学习目标：

1. 掌握文件操作的核心流程（打开、读写、关闭）
2. 能独立处理文本\二进制文件
3. 了解文件的编码

引入

为什么要学文件操作?

在 Python 中，“文件操作” 是实现**数据持久化**的核心手段（内存中的数据断电会消失，文件能长期保存）。比如：

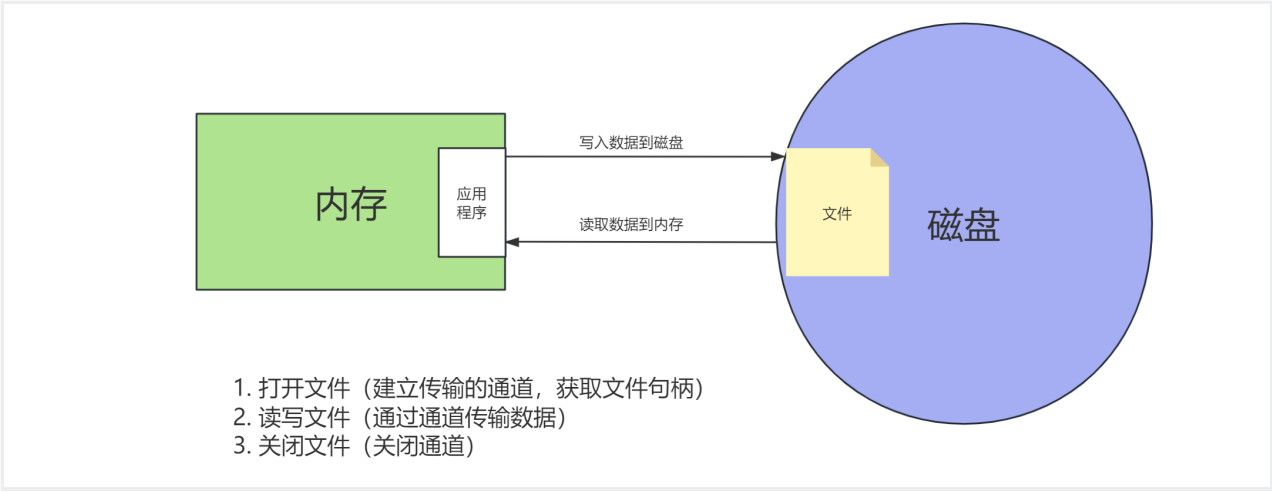
- 记录用户登录日志（log.txt）；
- 保存爬虫爬取的商品数据（products.csv）；
- 读取 Excel 表格中的学生成绩（基础版用文本文件，进阶用pandas）；
- 生成程序运行报告（report.md）。

一句话：程序需要 “和文件打交道”，才能实现 “数据存得下、取得到”。

文件操作的核心流程

所有文件操作都遵循 “打开→操作→关闭” 的流程，类比 “打开笔记本→写字 / 读字→合上笔记本”：

1. **打开文件**：获取文件的 “操作权限”（拿到 “笔记本”）；
2. **读写文件**：执行读（获取内容）或写（写入内容）操作（“写字 / 读字”）；
3. **关闭文件**：释放文件资源（“合上笔记本”，避免占用内存）。



