第 7 章

文件与数据格式化

拓展阅读

学习目标

- 了解文件，能够说出计算机中文件的完整标识和文件分类
- 掌握文件的基础操作，能够实现文件的打开、关闭、读写、定位读写操作
- 掌握文件与目录管理，能够通过 os 模块中的函数管理计算机中的文件与目录
- 了解数据的分类，能够区分一维数据、二维数据和多维数据
- 熟悉数据的存储与读写方式，能够归纳一维数据和二维数据的存储方式，以及实现一维数据和二维数据的读写操作
- 熟悉多维数据的格式化，能够归纳 JSON 和 XML 数据的特点

程序中通常会使用变量保存运行时产生的临时数据，但这些临时数据在程序运行结束后消失。然而，有些程序中的数据需要持久地保存，例如游戏程序中的角色属性数据、装备数据、物品数据等。那么，有没有一种方法可以持久地保存这些数据呢？答案是肯定的，计算机可以使用文件持久地保存数据。本章将从计算机中文件的分类、基础操作、管理方式与数据维度等方面对文件与数据格式化的相关知识进行介绍。

7.1 文件概述

在计算机中，文件是广泛应用的数据存储方式。它们以硬盘等外部介质为载体，在计算机内部存储各种数据，如文本文档、图像、程序、音频、视频等。

类似于程序中使用的变量，计算机中的每个文件也有唯一确定的标识，它用于识别和引用文件。文件标识由路径、文件名主干和扩展名 3 个部分组成，关于它们的介绍如下。

（1）路径：路径是文件在计算机中的位置表示，用于指明文件所在的目录结构，以帮助操作系统定位文件。路径可以分为相对路径或绝对路径，其中相对路径是相对于当前工作目录或某个特定目录的路径，绝对路径从根目录开始完整地指定文件的位置。

（2）文件名主干：文件名主干通常是用于描述文件的内容、用途或其他相关信息的字符串，有助于用户识别和区分不同的文件。

（3）扩展名：扩展名是文件名中表示文件类型或格式的部分，用于帮助操作系统和应用程序辨识和处理不同种类的文件，通常紧跟在文件名主干之后。

Windows 操作系统中一个文件的完整标识如图 7-1 所示。

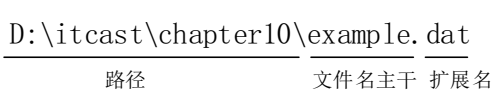

图7-1　Windows 操作系统中一个文件的完整标识

操作系统以文件为单位对数据进行管理，若想找到存放在外部介质上的数据，必须先按照文件标识找到指定的文件，再从文件中读取数据。根据图 7-1 所示的文件完整标识，用户可以在 Windows 操作系统中找到 D:\itcast\chapter10 路径下文件名主干为 example，扩展名为.dat 的文件。

根据数据的逻辑存储结构，人们将计算机中的文件分为文本文件和二进制文件，具体介绍如下。

1. 文本文件

文本文件是由字符组成的文件，包含可被人类读取和理解的文本信息。文本文件中的字符可以是字母、数字、标点符号和其他可输出字符。一般情况下，文本文件以纯文本形式存储，并使用 ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）、Unicode 或其他编码方式来表示其中的字符。常见的文本文件格式包括 TXT、CSV（Comma-Separated Values，逗号分隔值）、XML（Extensible Markup Language，可扩展标记语言）、HTML（Hypertext Markup Language，超文本标记语言）、JSON（JavaScript Object Notation，JavaScript 对象表示法）等。

由于文本文件以可读的文本形式存储，其内容在计算机以及不同操作系统和应用程序之间具有良好的可移植性和互操作性。文本文件可以被文本处理工具和编程语言读取、写入和处理，这使得文本文件格式成为广泛应用于各种场景的通用数据存储格式。

2. 二进制文件

二进制文件是由字节组成的文件。它以二进制形式存储数据，包含计算机能够直接解读和处理的信息。二进制文件可以包含各种类型的数据，如图像、音频、视频等，这类文件不能直接使用文字处理程序正常读写，必须使用相关程序才能正确获取文件信息。

在计算机中，文本文件和二进制文件这两种类型的文件是根据数据的逻辑存储结构而非物理存储结构进行划分的，由于计算机中的数据都是以二进制形式存储的，所以文本文件和二进制文件在物理层面上并没有区别。

多学一招：标准文件

Python 的 sys 模块中定义了 3 个标准文件，分别为 stdin（标准输入文件）、stdout（标准输出文件）和 stderr（标准错误文件）。标准输入文件对应输入设备，如键盘；标准输出文件和标准错误文件对应输出设备，如显示器。每个终端都有其对应的标准文件，这些文件在终端启动的同时打开。

在 Python 解释器中导入 sys 模块后便可对标准文件进行操作。以标准输出文件为例，向该文件中写入数据的示例代码如下：

```
import sys
file = sys.stdout
file.write("hello")
```

以上代码将标准输出文件赋给文件对象 file，又通过文件对象 file 调用内置方法 write()向标准输出文件中写入数据。程序执行后，字符串"hello"将被写入标准输出文件，即输出到终端。

