第十章

文件读写

数据是现代商业决策的基石，而文件操作是数据处理的核心环节。本章系统阐述 Python 文件输入输出 (File I/O) 的基本原理和实践方法。从文件的打开、读取、写入到关闭，将建立起完整的数据访问链路；从文本文件到二进制文件，将覆盖不同业务场景下的数据格式需求；从单行处理到批量操作，将掌握处理不同规模数据的最佳实践。这些知识和技能将为后续的商业数据分析、财务报表处理、销售数据统计等实际应用奠定坚实基础。

10.1 文件打开与关闭

重要性：★★★★★； **难易度：**★

10.1.1 open 函数

在 Python 编程中，`open()` 函数是进行文件操作的基础。该函数用于打开一个文件，并返回一个文件对象，之后可对该对象进行读取或写入操作。其基本语法如下：

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

其中，`file` 参数指定要打开的文件路径，`mode` 参数指定文件的打开模式。常用的模式包括：

- `'r'`：以只读模式打开文件（默认值）。
- `'w'`：以写入模式打开文件，若文件已存在，则会覆盖其内容。
- `'a'`：以追加模式打开文件，数据将写入文件末尾。
- `'b'`：以二进制模式打开文件，可与其他模式组合使用，如 `'rb'` 表示以二进制读取模式打开文件。
- `'t'`：以文本模式打开文件（默认值），可与其他模式组合使用，如 `'rt'` 表示以文本读取模式打开文件。
- `'+'`：同时打开文件进行读写操作，可与其他模式组合使用，如 `'r+'` 表示以读写模式打开文件。

以下是一个使用 `open()` 函数读取文件内容的示例：

```
# 打开文件 example.txt 进行读取
file = open('example.txt', 'r', encoding='utf-8')
content = file.read()
print(content)
file.close()
```

在上述代码中，`encoding='utf-8'`参数指定文件的编码方式，以正确读取包含非 ASCII 字符的文件。`file.close()`方法用于关闭文件对象，释放系统资源并确保数据完整性。

假设`example.txt`文件的内容如下：

```
Python是一种广泛使用的高级编程语言。
它具有简洁的语法和强大的功能。
```

运行上述代码将输出文件的全部内容。

此外，`open()`函数还可用于写入文件。以下是一个写入文件的示例：

```
# 打开文件 example.txt 进行写入
file = open('example.txt', 'w', encoding='utf-8')
file.write('这是一个新的内容。')
file.close()
```

执行此代码后，`example.txt`文件的内容将被替换为`'这是一个新的内容。'`。需要注意的是，使用`'w'`模式打开文件会覆盖原有内容，若需在文件末尾追加内容，应使用`'a'`模式。`file.close()`方法确保写入的数据被刷新到磁盘，避免数据丢失。

此外，为了简化文件关闭操作，可使用`with`语句自动管理文件资源：

```
# 使用 with 语句打开文件
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('这是一个新的内容。')
    # 无需显式调用 file.close()
```

在上述示例中，`with`语句会在代码块执行完毕后自动关闭文件，确保资源释放和数据完整性。

案例:商品数据读取与写入

在商品数据分析中,`open` 函数是处理文件读写操作的核心工具。例如,可以使用该函数从商品数据文件中读取信息,进行处理后再写入新的文件,便于分析和存储。

```
# 商品数据读取与写入示例

# 读取商品数据文件
with open('products.csv', 'r', encoding='utf-8') as infile:
    lines = infile.readlines()

# 处理数据，例如：过滤价格高于100的商品
filtered_lines = [line for line in lines if float(line.split(',')[2]) > 100]

# 将过滤后的数据写入新的文件
with open('filtered_products.csv', 'w', encoding='utf-8') as outfile:
    outfile.writelines(filtered_lines)
```

2.1 输入文件 `products.csv`

```
ID,Name,Price
1,Smartphone,299.99
2,Laptop,799.99
3,Headphones,49.99
4,Monitor,199.99
5,Keyboard,25.99
6,Mouse,15.99
7,Tablet,150.00
8,Smartwatch,99.99
9,Camera,120.00
10,Printer,85.00
```

输出文件 `filtered_products.csv`

```
1,Smartphone,299.99
2,Laptop,799.99
4,Monitor,199.99
7,Tablet,150.00
```

功能描述

- **读取文件**:通过 `open` 函数以只读模式(`'r'`)打开商品数据文件 `products.csv`。
- **数据处理**:将文件按行读取并过滤出价格高于 100 的商品数据。
- **写入文件**:使用 `open` 函数以写入模式(`'w'`)创建或覆盖一个新文件 `filtered_products.csv`,保存处理后的数据。

该案例通过实际的商品数据处理展示了 `open` 函数的综合应用,为数据分析任务提供了基础支持。

10.1.2 close 方法

`close()` 方法用于关闭已打开的文件对象。调用 `close()` 方法的必要性体现在以下几个方面：

1. **释放系统资源**：每个打开的文件都会占用系统资源，如文件描述符。在长时间运行的程序中，未及时关闭文件可能导致资源耗尽，进而引发程序错误。

2. **确保数据完整性**：在写入操作中，数据通常先写入缓冲区，随后再写入磁盘。调用 `close()` 方法会刷新缓冲区，将所有未写入的数据保存到磁盘，确保数据不丢失。

