# 一、文件操作

## 1. 文件处理基础

> **文件**：存储在计算机存储设备上的数据流（文本、图片、视频等），Python 主要处理文本文件（如 `.txt`、`.csv`）。
>
> **路径**：文件在计算机中的位置分两种：
>
> - **绝对路径**：从根目录开始的完整路径（如 `C:\data\test.txt` 或 `/home/user/test.txt`）。
> - **相对路径**：相对于当前程序所在目录的路径（如 `data/test.txt` 表示同级 `data` 文件夹下的 `test.txt`）。
> - **编码**：文本文件的字符编码方式（常用 `utf-8`，支持中文）。

## 2. 文件的打开与关闭

> 使用 `open()` 函数打开文件，处理完成后需用 `close()` 关闭（释放资源）。

> （1）基本语法

```
# 打开文件
文件对象 = open(文件路径, 模式, encoding=编码方式)

# 操作文件（读/写）

# 关闭文件
文件对象.close()
```

> （2）打开模式

> 常用模式：

| 模式 | 描述 |
| --- | --- |
| `r` | 只读模式（默认），文件不存在则报错 |
| `w` | 写入模式，文件不存在则创建，存在则清空内容 |
| `a` | 追加模式，文件不存在则创建，新内容添加到末尾 |
| `r+` | 读写模式，可读取和写入 |
| `w+` | 读写模式，会先清空文件内容 |
| `a+` | 读写模式，新内容追加到末尾 |

> （3）注意：文本模式下需指定 `encoding`（如 `encoding="utf-8"`），避免中文乱码。

> 入门案例

```
# 写入内容

# 1、打开文件
f = open('a.txt', 'w')
# 2、写入内容
f.write('Hello,Python!')
# 3、关闭文件
f.close()

# 读取内容
# 1、打开文件
f = open('a.txt', 'r')
# 2、读取内容
print(f.read())
# 3、关闭文件
f.close()

# 中文乱码问题  写
f = open('a.txt', 'w',encoding='utf-8')
f.write('人生苦短我用Python!')
f.close()

# 中文乱码问题  读
f = open('a.txt', 'r',encoding='utf-8')
print(f.read())
f.close()
```

## 3. 文件读取操作

> 读取文本文件内容的常用方法：

| 方法 | 描述 |
| --- | --- |
| `read()` | 读取整个文件内容，返回字符串 |
| `read(n)` | 读取前 `n` 个字符 |
| `readline()` | 读取一行内容 |
| `readlines()` | 读取所有行，返回列表（每行作为元素） |

> 案例 1：读取整个文件

```python
# 打开文件（只读模式）
file = open("test.txt", "r", encoding="utf-8")

# 读取内容
content = file.read()
print("文件内容：")
print(content)

# 关闭文件
file.close()
```

> 案例 2：逐行读取文件
>
> 适合大文件（避免一次性加载到内存）：

```python
file = open("test.txt", "r", encoding="utf-8")

print("逐行读取：")
# 方法1：用 readline() 循环
line = file.readline()
while line:
    print(line.strip())  # strip() 去除换行符
    line = file.readline()

file.close()

# 方法2：直接遍历文件对象（更简洁）
file = open("test.txt", "r", encoding="utf-8")
for line in file:
    print(line.strip())
file.close()
```

## 4. 文件写入操作

> 写入内容到文件的常用方法：
>
> - `write(str)`：写入字符串。
> - `writelines(iterable)`：写入可迭代对象（如列表，需自行添加换行符）。

> 案例 3：写入文件（覆盖模式 `w`）

```python
# 打开文件（写入模式，若文件存在则清空）
file = open("output.txt", "w", encoding="utf-8")

# 写入内容
file.write("第一行文本\n")  # \n 表示换行
file.write("第二行文本\n")

# 写入列表（需手动加换行符）
lines = ["第三行文本", "第四行文本"]
file.writelines([line + "\n" for line in lines])

# 关闭文件
file.close()
```

> 注意：`w` 模式会清空原有内容，如需保留原有内容并添加新内容，用 `a` 模式。

> 案例 4：追加内容（`a` 模式）

```
file = open("output.txt", "a", encoding="utf-8")
file.write("这是追加的内容\n")
file.close()
```

## 5. 上下文管理器（`with` 语句）

> 自动管理文件关闭，避免忘记 `close()` 导致的资源泄露，推荐优先使用。
>
> 语法:

```
with open(文件路径, 模式, encoding="utf-8") as 文件对象:
    # 操作文件（缩进代码块）
    # 无需手动调用 close()，退出代码块后自动关闭
```

> 案例 5：用 `with` 语句读写文件

```
# 读取文件
with open("test.txt", "r", encoding="utf-8") as f:
    content = f.read()
    print("读取内容：", content)

# 写入文件
with open("new_file.txt", "w", encoding="utf-8") as f:
    f.write("使用 with 语句写入的内容\n")
```