7.2 文件的基础操作

文件的打开、关闭与读写是文件的基础操作，任何复杂的文件操作都离不开这些操作。本节将介绍这些基础操作。

7.2.1 文件的打开与关闭

当用户需要对文件进行读取或写入操作时，首先需要将文件打开，建立与文件的连接，然后在完成操作以后将文件关闭，断开与文件的连接，以便及时释放资源并确保数据的完整性。下面介绍如何在程序中打开和关闭文件。

1. 打开文件

Python 中的内置函数 open()用于打开文件并返回相应的文件对象，该函数的语法格式如下：

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

open()函数中的参数 file 用于指定文件名或文件路径；参数 encoding 用于指定文件的编码方式，默认值为 None，表示使用系统默认的编码方式，常见的编码方式包括'utf-8'、'gbk'等；参数 mode 用于设置文件的打开模式，基本的打开模式有 r、w、a，它们也可以与 b、+组合使用，以满足不同打开文件操作的需求。常用的文件打开模式如表 7-1 所示。

表 7-1 常用的文件打开模式

打开模式	名称	描述
r 或 rb	读取或二进制读取模式	以只读的方式打开文本文件或二进制文件，若文件不存在或无法找到，则将无法成功打开文件。r 为默认的打开模式
w 或 wb	写入或二进制写入模式	以只写的方式打开文本文件或二进制文件。若文件已经存在，则会先清空文件内容再写入新的文本数据或二进制数据；若文件不存在，则会创建一个新的文件
a 或 ab	追加或二进制追加模式	以只写的方式打开文本文件或二进制文件，并在文件末尾追加数据，而不会清空文件原有的数据。若文件不存在，则会创建一个新的文件
r+或 rb+	读写或二进制读写模式	以读写的方式打开文本文件或二进制文件，允许在文件中进行读取和写入操作。若文件不存在或无法找到，则将无法成功打开文件
w+或 wb+	读写或二进制读写模式	以读写的方式打开文本文件或二进制文件，允许在文件中进行读取和写入操作。若文件已经存在，则会先清空文件内容再进行读取和写入的操作；若文件不存在，则会创建一个新的文件
a+或 ab+	追加或二进制追加读写模式	以读写的方式打开文本文件或二进制文件，允许在文件中进行读取和写入操作，写入的数据会直接追加到已有数据的后面，而不会清空文件原有的数据。若文件不存在，则会创建一个新的文件

若 open()函数成功打开文件，则会返回一个文件对象。打开文件的示例代码如下：

```
file1 = open('E:\\a.txt')          # 以只读的方式打开 E 盘的文本文件 a.txt
file2 = open('b.txt', 'w')         # 以只写的方式打开当前目录的文本文件 b.txt
file3 = open('c.txt', 'w+')        # 以读写的方式打开文本文件 c.txt
file4 = open('d.txt', 'wb+')       # 以二进制读写的方式打开文本文件 d.txt
```

以只读的方式打开文件时，若待打开的文件不存在，则文件会打开失败，导致程序出现报错信息。假设以上代码要打开的文件 a.txt 不存在，执行上述代码后出现的报错信息如下：

```
Traceback (most recent call last):
  File "E:\FastPrograms3\Chapter07\code02.py", line 1, in <module>
    file1 = open('E:\\a.txt')     # 以只读的方式打开 E 盘的文本文件 a.txt
            ^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'E:\\a.txt'
```

由上述报错信息“[Errno 2] No such file or directory: 'E:\\a.txt'”可知，程序在 E 盘下没有找到文本文件 a.txt。

2. 关闭文件

Python 中可通过 close()方法关闭文件，也可以使用 with 语句实现文件的自动关闭。下面分别介绍这 2 种关闭文件的方式。

（1）close()方法

close()方法是为文件对象提供的方法，用于关闭已经打开的文件。该方法无须传入任何参数，直接调用即可。例如，调用 close()方法关闭之前打开的文件对象 file，具体代码如下：

```
file.close()
```

（2）with 语句

当操作文件时，如果打开文件与关闭文件之间的操作较多，我们很容易遗漏关闭文件操作，为此 Python 引入 with 语句预定义清理操作，实现文件的自动关闭。with 语句的语法格式如下：

```
with open(文件路径, 打开模式) as 变量名:
    代码段
```

上述语法格式中，as 及其后面的变量名都是可选的。当程序执行 with 语句时，首先会调用 open()函数打开指定路径下的文件，并将该函数返回的文件对象赋给变量，然后执行 with 语句内的代码段，并在代码段执行完毕后自动调用 close()方法关闭已打开的文件。使用 with 语句可以简化文件操作的代码，并确保在任何情况下都会正确关闭文件。

使用 with 语句打开与关闭文件 a.txt 的示例代码如下：

```
with open('a.txt', 'w+') as file:
    print('我是 with 语句')
```

以上示例代码中的变量 file 用于接收 with 语句打开的文件对象。程序中无须调用 close()方法关闭文件，文件对象使用完毕后，with 语句会自动关闭文件。

思考：为什么要及时关闭文件？

虽然程序执行完毕后，系统会自动关闭由程序打开的文件，但计算机中可打开的文件数量是有限的，每打开一个文件，可打开的文件数量就减 1。打开的文件占用系统资源，若打开的文件过多，会降低系统性能。当文件以缓冲方式打开时，磁盘文件与内存间的读写并非即时的，若程序因异常关闭，可能产生数据丢失。因此，编写代码时应在程序中主动关闭不再使用的文件。