3. **避免文件锁定**：某些操作系统在文件打开期间会锁定文件，阻止其他进程访问。未关闭文件可能导致其他程序无法读取或修改该文件。

代码示例：

以下示例展示了在不调用 `close()` 方法的情况下，可能出现的数据丢失问题：

```
# 打开文件 example.txt 进行写入
file = open('example.txt', 'w')
file.write('这是一个测试内容。')
# 未调用 file.close()
```

在上述代码中，未调用 `file.close()` 方法，可能导致数据未实际写入磁盘。为确保数据完整性，应在写入操作后调用 `close()` 方法：

```
# 打开文件 example.txt 进行写入
file = open('example.txt', 'w')
file.write('这是一个测试内容。')
file.close()
```

10.1.3 with 语句

在 Python 编程中，`with` 语句用于简化对资源的管理，特别是在文件操作中。使用 `with` 语句打开文件时，系统会在代码块执行完毕后自动关闭文件，无需显式调用 `close()` 方法，从而确保资源的正确释放并提高代码的可读性。

代码示例：

以下示例展示了如何使用 `with` 语句读取文件内容：

```
# 使用 with 语句打开文件 example.txt 进行读取
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

在上述代码中，`with` 语句打开名为 `example.txt` 的文件，并将文件对象赋值给变量 `file`。在 `with` 代码块内，调用 `read()` 方法读取文件内容，并输出到控制台。当代码块执行完毕后，文件会被自动关闭。

示例文件内容：

假设 `example.txt` 文件的内容如下：

```
Python是一种广泛使用的高级编程语言。
它具有简洁的语法和强大的功能。
```

运行上述代码将输出文件的全部内容。

`with`语句在文件操作中能够自动管理资源，确保在代码块结束后文件被正确关闭，避免资源泄漏或数据丢失。这种方式不仅提高了代码的可读性，还能减少人为疏忽导致的错误。在编程实践中，强烈推荐使用`with`语句进行文件操作。

10.2 文件的读写方法和模式

重要性：★★★★★； **难易度：**★★★

10.2.1 读取方法

文件的读取方法主要包括`read()`、`readline()`、`readlines()`和直接迭代文件对象。这些方法适用于不同的场景，选择合适的方法有助于提高代码的效率和可读性。

1. `read()`方法：

`read()`方法用于一次性读取整个文件的内容，并将其作为一个字符串返回。需要注意的是，若文件较大，使用`read()`可能导致内存占用过高。

示例：

```
# 打开文件 example.txt 进行读取
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

假设`example.txt`文件的内容如下：

```
Python是一种广泛使用的高级编程语言。
它具有简洁的语法和强大的功能。
```

运行上述代码将输出文件的全部内容。

2. `readline()`方法：

`readline()`方法用于读取文件的一行内容，包括行末的换行符。每次调用`readline()`都会返回文件的下一行，适用于逐行读取文件的场景。

示例：

```
# 打开文件 example.txt 逐行读取
with open('example.txt', 'r', encoding='utf-8') as file:
    line = file.readline()
    while line:
        print(line, end='')  # end='' 防止重复添加换行
        line = file.readline()
```

运行上述代码将逐行输出文件内容。

3. `readlines()`方法：

`readlines()` 方法用于读取文件的所有行，并将其作为一个列表返回，每个元素为文件中的一行。需要注意的是，若文件较大，使用 `readlines()` 可能导致内存占用过高。

示例：

```
# 打开文件 example.txt 读取所有行
with open('example.txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()
    for line in lines:
        print(line, end='')
```

运行上述代码将输出文件的全部内容。

4. 直接迭代文件对象：

文件对象本身是可迭代的，直接对文件对象进行迭代可逐行读取文件内容。这种方法内存占用较低，适用于大文件的读取。

示例：

```
# 打开文件 example.txt 直接迭代
with open('example.txt', 'r', encoding='utf-8') as file:
    for line in file:
        print(line, end='')
```

运行上述代码将逐行输出文件内容。

注意事项：

- 在读取文件时，建议使用 `with` 语句管理文件上下文。`with` 语句会在代码块执行完毕后自动关闭文件，确保资源的正确释放。

- 在读取包含非 ASCII 字符的文本文件时，需指定编码方式，如 `encoding='utf-8'`，以确保字符编码正确。

10.2.2 读取模式

文件的读取模式决定了文件的打开方式和操作权限。常用的读取模式包括：

1. `'r'`（只读模式）：以只读方式打开文件，文件必须存在，指针位于文件开头。
2. `'rb'`（二进制只读模式）：以二进制格式只读方式打开文件，适用于非文本文件，如图片、音频等。
3. `'r+'`（读写模式）：以读写方式打开文件，文件必须存在，指针位于文件开头。
4. `'rb+'`（二进制读写模式）：以二进制格式读写方式打开文件，适用于需要读写非文本文件的情况。

代码示例：

以下示例展示了如何使用不同的读取模式读取文件内容。假设存在一个名为 `example.txt` 的文本文件，其内容如下：

```
Python是一种广泛使用的高级编程语言。
它具有简洁的语法和强大的功能。
```

- 使用 `'r'` 模式读取文件：

```
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

- 使用 `'rb'` 模式读取文件：

```
with open('example.txt', 'rb') as file:
    content = file.read()
    print(content.decode('utf-8'))
```

- 使用 `'r+'` 模式读取并写入文件：

```
with open('example.txt', 'r+', encoding='utf-8') as file:
    content = file.read()
    print(content)
    file.write('\n这是追加的内容。')
```

- 使用 `'rb+'` 模式读取并写入二进制文件：