## 6. 文件操作综合案例

> 案例 6：复制文件内容
>
> 将 `source.txt` 的内容复制到 `copy.txt`：

```
with open("source.txt", "r", encoding="utf-8") as source:
    content = source.read()

with open("copy.txt", "w", encoding="utf-8") as copy:
    copy.write(content)

print("文件复制完成")
```

> 案例 7：统计文件中的单词数量

```
Python is a powerful programming language. Python is widely used for web development, data analysis, artificial intelligence, and scientific computing. Python's simplicity and readability make it a great choice for beginners and experts alike.

The Python community is large and active, with many libraries and frameworks available.
Learning Python can open doors to many career opportunities in technology.

Python, python, PYTHON - all refer to the same amazing language!
```

```python
def count_words(filename):
    """统计文件中单词的数量（以空格分隔）"""
    with open('c.txt','r',encoding='utf-8') as f:
        count = 0
        for line in f:
            line = line.strip() # 去除空白字符
            line = line.replace(',','').replace('.','') # 替换,和.
            words_list = line.split(' ') # 按照空格进行切割
            count += len(words_list)
        return count

word_count = count_words("article.txt")
print(f"文件中的单词数量：{word_count}")
```

### 7. 常见错误与注意事项

1. **文件路径错误**：路径不存在或拼写错误，导致 `FileNotFoundError`。
2. **忘记关闭文件**：长期运行的程序可能耗尽系统资源，推荐用 `with` 语句。
3. **编码问题**：未指定 `encoding="utf-8"` 或编码不匹配，导致中文乱码或 `UnicodeDecodeError`。
4. **模式使用错误**：用 `r` 模式写入文件（报错 `IOError`），或用 `w` 模式误删文件内容。
5. **大文件读取**：用 `read()` 一次性读取大文件（如 1GB），可能导致内存溢出，应逐行读取。

### 8. 课堂练习

1. 编写程序，读取 `scores.txt`（每行一个学生成绩），计算平均分并写入 `result.txt`。
2. 用 `a` 模式向 `log.txt` 中追加一条日志，格式为 “[2025-10-01 12:00:00] 操作内容”（时间可用固定字符串）。
3. 读取 `poem.txt`，统计其中包含 “月” 字的行数(文件内容自定义)。

## 二、Python模块

### 1. 模块的基本概念

- **定义**：模块是一个包含 Python 代码的文件（扩展名为 `.py`），可包含函数、类、变量和可执行代码。
- **作用**：
  - 将代码拆分到不同文件，便于管理和维护（模块化）。
  - 实现代码复用（多个程序可共享同一模块）。
  - 避免命名冲突（不同模块中的同名函数 / 变量互不影响）。

生活类比：像乐高积木 —— 每个模块是一个功能组件，可单独开发，也可组合成复杂系统。

## 2. 模块的导入与使用

使用 `import` 语句导入模块，然后通过模块名访问其中的内容。

（1）基本导入方式：`import 模块名`

```
# 导入标准库中的 math 模块（包含数学相关功能）
import math

# 使用模块中的函数/变量（格式：模块名.功能）
print("圆周率：", math.pi)  # 访问变量
print("3的平方：", math.pow(3, 2))  # 调用函数
print("向上取整：", math.ceil(2.3))  # 调用函数
```

（2）导入指定内容：`from 模块名 import 功能`

直接导入模块中的特定函数 / 变量，无需通过模块名访问。

```
# 从 math 模块导入 pi 和 sqrt 功能
from math import pi, sqrt

print("圆周率：", pi)  # 直接使用变量
print("16的平方根：", sqrt(16))  # 直接调用函数
```

（3）导入所有内容：`from 模块名 import \*`

导入模块中所有公开的功能（不推荐，可能导致命名冲突）。

```
from math import *

print('列表求和：',fsum([1,2,3,4,5])) # 直接使用fsum函数
```

（4）给模块 / 功能起别名：`as`

解决命名冲突或简化代码。

```
# 给模块起别名
import math as m
print("2的自然对数：", m.log(2))

# 给功能起别名
from math import power as pow2  # 注意：math 中实际是 pow，此处为示例
print("2的3次方：", pow2(2, 3))
```

## 3. 自定义模块

自己编写 `.py` 文件作为模块，供其他程序导入使用。

（1）创建自定义模块

新建文件 `my_module.py`，写入以下内容：

```python
# my_module.py
"""这是一个自定义模块，包含简单的功能"""

# 变量
version = "1.0"

# 函数
def add(a, b):
    """计算两个数的和"""
    return a + b

def multiply(a, b):
    """计算两个数的积"""
    return a * b

# 执行代码（模块被导入时会执行，通常用 __name__ 控制）
if __name__ == "__main__":
    # 仅在当前模块直接运行时执行，被导入时不执行
    print("这是 my_module 模块的自测代码")
    print(add(2, 3))  # 输出：5
```

