 # 读取文件全部内容,file 为文件句柄,r 为读取模式
3  while True:                                 # while 永真循环
4      line = file.readline()                  # 读取文件行
5      if (line ! = ''):                       # 如果 line 不等于空,则文件没有结束
6          print(line)                         # 打印行内容
7      else:
8          break                               # 强制退出循环
9  file.close()                                # 关闭文件
   >>>[唐] 王之涣…                             # 程序输出(略,输出有空行)
```

程序说明：程序第 5 行，语句 if (line!＝'')为判断第 4 行 readline()读到的内容是否为空，行内容为空意味着文件结束。如果语句 readline()读到一个空行，也会判断为文件结束吗？事实上空行并不会返回空值，因为空行的末尾至少还有一个换行符(\n)。所以，即使文件中包含空行，读入行的内容也不为空，这说明语句 if (line!＝'')判断是正确的。

5.1.4 文件写入数据

1. 覆盖写入文件

Python 提供了两个文件写入函数，语法如下。

```
1  文件句柄.write('单字符串')                  # 语法 1:向文件写入一个字符串或字节流
2  文件句柄.writelines('行字符串')             # 语法 2:将多个字符串元素写入文件
```

函数 write()是将字符串写入一个打开的文件。注意，字符串也可以是二进制数据；函数 write()不会在字符串结尾添加换行符(\n)。

【例 5-16】 用函数 write()将字符串内容写入名为“杜甫诗歌 1.txt”的文件。

```
1  str1 = '百年已过半,秋至转饥寒。\n 为问彭州牧,何时救急难? \n'    # 符号\n 为换行符
2  file = open('d:\\test\\05\\杜甫诗歌 1.txt', 'w')                 # 以写模式打开文件
3  file.write(str1)                                                 # 字符串内容写入文件
4  file.close()                                                     # 关闭文件
5  print('字符串写入成功。')
   >>>字符串写入成功。                                             # 程序输出
```

程序说明：

程序第 1 行，由于函数 write()不会在字符串结尾自动添加换行符(\n)，因此字符串中必须根据需要人为加入换行符。

程序第 2 行，如果这个文件已经存在，那么源文件内容将会被新内容覆盖。

【例 5-17】 用函数 writelines()将内容写入名为“寄征衣 out.txt”的文件。

```
1  s = ['欲寄君衣君不还,', '不寄君衣君又寒。', '寄与不寄间,',          # 定义列表
2       '妾身千万难。\n ——[元] 姚燧《寄征衣》']                     # 写模式打开文件
```

```
3  file = open('d:\\test\\05\\寄征衣 out.txt', 'w')                    # 列表写入文件
4  file.writelines(s)                                                  # 关闭文件
5  file.close()
6  print('列表写入成功。')
   >>>列表写入成功。                                                   # 程序输出
```

2. 追加写入文件

【例 5-18】 将字符串内容追加写入“杜甫诗歌 1.txt”文件的结尾。

```
1  file = open('d:\\test\\05\\杜甫诗歌 1.txt', 'a + ')      # 以追加模式打开已存在的文件
2  file.write('——杜甫《因崔五侍御寄高彭州一绝》\n')        # 将字符串内容追加写入文件末尾
3  file.close()                                             # 关闭文件
4  print('追加写入成功。')
   >>>追加写入成功。                                        # 程序输出
```

3. 二进制文件的读写

【例 5-19】 读取“图片.png”文件,并且写入“图片复制.png”文件。

```
1  file_read = open('d:\\test\\05\\图片.png', 'rb')          # 以二进制读方式打开文件
2  file_write = open('图片复制 out.png', 'wb')               # 以二进制写方式创建文件
3  file_write.write(file_read.read())                        # 读二进制文件并写二进制文件
4  file_write.close()                                        # 关闭写文件
5  file_read.close()                                         # 关闭读文件
   >>>                                                       # 程序输出
```

5.2 常用文件操作

5.2.1 文件的格式化

1. 结构化数据和非结构化数据

结构化数据也称为行数据,它有规定的数据类型、规定的存储长度、规范化的数据结构等要求,可以用数据库进行存储和管理。常见的结构化数据有数据库文件、Excel 文件、CSV 文件、部分结构规范的文本文件等。结构化数据非常适合程序进行处理。

非结构化数据主要有办公文档(文档编码不一)、数据结构不一的文本文件(如字符串、数值、日期格式不一致)、网页(各种 HTML 标签和控制符)、各类图片、音频、视频等数据。非结构化数据不适宜用关系数据库进行存储和管理,程序处理非结构化数据非常麻烦。数据清洗的主要工作是将非结构化数据转换为结构化数据,便于程序处理。

2. 英文字母大小写转换

【例 5-20】 将莎士比亚(Shakespeare)名言中的英语单词进行各种转换。

```
1  >>> char = 'The pyramid is built with stones pieces of.'     # 金字塔是用一块块石头堆砌而成
2  >>> print(char.upper())                                      # 所有字符转换为大写字母
   THE PYRAMID IS BUILT WITH STONES PIECES OF.
3  >>> print(char.lower())                                      # 所有字符转换为小写字母
   the pyramid is built with stones pieces of.
4  >>> print(char.capitalize())                                 # 语句第一个字母转换为大写字母
   The pyramid is built with stones pieces of.                  # 其余为小写
```

```
5  >>> print(char.title())                                  # 每个单词的第一个字母转换为大
   The Pyramid Is Built With Stones Pieces Of.             # 写,其余为小写
```

3. 文本格式对齐

【例 5-21】 如图 5-1 所示,文件“成绩 1.txt”中文本行中的空格参差不齐,试对文本行中的空格进行对齐处理(见图 5-2)。

