9. 文件与IO

如前所述，程序中数据的输入可通过 input() 函数经由键盘读入，但当数据量较大时，用户工作量将会很大，而且每次运行时都需要重复输入工作。此外，程序的运行结果可通过 print() 函数直接输出到屏幕上，但程序每次运行完毕后，程序运行结果就会被“清空”。因此，如果将数据保存在文件中，每次程序运行时对文件进行读取，并且将程序运行结果保存在另一个文件中，这样可大大减少工作量，也可长期保存数据。

本章首先介绍文件的概念；然后介绍文件的**打开**、**关闭**、读写和定位等基本操作；接着介绍文件与文件夹的相关操作，如文件与文件夹的重命名、移动、删除等。

9.1. 文件的打开与关闭

文件指存储在外部介质（如磁盘等）上有序的数据集合，这个数据集有一个名称，称为文件名。常见的文件有记事本文件、日志文件、各种配置文件、数据库文件、图像文件、音频和视频文件等。按数据的组织形式不同，可以将文件分为**文本文件**和**二进制文件**两大类。

文本文件一般由单一特定编码的字符组成，如 **UTF-8** 编码，内容容易统一展示和阅读。大部分文本文件都可以通过文本编辑软件或文字处理软件创建、修改和阅读。由于文本文件存在编码，因此。它可以被看作是存储在磁盘上的长字符串。例如，在 Windows 平台中，扩展名为 txt，log，ini 的文件都属于文本文件，可以使用记事本进行编辑。

二进制文件直接由比特 0 和比特1 组成，没有统一字符编码，文件内部数据的组织格式与文件用途有关。例如，图形图像文件、音频视频文件、可执行文件、各种数据库文件、各类 Office 文件等都属于二进制文件。二进制文件把信息以字节流形式存储，无法用记事本或其他普通文字处理软件直接修改和阅读，需要使用正确的软件进行解码或反序列化之后才能正确地读取、显示、修改或执行。

9.1.1. 文件打开

Python 对文本文件和二进制文件采用统一的操作步骤：

（1）打开文件，或者新建文件;

（2）读/写文件；

（3）关闭文件。

- 操作系统中的文件默认处于**存储状态**，首先需要将其打开，使得当前程序有权操作这个文件，如果打开不存在的文件可以创建文件。
- 打开后的文件处于**占用状态**，此时，另一个进程不能操作该文件。
- 接下来，可以通过一组方法**读取**文件的内容或向文件**写入**内容。
- 操作完成后需要**关闭文件**，关闭操作将释放对文件的控制，使文件恢复存储状态，此时，另一个进程才能操作该文件。

Python 内置了文件对象，通过 `open()` 函数即可按照指定**模式**打开指定文件，并创建文件对象，其语法格式如下:

```
文件对象名 = open(文件名字, [打开方式])
```

完整语法

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

参数说明:

- file: 必需，文件路径（相对或者绝对路径）。
- mode: 可选，文件打开模式
- buffering: 设置缓冲
- encoding: 一般使用utf8
- errors: 报错级别
- newline: 区分换行符
- closefd: 传入的 file 参数类型
- opener: 设置自定义开启器，开启器的返回值必须是一个打开的文件描述符。

其中，文件名指定了被打开的文件名称，如果使用 open() 函数打开文件时，只带一个文件名参数，那么是以**只读方式**打开文件，而且当文件不存在时会抛出异常。例如，打开一个名为 “1.txt” 的文件，代码如下：

```
FileNotFoundError: [Error 2] No sush file or directory:
'1.txt'
```

> 提示
>
> 文件名亦可包含文件路径，写文件路径时注意斜杆问题。若路径和文件名为 `c:\myfile`，应写成 `c:\\myfile`。
>
> 例如：file = open('c:\myfile')