```
with open('example.txt', 'rb+') as file:
    content = file.read()
    print(content.decode('utf-8'))
    file.write('\n这是追加的内容。'.encode('utf-8'))
```

在上述示例中，`with` 语句用于确保文件在操作完成后自动关闭，`encoding='utf-8'` 参数指定了文件的编码方式，以正确处理包含非 ASCII 字符的内容。

10.2.3 写入方法

文件的写入方法主要包括 `write()` 和 `writelines()`。这两种方法用于将数据写入文件，但适用场景有所不同。

1. `write()` 方法：

`write()` 方法用于将字符串写入文件。需要注意的是，`write()` 方法不会自动添加换行符，若需换行，需手动在字符串中加入\n。

示例：

```
# 打开文件 example.txt 进行写入
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('这是第一行内容。\n')
    file.write('这是第二行内容。')
```

执行上述代码后，`example.txt` 文件的内容为：

```
这是第一行内容。
这是第二行内容。
```

2. `writelines()` 方法：

`writelines()` 方法用于将字符串列表写入文件。与 `write()` 方法类似，`writelines()` 不会自动添加换行符，需在列表中的每个字符串末尾手动添加\n。

示例：

```
# 打开文件 example.txt 进行写入
with open('example.txt', 'w', encoding='utf-8') as file:
    lines = ['这是第一行内容。\n', '这是第二行内容。\n', '这是第三行内容。']
    file.writelines(lines)
```

执行上述代码后，`example.txt` 文件的内容为：

```
这是第一行内容。
这是第二行内容。
这是第三行内容。
```

3. `print()` 函数：

`print()` 函数不仅用于在控制台输出信息，还可用于将内容写入文本文件。通过指定 `print()` 函数的 `file` 参数，可以将输出重定向到文件对象，从而实现文件写入操作。

示例文件内容：

假设需要将以下内容写入名为 `output.txt` 的文件：

```
Hello, this is a sample text.
This text will be written to a file using the print function.
```

使用 `print()` 函数写入文件的示例代码：

```
# 打开文件进行写入操作
with open('output.txt', 'w', encoding='utf-8') as file:
    # 使用 print() 函数将内容写入文件
    print("Hello, this is a sample text.", file=file)
    print("This text will be written to a file using the print function.", file=file)
```

代码解析：

1. 使用 `open()` 函数以写入模式（`'w'`）打开或创建名为 `output.txt` 的文件，并指定编码为 UTF-8。

2. 使用 `with` 语句管理文件上下文，确保在操作完成后文件自动关闭，避免资源泄漏。

3. 在 `print()` 函数中，指定 `file` 参数为打开的文件对象 `file`，将字符串内容写入文件。

注意事项：

- 在使用 `print()` 函数写入文件时，默认会在每次调用后添加换行符。如需避免换行，可设置 `end` 参数，例如：

```
print("This is a line without newline.", file=file, end='')
```

- 如果文件已存在且使用写入模式（`'w'`）打开，文件内容将被覆盖。如需在文件末尾追加内容，可使用追加模式（`'a'`）：

```
with open('output.txt', 'a', encoding='utf-8') as file:
    print("This line will be appended to the file.", file=file)
```

10.2.4 写入模式

文件的写入模式决定了文件的打开方式和操作权限。常用的写入模式包括：

1. `'w'`（写入模式）：以写入方式打开文件，若文件已存在，则清空其内容；若文件不存在，则创建新文件。

2. `'wb'`（二进制写入模式）：以二进制格式写入方式打开文件，适用于非文本文件，如图片、音频等。

3. `'a'`（追加模式）：以追加方式打开文件，若文件已存在，指针位于文件末尾，新的内容将添加到现有内容之后；若文件不存在，则创建新文件。

4. `'ab'`（二进制追加模式）：以二进制格式追加方式打开文件，适用于需要在非文本文件末尾添加内容的情况。

5. `'w+'`（读写模式）：以读写方式打开文件，若文件已存在，则清空其内容；若文件不存在，则创建新文件。

6. `'a+'`（追加读写模式）：以追加和读写方式打开文件，若文件已存在，指针位于文件末尾，可读取和追加内容；若文件不存在，则创建新文件。

代码示例：

以下示例展示了如何使用不同的写入模式操作文件。假设存在一个名为`example.txt`的文本文件，其初始内容如下：

```
Python是一种广泛使用的高级编程语言。
它具有简洁的语法和强大的功能。
```

- 使用`'w'`模式写入文件：

```
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('这是使用 w 模式写入的新内容。')
```

执行上述代码后，`example.txt`的内容将被替换为：

```
这是使用 w 模式写入的新内容。
```

- 使用`'a'`模式追加内容：

```
with open('example.txt', 'a', encoding='utf-8') as file:
    file.write('\n这是使用 a 模式追加的内容。')
```

执行上述代码后，`example.txt`的内容将变为：

```
这是使用 w 模式写入的新内容。
这是使用 a 模式追加的内容。
```

- 使用`'w+'`模式读写文件：

```
with open('example.txt', 'w+', encoding='utf-8') as file:
    file.write('这是使用 w+ 模式写入的新内容。')
    file.seek(0)
    content = file.read()
    print(content)
```

执行上述代码后，`example.txt` 的内容为：

```
这是使用 w+ 模式写入的新内容。
```

控制台输出：

```
这是使用 w+ 模式写入的新内容。
```

- 使用 `'a+'` 模式追加并读取内容：

```
with open('example.txt', 'a+', encoding='utf-8') as file:
    file.write('\n这是使用 a+ 模式追加的内容。')
    file.seek(0)
    content = file.read()
    print(content)
```

执行上述代码后，`example.txt` 的内容为：