7.2.2　文件的读写

Python 提供了一系列读写文件的方法，包括读取文件的 read()、readline()、readlines()方法和写文件的 write()、writelines()方法，下面将介绍如何使用这些方法实现文件的读写。

1. 读取文件

（1）read()方法

read()方法用于从指定文件中读取一定数量的字节或字符，并将读取到的数据返回。read()方法的语法格式如下：

```
read(size=-1)
```

以上语法格式中的参数 size 用于指定从文件中读取的数据的字节数或字符数，默认值为 -1，表示一次性从文件中读取所有数据。

下面以文件 test.txt 为例，演示使用 read()方法读取该文件中指定数量的字符，test.txt 文件中的内容具体如下：

```
合抱之木，生于毫末；
九层之台，起于累土；
千里之行，始于足下。
老子《老子》
```

使用 read()方法读取 test.txt 文件的示例代码如下：

```
with open('test.txt') as file:
    result = file.read(3)              # 读取 3 个字符
    print(result)
    result = file.read(3)              # 继续读取 3 个字符
    print(result)
    result = file.read()               # 继续读取剩余的全部数据
    print(result)
```

运行代码，结果如下所示：

```
合抱之
木，生
于毫末；
九层之台，起于累土；
千里之行，始于足下。
老子《老子》
```

（2）readline()方法

readline()方法用于从指定文件中读取一行数据，并保留该行数据末尾的换行符\n。readline()方法的语法格式如下：

```
readline()
```

下面以文件 test.txt 为例，演示使用 readline()方法读取该文件中的数据，示例代码如下：

```
with open('test.txt') as file:
    result = file.readline()  # 第 1 次读取，读取第 1 行数据
    print(result)
    result = file.readline()  # 第 2 次读取，读取第 2 行数据
    print(result)
    result = file.readline()  # 第 3 次读取，读取第 3 行数据
    print(result)
    result = file.readline()  # 第 4 次读取，读取第 4 行数据
    print(result)
```

运行代码，结果如下所示：

```
合抱之木，生于毫末；

```

```
九层之台，起于累土；

千里之行，始于足下。

老子《老子》
```

（3）readlines()方法

readlines()方法用于一次性读取文件中的所有数据，若读取成功返回一个列表，文件中的每一行对应列表中的一个元素。readlines()方法的语法格式如下：

```
readlines(hint=-1)
```

以上语法格式中，参数 hint 的单位为字节，它用于控制要读取的行数，如果一行中数据的总大小超出了 hint 指定的字节，readlines()方法将不会继续读取文件，返回的列表中的行数可能会少于指定的行数。

下面以文件 test.txt 文件为例，演示使用 readlines()方法读取该文件的所有数据，示例代码如下：

```
with open('test.txt') as file:
    print(file.readlines())     # 使用 readlines() 方法读取所有的数据
```

运行代码，结果如下所示：

```
['合抱之木，生于毫末；\n', '九层之台，起于累土；\n', '千里之行，始于足下。\n', '老子《老子》']
```

以上介绍的 3 个方法中，read()和 readlines()方法都可一次性读取文件中的全部数据，但这两个方法可能会引起内存问题。因为计算机的内存是有限的，若文件较大，read()和 readlines()的一次性读取便会耗尽系统内存，这显然是不可取的。为了保证读取安全，通常会采用 read(size)方法，通过多次调用 read()方法每次读取 size 个字节或字符。

2. 写入文件

（1）write()方法

write()方法可以将字符串写入文件，其语法格式如下：

```
write(data)
```

以上语法格式中的参数 data 表示要写入文件的数据，若数据写入成功，write()方法会返回本次写入文件的数据的字节数或字符数。

使用 write()方法向 write_file.txt 文件中写入数据，示例代码如下：

```
string = "Nothing in the world is difficult " \
         "for one who sets his mind to it."
with open('write_file.txt', mode='w') as file:
    size = file.write(string)               # 向文件中写入数据
    print(size)                             # 输出字符数
```

运行代码，结果如下所示：

```
66
```

此时打开 write_file.txt 文件，可以在该文件中看到“Nothing in the world is difficult for one who sets his mind to it.”，具体如图 7-2 所示。

（2）writelines()方法

writelines()方法用于将字符串或字符串列表写入文件，其语法格式如下：

```
writelines(lines)
```

图7-2 使用write()方法写入文件的字符串

以上语法格式中的参数 lines 表示要写入文件中的数据，该参数可以是一个字符串或字符串列表。需要说明的是，若写入文件的数据在文件中需要换行，应显式插入换行符。

使用 writelines()方法向文件 write_file.txt 中写入数据，示例代码如下：

```
string_list = ["Interest is the best teacher!\n",
               "Interest is the best teacher!\n",
               "Interest is the best teacher!"]
with open('write_file.txt', mode='w') as file:
    file.writelines(string_list)
```