文件操作

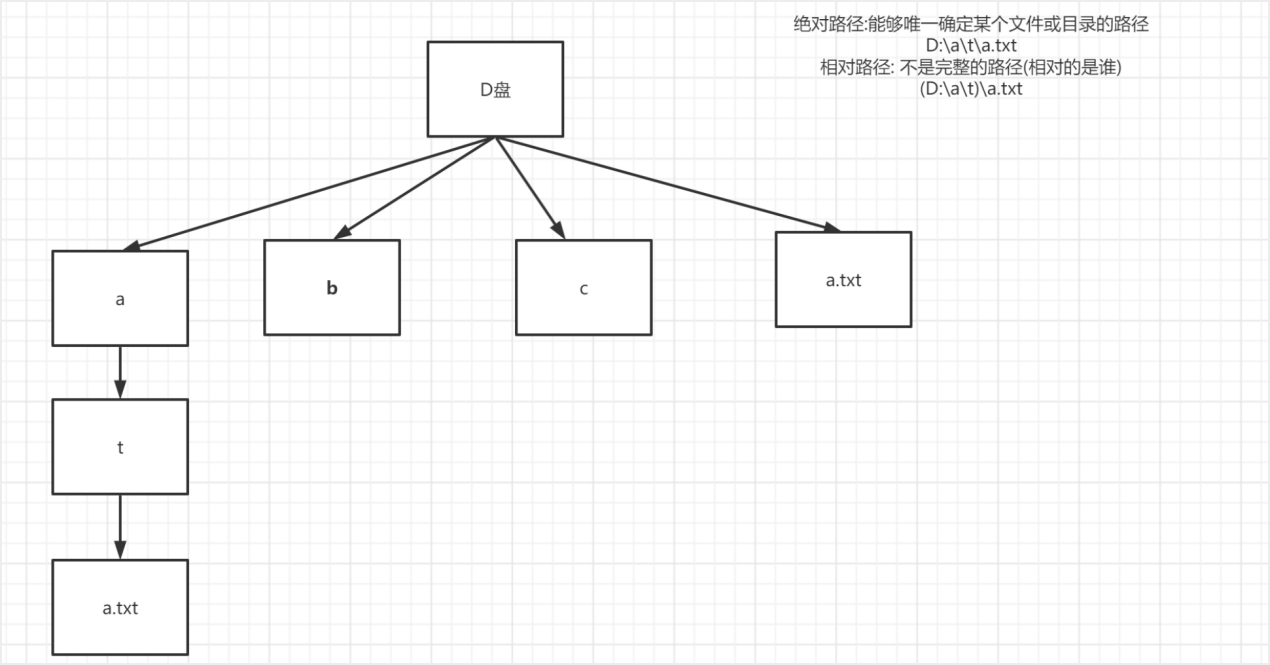
这是这一章内容的核心。

学习目标：

1. 知道什么是绝对路径，什么是相对路径
2. 会打开文件，关闭文件，读写文件
3. 掌握几种重要的打开模式
4. 掌握安全的操作文件的方式

绝对路径与相对路径

要学会操作文件，得学会两个概念：绝对路径与相对路径



绝对路径:

- 绝对路径名是 完整的路径名，不需要任何其他信息就可以定位它所表示的文件
- windows： E:\demo\first\a.txt

相对路径

- 相反，相对路径名必须使用取自其他路径名的信息进行解释(不完整的路径名)
- windows: second\a.txt

对于类 **UNIX**(Mac/Linux) 平台，绝对路径名的前缀始终是 "/"。相对路径名没有前缀。表示根目录的绝对路径名的前缀为 "/" 且名称序列为空。

- 绝对路径： /home/ciggar/6379.conf
- 相对路径： st/a.txt
- 根目录： /

对于 **Microsoft Windows** 平台，包含盘符的路径名前缀由驱动器号和一个 ":" 组成。如果路径名是绝对路径名，还可能后跟 "\“

- 绝对路径： e:\st\a.txt
- 相对路径：没有盘符前缀 st\a.txt

在Python中，相对路径相对的是谁呢？

相对的是当前目录，可以使用 `os.getcwd()` 查看当前运行的目录，避免路径混乱。

```
import os
print(os.getcwd())   # 输出当前程序所在目录，如 "D:/PythonClass"
```

基础操作：打开文件

`open()` 是 Python 内置函数，用于“打开文件并返回文件句柄”。

文件句柄 = 操作文件的“工具”，类似笔记本的“笔”

基本语法：

```
文件句柄 = open(文件路径, 打开模式, encoding=编码格式)
```

- **文件路径**：告诉 Python 文件在哪里（如"test.txt"或"D:/data/log.txt"）；
- **打开模式**：决定文件能执行读 / 写 / 追加等操作（核心参数，下节重点讲）；
- **encoding**：指定文件编码（文本文件必加，避免乱码，常用utf-8或gbk）。

初步体验：

```
# 1. 打开当前目录下的aaa.txt 文件
f = open('aaa.txt','r',encoding='utf-8')

# 2. 读取一行
s = f.readline()
print(s)

# 3. 关闭文件
f.close()
```