```
这是使用 w+ 模式写入的新内容。
这是使用 a+ 模式追加的内容。
```

控制台输出：

```
这是使用 w+ 模式写入的新内容。
这是使用 a+ 模式追加的内容。
```

在上述示例中，`with` 语句用于确保文件在操作完成后自动关闭，`encoding='utf-8'` 参数指定了文件的编码方式，以正确处理包含非 ASCII 字符的内容。

10.3 文件路径

重要性：★★★★； **难易度：**★★

文件路径的指定方式主要分为相对路径和绝对路径。理解并正确使用这两种路径对于文件操作至关重要。

1. 相对路径：

相对路径是相对于当前工作目录（current working directory, CWD）指定的文件或目录位置。使用相对路径时，路径的起点是当前正在访问的文件夹。相对路径更灵活，易于在不同机器或文件夹结构间移植代码。但如果更改了工作目录，可能导致路径无效。

示例：

假设当前工作目录为 `/home/user/project`，目录结构如下：

```
/home/user/project/
├── main.ipynb
└── data/
    └── example.txt
```

在 `main.ipynb` 中，使用相对路径读取 `example.txt`：

```
with open('data/example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

此时，`data/example.txt` 即为相对路径。

2. 绝对路径：

绝对路径是从文件系统的根目录开始的完整路径，提供了到达指定文件或目录的完整地址。绝对路径明确无误地指向文件位置，不受当前工作目录的影响。但不够灵活，当文件系统结构变化或在不同系统之间迁移代码时可能需要修改。

示例：

基于上述目录结构，使用绝对路径读取 `example.txt`：

```
with open('/home/user/project/data/example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

此时，`/home/user/project/data/example.txt` 即为绝对路径。

获取当前工作目录：

在 Python 中，可使用 `os` 模块的 `getcwd()` 函数获取当前工作目录：

```
import os

current_directory = os.getcwd()
print(f"当前工作目录：{current_directory}")
```

输出示例：

```
当前工作目录：/home/user/project
```

将相对路径转换为绝对路径：

可使用 `os.path` 模块的 `abspath()` 函数将相对路径转换为绝对路径：

```
import os

relative_path = 'data/example.txt'
absolute_path = os.path.abspath(relative_path)
print(f"绝对路径：{absolute_path}")
```

输出示例：

```
绝对路径：/home/user/project/data/example.txt
```

3. 路径分隔符的处理

在 Windows 系统中，路径分隔符通常使用反斜杠\，但在 Python 中，反斜杠是转义字符，可能导致路径解析错误。为避免此类问题，建议在处理 Windows 路径时采取以下方法：

1. 使用原始字符串（Raw String）：

在字符串前添加字母 `r`，将字符串声明为原始字符串，避免反斜杠被解释为转义字符。

示例：

```
# 使用原始字符串表示Windows路径
path = r'C:\Users\Username\Documents\example.txt'

# 打开文件进行读取
with open(path, 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

假设 `example.txt` 文件内容如下：

```
Python是一种广泛使用的高级编程语言。
```

运行上述代码将正确读取并输出文件内容。

2. 使用双反斜杠：

在路径字符串中使用双反斜杠\\表示单个反斜杠，避免转义字符的问题。

示例：

```
# 使用双反斜杠表示Windows路径
path = 'C:\\Users\\Username\\Documents\\example.txt'

# 打开文件进行读取
with open(path, 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

此方法同样可正确读取并输出文件内容。

3. 使用 `os.path` 模块：

Python 的 `os.path` 模块提供了跨平台的路径操作函数，自动处理路径分隔符，避免手动处理反斜杠，此方法确保路径在不同操作系统上的兼容性。

示例：

```
import os

# 动态构建路径，适配不同操作系统
directory = 'data'
filename = 'example.txt'
path = os.path.join(directory, filename)

with open(path, 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

假设当前工作目录为 `/home/user/project`，代码运行后，将读取 `/home/user/project/data/example.txt` 文件的内容，而无需显式处理路径分隔符。

注意事项：

- 在处理包含非 ASCII 字符的文件路径或文件内容时，需指定编码方式，如 `encoding='utf-8'`，以确保字符编码正确。

- 在跨平台开发时，建议使用 `os.path` 或 `pathlib` 模块处理路径，避免手动拼接路径字符串，以减少错误并提高代码的可移植性。

10.4 商业数据分析中常用的数据文件格式

在商业数据分析中，常用的文件格式主要包括以下几种：

1. TXT（纯文本）文件：

TXT 文件是最基本的文件格式，仅包含纯文本数据，不包含格式化信息或元数据。其具有广泛的兼容性，可以在各种文本编辑器中打开和编辑。然而，TXT 文件缺乏结构化信息和元数据描述，不适合存储复杂数据。

应用场景：

- 用于存储非结构化数据，如系统日志、文本记录和文档。
- 具有最佳的跨平台兼容性，便于在不同平台之间传输数据。
- 适合进行文本挖掘和自然语言处理。

2. CSV（Comma-Separated Values）文件：

CSV 文件以纯文本形式存储数据，使用逗号分隔各字段。其简单、易于使用，广泛应用于数据交换和存储。然而，CSV 文件缺乏对数据类型和结构的描述，可能导致数据解析时出现问题。