如果想要编辑文件，就需要在打开文件时指明文件的打开方式。Python 中文件的打开方式有多种，具体表示方式及含义如下表所示。

模式	描述
t	文本模式 (默认)。
x	写模式，新建一个文件，如果该文件已存在则会报错。
b	二进制模式。
+	打开一个文件进行更新(**可读可写**)。
U	通用换行模式（**Python 3 不支持**）。
r	以**只读**方式打开文件。文件的指针将会放在文件的开头。这是**默认模式**。
rb	以二进制格式打开一个文件用于**只读**。文件指针将会放在文件的开头。这是默认模式。一般用于非文本文件如图片等。
r+	打开一个文件用于**读写**。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于**读写**。文件指针将会放在文件的开头。一般用于非文本文件如图片等。
w	打开一个文件只用于**写入**。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于**写入**。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
w+	打开一个文件用于**读写**。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于**读写**。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
a	打开一个文件用于**追加**。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。

模式	描述
ab	以二进制格式打开一个文件用于**追加**。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于**读写**。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于**追加**。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

- （1）用只读方式“r”打开文件时，该文件必须已经存在，否则出错，且只能进行读取操作，打开时文件位置指针在文件的开头。
- （2）用只写方式“w”打开文件时，如文件不存在，则以指定的文件名新建文件。若打开的文件已经存在，则原文件内容消失，重新写入内容且只能进行写操作。
- （3）用追加方式“a”打开文件时，如果文件已存在，文件位置指针在文件的结尾，也就是说，新的内容被写入到已有内容之后，如果文件不存在，则创建新文件进行写入。
- （4）“r+”“w+”“a+”都是既可读亦可写，区别在于“r+”与“r”一样，文件必须已经存在：“w+”和“w”一样，如文件不存在则新建文件，写后可以读：“a+”则是打开文件后可以在文件末尾增加新数据亦可以读取文件。
- （5）打开方式带上“b”表示是以二进制文件格式进行操作。

9.1.2. 文件关闭

在 Python 中，虽然文件会在程序退出后自动关闭，但是考虑到**数据的安全性**，在每次使用完文件后，都需要使用 close() 方法关闭文件，其语法格式如下：

```
文件对象名.close()
```

例如，以只写方式打开一个名为 “test.txt” 的文件，然后关闭文件，代码如下：

```
file = open('test.txt, 'w')      #以只写方式打开一个名为“test.txt”的文件
file.close()                     #关闭文件
```

9.1.3. 上下文管理语句 with

Python 中的 with 语句用于对资源进行访问，保证不管处理过程中是否发现错误或者异常，都会执行规定的 `__exit__(清理)` 操作，释放被访问的资源，常用于文件操作、数据库连接、网络通信连接、多线程与多进程同步时的锁对象管理等场所。

语法格式如下：

```
with context_expression[as target(s)]:
    with-body
```

其中，context_expression 为表达式，target(s) 为对象名。

例如，用于文件内容读写时，with 语句的用法如下：

```
with open(文件名[,打开方式]) as 文件对象名
    # 通过文件对象名读写文件内容语句
```

> 提示
>
> 在实际开发中，读写文件应优先考虑使用上下文管理语句 with。

9.2. 文件的读写

当文件被打开后，根据打开的方式不同可以对文件进行相应的读写操作。当文件以文本方式打开时，按照字符串方式进行读写，采用当前计算机使用的编码或指定编码；当文件以二进制文件方式打开时，按照字节流方式进行读写。

> 提示
>
> 对于所有读操作，文件都必须以读或者读写方式打开；对于所有写操作，文件都必须以写、读写或者追加方式打开；如希望重建文件，可采用只写或读写方式打开文件；如希望保留原文件内容，从后面开始增加新内容，可采用追加或追加式读写方式打开文件。

9.2.1. 写文件

Python 提供了两个与文件写入有关的方法：**write()** 和 **writelines()** 方法。

1. write() 方法

   write() 方法用于像文件中写入只当字符串，其语法格式如下：

   ```
   文件对象名.write(str)
   ```

   其中，str 为要写入文件的字符串。

   示例：

   向 “testfile.txt” 文件中写入下列数据：

   - Interface options
   - Generic options
   - Miscellaneous options
   - Options you should't use