打开模式

打开模式决定了文件的操作权限，重点记忆前 6 种文本模式，后 3 种二进制模式按需了解。

模式	类型	核心功能	注意事项
r	文本读	**只读（默认模式）**，文件不存在则报错	不能写操作
w	文本写	**覆盖写**，文件不存在则创建，存在则清空内容	会覆盖原有内容，谨慎使用
a	文本追	**追加写**，文件不存在则创建，内容加在末尾	不会覆盖原有内容，适合写日志
r+	文本读写	**可读可写**，文件不存在则报错	写操作从文件开头覆盖
w+	文本读写	**可读可写**，文件不存在则创建，存在则清空	先清空再读写，慎用

a+	文本读写	**可读可写**，文件不存在则创建，写在末尾	读操作需先移动指针（后续讲seek()）
rb	二进制读	以二进制格式读（如图片、视频）	不指定encoding，避免乱码
wb	二进制写	以二进制格式写（如保存图片）	常用于文件传输、保存非文本数据
ab	二进制追	以二进制格式追加写	如给视频文件追加内容

模式对比：`w` 和 `a`

```
# 1. 用w模式写文件：覆盖原有内容
f = open("test.txt", "w", encoding="utf-8")
f.write("第一次写入：w模式会覆盖\n")
f.close()  # 必须关闭文件，否则内容可能没保存

# 2. 再用w模式写：覆盖上一次内容
f = open("test.txt", "w", encoding="utf-8")
f.write("第二次写入：覆盖了第一次\n")
f.close()

# 3. 用a模式追加：在末尾加内容
f = open("test.txt", "a", encoding="utf-8")
f.write("第三次写入：追加在末尾\n")
f.close()

# 读取结果：
f = open("test.txt", "r", encoding="utf-8")
print(f.read())  # 输出：第二次写入：覆盖了第一次\n第三次写入：追加在末尾
f.close()
```

核心操作：文件的读与写

掌握 4 种核心方法：读（read()/readline()/readlines()）、写（write()/writelines()）。

读取

- read(size)：读指定长度 / 全部内容
  - size为可选参数，指定读取的字符数（二进制模式如rb为字节数）
  - 如果不写size默认读取文件全部内容（大文件慎用，会占满内存）

```python
# 读取小文件（全部内容）
f = open("test.txt", "r", encoding="utf-8")
content = f.read()  # 读全部内容
print(content)
f.close()

# 读取部分内容（前10个字符）
f = open("test.txt", "r", encoding="utf-8")
content_part = f.read(10)  # 读前10个字符
print("前10个字符：", content_part)
f.close()
```

- readline()：逐行读取（适合大文件）
  - 每次调用读“一行内容”，包括换行符\n
  - 读到文件末尾返回空字符串""，可用于循环读取

```python
# 逐行读取大文件（避免内存溢出）
f = open("large_file.txt", "r", encoding="utf-8")
while True:
    line = f.readline()  # 每次读一行
    if not line:  # 读到末尾，退出循环
        break
    print("当前行：", line.strip())  # strip()去掉换行符
f.close()
```

- readlines()：读取所有行，返回列表

  把文件每一行作为列表的一个元素，适合处理“需要按行操作”的场景（如统计行数）

```python
# 读取所有行，统计行数
f = open("test.txt", "r", encoding="utf-8")
lines = f.readlines()  # 返回列表：["第二次写入...\n", "第三次写入...\n"]
print("总行数：", len(lines))  # 输出：2
# 遍历每行处理
for line in lines:
    print("处理后：", line.strip())  # 去掉换行符
f.close()
```

写入

写入内容到文件。

- write(content)：写入字符串 / 二进制数据

- 文本模式：content必须是字符串
- 二进制模式：content必须是字节串（如b"hello"）

```
# 写入字符串
f = open("write_test.txt", "w", encoding="utf-8")
f.write("第一行内容\n")  # 写单行
f.write("第二行内容\n")  # 继续写
f.close()  # 关闭后内容才会保存
```

- writelines(lines)：写入列表（元素为字符串）

  用于批量写入多行内容，列表中每个元素是一行字符串（需手动加\n）

```
# 批量写入多行
lines = ["张三,20\n", "李四,19\n", "王五,21\n"]  # 列表元素带换行符
f = open("student.txt", "w", encoding="utf-8")
f.writelines(lines)  # 一次性写入列表
f.close()

# 读取验证
f = open("student.txt", "r", encoding="utf-8")
print(f.read())
f.close()
```