应用场景：

- 最常用于存储表格形式的业务数据。
- 适用于财务报表、销售记录和客户信息等结构化数据。
- 可方便地导入 Excel 或数据库系统，用于进一步分析和处理。

3. Excel 文件（.xls, .xlsx）：

Excel 文件由 Microsoft Excel 创建，支持多种数据类型和复杂的表格结构。其直观的界面和强大的功能使其在商业和数据分析中广泛使用。但由于其为专有格式，可能在不同软件之间存在兼容性问题。

应用场景：

- 商业分析中使用最广泛的电子表格格式。
- 支持多个工作表、公式计算和数据透视表。
- 适合财务建模、预算规划和销售数据分析等业务场景。

4. JSON（JavaScript Object Notation）文件：

JSON 是一种轻量级的数据交换格式，易于人和机器读取和编写。其结构化的键值对形式适合表示复杂的嵌套数据结构，广泛用于 Web 应用程序的数据传输。然而，JSON 文件可能在处理大型数据集时效率较低。

应用场景：

- 常见于网络应用程序和 API 数据交换。
- 适合存储层次化的业务对象，如产品目录和客户画像。
- 支持嵌套结构，便于表达复杂的业务实体关系。

10.4.1 TXT 文件读写

参见本章10.2节的内容。

10.4.2 CSV 文件读写

在商业数据分析中，CSV（Comma-Separated Values）文件是一种常用的文本格式，用于存储表格数据。Python 提供了内置的 `csv` 模块，方便地读取和写入 CSV 文件。

读取 CSV 文件：

假设存在一个名为 `data.csv` 的文件，内容如下：

```
Name,Age,Occupation
Alice,30,Data Scientist
Bob,25,Software Engineer
Charlie,35,Product Manager
```

使用 `csv` 模块读取该文件的示例代码如下：

```
import csv

with open('data.csv', mode='r', newline='', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    header = next(csv_reader)  # 读取表头
        for row in csv_reader:
            print(f'Name: {row[0]}, Age: {row[1]}, Occupation: {row[2]}')
```

上述代码打开 `data.csv` 文件，使用 `csv.reader` 创建一个读取器对象。首先读取表头，然后遍历每一行数据，按字段输出。

写入 CSV 文件：

要将数据写入 CSV 文件，可使用 `csv.writer`。以下示例将数据写入名为 `output.csv` 的文件：

```
import csv

data = [
['Name', 'Age', 'Occupation'],
['David', '28', 'Data Analyst'],
['Eva', '22', 'Marketing Specialist'],
['Frank', '40', 'Sales Manager']
]

with open('output.csv', mode='w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows(data)
```

此代码定义了一个包含数据的列表 `data`，然后使用 `csv.writer` 将其写入 `output.csv` 文件。`writerows` 方法用于写入多行数据。

使用字典读取和写入 CSV 文件：

`csv` 模块还提供了 `DictReader` 和 `DictWriter` 类，允许将每行数据作为字典处理。

读取：

```
import csv

with open('data.csv', mode='r', newline='', encoding='utf-8') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
        print(f"Name: {row['Name']}, Age: {row['Age']}, Occupation: {row['Occupation']}")
```

写入：

```
import csv

fieldnames = ['Name', 'Age', 'Occupation']
rows = [
{'Name': 'George', 'Age': '29', 'Occupation': 'Consultant'},
{'Name': 'Hannah', 'Age': '24', 'Occupation': 'Designer'}
]

with open('output_dict.csv', mode='w', newline='', encoding='utf-8') as file:
    csv_writer = csv.DictWriter(file, fieldnames=fieldnames)
    csv_writer.writeheader()  # 写入表头
    csv_writer.writerows(rows)
```

在上述示例中，`DictReader` 和 `DictWriter` 允许使用字典方式读取和写入 CSV 文件，更加直观。

注意事项：

- 在打开文件时，使用 `newline=''` 参数可避免在不同平台上出现额外的空行。
- 指定 `encoding='utf-8'` 以确保正确处理非 ASCII 字符。
- 使用 `with` 语句管理文件上下文，确保文件在操作完成后自动关闭。

10.4.3 Excel 文件读写

在商业数据分析中，Excel 文件（.xls 和.xlsx）是常用的电子表格格式。Python 提供了多种库来读取和写入 Excel 文件，其中 `pandas` 和 `openpyxl` 是常用的选择。

1. 使用 `pandas` 库读取和写入 Excel 文件：

`pandas` 库提供了简洁的接口来处理 Excel 文件。以下示例展示了如何读取和写入 Excel 文件。

(1) 读取 Excel 文件：

假设存在一个名为 `data.xlsx` 的 Excel 文件，内容如下：

```
| Name    | Age | Occupation         |
|---------|-----|--------------------|
| Alice   | 30  | Data Scientist     |
| Bob     | 25  | Software Engineer  |
| Charlie | 35  | Product Manager    |
```

使用 `pandas` 读取该文件的代码如下：

```
import pandas as pd

```

```
# 读取Excel文件
df = pd.read_excel('data.xlsx')

# 显示前几行数据
print(df.head())
```

(2) 写入 Excel 文件：

将数据写入 Excel 文件的代码如下：

```
import pandas as pd

# 创建数据框
data = {
    'Name': ['David', 'Eva', 'Frank'],
    'Age': [28, 22, 40],
    'Occupation': ['Data Analyst', 'Marketing Specialist', 'Sales Manager']
}
df = pd.DataFrame(data)

# 写入Excel文件
df.to_excel('output.xlsx', index=False)
```