运行代码，控制台没有输出信息。此时打开 write_file.txt 文件，可在该文件中看到写入的多个字符串，具体如图 7-3 所示。

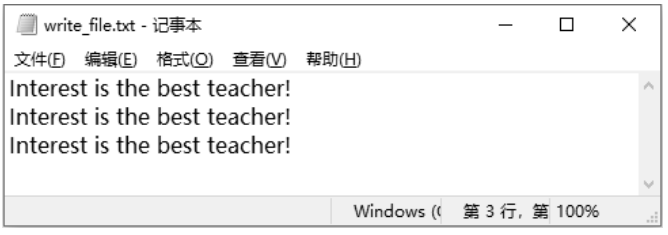

图7-3 使用writelines()方法写入文件的字符串

多学一招：字符与编码方式

文本文件支持多种编码方式，不同编码方式下字符数与字节数的对应关系不同，常见的编码方式以及字符数与字节数的对应关系如表 7-2 所示。

表 7-2 常见的编码方式及字符数与字节数的对应关系

编码方式	语言	字符数	字节数
ASCII	中文	1	
	英文	1	1
UTF-8	中文	1	3 或 4
	英文	1	1
UTF-16	中文	1	2
	英文	1	2
UTF-32	中文	1	4
	英文	1	4
GBK	中文	1	2
	英文	1	1

7.2.3 文件的定位读写

7.2.2 小节使用 read()方法读取了文件 test.txt，结合代码与运行结果分析后可以发现，程序第 1 次读取到了 3 个字符，第 2 次从第 4 个字符“木”开始读取。之所以出现上述情况，

是因为在文件的一次打开与关闭之间进行的读写操作是连续的，程序总是从上次读写的位置继续往后进行读写操作。

实际上，每个文件对象都有一个称为“读写指针”的属性，该属性用于记录当前的读写位置，默认值为 0，即文件读写位置默认在文件开头位置。Python 中提供了一些获取与修改文件读写位置的方法，以实现文件的定位读写，下面分别对这些方法进行详细讲解。

1. tell()方法

tell()方法用于获取文件当前的读写位置，会返回一个表示文件读写位置的整数，这个整数以字节为单位。例如，当程序从文件开头位置读取了 3 个英文字符后，通过 tell()方法获取的文件读写位置为 3，说明文件读写位置当前为从文件开头偏移 3 个字节的位置。

下面以文件 write_file.txt 为例，演示通过 tell()方法获取当前的文件读写位置，具体代码如下：

```
with open('write_file.txt') as file:
    read_location = file.tell()    # 获取文件读写位置
    print(read_location)
    file.read(5)                   # 通过 read()方法读取数据，移动文件读写位置
    read_location = file.tell()    # 再次获取文件读写位置
    print(read_location)
```

运行代码，结果如下所示：

```
0
5
```

由上述代码的运行结果可知，程序第 1 次获取到的文件读写位置为 0，说明文件读写位置当前为文件开头位置；第 2 次获取到的文件读写位置为 5，说明文件读写位置当前为从文件开头偏移 5 个字节的位置。

2. seek()方法

为了满足程序在读写文件时从指定位置开始的需求，Python 中提供了 seek()方法。该方法用于移动文件读写位置到指定的位置，以便用户能够从任意位置开始读写文件。seek()方法的语法格式如下：

```
seek(offset, whence=os.SEEK_SET, /)
```

seek()方法中的参数 offset 表示偏移量，即文件读写位置需要移动的字节数，它的取值可以为正数、负数或 0，其中正数表示相对于指定位置向文件末尾移动的字节数，负数表示相对于指定位置向文件开头移动的字节数，0 表示不移动，即保持位置不变；参数 whence 用于指定偏移量的参考位置，该参数支持的取值及其代表的含义分别如下。

- os.SEEK_SET 或 0：默认值，表示从文件开头位置开始偏移。
- os.SEEK_CUR 或 1：表示从当前位置开始偏移。
- os.SEEK_END 或 2：表示从文件末尾位置开始偏移。

seek()方法调用成功后会返回当前的文件读写位置。

下面以文件 write_file.txt 为例，演示通过 seek()移动文件读写位置，具体代码如下：

```
with open('write_file.txt') as file:
    read_location = file.seek(5, 0)        # 从文件开头移动 5 个字节
    print(read_location)                   # 输出当前的文件读写位置
    result = file.read(3)                  # 读取 3 个字符
    print(result)
```

以上代码中首先通过文件对象 file 调用 seek()方法移动文件读写位置。seek()方法的第 1 个参数为 5，第 2 个参数为 0，说明从文件开头位置开始移动 5 个字节，然后通过文件对象调用 read()方法继续读取 3 个字符，也就是说，读取文件中的第 6～8 个字符。

运行代码，结果如下所示：

```
5
est
```

需要注意的是，当使用 seek()方法操作文本文件时，只能在 from 参数值为 0 的情况下从文件开头位置开始移动文件读写位置，而不能在 from 参数值为 1 或 2 的情况下进行相对移动，这样会导致程序出现错误。示例代码如下：

```
with open('write_file.txt') as file:           # 打开文本文件
    read_location = file.seek(5, 0)            # 从文件开头移动 5 个字节
    print(read_location)
    read_location = file.seek(-3, 2)           # 从文件末尾向前移动 3 个字节
    print(read_location)
```

运行代码，结果如下所示：

```
5
Traceback (most recent call last):
  File "E:\FastPrograms3\Chapter07\code02.py", line 4, in <module>
    read_location = file.seek(-3, 2)              # 从文件末尾向前移动 3 个字节
                    ^^^^^^^^^^^^^^^^
io.UnsupportedOperation: can't do nonzero end-relative seeks
```