```
01    贾宝玉   85.5
  02      林黛玉      90
03      薛宝钗       88.5
   04       袭人      65
```

图 5-1 成绩 1.txt

```
01  贾宝玉  85.5
02  林黛玉  90
03  薛宝钗  88.5
04  袭人    65
```

图 5-2 对齐后的文本格式

案例分析:图 5-1 中,行中空格数参差不齐,可以用函数 split()对字符串进行切分(参见 2.1.3 节),将每行切分为 3 个元素;然后用函数 format()对一行中的字符串进行格式化排列对齐(参见 3.1.3 节)。

```
1  with open('d:\\test\\05\\成绩 1.txt', 'r') as file:          # 用绝对路径打开文件
2      for s in file:
3          L = s.split()                                       # 用空格符对字符串进行切分
4          t = '{0:<4}{1:<5}{2:4}'.format(L[0], L[1], L[2])    # 字符串格式化对齐
5          print(str(t))
   >>> 01   贾宝玉    85.5 …                                  # 程序输出见图 5-2(略)
```

程序说明:程序第 4 行,参数{0:<4}为 0 号元素(序号),占 4 个字符;参数{1:<5}为 1 号元素(姓名),占 5 个字符;参数{2:4}为 2 号元素(成绩),占 4 个字符。格式化函数 format()中,L[0]为 0 号元素,L[1]为 1 号元素,L[2]为 2 号元素。

5.2.2 多个文件合并

文本处理时,往往需要将多个文件合并在一起,再进行数据处理。多个文件合并时,先创建一个新文件,其次读入文件 A,然后将文件 B 添加在文件 A 之后,文件 C 添加在文件 B 之后,以此类推。文件合并后,还需要对合并后的文件进行清理,如排列格式化、删除空格、删除空行、删除乱码等操作。

【例 5-22】 文件如图 5-3~图 5-5 所示,将“琴诗 1.txt”“琴诗 2.txt”“琴诗 3.txt”文件合并成一个文件。

```
[宋] 苏轼 《琴诗》
```

图 5-3 琴诗 1.txt

```
若言琴上有琴声,
放在匣中何不鸣?
```

图 5-4 琴诗 2.txt

```
若言声在指头上,
何不于君指上听?
```

图 5-5 琴诗 3.txt

案例分析:多个文本文件合并时,首先将所有文件合并成一个新文件,然后对新文件进行格式化处理,如删除多余的空行、对齐文本行、对文件乱码进行处理等。

(1) 合并文件。

```
1  file_list = ['d:\\test\\05\\琴诗 1.txt',          # 定义文件列表
2       'd:\\test\\05\\琴诗 2.txt',
```

```
3       'd:\\test\\05\\琴诗 3.txt']
4   file = open('d:\\test\\05\\琴诗 4.txt', 'w')        # 创建新的临时文件琴诗 4.txt
5   for data in file_list:                              # 外循环遍历读取源文件列表
6       for txt in open(data, 'r'):                     # 内循环读取某个文件中的每一行
7           file.write(txt + '\r')                      # 行内容写入文件,行尾加回车符(\r)
8   file.close()                                        # 关闭文件
    >>>                                                 # 程序输出
```

程序说明：用记事本程序查看新创建的“琴诗 4.txt”文件，会发现文件中出现了多余的空行，因此需要进行删除空行的处理。

(2) 删除文件“琴诗 4.txt”中的空行。

```
1   file = open('d:\\test\\05\\琴诗 4.txt', 'r')          # 以读方式打开临时文件
2   tup = list()                                        # 函数 list()将元组转换为列表
3   for line in file.readlines():                       # 循环读取文件中每一行
4       line = line.strip()                             # 删除每行头尾的空白字符
5       if not len(line) or line.startswith('#'):       # 判断是否为空行,或是以#开始的行
6           continue                                    # 空行跳过不处理,返回循环头部
7       tup.append(line)                                # 插入一个新行
8   open('琴诗 5.txt', 'w').write('%s' % '\n'.join(tup))  # 创建新文件,并将内容写入文件
9   file.close()                                        # 关闭文件
    >>>                                                 # 程序输出
```

程序说明：

程序第 4 行，函数 line.strip()用于删除指定字符，如果没有指定字符则删除空白字符(包括换行符'\n'、回车符'\r'、制表符'\t'、空格' ')。函数从原字符尾部开始寻找，找到指定字符就将其删除，直到遇到一个不是指定字符就停止删除。

程序第 5 行，函数 startswith('#')用于检查字符串是否以'#'子字符串开头，如果是则返回 True，否则返回 False。也可以用“if line.count('\n')==len(line)”判断是否为空行。

程序第 8 行，函数 write('%s'%'\n'.join(s))中，%s 为字符串占位符(参见 3.1.3 节)；%'\n'为换行占位符；函数 join(s)用于将字符串 s 连接成一个新字符串。

5.2.3 多个文件连接

文件连接与文件合并的相同之处是两个文件合并形成一个新文件，不同之处是文件合并是文件 B 连接在文件 A 之后，而文件连接是文件 A 与文件 B 的第 1 行连接在一起，文件 A 与文件 B 的第 2 行连接在一起，以此类推，最后形成一个新文件。

文件连接有以下要求：一是两个文件的行与行之间为一一对应的关系；二是两个文件的行数一致；三是两个文件的编码一致(如均为 UTF8 编码)。

【例 5-23】 “企业 A.txt”记录了企业职工工号和姓名(见图 5-6)；“企业 B.txt”记录了职工工号和工资(见图 5-7)。要求将两个文件按行连接，即企业 A 的第 1 行连接到企业 B 的第 1 行，其余以此类推；然后打印输出连接后的文件。

案例分析：首先创建“企业 A.txt”和“企业 B.txt”的文件句柄；其次创建一个新的输出文件“企业 AB_out.txt”；然后循环读入文件句柄 a 中的一行，与文件句柄 b 中的一行连接在一起；最后将连接好的新行写入新建文件即可。

100 Jason Smith（杰森·史密斯）
200 John Doe（约翰·多伊）
300 Sanjay Gupta（桑贾伊·古普塔）
400 Ashok Sharma（阿肖克·夏尔马）

图 5-6　企业 A.txt(工号,姓名)

100 $5000
200 $500
300 $3000
400 $1250

图 5-7　企业 B.txt(工号,工资)