【问题分析】首先以只写方式打开文件（当文件不存在时会创建文件）；然后向文件中写入数据，这里需要注意的是 write() 方法不会自动在字符串的末尾添加换行符，因此，当输入多行时，需要在 write() 语句中包含换行符；最后关闭文件。

```
file = open('testfile.txt', 'w')
# 向文件中输入字符串
file.write('Interface options\n')
file.write('Generic options\n')
file.write('Miscellaneous options\n')
file.write('Options you should't use\n')
file.close()
```

> 提示
>
> 如果打开文件时，文件打开方式带 “b”，那么写入文件内容时，str（参数）要用 encode 方法转化为字节流形式，否则报错。
>
> 例如，`file.write('Interface options'.encode())`

2. writelines() 方法

   writelines() 用于向文件中写入一序列的字符串。这一序列字符串可以是由**迭代对象产生**的，如一个字符串列表，语法：

   ```
   文件对象名.writelines(sequence)
   ```

   其中，sequence 为要写入文件的字符串序列。

   使用 writelines() 方法向已有的 “testfile.txt” 文件中追加如下数据。

   - Environment
   - variable

   【问题分析】要向文件中追加数据，需要用追加方式 “a” 打开文件。使用 writelines() 方法写入数据时，同样不会自动在列表后面增加换行符，需要手动加入。

```
ls = ['Environment\n', 'variables']
with open('testfile.txt', 'a') as file:
    file.writelines(ls)
```

9.2.2. 读文件

3个常用的文件内容读取方法：

- read()
- readline()
- readlines()

1. read() 方法

   read() 方法用于从文件中读取**指定的字节数**，如果未给定参数为负，则读取整个文件内容，语法如下：

   ```
   文件对象名.read([size])
   ```

   其中，size 为从文件读取的字节数，该方法返回**从文件中读取的字符串。**

   使用 read() 方法读取 testfile.txt 文件

   ```
   with open('testfile.txt', 'r') as file:
       line = file.read(10)
       print(line)
       print('*' * 30)
       context = file.read()
       print(context)
   ```

   【程序说明】打开文件时，打开文件时，文件位置指针在文件的开头，运行 “line = fileread(10)” 语句，就会从文件的开头读取 10 个字符，因此输出 “Interface ”（后面包含了一个空格）当执行 “content = file.read()” 语句时，文件的指针已经在第 10 个字符处，因此，执行该语句时，读取了文件中剩余的所有内容（不包括前 10 个字符)。

2. readline() 方法

   readline() 方法用于从文件中**读取整行**，包括 “\n” 字符。如果指定了一个非负数的参数，则表示读入指定大小的字符串，其语法格式如下：

```
文件对象名.readline([size])
```

   其中，size 为从文件中读取的字节数。

   【实例】使用 readline() 方法读取 “testfile.txt” 文件。

```
with open('testfile.txt', 'r') as file:
    line = file.readline()
    print(line)
    print('*' * 30)
    line = file.readline(10)
    print(line)
```

3. readlines() 方法

   readlines() 方法用于读取所有行（**直到结束符 EOF**）并返回**列表**，列表中每个元素为文件中的一行数据，其语法格式如下：

```
文件对象名.readlines()
```

   【实例】使用 readlines() 方法读取 “testfile.txt” 文件。

```
with open('testfile.txt', 'r') as file:
    content = file.realines()
print('*'*60)
for temp in content:
    print(temp)
```

   【程序说明】从运行结果可以看出，使用 readlines() 方法读取文件后返回的值为列表。遍历列表时，由于每个元素后面有一个“\n”，而 print 语句也会加上一个换行符，因此会多出来空白行。

   上述代码存在一个缺点：当读取文件非常大时，一次性将内容读取到列表中会占用很多内存，影响程序执行速度。可以将文件本身作为一个行序列进行读取，遍历文件的所有行可以直接用下面的代码实现。

```
with open('testfile.txt', 'r') as file:
    for line in file:
        print(line)
```