在上述代码中，`to_excel` 方法用于将数据框写入 Excel 文件，`index=False` 参数表示不写入行索引。

2. 使用 `openpyxl` 库读取和写入 Excel 文件：

`openpyxl` 是一个专门用于处理 Excel 2010 及更新版本（即.xlsx 格式）的 Python 库。

(1) 读取 Excel 文件：

```
from openpyxl import load_workbook

# 加载Excel工作簿
wb = load_workbook('data.xlsx')

# 选择活动工作表
ws = wb.active

# 读取单元格值
# 从第2行开始读取，第1行通常为表头，仅读取单元格中的值
for row in ws.iter_rows(min_row=2, values_only=True):
    print(row)
```

(2) 写入 Excel 文件：

```
from openpyxl import Workbook

# 创建新的工作簿
wb = Workbook()
ws = wb.active

# 写入表头
ws.append(['Name', 'Age', 'Occupation'])
```

```
# 写入数据行
data = [
['George', 29, 'Consultant'],
['Hannah', 24, 'Designer']
]
for row in data:
    ws.append(row)

# 保存工作簿
wb.save('output_openpyxl.xlsx')
```

在上述代码中，`append` 方法用于向工作表添加行数据。

注意事项：

- `pandas` 库依赖于 `openpyxl` 来处理.xlsx 文件，确保已安装 `openpyxl` 库。

- 对于旧版 Excel 文件（.xls），可使用 `xlrd` 库读取，但需注意 `xlrd` 从 2.0.0 版本开始不再支持.xlsx 文件。

- 使用 `with` 语句管理文件上下文，确保文件在操作完成后自动关闭。

10.4.4 JSON 文件读写

在商业数据分析中，JSON（JavaScript Object Notation）文件是一种常用的轻量级数据交换格式，广泛应用于数据存储和传输。Python 提供了内置的 `json` 模块，方便地读取和写入 JSON 文件。

1. 读取 JSON 文件：

假设存在一个名为 `data.json` 的文件，内容如下：

```
{
    "employees": [
    {"name": "Alice", "age": 30, "department": "Data Science"},
    {"name": "Bob", "age": 25, "department": "Software Engineering"},
    {"name": "Charlie", "age": 35, "department": "Product Management"}
    ]
}
```

使用 Python 读取该 JSON 文件的代码如下：

```
import json

# 打开并读取 JSON 文件
with open('data.json', 'r', encoding='utf-8') as file:
    data = json.load(file)

# 输出读取的数据
print(data)
```

上述代码中，使用 `open` 函数以读取模式（`'r'`）打开 `data.json` 文件，并指定编码为 UTF-8。通过 `json.load` 函数将文件内容解析为 Python 字典对象，存储在变量 `data` 中。

2. 写入 JSON 文件：

将 Python 数据写入 JSON 文件的代码如下：

```
import json

# 定义要写入的数据
data_to_write = {
    "employees": [
    {"name": "David", "age": 28, "department": "Data Analysis"},
    {"name": "Eva", "age": 22, "department": "Marketing"},
    {"name": "Frank", "age": 40, "department": "Sales"}
    ]
}

# 将数据写入 JSON 文件
with open('output.json', 'w', encoding='utf-8') as file:
    json.dump(data_to_write, file, ensure_ascii=False, indent=4)
```

在上述代码中，定义了一个包含员工信息的字典`data_to_write`。使用`open`函数以写入模式（`'w'`）打开（或创建）名为`output.json`的文件，并指定编码为 UTF-8。通过`json.dump`函数将字典数据写入文件。参数`ensure_ascii=False`确保非 ASCII 字符（如中文）能够正确写入，`indent=4`使输出的 JSON 文件具有良好的可读性。

注意事项：

- 使用`with`语句管理文件上下文，确保文件在操作完成后自动关闭，避免资源泄漏。
- 在读取和写入 JSON 文件时，指定编码为 UTF-8 以支持多语言字符集。
- 在写入 JSON 文件时，使用`indent`参数格式化输出，提升可读性。

10.4.5 常见数据文件格式在商业数据分析中的应用

1. 客户反馈情感分析

在客户反馈数据分析中，通过合理使用文本文件的读写方法，可以高效地处理客户反馈数据。以下案例展示了如何综合应用文本文件的读写方法读取、处理和写入客户反馈数据。

```
# 安装所需的包
!pip install snownlp # 情感分析
!pip install jieba   # 中文分词

from snownlp import SnowNLP

# 读取客户反馈文件
with open('customer_feedback.txt', 'r', encoding='utf-8') as infile:
    feedbacks = infile.readlines()

# 处理数据，进行情感分析
results = []
for feedback in feedbacks:
    s = SnowNLP(feedback.strip())
    sentiment = s.sentiments  # 返回情感得分（0-负面，1-正面）
    sentiment_label = '正面' if sentiment > 0.5 else '负面'
```

```
    results.append(f'{feedback.strip()} - 情感：{sentiment_label}\n')

# 将分析结果写入新的文件
with open('feedback_analysis.txt', 'w', encoding='utf-8') as outfile:
    outfile.writelines(results)
```

示例文件内容

输入文件 `customer_feedback.txt`

```
产品质量非常好，值得购买。
客服态度差，不会再来。
物流速度快，包装完好。
价格偏高，性价比一般。
使用体验不错，推荐给朋友。
```

输出文件 `feedback_analysis.txt`

```
产品质量非常好，值得购买。 - 情感：正面
客服态度差，不会再来。 - 情感：负面
物流速度快，包装完好。 - 情感：正面
价格偏高，性价比一般。 - 情感：正面
使用体验不错，推荐给朋友。 - 情感：正面
```