```
1  file_a = open('d:\\test\\05\\企业 A.txt', 'r', encoding = 'utf8')      # 创建文件句柄
2  file_b = open('d:\\test\\05\\企业 B.txt', 'r', encoding = 'utf8')
3  new_file = open('d:\\test\\05\\企业 AB_out.txt', 'a', encoding =       # 创建新文件,追加模式
   'utf8')
4  for i in file_a:                                                      # 循环写入文件内容
5      line_c = i.strip() + '  ' + file_b.readline().strip() + '\n'      # 连接文件行(很重要)
6      new_file.write(line_c)                                            # 连接好的行写入新文件
7      print(line_c)                                                     # 打印连接后的新行
8  file_a.close()                                                        # 关闭文件
9  file_b.close()
10 new_file.close()
```

```
>>>                                                                      # 连接文件如下
100 Jason Smith(杰森·史密斯) 100  $5000
200 John Doe(约翰·多伊) 200  $500
300 Sanjay Gupta(桑贾伊·古普塔) 300  $3000
400 Ashok Sharma(阿肖克·夏尔马) 400  $1250
```

程序说明：程序第 5 行,变量 line_c 为连接后的新行；函数 i.strip()为删除文件 a 中一行字符串的前后空格；参数' '为元素之间添加 2 个空格；函数 file_b.readline().strip()为读取文件 b 中的一行,并删除前后空格；参数'\n'为在行尾添加换行符。

从程序输出结果可见,工号在同一行中有重复,虽然可以用集合函数将重复元素删除,但是集合的无序性会导致行中元素混乱,因此对重复元素需要另外编程处理。

【例 5-24】　文件内容如图 5-8～图 5-10 所示,将三个文件连接成一个新文件,要求将文件按行连接,即 test1.txt 第 1 行后面连接 test2.txt 第 1 行,其余以此类推。

01 杨天黄
02 何冷烟
03 李屏西

图 5-8　test1.txt

程序设计 70
程序设计 80
程序设计 85

图 5-9　test2.txt

计算科学导论 82
计算科学导论 88
计算科学导论 85

图 5-10　test3.txt

案例分析：以下解决方案比例 5-23 复杂,程序用了函数 zip(* files)进行解包。

```
1  import itertools as it                                                # 导入标准模块
2
3  file = 'd:\\test\\05\\test1.txt d:\\test\\05\\test2.txt d:\\test\\    # 定义文件变量
   05\\test3.txt'.split()
4  with open('d:\\test\\05\\test_out.txt', 'w', encoding = 'utf8') as out_file:   # 创建新文件
5      files = [open(fname, 'r', encoding = 'utf8') for fname in file]   # 读入文件
6      for text in it.zip_longest( * files):                             # 用函数 zip()解包
7          out_file.write('\t'.join(t.strip() if t else '' for t in text)   # 新行写入
   + '\n')
8  for f in files:
9      f.close()                                                         # 循环关闭文件
```

```
>>>                                                                      # 程序输出
```

程序说明:

程序第 5 行,读入 test1.txt、test2.txt、test3.txt 三个文件(均为 UTF8 编码),注意读取和写入文件时采用 UTF8 编码(encoding='utf8'),否则程序会异常退出。

程序第 6 行,函数 it.zip_longest(*files)为解包(参见 4.1.2 节)。

5.2.4 文件内容去重

网络上下载的文件(如文本小说、密码字典等)有时会存在大量的重复行、空行、空格等;多个文件合并时,合并后的文件中可能会含有相同的内容。因此,需要对文本进行清洗,消除重复行(去重)、空行、空格等。

对两个内容疑似相同的文件,可以通过函数 md5()计算两个文件的哈希值,通过哈希值比较,判断文件内容是否相同,然后删除重复文件。

在一个文本文件内,内容去重的方法很多:一是利用 set()集合函数删除重复数据;二是利用正则表达式删除重复数据;三是利用程序语句删除文件中指定行或指定内容;四是将文件 A 内容读入列表变量中,然后读入文件 B 的一行内容,如果这行内容没有出现在列表变量中,则将这行内容写入新文件 C 中。

【例 5-25】 如图 5-11 所示,对"成绩 2.txt"文件进行去重操作,要求:①删除空行;②删除重复行;③删除空格;④生成一个新文件(见图 5-12)。

学号, 姓名, 班级,古文, 诗词, 平均
1, 宝玉,01, 70, 85, 0
2, 黛玉, 01, 85, 90, 0

3,晴雯,02, 40, 65, 0
2, 黛玉, 01, 85, 90, 0

图 5-11 源文件"成绩 2.txt"

学号,姓名,班级,古文,诗词,平均
1,宝玉,01,70,85,0
2,黛玉,01,85,90,0
3,晴雯,02,40,65,0

图 5-12 去重后新文件"成绩 3.txt"

案例分析:首先创建一个新文件,读入源文件中的行,删除字符串前后的空格(行中间空格后面再处理),然后写入临时文件 1 中。再创建临时文件 2,读入临时文件 1 中的行,删除其中的重复行、空行、行中间的空格等,将数据写入临时文件 2 即可。