安全操作 with

场景：我们在以后开发的时候，经常容易忘记`close()`，就会导致文件资源泄露。with语句能**自动关闭文件**（退出with块时触发），是 Python 推荐的最佳操作方式。

语法格式：

```
with open(文件路径, 模式, encoding=编码) as 文件句柄:
    # 缩进内执行读写操作
    读/写代码
# 退出缩进后，文件自动关闭，无需手动调用close()
```

案例：用with重构之前的代码

```python
# 读文件：with自动关闭
with open("test.txt", "r", encoding="utf-8") as f:
    content = f.read()
    print(content)  # 缩进内操作文件

# 写文件：with自动关闭
with open("with_write.txt", "w", encoding="utf-8") as f:
    f.write("用with写的内容，无需手动close()\n")
    f.write("自动关闭更安全！")

# 验证：文件已关闭
# print(f.read())  # 报错：ValueError: I/O operation on closed file.
```

处理二进制文件

文本文件用r/w/a模式，二进制文件（如图片、视频、Excel）必须用rb/wb/ab模式，且**不指定encoding**。

案例：复制一张图片

```python
# 思路：用rb读原图片 → 用wb写新图片
with open("original.jpg", "rb") as read_f:  # 二进制读
    img_data = read_f.read()  # 读取图片的二进制数据

with open("copy.jpg", "wb") as write_f:  # 二进制写
    write_f.write(img_data)  # 写入二进制数据

print("图片复制完成！")
```

注意：二进制模式下，read()返回的是字节串（如b'\xff\xd8\xff\xe0\x00\x10'），不能直接用字符串操作。

文件指针与编码问题

主要介绍文件操作的进阶技巧，学习目标：

1. 了解文件指针
2. 掌握文件的编码，知道为什么会产生**乱码问题**

文件指针

文件指针（光标）：控制读写位置

文件指针类似“书签”，记录当前读写的位置（默认在文件开头）。用seek(offset，whence)调整指针位置

- offset：偏移量（正数向右移，负数向左移）
- whence：基准位置（0 = 文件开头，1 = 当前位置，2 = 文件末尾）

案例：a+模式下读内容（需移动指针）

```
# a+模式默认指针在文件末尾，直接读会读不到内容
with open("test.txt", "a+", encoding="utf-8") as f:
    f.write("追加一行，用于测试指针\n")
    # 移动指针到文件开头，才能读到内容
    f.seek(0, 0)  # 0=开头，偏移0字节
    content = f.read()
    print("读取全部内容：", content)
```

注意：偏移量是指偏移的字节量，在utf-8编码中，一个中文字符通常占用3个字节

文本模式下offset只能为非负值，往后移，中间，或者尾部的相对位置时，只能是0；汉字偏移要是3的倍数 二进制模式下才可以往前偏，二进制模式下offset才能为负值

乱码问题

因编码不匹配导致文件乱码，核心原则：**读文件的编码必须和写文件的编码一致**。

常用编码：

- utf-8（通用，支持中文）
- gbk（Windows 默认中文编码）
- 其他编码：
  - ASCII、欧洲码表、大五码...

```python
# 1. 用gbk编码写文件
with open("gbk_file.txt", "w", encoding="gbk") as f:
    f.write("这是GBK编码的文件")

# 2. 用utf-8读会乱码（错误示范）
with open("gbk_file.txt", "r", encoding="utf-8") as f:
    # print(f.read())  # 报错或显示乱码：����GBK编码的文件

# 3. 用gbk读才正确（正确示范）
with open("gbk_file.txt", "r", encoding="gbk") as f:
    print("正确读取：", f.read())  # 输出：这是GBK编码的文件
```

练一练

1 小文本文件复制与备份

需求： 把本地操作系统中的一个文本文件，复制到系统中的另外一个路径下

2 大文本文件的复制

需求：打开一个已有文件，逐行读取内容，并顺序写入到另外一个文件

1 统计文本文件的单词数

需求：读取article.txt，统计文件中总单词数（单词用空格分隔）

总结

1 **核心流程**：打开（open()）→ 读写（read()/write()）→ 关闭（close()或with自动关闭）

2 **必记模式**：r（读）、w（写覆盖）、a（追加）、rb/wb（二进制）

3 **安全操作**：优先用with语句，避免忘记关闭文件

4 **避坑要点**：路径写对（相对 / 绝对）、编码一致（utf-8/gbk）、大文件用readline()逐行读。

目录文件的常用操作

在 **终端 / 文件浏览器**、 中可以执行常规的 **文件 / 目录** 管理操作，例如：

创建、重命名、删除、改变路径、查看目录内容...