若要相对于当前的文件读写位置或文件末尾移动文件读写位置，则需要以二进制读取的方式打开文件，示例代码如下：

```
with open('write_file.txt', 'rb') as file:     # 以二进制读取的方式打开文件
    read_location = file.seek(5, 0)            # 从文件开头移动 5 个字节
    print(read_location)
    read_location = file.seek(-3, 2)           # 从文件末尾向前移动 3 个字节
    print(read_location)
```

运行代码，结果如下所示：

```
5
88
```

7.3 文件与目录管理

对于用户而言，文件和目录以不同的形式展现，但对于计算机而言，目录是文件属性信息的集合，其本质上也是一种文件。除了 Python 的内置方法外，os 模块中也定义了与文件操作相关的函数，利用这些函数可以实现删除文件、重命名目录或文件、创建或删除目录、获取当前目录、更改默认目录与获取文件名列表等操作。本节将对 os 模块中的常用函数进行讲解。

1. 删除文件——remove()函数

os 模块中的 remove()函数用于删除指定路径下的文件，若待删除的文件不存在，则会导致程序报错。remove()函数的语法格式如下：

```
remove(path)
```

以上语法格式中，参数 path 表示待删除文件的路径，其取值可以为绝对路径或相对路径，

若该参数的值为一个文件的名称，则表明删除当前目录下指定名称的文件。

例如，使用 remove()函数删除文件 a.txt，具体代码如下：

```
import os
os.remove('a.txt')
```

2. 重命名目录或文件——rename()函数

使用 os 模块中的 rename()函数可以重命名目录或文件，该函数要求目标目录或文件必须存在，若不存在会导致程序报错。rename()函数的语法格式如下：

```
rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
```

以上语法格式中各参数的含义如下。

- src：表示旧的目录名或文件名。
- dst：表示新的目录名或文件名。
- src_dir_fd：表示旧目录或文件对应的文件描述符，文件描述符是操作系统用来标识已经打开的文件的一种方式，如果指定了此参数，则会将 src 视为 src_dir_fd 指定的目录的相对路径。例如，重命名/home/user/documents/file.txt，src_dir_fd 参数指定的文件描述符为/home/user，src 参数的值可以简化为 documents/file.txt，而不需要使用绝对路径。
- dst_dir_fd：表示新目录或文件对应的文件描述符，如果指定了此参数，则会将 dst 视为 dst_dir_fd 指定的目录的相对路径。

例如，使用 rename()函数重命名文件 write_file.txt，具体代码如下：

```
os.rename('a.txt', 'test.txt')
```

3. 创建或删除目录——mkdir()或 rmdir()函数

os 模块中的 mkdir()函数用于创建目录，rmdir()函数用于删除目录，这两个函数都必须传入一个目录名。例如，使用 mkdir()函数创建名称为 dir 的目录，具体代码如下：

```
os.mkdir('dir')
```

运行以上代码后，当前路径下增加了一个名称为 dir 的目录。需要注意的是，待创建的目录不能与已有目录重名，否则将创建失败。

例如，使用 rmdir()函数删除名称为 dir 的目录，具体代码如下：

```
os.rmdir('dir')
```

4. 获取当前目录——getcwd()函数

当前目录即 Python 当前的工作路径。os 模块中的 getcwd()函数用于获取当前目录，调用该函数可获取当前目录的绝对路径。示例代码如下：

```
result = os.getcwd()
print(result)
```

5. 更改默认目录——chdir()函数

os 模块中的 chdir()函数用于更改默认目录。若对文件或文件夹进行操作时传入的是文件名而非路径，Python 解释器会从默认目录中查找指定文件，或在默认目录下创建新的文件。若没有特别设置，当前目录即为默认目录。

使用 chdir()函数更改默认目录为“E:\”，再次使用 getcwd()函数获取当前目录，示例代码如下：

```
os.chdir('E:\\')                    # 更改默认目录
result = os.getcwd()                # 获取当前目录
print(result)
```

运行代码，结果如下所示：

```
'E:\'
```

6. 获取文件名列表——listdir()函数

实际应用中常常需要先获取指定目录下的所有文件，再对目标文件进行相应操作。os 模块中提供了 listdir()函数，使用该函数可方便快捷地获取指定目录下所有文件的文件名列表。例如，获取当前目录下的所有文件的文件名列表，具体代码如下：

```
dirs = os.listdir('./')       # 获取文件名列表
print(dirs)
```

7.4 实训案例

7.4.1 信息安全策略——文件备份

信息安全策略——文件备份

当今是“信息时代”，信息在当今社会占据的地位不言而喻，信息安全问题更是人们当前重视的问题之一。人们考虑从传输和存储两个方面来保障信息的安全，备份是在存储方面保障信息安全的有效方式。