关键：`if __name__ == "__main__":` 用于模块自测，避免导入时执行多余代码。

（2）使用自定义模块

在同一目录下新建 `main.py`，导入 `my_module`：

```python
# main.py
import my_module

print("模块版本：", my_module.version)
print("2+3 =", my_module.add(2, 3))
print("4×5 =", my_module.multiply(4, 5))

# 也可使用 from...import 导入
from my_module import add
print("10+20 =", add(10, 20))
```

## 4. 包（Package）

当模块数量较多时，用包（包含 `__init__.py` 的文件夹）组织模块，形成层级结构。

（1）包的结构

```
my_package/          # 包文件夹
├── __init__.py      # 包的初始化文件（可空，标识为包）
├── module1.py       # 子模块1
└── module2.py       # 子模块2
```

(2) `__init__.py` 的作用

- 标识该文件夹为 Python 包。
- 可在其中定义包的公共接口（如指定默认导入内容）。

(3) 导入包中的模块

```
# 导入包中的模块
import my_package.module1

# 导入包中的模块并起别名
import my_package.module2 as m2

# 从包中导入模块的功能
from my_package.module1 import func1
```

## 5. 常用标准库模块

Python 内置了大量实用模块（标准库），无需安装即可使用。

(1) `random` 模块（随机数）

```
import random

# 生成 0-1 之间的随机浮点数
print(random.random())

# 生成 1-10 之间的随机整数
print(random.randint(1, 10))

# 从列表中随机选择一个元素
fruits = ["苹果", "香蕉", "橙子"]
print(random.choice(fruits))
```

(2) `datetime` 模块（日期时间）

```
import datetime

# 获取当前时间
now = datetime.datetime.now()
print("当前时间：", now)
print("年份：", now.year)
print("月份：", now.month)

# 格式化时间
print("格式化时间：", now.strftime("%Y-%m-%d %H:%M:%S"))
```

(3) `os` 模块（操作系统交互）

```python
import os

# 获取当前工作目录
print("当前目录：", os.getcwd())

# 列出目录下的文件
print("目录内容：", os.listdir("."))

# 创建文件夹
# os.mkdir("new_folder")  # 仅创建一级目录
# os.makedirs("a/b/c")    # 创建多级目录
```

## 6. 综合案例

> 案例：学生成绩分析工具（模块化实现）
>
> 创建 `score_functions.py`（功能模块）：

```python
# score_functions.py
def calculate_average(scores):
    """计算平均分"""
    return sum(scores) / len(scores) if scores else 0

def get_grade(score):
    """返回成绩等级"""
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 60:
        return "C"
    else:
        return "D"
```

> 创建 `data_io.py`（文件操作模块）：

```python
# data_io.py
def read_scores(filename):
    """从文件读取成绩"""
    scores = []
    with open(filename, "r", encoding="utf-8") as f:
        for line in f:
            scores.append(float(line.strip()))
    return scores

def write_result(filename, avg, grade_counts):
    """写入分析结果"""
    with open(filename, "w", encoding="utf-8") as f:
        f.write(f"平均分：{avg:.1f}\n")
        f.write("等级分布：\n")
        for grade, count in grade_counts.items():
            f.write(f"{grade}: {count}人\n")
```

> 创建 `main.py`（主程序）：

```python
# main.py
import score_functions as sf
import data_io as io

# 读取数据
scores = io.read_scores("scores.txt")
if not scores:
    print("没有成绩数据")
    exit()

# 分析数据
avg = sf.calculate_average(scores)
grade_counts = {"A": 0, "B": 0, "C": 0, "D": 0}
for s in scores:
    grade = sf.get_grade(s)
    grade_counts[grade] += 1

# 保存结果
io.write_result("analysis.txt", avg, grade_counts)
print("分析完成，结果已保存到 analysis.txt")
```

## 7. 课堂练习

1. 编写一个自定义模块 `string_utils.py`，包含 `reverse(s)`（反转字符串）和 `count_vowels(s)`（统计元音字母数量）函数，在另一个程序中导入使用。
2. 使用 `random` 模块编写一个猜数字游戏：程序随机生成 1-100 的数字，用户最多猜 5 次，每次提示“太大”或“太小”。
3. 使用 `os` 模块遍历当前目录下所有 `.txt` 文件，并统计总个数。

## 8. 常见错误与注意事项

1. **模块找不到**：模块文件不在搜索路径中，或文件名与标准库模块重名（如命名为 `math.py` 会覆盖标准库 `math`）。
2. **循环导入**：模块 A 导入模块 B，同时模块 B 导入模块 A，导致报错。
3. **命名冲突**：使用 `from...import *` 导入多个模块时，可能出现同名功能覆盖。
4. **包结构错误**：忘记在包文件夹中添加 `__init__.py`，导致无法识别为包。