```
1  file1 = open('d:\\test\\05\\成绩 2.txt', 'r', encoding = 'utf8')   #【1.删除空行】
2  file_new = open('d:\\test\\05\\成绩 tmp1.txt', 'w')                #  创建临时文件 1
3  for line in file1.readlines():                                     #  循环读入每一行
4      data = line.strip()                                            #  删除本行字符头尾空格
5      if len(data) ! = 0:                                            #  如果此行长度不等于 0
6          file_new.write(data)                                       #  写入本行数据
7          file_new.write('\n')                                       #  写入换行符
8  file1.close()                                                      #  关闭源文件
9  file_new.close()                                                   #  关闭临时文件
10                                                                    #【2.删除重复行】
11 tmp_file = open('d:\\test\\05\\成绩 tmp2.txt', 'w')                #  创建临时文件 2
12 lst1 = []                                                          #  建立空列表 lst1
13 for line in open('d:\\test\\05\\成绩 tmp1.txt', 'r'):              #  循环读入临时文件 1 中字符
14     tmp = line.strip()                                             #  删除字符串首尾的空格
15     if tmp not in lst1:                                            #  判断是否为重复行
16         lst1.append(tmp)                                           #  在列表末尾添加新对象
17         tmp_file.write(line)                                       #  逐行写入临时文件
18 tmp_file.close()                                                   #  关闭临时文件
19                                                                    #【3.删除空格】
```

```
lst2 = []                                                    # 建立空列表 lst2
with open('d:\\test\\05\\成绩 tmp2.txt', 'r') as file2:       # 打开临时文件 2
    data = file2.read()                                      # 读临时文件 2
    s1 = data.split(' ')                                     # 通过空格对字符串进行切片
lines = s1
for i in lines:                                              # 循环替换
    if i != '\r':                                            # 是否不为回车符
        k = i.strip()                                        # 不是回车符时删除头尾空格
    if i != '\n':                                            # 是否不为换行符
        k = i.replace('\t', '')                              # 不是换行符时删除空格
        lst2.append(k)                                       # 在列表末尾添加新对象
                                                             #【4.写入新文件】
with open('d:\\test\\05\\成绩 3out.txt', 'w') as file3:       # 创建新文件
    for i in lst2:
        file3.write(i)                                       # 写入处理过的内容到'成绩 3.txt'
>>>                                                          # 程序输出(生成文件见图 5-12)
```

程序说明：

程序第 4 行，函数 line.strip()用于删除字符串头尾指定字符(默认为空格或换行符)，返回一个新字符串。注意，该函数只能删除开头或结尾的字符，不能删除中间的字符。

程序第 23 行，函数 data.split(' ')为指定分隔符(如空格符)对字符串进行切片，如果不指定分隔符，默认分隔符为换行符(\n)、回车符(\r)、制表符(\t)。

程序第 29 行，函数 i.replace('\t','')表示把旧字符串(此处为\t，即空格)替换成新字符串(此处新字符为''，即删除空格)。

5.2.5 案例：文件字符统计

【例 5-26】 统计“全唐诗.txt”文本中的字数和行数。

```
with open('d:\\test\\05\\全唐诗.txt', encoding = 'utf8') as file1:   # 打开当前目录下的文件
    word = file1.read()                                             # 读取全部字符
print('全唐诗总字数:', len(word))                                    # 打印全部字符数
with open('d:\\test\\05\\全唐诗.txt', encoding = 'utf8') as file2:   # 打开当前目录下的文件
    lines = file2.readlines()                                       # 按行读入文件
print('全唐诗总行数:', len(lines))                                   # 打印文件行数
```

```
>>>                                                               # 程序输出
全唐诗总字数：4646026
全唐诗总行数：387171
```

【例 5-27】 统计某个字符串中汉字、英文、空格、数字、标点个数。

案例分析：

(1) 用 string.ascii_letters 方法生成英文大小写字母，然后判断字符串中是否含有英文字母；用函数 isdigit()判断字符串中是否含有数字；用函数 isspace()判断字符串中是否含有空格；用函数 isalpha()判断字符串中是否有汉字；其余字符串为标点符号。

(2) 函数有多个返回值时，可以用命名元组的方法使用返回值。

```
import string                                                # 导入标准模块——字符串模块
from collections import namedtuple                           # 导入标准函数——命名元组

```

```
4  def str_count(s):                                          # 定义字符统计函数
5      en = dg = sp = zh = pu = 0                             # 变量初始化
6      for c in s:                                            # 循环统计各种字符数
7          if c in string.ascii_letters:                      # 生成 a～z、A～Z 字母并且判断
8              en += 1                                        # 英文字符统计
9          elif c.isdigit():                                  # 判断字符串中是否含有数字
10             dg += 1                                        # 数字统计
11         elif c.isspace():                                  # 判断字符串中是否含有空格
12             sp += 1                                        # 空格统计
13         elif c.isalpha():                                  # 判断是否有其他语言中的字母
14             zh += 1                                        # 中文字符统计
15         else:
16             pu += 1                                        # 标点符号统计
17     total = zh + en + sp + dg + pu                         # 统计所有字符
18     return namedtuple('Count', ['total','zh','en','space','digit','punc'])(total,zh,en,sp,dg,pu)
19                                                            # 返回值用命名元组(很重要)
20 s = '66The pyramid is built with stones pieces of. 金字塔是用一块块石头堆砌而成 88。'
21 count = str_count(s)                                      # 调用字符统计函数
22 print(f'共{count.total}个字符,其中{count.zh}个汉字,{count.en}\
23 个英文,{count.space}个空格,{count.digit}个数字,{count.punc}个标点符号')
   >>>共 63 个字符,其中 14 个汉字,35 个英文,8 个空格,4 个数字,2 个标点符号
```

程序说明:

程序第 7 行,函数 string.ascii_letters 生成所有 a～z、A～Z 的字符串。

程序第 18 行,函数 namedtuple()为命名元组,普通元组中的元素只能按索引号访问,不能为元组中的每个元素命名。命名元组可以构造一个带字段名的元组,命名元组有两个参数:参数'Count'是元组名;参数['total','zh','en','space','digit','punc']是每个元素的名称。参数(total,zh,en,sp,dg,pu)是函数中的变量名,它们与元素名称一一对应。

【例 5-28】 命名元组应用案例。