本案例要求编写程序，根据用户输入的要备份的文件或目录，实现一个具有备份文件与目录功能的备份工具。备份工具的具体要求如下。

（1）如果用户输入的要备份的文件或目录不存在，则会创建该文件或目录后再进行备份操作；如果用户输入的要备份的文件或目录存在，则会直接进行备份操作。

（2）如果用户输入的要备份的目录是一个文件夹，则遍历该文件夹下的所有文件并逐个备份。

（3）如果用户输入的要备份的文件存在，则会直接对该文件进行备份操作，否则提示用户要备份的文件不存在，并退出程序。

（4）备份操作是将源文件内容逐行复制到新文件中，保存在备份目录下，并以原文件名命名新文件。

7.4.2 用户账户管理

用户账户管理

如今，许多网站要求访问者在访问网站内容之前进行登录；若访问者没有该网站的账户，则需要先进行注册。注册后，网站的服务器会保存账户信息，以便访问者下次访问时可根据保存的信息验证身份。为了保障账户安全，访问者应定期修改密码；若决定不再访问此网站，可以选择注销账户。

作为网络环境中的一员，我们需要加强自身信息安全意识，提高密码的复杂度和使用频率，定期更换和更新密码，从而保障在网络上的信息安全。

本案例要求编写一个用户账户管理程序，使用文件存储用户的账户信息。用户账户管理程序的具体要求如下。

（1）开始使用时用户账户管理程序提供一个功能菜单，便于提示用户程序具有哪些功能。功能菜单如下所示。

```
欢迎使用用户账户管理程序!
====================
1.用户注册
```

```
2.用户登录
3.用户注销
4.修改密码
5.退出
====================
```

（2）用户账户管理程序总共有 5 个功能，用户可以根据需要输入相应的编号选择相应的功能。如果用户选择退出功能，则会直接退出程序，否则可以重复选择相应的功能。其他功能的说明如下。

- 用户注册：接收用户输入的用户名，判断用户输入的用户名是否已经存在，如果存在，则输出用户已注册的提示信息；如果不存在，则继续接收用户输入的密码，并将用户名和密码保存到文件中。
- 用户登录：接收用户输入的用户名和密码，依次判断用户名与密码是否正确，如果都正确则输出登录成功的提示信息，否则输出用户名或密码不正确的提示信息。
- 用户注销：接收用户输入的要注销的用户名和密码，依次判断用户名与密码是否正确，如果都正确则将对应的用户信息从文件中删除，输出注销成功的提示信息，否则输出用户不存在或者密码不正确的提示信息。
- 修改密码：接收用户输入的用户名和旧密码，依次判断用户名和密码是否正确，如果都正确则要求用户输入新密码，并将修改后的用户信息保存到文件中；如果用户名或者密码不正确，则输出用户名或密码不正确的提示信息。

7.5　数据维度与数据格式化

从广义上讲，维度是描述事物之间联系的概念的数量，根据概念的数量，事物可以被分为不同的维度。例如，长度是与线有联系的概念，因此线是一维事物；长度和宽度是与长方形面积有联系的概念，因此面积为二维事物；长度、宽度和高度是与长方体体积有联系的概念，因此体积为三维事物。

在计算机领域中，数据根据与其关联的参数数量分为不同的维度，本节将对数据维度和与不同维度数据格式化的相关知识进行讲解。

7.5.1　基于维度的数据分类

根据组织数据时与数据有联系的参数的数量，数据可分为一维数据、二维数据和多维数据。

1. 一维数据

一维数据是一组具有对等关系的线性数据，类似于数学中的集合和一维数组。Python 中可以使用一维列表、一维元组和一维集合表示一维数据。一维数据的各个元素可以通过英文逗号、空格等符号进行分隔。例如，我国 2023 年公布的“新一线”城市便是一组一维数据，通过英文逗号分隔此组数据，具体如下所示：

```
成都,重庆,杭州,西安,武汉,苏州,郑州,南京,天津,长沙,东莞,宁波,昆明,合肥,青岛
```

2. 二维数据

二维数据是具有两个关联参数的数据集合，类似于数学中的矩阵和二维数组。Python 中可以使用二维列表、二维元组等表示二维数据。表格是日常生活中比较常见的二维数据组织

形式，因此二维数据也常被称为表格数据。高三一班考试的成绩表就是一种表格数据，如图 7-4 所示。

姓名	语文	数学	英语	理综
小红	124	137	145	260
小明	116	143	139	263
小白	120	130	148	255
小兰	115	145	131	240
小刚	123	108	121	235
小华	132	100	112	210

图7-4　高三一班考试的成绩表

3. 多维数据

多维数据利用键值对等简单的二元关系展示数据的复杂结构，Python 中可以使用字典表示多维数据。多维数据在网络应用中非常常见，计算机中常见的多维数据格式有 HTML、JSON 等。例如，使用 JSON 格式描述高三一班考试的成绩，具体如下所示：

```
"高三一班考试成绩":[
                    {"姓名": "小红",
                     "语文": "124",
                     "数学": "137",
                     "英语": "145",
                     "理综": "260" };
                    {"姓名": "小明",
                     "语文": "116",
                     "数学": "143",
                     "英语": "139",
                     "理综": "263" };
                     …
                ]
```