【实例】将文件 “testfile.txt” 中的内容复制到另一个文件 “copy.txt” 中。

```
with open('testfile.txt', 'r') as file1, open('copy.txt', 'w') as file2:
    file2.write(file1.read())
```

9.3. 文件的定位

所谓文件位置指针，是系统设置的用来指向文件当前读写位置的指针，不需要用户定义，但会随着文件的读写操作而移动，因此，在对文件进行操作前，需先清楚当前文件位置指针的位置，在不同位置进行操作时，也需将文件位置指针定位在相应位置。

9.3.1. 获取当前读写位置

在读写文件的过程中，如果想知道当前文件位置指针的位置，可以通过调用 **tell()** 方法来获取。**tell()** 方法返回文件的当前位置，即文件位置指针当前位置。其语法格式如下:

```
文件对象名.tell()
```

【实例】使用 tell() 方法获取文件当前的读写位置。

```
with open('testfile.txt', 'r') as file:
    line = file.read(8)
    print(line)
    p = file.tell()
    print('当前位置：', p) # 8
    line = file.read(4)
    print(line)
    p = file.tell()
    print('当前位置：', p) # 12
```

9.3.2. 定位到某个位置

如果在读写文件的过程中，需要从指定的位置开始读写操作，就可以使用 seek() 方法实现。seek() 方法用于移动文件位置指针到指定位置，其语法格式如下：

```
文件对象名.seek(offset[,whence])
```

其中，参数介绍如下：

- （1）**offset**：表示偏移量，也就是需要**偏移的字节数**。
- （2）**whence**：可选，默认值为 0，表示起始点，即位移量的参考点，有三种取值，0代表“文件开始位置”，1 代表“当前位置”，2 代表“文件末尾位置”。下面通过实例进行介绍。

【实例】创建名为 “seek.txt” 的文件，输入 “This is a test!” 并存放进文件中，读取单词 “test” 并输出到终端。

【问题分析】首先创建并打开指定的文件，文件名由终端输入。然后在文件中写入 “This is a test!” 字符串，接着利用 seek() 方法将文件位置指针指向 “test” 单词的字母 “t” 处，最后读取单词 “test” 并输出到终端。

```
filename = input('请输入新建的文件名：')
with open(filename, 'w+') as file:
    file.write('This is a test!')
    file.seek(10)
    con = file.read(4)
    print(con)
```

【程序说明】将文件位置指针移到从文件起始位置开始的第 10 个字符处，这里省略了 `whence` 参数，该语句也可以写成“`file.seek(10, 0)`”。

> 提示
>
> 以**文本文件格式**打开文件时，seek() 方法中的 whence 参数取值只能是 0，即只允许从文件开始位置计算偏移量。若想从**当前位置**或文件**末尾位置**计算偏移量，**需要使用“b”模式（二进制格式）打开文件**。

【示例】读取“seek.txt”文件中倒数第 2 个字符。

```
with open('seek.txt', 'rb') as file:
    # 将文件位置指针定位到倒数第二个字符处
    file.seek(-2, 2)
    con = file.read(1)
    print(con)
```

【程序说明】上述代码中，如果以文本文件格式打开文件，即第一条语句中的 `open()` 方法改为 `open('seek.txt', 'r')`，运行程序将会提示错误信息。