输出结果中，第四条反馈的情感分析结果被错误地判断为正面，其实际情感为负面。

功能描述

- **读取文件：**使用 `open` 函数以只读模式（`'r'`）打开名为 `customer_feedback.txt` 的客户反馈数据文件，并读取其内容。
- **数据处理：**遍历读取的反馈数据，使用 `SnowNLP` 库对每条反馈进行情感分析，判断其情感倾向（正面或负面）。
- **写入文件：**使用 `open` 函数以写入模式（`'w'`）创建或覆盖一个新文件 `feedback_analysis.txt`，并将分析结果写入其中。

该示例通过实际的客户反馈数据处理，展示了 `open` 函数与 `SnowNLP` 的综合应用，为客户反馈数据分析提供了基础支持。

2. 财务账户余额计算

在财务数据分析中，CSV 文件是保存财务数据的一种重要文件格式，通过合理使用 CSV 文件读写方法，可以高效地处理财务数据文件。以下案例展示了如何综合应用 CSV 文件读写方法读取、处理和写入财务数据。

```
import csv

# 读取财务数据文件
with open('financial_data.csv', mode='r', encoding='utf-8') as infile:
    reader = csv.reader(infile)
    header = next(reader)  # 读取表头
    data = [row for row in reader]
```

```
# 处理数据，例如：计算每个账户的总余额
account_balances = {}
for row in data:
    account_id = row[0]
    transaction_amount = float(row[2])
    if account_id in account_balances:
        account_balances[account_id] += transaction_amount
    else:
        account_balances[account_id] = transaction_amount

# 将处理后的数据写入新的文件
with open('account_balances.csv', mode='w', encoding='utf-8', newline='') as outfile:
    writer = csv.writer(outfile)
    writer.writerow(['Account ID', 'Total Balance'])
    for account_id, total_balance in account_balances.items():
        writer.writerow([account_id, total_balance])
```

示例文件内容

输入文件 `financial_data.csv`

```
Account ID,Date,Transaction Amount
1001,2024-01-15,500.00
1002,2024-01-16,-200.00
1001,2024-01-17,150.00
1003,2024-01-18,300.00
1002,2024-01-19,100.00
```

输出文件 `account_balances.csv`

```
Account ID,Total Balance
1001,650.00
1002,-100.00
1003,300.00
```

功能描述

- **读取文件**：使用 `open` 函数以只读模式（`'r'`）打开名为 `financial_data.csv` 的财务数据文件，并使用 `csv.reader` 读取其内容。
- **数据处理**：遍历读取的数据，计算每个账户的总交易金额（余额）。
- **写入文件**：使用 `open` 函数以写入模式（`'w'`）创建或覆盖一个新文件 `account_balances.csv`，并使用 `csv.writer` 将处理后的数据写入其中。

该案例通过实际的财务数据处理，展示了 CSV 文件读写方法的综合应用，为财务数据分析提供了技术支持。

3. 国际贸易数据汇总

在国际贸易数据分析中，`pandas` 库结合 `openpyxl` 库，可高效地处理 Excel 文件的读写操作。以下案例展示了如何综合应用 Excel 文件读写方法读取、处理和写入国际贸易数据。

```
import pandas as pd

# 读取国际贸易数据
df_import = pd.read_excel('import_data.xlsx', sheet_name='Imports')
df_export = pd.read_excel('export_data.xlsx', sheet_name='Exports')

# 数据处理：计算每个国家的贸易总额
df_import['Total Import'] = df_import.iloc[:, 1:].sum(axis=1)
df_export['Total Export'] = df_export.iloc[:, 1:].sum(axis=1)

# 合并进出口数据
df_trade = pd.merge(df_import[['Country', 'Total Import']],
df_export[['Country', 'Total Export']],
on='Country', how='outer')

# 计算贸易差额
df_trade['Trade Balance'] = df_trade['Total Export'] - df_trade['Total Import']

# 将结果写入新的 Excel 文件
df_trade.to_excel('trade_summary.xlsx', index=False)
```

示例文件内容

输入文件 `import_data.xlsx`（工作表名:Imports）

```
Country,Product A,Product B,Product C
USA,1000,1500,2000
China,2000,2500,3000
India,1500,1000,500
```

输入文件 `export_data.xlsx`（工作表名:Exports）

```
Country,Product A,Product B,Product C
USA,1200,1600,2100
China,2200,2600,3100
India,1400,1100,600
```

输出文件 `trade_summary.xlsx`

```
Country,Total Import,Total Export,Trade Balance
China,7500,7900,400
India,3000,3100,100
USA,4500,4900,400
```

功能描述

- **读取文件**:使用 `pandas` 库的 `read_excel` 函数分别读取名为 `import_data.xlsx` 和 `export_data.xl` 的 Excel 文件,指定工作表名称分别为'Imports' 和'Exports'。
- **数据处理**:计算每个国家的进口总额和出口总额,将结果存储在新的列 `Total Import` 和 `Total Export` 中。

- **数据合并**：使用 `merge` 函数根据 `Country` 列合并进口和出口数据，生成包含每个国家进口、出口总额和贸易差额的汇总表。
- **写入文件**：使用 `to_excel` 函数将汇总结果写入新的 Excel 文件 `trade_summary.xlsx`，不包含索引列。