7.5.2　一维数据和二维数据的存储与读写

程序中与数据相关的操作分为数据的存储与读写，本小节将对如何存储与读写不同维度的数据进行讲解。

1. 数据存储

数据通常存储在文件中。为了方便后续的读写操作，数据通常需要按照约定的组织方式进行存储。

一维数据呈线性排列，一般用特殊字符分隔，具体示例如下。

- 使用空格分隔：成都 杭州 重庆 武汉 苏州 西安 天津。
- 使用英文逗号分隔：成都,杭州,重庆,武汉,苏州,西安,天津。
- 使用&分隔：成都&杭州&重庆&武汉&苏州&西安&天津。

由此可见，在存储一维数据时可使用不同的特殊字符（即分隔符）分隔数据元素，但有以下几点需要注意。

（1）同一文件或同组文件一般使用同一种分隔符分隔。

（2）用于分隔数据的分隔符不应出现在数据中。

（3）分隔符为英文符号，一般不使用中文符号作为分隔符。

二维数据可视为多个一维数据的集合，当二维数据只有一个元素时，这个二维数据被视为一维数据。国际上通用的一维数据和二维数据的存储格式为 CSV。CSV 文件以纯文本形式存储表格数据，文件的每一行对应表格中的一条数据记录，每条记录由一个或多个字段组成，

字段之间使用英文逗号分隔。因为字段之间可能使用除英文逗号外的其他分隔符，所以 CSV 也称为字符分隔值。具体示例如下：

```
姓名,语文,数学,英语,理综
小红,124,137,145,260
小明,116,143,139,263
小白,120,130,148,255
小兰,115,145,131,240
小刚,123,108,121,235
小华,132,100,112,210
```

CSV 广泛应用于不同体系结构下网络应用程序之间表格信息的交换，它本身并无明确格式标准，具体标准一般由传输双方协商决定。

2. 数据读取

计算机中采用 CSV 格式存储数据的文件的扩展名一般为.csv，此种文件在 Windows 平台上，可通过办公软件 Excel 或记事本打开。例如，将以上示例中 CSV 格式的数据存储到当前路径下的 score.csv 文件中，之后通过 Python 程序读取该文件中的数据并以列表形式输出，具体代码如下：

```
csv_file = open('score.csv')
lines = []
for line in csv_file:
  line = line.replace('\n', '')
  lines.append(line.split(','))
print(lines)
csv_file.close()
```

以上程序首先调用 open()函数打开 score.csv 文件，然后通过 for 循环对打开的文件对象进行迭代，在循环中逐条获取文件中的记录，去掉每条记录末尾的换行符，利用分隔符“,”分隔记录，将记录存储到列表 lines 中。

运行程序，结果如下所示：

```
[['姓名', '语文', '数学', '英语', '理综'], ['小红', '124', '137', '145', '260'], ['小明', '116', '143', '139', '263'], ['小白', '120', '130', '148', '255'], ['小兰', '115', '145', '131', '240'], ['小刚', '123', '108', '121', '235'], ['小华', '132', '100', '112', '210']]
```

3. 数据写入

将一维数据、二维数据写入文件中，即按照数据的组织形式，在文件中添加新的数据。在保存学生成绩的文件 score.csv 中写入每名学生的总分，具体代码如下：

```
csv_file = open('score.csv')
file_new = open('count.csv', 'w+')
lines = []
for line in csv_file:
    line = line.replace('\n', '')
    lines.append(line.split(','))
# 添加表头字段
lines[0].append('总分')
# 添加总分
for i in range(len(lines) - 1):
    idx = i + 1
    sun_score = 0
```

```
        for j in range(len(lines[idx])) :
            if lines[idx][j].isnumeric():
                sun_score += int(lines[idx][j])
        lines[idx].append(str(sun_score))
    for line in lines:
        print(line)
        file_new.write(','.join(line) + '\n')
    csv_file.close()
    file_new.close()
```

执行以上代码后，当前目录中会新建写有学生每科成绩与总分的文件 count.csv，使用 Excel 打开该文件，该文件的内容如图 7-5 所示。

由图 7-5 可知，内容比之前多了“总分”一列，说明程序成功将总分写入 count.csv 文件。

姓名	语文	数学	英语	理综	总分
小红	124	137	145	260	666
小明	116	143	139	263	661
小白	120	130	148	255	653
小兰	115	146	131	240	631
小刚	123	108	121	235	587
小华	132	100	112	210	554

图7-5 count.csv文件的内容

7.5.3 多维数据的格式化

二维数据可以看作一维数据的集合，三维数据可以看作二维数据的集合，以此类推。然而，通过层层嵌套的方式组织数据会导致多维数据的表示变得非常复杂。为了直观地表示多维数据，并方便地组织和操作多维数据，通常采用键值对的形式对三维及以上的多维数据进行格式化。

在网络应用中，经常传递的数据大多是多维数据。其中，JSON 是一种常见的轻量级数据交换格式，JSON 格式的数据本质上是一种格式化的字符串，既易于被人们阅读和编写，也容易被机器解析和生成。JSON 语法是 JavaScript 语法的子集，因此它以 JavaScript 对象的形式来表示数据。