9.4. 文件与文件夹操作

前面主要介绍了对文件内容进行操作的方法，接下来介绍文件级别的操作和文件夹操作，例如文件重命名、文件删除、创建文件夹、删除文件夹等。

9.4.1. os 模块

Python 标准库的 os 模块除了提供使用操作系统功能和访问文件系统的简便方法之外，还提供了大量文件级操作的方法，下面列举几个常用的方法。

方法	功能说明
os.rename(src, dst)	重命名（从 src 到 dst）文件或者目录，可以实现文件的移动，若目标文件已存在则抛出异常
os.remove(path)	删除路径为 path 的文件，如果 path 是一个文件夹，则抛出异常
os.mkdir(path[,mode])	创建目录，要求上级目录必须存在，参数 mode 为创建目录的权限，默认创建的目录权限为可读可写可执行
os.getcwd()	返回当前工作目录
os.chdir(path)	将 path 设为当前工作目录
os.listdir(path)	返回 path 目录下的文件和目录列表
os.rmdir(path)	删除 path 指定的空目录，如果目录非空，则抛出异常
os.removedirs(path)	删除多级目录，目录中不能有文件

【示例】

```
import os
os.getcwd()

os.mkdir('ostest')
os.chdir('ostest')
os.mkdir('mktest')

f = open('1.txt', 'w')
f.close()

os.rename('1.txt', '2.txt')
os.listdir('ostest')

os.rmdir('mktest')
```

```
os.listdir('ostest')

os.remove('2.txt')
os.listdir('ostest')
```

【示例】批量修改文件名，在 “ostest” 目录下的文件名前加上编号，修改前和修改后的文件名。

- 修改前
  - 计科
  - 软件
  - 通信
  - 信工
- 修改后
  - 1信工
  - 2计科
  - 3软件
  - 4通信

【问题分析】首先将当前工作目录切换到 “ostest” 目录下，最后利用 for 循环遍历列表的同时调用 rename() 方法重命名每个文件名。

```
import os
dir_list = os.listdir('ostest')
i = 1
os.chdir('ostest')
for name in dir_list:
    print(name)
    new_name = str(i) + name
    i += 1
    print(new_name)
    os.rename(name, new_name)
```

9.4.2. os.path 模块

os.path 模块提供了大量用于路径判断、文件属性获取的方法，列举常用方法。

方法	功能说明
os.path.abspath(path)	返回给定路径的绝对路径
os.path.split(path)	将 path 分割成目录和文件名二元组返回
os.path.splitext(path)	分离文件名与扩展名；默认返回（fname，fextension）元组，可做分片操作
os.path.exists(path)	如果 path 存在，返回 True；如果 path 不存在，返回 False
os.path.getsize(path)	返回 path 文件的大小（字节）
os.path.getatime(path)	得到指定文件最后一次的访问时间
os.path.getctime(path)	得到指定文件的创建时间
os.path.getmtime(path)	得到指定文件最后一次的修改时间

【示例】

```
import os.path
p1 = os.path.abspath('ostest')
print(p1)

os.path.split('ostest')

os.path.splitext('test.py')

os.path.exists('ostest')

os.path.getsize('test.py')
```

> 知识库

> getatime、getctime、getmtime 分别用于获取文件的最近访问时间、创建时间和修改时间。不过返回值是浮点型秒数，可用 time 模块的 gmtime 或 localtime 方法换算。例如：
>
> ```
> import os.path
> import time
> temp = time.localtime(os.path.getatime('testfile.txt'))
> print('testfile.txt被访问时间是：
> {}'.format(time.strftime('%d %b %Y %H:%M:%S', temp)))
> ```

9.4.3. shutil 模块

shutil 模块也是提供了大量方法支持文件和文件夹操作，常用方法如下。

方法	功能说明
shutil.copy(src, dst)	复制文件内容以及**权限**
shuhil.copy2(src, dst)	复制文件内容以及**文件的所有状态信息**
shutil.copyfile(src, dst)	复制文件，**不复制文件属性**，如果目标文件已存在则直接覆盖
shutil.copytree(src, dst)	递归复制文件内容及**状态信息**
shutil.rmtree(path)	递归删除文件夹
shutil.move(src, dst)	移动文件或递归移动文件夹，也可给文件和文件夹重命名

【示例】

```
import shutil
shuti.copy('testfile.txt', 'copytest.txt')

shutil.copytree('ostest','copytest')

shutil.rmtree('copytest')
```