在 Python 中，如果希望通过程序实现上述功能，**需要导入 os模块**

文件操作

序号	方法名	说明	示例
01	rename	重命名文件	os.rename(源文件名，目标文件名)
02	remove	删除文件,不能删除文件夹	os.remove(文件名)

提示：文件或者目录操作都支持 **相对路径** 和 **绝对路径**

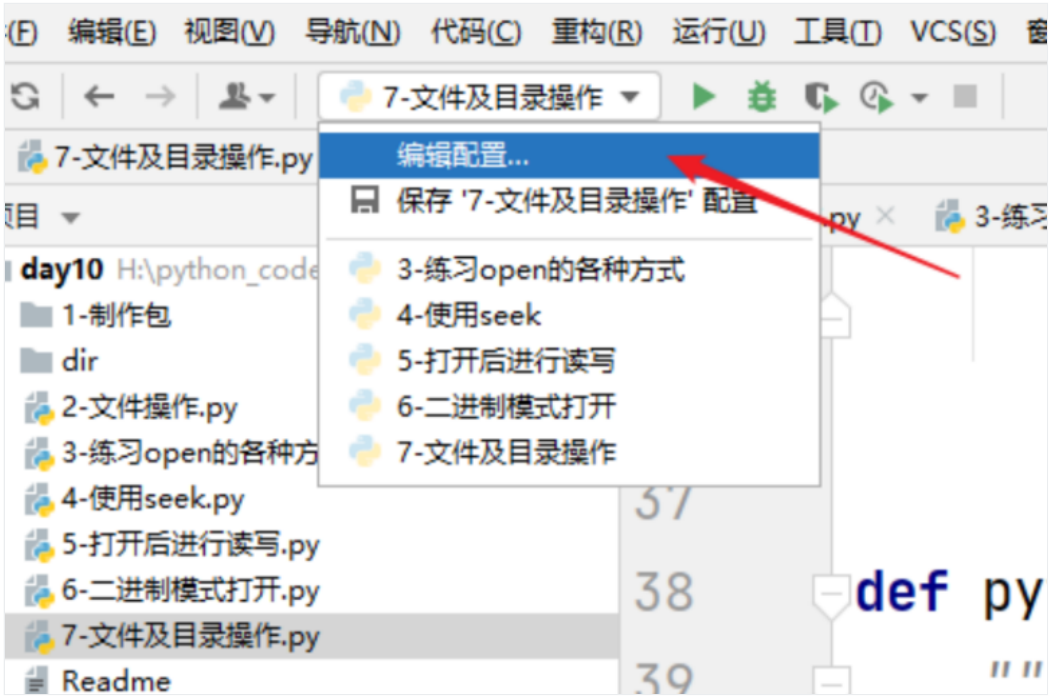
目录操作

序号	方法名	说明	示例
01	listdir	列出指定目录下的所有文件	os.listdir(目录名)
02	mkdir	创建目录文件	os.mkdir(目录名)
03	rmdir	删除目录文件，注意只能删除空的	os.rmdir(目录名)
04	getcwd	获取当前目录	os.getcwd()
05	chdir	修改工作目录	os.chdir(目标目录)
06	path.isdir	判断是否是文件夹	os.path.isdir(文件路径)

获取文件大小，是stat（windows的权限不正确，Linux权限正确）

```
import os
file_info=os.stat('file.txt')
print('size{},uid{},mtime{}'.format(file_info.st_size,file_info.st_uid
, \
                      file_info.st_mtime))
from time import strftime
from time import gmtime
#把秒数转为字符串时间
print(strftime("%Y-%m-%d %H:%M:%S", gmtime(file_info.st_mtime)))
```

Python程序执行时如何传递参数



命令行也可以传递参数。

import sys

通过print(sys.argv)即可打印传递的所有参数,sys.argv是一个列表。

eval读配置

当我们把一个字典放在文件中时，如何快速把它变为一个字典变量呢？