JSON 格式的数据遵循以下语法规则。

（1）数据存储在键值对中，例如"姓名":"张华"。

（2）数据的字段由英文逗号分隔，例如"姓名":"张华","语文":"116"。

（3）一个大括号保存一个 JSON 对象，例如{"姓名":"张华","语文":"116"}。

（4）一个中括号保存一个数组，例如[{"姓名":"张华","语文":"116"}]。

假设目前有高三一班考试成绩的 JSON 数据，具体如下所示：

```
"高三一班考试成绩":[
                    {"姓名": "小红",
                     "语文": "124",
                     "数学": "137",
                     "英语": "145",
                     "理综": "260" };
                    {"姓名": "小明",
                     "语文": "116",
                     "数学": "143",
                     "英语": "139",
                     "理综": "263" };
                     …
                ]
```

以上数据包含多个键值对，其中一个键为“高三一班考试成绩”，之后跟着以英文冒号

分隔的值。此值本身是一个数组，该数组中存储了多名学生的成绩，通过中括号组织，其中的元素通过英文分号“;”分隔；作为数组元素的学生成绩亦为键值对，通过英文逗号“,”分隔。

除了 JSON 外，网络平台也会使用 XML、HTML 等格式组织多维数据，XML 和 HTML 格式通过标签组织数据。例如，将学生成绩以 XML 格式存储，具体如下所示：

```
<高三一班考试成绩>
 <姓名>小红</姓名><语文>124</语文><数学>137<数学/><英语>145<英语/><理综>260<理综/>
 <姓名>小明</姓名><语文>116</语文><数学>143<数学/><英语>139<英语/><理综>263<理综/>
 …
</高三一班考试成绩>
```

对比 JSON 格式与 XML 格式可知，采用 JSON 格式组织的多维数据更为直观，且数据的键只需存储一次，在网络中进行数据交换时耗费的流量更少。

7.6　本章小结

本章主要介绍了文件与数据格式化的相关知识，包括文件概述、文件的基础操作、文件与目录管理、数据维度与数据格式化。通过学习本章的内容，读者能够对计算机中的文件有基本的认识，熟练进行文件操作、文件和目录管理，并掌握常见的不同维度数据的格式化操作。

7.7　习题

一、填空题

1. 打开文件并对文件进行读写后，应调用_____方法关闭文件以释放资源。
2. seek()方法用于移动文件读写位置，该方法的_____参数表示要偏移的字节数。
3. _____是一组具有对等关系的线性数据，类似于数学中的集合和一维数组。
4. os 模块中的_____ 函数用于创建目录。
5. _____路径从根目录开始完整地指定文件的位置。

二、判断题

1. Python 中打开文件时默认使用的模式为读取。(　　)
2. 以读写的方式打开一个文件，若文件已存在，文件内容会被清空。(　　)
3. 使用 write()方法写入文件时，数据会追加到文件的末尾。(　　)
4. Python 中使用 os 模块中的函数可实现与目录相关的操作。(　　)
5. 使用 read()方法只能一次性读取文件中的所有数据。(　　)

三、选择题

1. 若打开一个已有文件后在文件末尾添加数据，则应该使用下列哪种模式打开文件？(　　)

A. r　　B. w　　C. a　　D. w+

2. 通过 open()方法打开不存在的文件时，使用以下哪种模式会使程序报错？(　　)

A. r　　B. w　　C. a　　D. w+

3. 假设 file 是文本文件对象，下列哪个选项用于读取 file 的一行内容？（　　）

A. file.read()　　B. file.read(200)　　C. file.readline()　　D. file.readlines()

4. 下列函数或方法中，用于向文件中写入数据的是（　　）。

A. open()　　B. write()　　C. close()　　D. read()

5. 下列函数或方法中，用于获取当前目录的是（　　）。

A. open()　　B. write()　　C. getcwd()　　D. read()

6. 下列代码要打开的文件应该在（　　）。

```
f = open('itheima.txt', 'w')
```

A. C 盘根目录　　B. D 盘根目录　　C. Python 安装目录　　D. 程序所在目录

7. 假设文本文件 abc.txt 中的内容如下：

```
abcdef
```

编写程序读取上述文本文件的数据，具体如下：

```
file = open('abc.txt', 'r')
data = file.readline()
data_list = list(data)
print(data_list)
```

以上程序执行后结果为（　　）。

A. ['abcdef']　　B. ['abcdef\n']

C. ['a', 'b', 'c', 'd', 'e', 'f']　　D. ['a', 'b', 'c', 'd', 'e', 'f', '\n']

四、简答题

1. 请简述文本文件和二进制文件的区别。

2. 请简述读取文件的 3 种方法 read()、readline()、readlines()的区别。

五、编程题

1. 假设现有一个文件 file.txt，该文件中的内容具体如下：

```
# 这是一首李白写的诗
金樽清酒斗十千，玉盘珍羞直万钱。
停杯投箸不能食，拔剑四顾心茫然。
欲渡黄河冰塞川，将登太行雪满山。
闲来垂钓碧溪上，忽复乘舟梦日边。
行路难！行路难！多歧路，今安在？
长风破浪会有时，直挂云帆济沧海。
```

编写程序，从 file.txt 文件中逐行读取内容，但不读取以#开头的行。

2. 编写程序，实现备份文件 file.txt 内容的功能，新文件的名称为 file[复件].txt。

3. 假设现有一个文件 num.txt，该文件中的内容具体如下：

```
10 6 55
20
80
30
50
5
8
```

编写程序，从 num.txt 文件中逐行读取内容，之后继续将每行的内容按照空格分割得到所有的数字，并将所有的数字排序后输出。