关键代码片段解析

(1) 代码片段 `df_import.iloc[:, 1:].sum(axis=1)`

- `df_import.iloc[:, 1:]`：通过位置索引选取第 1 列至最后一列的数据，这些列通常为数值型数据（如'Product A', 'Product B', 'Product C'）。
- `.sum(axis=1)`：按行求和，计算每个国家的总进口额或总出口额。
- `df_import['Total Import']`：将结果存储为新列'Total Import'，表示每个国家的进口总额。

(2) 代码片段 `pd.merge()`

- `pd.merge`：用于基于指定键合并两个数据表。
- `df_import[['Country', 'Total Import']]` 和 `df_export[['Country', 'Total Export']]`：只保留'Country' 和总额列，减少冗余数据。
- `on='Country'`：以'Country' 列为键，按国家名称匹配数据。
- `how='outer'`：外连接方式，保留两表中的所有国家，缺失值填充为 `NaN`。

4. 电子商务库存数据处理

在电子商务平台的库存数据分析中，JSON（JavaScript Object Notation）是一种常用的数据交换格式，具有结构清晰、易于解析的特点。Python 提供了内置的 `json` 模块，可方便地对 JSON 文件进行读写操作。以下案例展示了如何综合应用 JSON 文件的读写方法来读取、处理和写入库存数据。

```
import json

# 读取库存数据
with open('inventory.json', 'r', encoding='utf-8') as file:
    inventory_data = json.load(file)

# 数据处理：计算每种商品的总库存
total_stock = {}
for item in inventory_data:
    product = item['product_name']
    quantity = item['quantity']
    if product in total_stock:
        total_stock[product] += quantity
    else:
        total_stock[product] = quantity

# 将结果写入新的 JSON 文件
```

```
with open('total_stock.json', 'w', encoding='utf-8') as file:
    json.dump(total_stock, file, ensure_ascii=False, indent=4)
```

示例文件内容

输入文件 `inventory.json`

```
[
{"product_name": "Laptop", "quantity": 50, "warehouse": "A"},
{"product_name": "Smartphone", "quantity": 200, "warehouse": "B"},
{"product_name": "Laptop", "quantity": 30, "warehouse": "B"},
{"product_name": "Tablet", "quantity": 70, "warehouse": "A"}
]
```

输出文件 `total_stock.json`

```
{
    "Laptop": 80,
    "Smartphone": 200,
    "Tablet": 70
}
```

功能描述

- **读取文件**：使用 `open` 函数以读取模式（`'r'`）打开名为 `inventory.json` 的文件，指定编码为 `'utf-8'`，确保正确处理文件中的字符。使用 `json.load` 函数将 JSON 数据解析为 Python 对象。
- **数据处理**：遍历读取的库存数据，计算每种商品的总库存量。通过检查商品名称是否已在 `total_stock` 字典中，累加相同商品的数量。
- **写入文件**：使用 `open` 函数以写入模式（`'w'`）打开名为 `total_stock.json` 的文件，指定编码为 `'utf-8'`。使用 `json.dump` 函数将处理后的数据写入文件，设置参数 `ensure_ascii=False` 以确保非 ASCII 字符正确写入，`indent=4` 使输出的 JSON 数据格式化，便于阅读。

10.5 文件随机读写

重要性：★★； **难易度**：★★★★

在 Python 中，`seek()` 和 `tell()` 方法用于控制文件指针的位置，从而实现对文件的随机读写操作。`seek()` 方法用于移动文件指针到指定位置，`tell()` 方法用于获取当前文件指针的位置。

示例文件内容：

假设有一个名为 `example.txt` 的文本文件，内容如下：

```
1 Hello, this is a sample text file.
2 It contains multiple lines of text.
3 Each line serves as an example.
```

使用 `seek()` 和 `tell()` 进行文件操作的示例代码：

```python
# 打开文件进行读写操作
with open('example.txt', 'r+') as file:
    # 读取并打印第一行
    first_line = file.readline()
    print(f"第一行内容：{first_line.strip()}")

    # 获取当前文件指针位置
    current_position = file.tell()
    print(f"当前文件指针位置：{current_position}")

    # 移动文件指针到文件开头
    file.seek(0)
    print(f"文件指针已移动到位置：{file.tell()}")

    # 在文件开头插入新内容
    new_content = "Inserted line at the beginning.\n"
    original_content = file.read()
    file.seek(0)
    file.write(new_content + original_content)

    # 移动文件指针到文件末尾
    file.seek(0, 2)
    print(f"文件指针已移动到文件末尾位置：{file.tell()}")

    # 在文件末尾追加新内容
    file.write("\nAppended line at the end.")
```

代码解析：

1. 使用 `with open('example.txt', 'r+')` 打开文件，模式为 `'r+'`，表示以读写模式打开文件。

2. 使用 `readline()` 方法读取并打印文件的第一行内容。

3. 使用 `tell()` 方法获取当前文件指针的位置，并打印该位置。

4. 使用 `seek(0)` 将文件指针移动到文件开头，并验证指针位置。

5. 在文件开头插入新内容。为此，先读取原始内容，移动指针到开头，然后将新内容和原始内容写入文件。

6. 使用 `seek(0, 2)` 将文件指针移动到文件末尾，`0` 表示偏移量，`2` 表示从文件末尾开始计算。

7. 在文件末尾追加新内容。

注意事项：

- 使用 `seek()` 方法时，第二个参数 `whence` 的取值：`0` 表示从文件开头计算（默认值），`1` 表示从当前位置计算，`2` 表示从文件末尾计算。

- 在文本模式下（如 `'r+'`），`seek()` 和 `tell()` 的偏移量和位置是以字节为单位的。