```
1  >>> from collections import namedtuple                    # 导入标准函数——命名元组
2  >>> user = namedtuple('user', ['name', 'lesson', 'score'])  # 定义命名元组,['字段名 1', …]
3  >>> s1 = user('宝玉', '古文', '60')                        # 实例化命名元组
4  >>> print(s1.name, s1.score)                              # 通过类属性 user 获取值
   宝玉 60
5  >>> print(s1._fields)                                     # 通过_fields 获取获字段名
   ('name', 'lesson', 'score')
```

5.3 文本编码处理

5.3.1 字符集的编码

字符集是各种文字和符号的总称,它包括文字、符号、图形、数字等。字符集种类繁多,每个字符集包含的字符个数不同,编码方法不同。如 ASCII(America Standard Code for Information Interchange,美国信息交换标准码)、GBK(国家标准扩展的汉语拼音)字符集、Unicode(统一码)字符集等。计算机要处理各种字符集的文字,就需要对字符集中每个字符

进行唯一性编码，以便计算机能够识别和存储各种文字。

1. Unicode 字符集和编码

(1) Unicode 字符集。Unicode 是一个信息领域的国际字符集标准。Unicode 字符集目前有 $2^{21}=2\,097\,152$ 个编码(理论值)。Unicode 为全球每种语言和符号中的每个字符都规定了一个唯一代码点和名称，如“汉”字的名称和码点是“U+6C49”(U 为名称，6C49 为码点)。Unicode 目前规定了 17 个语言符号平面(大约 110 万编码)，如 CJK(中日韩统一表意文字)平面收录了中文简体汉字、中文繁体汉字、日文假名、韩文谚文、越南喃字。**Unicode 字符集有多种编码**，如 UTF8(UTF8-1，UTF8-2、UTF8-3、UTF8-4)、UTF16、UTF32 等，其他大多数字符集(如 GB2312、ASCII 等)都只有一种编码。**Unicode 字符集中汉字码点按《康熙字典》的偏旁部首和笔画数排列**，编码排序与 GB(国家标准)字符集排序不同。

【例 5-29】 打印字符串的 Unicode 编码。Python 程序如下。

```
str = input('请输入一个字符串:')                    # 输入字符串
a = [0] * len(str)                                  # 计算字符串长度
i = 0                                               # 计数器初始化
for x in str:                                       # 循环计算编码
    a[i] = hex(ord(x))                              # 将字符 x 转换为 Unicode 编码
    i = i+1                                         # 计数器累加
result = list(a)                                    # 转换为列表
print('字符串的 Unicode 编码为:', result)           # 打印 Unicode 编码
```

```
>>>                                                 # 程序输出
请输入一个字符串:a 中国
字符串的 Unicode 编码为: ['0x61', '0x4e2d', '0x56fd']   # 输出字符串的十六进制编码
```

(2) UTF8 编码。UTF8 采用变长编码。UTF8-1 编码中，128 个 ASCII 字符只需要 1 字节；UTF8-2 编码中，带有变音符号的拉丁文、希腊文、西里尔字母、亚美尼亚语、希伯来文、阿拉伯文、叙利亚文等需要 2 字节；UTF8-3 编码中，汉字为 3 字节；UTF8-4 编码中，其他辅助平面符号为 4 字节。Python 3.x 程序采用 UTF8 编码，Linux 系统、因特网协议、浏览器等均采用 UTF8 编码。

(3) UTF16 编码。UTF16 采用 2 字节或 4 字节编码。Windows 内核采用 UTF16 编码，但支持 UTF16 与 UTF8 的自动转换。UTF16 编码有 Big Endian(大端字节序)和 Little Endian(小端字节序)之分，UTF8 编码没有字节序问题。

(4) UTF32 编码。UTF32 采用 4 字节编码，由于浪费存储空间，因此很少使用。

2. 中文字符集标准

(1) GB 2312 编码。GBK 2312 是最早的国家标准中文字符集，它收录了 6763 个常用汉字和符号。GBK 2312 采用定长 2 字节编码。

(2) GBK 编码。GBK 是 GB 2312 的扩展，加入了对繁体字的支持，兼容 GB 2312，也与 Unicode 编码兼容。GBK 使用 2 字节定长编码，共收录 21 003 个汉字。

(3) GB 18030 编码。GB 18030 与 GB 2312 和 GBK 兼容。GB 18030 共收录 70 244 个汉字，它采用变长多字节编码，每个汉字或符号由 1～4 字节组成。Windows 7/8/10 默认支持 GB 18030 编码。

(4) 繁体中文 Big5 编码。Big5 是港澳台地区繁体汉字编码。它对汉字采用 2 字节定

长编码,一共可表示 13 053 个中文繁体汉字。Big5 编码的汉字先按笔画再按部首进行排列。Big5 编码与 GB 系列编码互不兼容。

5.3.2 字符编码转换

1. 字符的编码和解码

Python 程序中定义的字符串默认为 UTF8 编码(即 Unicode 码),字符串解码函数为 decode(),字符串编码函数为 encode()。函数 encode()的作用是将字符串编码为字节码(bytes),函数 decode()的作用是将字节码解码成字符串,语法如下。

```
decode([解码标准], [errors = 'ignore'])
encode([编码标准], [errors = 'ignore'])
```

参数"解码标准"有 utf8(也可写为 utf-8、utf、UTF8、UTF-8)、gbk 等。

参数 errors='ignore'为忽略非法字符,或者设置为 errors='replace'(用?号取代非法字符);如果为 errors='strict'(默认),则表示遇到非法字符时抛出异常。

【例 5-30】 字符串的编码与解码。

```
>>> s1_utf8 = '汉字 hz'                          # 定义字符串,默认为 UTF8 编码
>>> s2_utf8 = s1_utf8.encode('utf8')             # 对字符串进行编码
>>> print(s2_utf8)                               # 打印字符串编码
b'\xe6\xb1\x89\xe5\xad\x97hz'                    # 字符串的 UTF8 编码
>>> s3_utf8 = s2_utf8.decode('utf8')             # 对字符串进行解码
>>> print(s3_utf8)                               # 打印解码后的字符串
汉字 hz
```

【例 5-31】 网址中的"%xx"字符编码转换。

```
>>> url = 'https://www.baidu.com/s? wd=code520 中国'      # 定义 URL
>>> from urllib.parse import quote                        # 导入标准函数——字符编码
>>> url_utf = quote(url, safe=';/?:@&=+$,', encoding='utf8')
                                                          # ';/?:@&=+$,'字符不编码
>>> print('url_utf 编码:%s' % url_utf)                    # 打印 UTF 编码的 URL
url_utf 编码:https://www.baidu.com/s? wd=code520%E4%B8%AD%E5%9B%BD
>>> from urllib import parse                              # 导入标准函数——编码转字符
>>> print(parse.unquote('https://www.baidu.com/s? wd=code520%E4%B8%AD%E5%9B%BD'))
https://www.baidu.com/s? wd=code520 中国
```

程序说明:

程序第 3 行,函数 quote()将 url 中的字符转换为"%xx"形式;参数 safe=';/?:@&=+$,'为特殊字符(;/?:@&=+$)不转换为"%xx"形式;字符串编码为 UTF8。

程序第 5 行,函数 urllib.parse.unquote()将 url 中的"%xx"序列解码为 Unicode 字符;url 必须是字符串;默认编码为 UTF8。

2. GBK 与 UTF8 编码的转换

【例 5-32】 字符串的 GBK 与 UTF8 编码的相互转换。

案例分析：当字符串为 GBK 编码，希望转换为 UTF8 编码时，可以采用函数 s.decode('gbk').encode('utf8')；当字符串为 UTF8 编码，希望转换为 GBK 编码时，可以采用函数 s.decode('utf8').encode('gbk')。

```
1  >>> s = '汉字 hz'                                          # 字符串默认为 UTF8 码
2  >>> s_gbk = s.encode('gbk')                                # 无须解码，直接编码为 GBK
3  >>> s_gbk
   b'\xba\xba\xd7\xd6hz'
4  >>> gbk_utf8 = s_gbk.decode('gbk', 'ignore').encode('utf8')  # 字符串的 GBK 编码
5  >>> gbk_utf8                                               # 解码 GBK，编码 UTF8
   b'\xe6\xb1\x89\xe5\xad\x97hz'
6  >>> s_utf8 = gbk_utf8.decode('utf8')                       # 字符串的 UTF8 编码
7  >>> s_utf8                                                 # 将 GBK 解码为 UTF8
   '汉字 hz'                                                  # 字符串的 UTF8 编码
```

程序说明：程序第 4 行，函数 s_gbk.decode('gbk','ignore').encode('utf8')中，参数'gbk'为读取 GBK 编码文件，参数'ignore'为忽略非法字符；解码后的字符串编码为 UTF8。

5.3.3 文件编码转换

1. Windows 文件编码形式

Windows 系统默认采用 GBK 编码，但是系统可以进行 GBK 与 Unicode 编码的处理。Windows 下部分软件可以自动识别文件的编码形式，如“记事本”软件；但是大部分软件没有自动识别文件编码功能，如“写字板”软件。

【例 5-33】 用“记事本”程序打开 test_utf8.txt 文件时，程序可以自动识别文件的编码形式，不会发生乱码(见图 5-13)；但是用“写字板”程序打开 test_utf8.txt 文件时，由于写字板程序不能自动识别文件编码，因此文件会显示出现乱码(见图 5-14)。

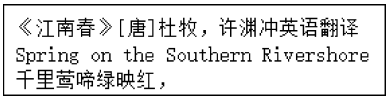


图 5-13 用“记事本”程序打开 test_utf8.txt 文件

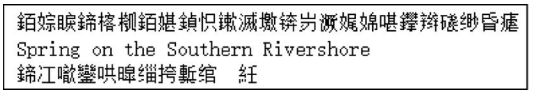


图 5-14 用“写字板”程序打开 test_utf8.txt 文件

记事本程序如何确定文件编码呢？记事本程序的处理方法是在文件最前面保存一个编码标签。程序检查到文件头部标签是 FF FE 时，说明文件采用 UTF-16LEB 编码(小端 Unicode 码)；如果文件头部标签是 FE FF，则文件采用 UTF-16BE 编码(大端 Unicode 码)；如果头部标签是 EF BB BF，则是 UTF8 编码(注意，头部标签不是 UTF8 标准的规定，这个头部标签也很容易导致 Python 程序出现异常)；没有以上 3 个头部标签的文件是 ANSI 编码，如果系统是简体中文 Windows，ANSI 编码就是 GBK 编码。

2. 检查文件的编码格式

不知道文件的编码格式时，文件打开时会无法设置编码格式，使得读取的内容出现乱码。可以用第三方软件包 chardet 检测文件编码格式，软件包 chardet 安装方法如下。

```
1  > pip install -i https://pypi.tuna.tsinghua.edu.cn/simple chardet        # 版本为 2.1.1
```

【例 5-34】 新建测试文件“江南春.txt”，保存时编码为 ANSI(即 GBK 编码)。

```
1  import chardet                                                    # 导入第三方包——编码
2  file = open('d:\\test\\05\\江南春.txt', 'rb')                     # 二进制读文件
3  data = file.read()                                                # 读入文件内容
4  print(chardet.detect(data))                                       # 打印文件编码
5  file.close()                                                      # 关闭文件
   >>>{'encoding': 'GB2312', 'confidence': 0.711, 'language': 'Chinese'}
                                                                     # GB 2312 可信度为 71 %
```

程序说明：程序第 2 行，**采用字节码读模式(rb)打开文件可以避免很多读错误**，指定编码格式反而可能报错。测试文件内容过少时，检测的语言可能会有偏差。

3. GBK 编码文件与 UTF8 文件的相互转换

【例 5-35】 将 d:\test\05\目录下的“江南春 gbk.txt”文件转换为 UTF8 编码，将文件重命名为“江南春 utf8.txt”。

案例分析：中文 Windows 下用函数 open()打开文件时，如果没有传递 encoding 参数，Python 会自动采用 GBK 编码打开文件，这容易引起读取文件时出错。解决方法是在 open()函数中传递编码参数 encoding='gbk'，或者 encoding='utf8'。

```
1  with open('d:\\test\\05\\江南春.txt', encoding = 'gbk', errors = 'ignore') as file:
                                                                     # 以 GBK 编码打开文件
2      while True:                                                   # while 永真循环
3          data = file.read()                                        # 读取文件内容
4          if data:
5              open('d:\\test\\05\\江南春 utf8.txt', 'a', encoding = 'utf8', errors = 'ignore').
   write(data)
6          else:
7              break                                                 # 强制退出循环
8  print('文件 GBK 转换为 UTF8 完成')
   >>>文件 GBK 转换为 UTF8 完成                                      # 程序输出
```

【例 5-36】 将当前目录下的“江南春 utf8.txt”文件转换为 GBK 编码，将文件重命名为“江南春 gbk.txt”。

```
1  def utf8_to_gbk(inFilePath, outFilePath):                         # 定义转换函数
2      with open(inFilePath, 'rb') as file1:                         # 创建文件(二进制码)
3          a = file1.read()                                          # 读取文件内容
4          b = a.decode('utf8', 'ignore')                            # 文件编码
5          with open(outFilePath, 'w', encoding = 'gbk') as file2:   # 创建新文件(GBK 编码)
6              file2.write(b)                                        # 写入文件内容
7          print('文件转换完成')
8
9  utf8_to_gbk('d:\\test\\05\\江南春 utf8.txt','江南春 gbk.txt')     # 调用转换函数
   >>>文件转换完成                                                   # 程序输出
```

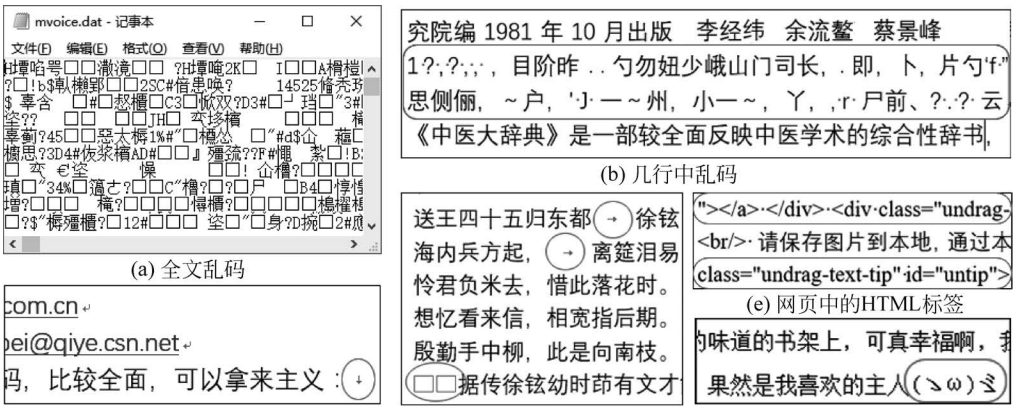
5.3.4 文本乱码处理

1. 文本文件中的乱码

乱码是用文本编辑器(如笔记本程序或 Word 程序)打开源文件时，文本中部分字符是

无法阅读或无法理解的一系列杂乱符号。在数据处理过程中，乱码问题让人头疼。

【例 5-37】 文本文件中的乱码现象如图 5-15 所示。如果文本内容全是乱码(见图 5-15(a))，说明文件是二进制编码，或者编辑器不支持这种文本编码。如果文本中只有某几行出现乱码(见图 5-15(b))，说明文本出现了编码错误，程序读取这种文本时非常容易出错，处理起来也非常麻烦。有些文本中含有不可见的控制符(见图 5-15(c)、图 5-15(d))，程序读取这些文本时也很容易出错。有些文件中含有 HTML(Hyper Text Mark Language，超文本标记语言)标识符(见图 5-15(e))和一些特殊符号(见图 5-15(f))，这都需要编程处理。



(a) 全文乱码　(b) 几行中乱码　(e) 网页中的HTML标签

(c) 软回车控制符　(d) Tab控制符+汉字空格符　(f) 特殊符号

图 5-15　文本中的乱码现象

2. 文本文件乱码原因

(1) 软件包原因。Python 程序经常会用到第三方软件包，一些国外软件包可能采用单字节编码(如 ISO 8859)，这些软件包打开双字节语言(如 GBK)文件时，如果不能正确识别文件分割符，就容易把一个汉字编码(2 字节)从中分割为两段，这会导致紧接在后面的整个一行全部都是乱码(见图 5-15(b))。

(2) 数据库原因。数据库字符编码与程序字符编码不一致，如数据库采用 GBK 编码，客户端程序采用 UTF8 编码，数据导出时就容易出现乱码(见图 5-15(a))。

(3) 数据错误。例如，大部分网页都采用了 JavaScript 脚本程序，而 JavaScript 语言默认编码为 ISO 8859。网络爬虫在爬取 JSP 网页数据后，如果存储为 UTF8 或 GBK 编码文件，以后 Python 程序读取这些文本时，就容易造成读写错误。

(4) 存储格式原因。例如，一些文本文件采用了字节保存模式，而 Python 程序采用文本读写操作时，就会出现读写错误。

3. 文本读错误的处理方法

(1) 开发环境设置。Python 3.x 默认使用 UTF8 编码，因此程序开发和文本文件存储时应尽量选用 UTF8 编码，而不是 GBK 编码(Windows 下为 ANSI 编码)。第一次使用 IDE(如 PyCharm)时，应将默认文本编辑器修改为 UTF8 编码。

(2) 对输入文本采用字节读的模式。

【例 5-38】 文本中夹杂有二进制字节码时，可对文本采用字节读(rb)模式。

```
1  file = open('d:\\test\\05\\三国演义.txt', 'rb')          # 采用二进制字节读模式
```

(3) 指明文本编码。

【例 5-39】 如果文本中有中文,可以在打开文件时指明文本编码模式。

```
1  file = open('d:\\test\\05\\三国演义.txt', 'rb', encoding = 'gbk')   # 说明文本编码模式(GBK
                                                                      # 编码)
```

(4) 用解码函数忽略非法字符。

【例 5-40】 利用函数 decode()忽略文本中的非法字符。

```
1  with open('d:\\test\\05\\春.txt', 'rb') as file:      # 打开文本文件(绝对路径)
2      data = file.read()                                # 读取文本文件
3      txt = data.decode('GB2312', errors = 'ignore')    # 参数 errors = 'ignore'为忽略非法字符
4      print(txt)
```

(5) 设置常用编码集。

【例 5-41】 文件“射雕英雄传.txt”编码不明,尝试读出该文件中的 20 个关键字。

案例分析:对某些不明编码的文本,可以在程序中设置多种编码集进行文本读,如设置 UTF8、GB 18030、GBK、GB 2312 编码集,必有一款编码适合读出文本。对文本文件的关键字提取可以采用“结巴分词”软件包(安装方法参见 7.3.2 节)。提高中文人名识别率的第三方软件包有 LTP(中规中矩)、LAC(提取量少,但是正确率高)等。

```
1  import jieba.analyse                                  # 导入第三方包——结巴分词
2
3  def read_from_file(directions):                       # 定义文本解码函数
4      decode_set = ['utf8', 'gb18030', 'gbk', 'gb2312', 'ISO-8859-2', 'Error']
                                                         # 定义编码集
5      for k in decode_set:                              # 编码集循环
6          try:
7              file = open(directions, 'r', encoding = k)
8              read_file = file.read()                   # 如果解码失败引发异常,就跳到 except
9              file.close()
10             break                                     # 如果文本打开成功,则跳出编码匹配
11         except:
12             if k == 'Error':                          # 如果出现异常就终止程序运行
13                 raise Exception('射雕英雄传.txt 文件无法解码!')      # 出错提示信息
14             continue
15     return read_file
16 file_data = str(read_from_file('d:\\test\\05\\射雕英雄传.txt'))     # 读取文本文件
17 tfidf = jieba.analyse.extract_tags(file_data, topK = 20)           # 提取关键字
18 print('关键字:', set(tfidf))
```

```
>>>                                                      # 程序输出
关键字:{'洪七公', '郭靖', '欧阳锋', '周伯通', '黄蓉', '武功', '爹爹', '说道', '郭靖', '黄蓉道',
'欧阳克', '柯镇恶', '裘千仞', '丘处机', '师父', '梅超风', '黄药师', '功夫', '完颜洪烈', '两人'}
```

程序说明:程序第 4 行,用多种编码读文本,常用编码放在前面,减少程序试错时间。由于 GB 18030 字符集较大,汉字覆盖较好,不容易出错,因此放在 GBK 编码前面。

习　题　5

5-1　简要说明“文件句柄”的功能和在程序中的作用。

5-2　简要说明函数 read()与函数 readlines()的相同和不同之处。

5-3　open()语句或 with open()语句都可以实现文件读写，它们有哪些差别？

5-4　简要说明 UTF8 编码的特点。

5-5　简要说明文本读错误的处理方法。

5-6　编程：统计“唐诗三百首.txt”文件中的字数和行数。

5-7　编程：数据文件 test.txt 内容如图 5-16 所示，把数据读入二维矩阵中。

5-8　编程：读取并打印“鸢尾花数据集.csv”。

5-9　编程：读取并打印“鸢尾花数据集.csv”数据集中第 2 行。

5-10　编程：将图 5-17 中数据保存到“成绩.csv”文件中。

1 2 2.5
3 4 4
7 8 7

图 5-16　数据集 1

学号	姓名	性别	班级	古文	诗词
100001	宝玉	男	1 班	85	70
100002	黛玉	女	2 班	88	85

图 5-17　数据集 2