# 第七章 文件概述

# 主要内容

- 文件的概念
- 文件基本操作
- 示例程序
- JSON（*）



# 文件的概念

- 文件是存储在持久化存储介质（外存储器）上的有序数据集合。
- 每个文件都有唯一的名称和类型。
- 文件操作用于数据的持久化存储、交换和共享。
- Python 通过文件对象来操作文件，文件可以看作是一个数据流。

# 文件类型

从逻辑上可分为二大类：示例1

- **文本文件：** 主要存储人类可读的字符数据。例如：用ASCII, UTF-8, GBK 等编码的字符。用文本编辑器打开时，程序会按照相应的编码将文件中的字节转换成我们可以理解的文字。如：.txt 文件, .csv 文件, .html 文件, .py 文件（源代码本质上是文本）。

- **二进制文件：**存储的是**任意格式的原始字节数据**。这些字节数据不一定对应于可打印字符，它们可能代表图像像素、音频采样、视频帧、程序指令、压缩数据等等。用文本编辑器尝试打开二进制文件时，通常会看到乱码，因为文本编辑器会尝试将任意字节解释为字符。如： .jpg 文件, .mp3 文件, .mp4 文件, .exe 文件, .zip 文件, .dat 文件。

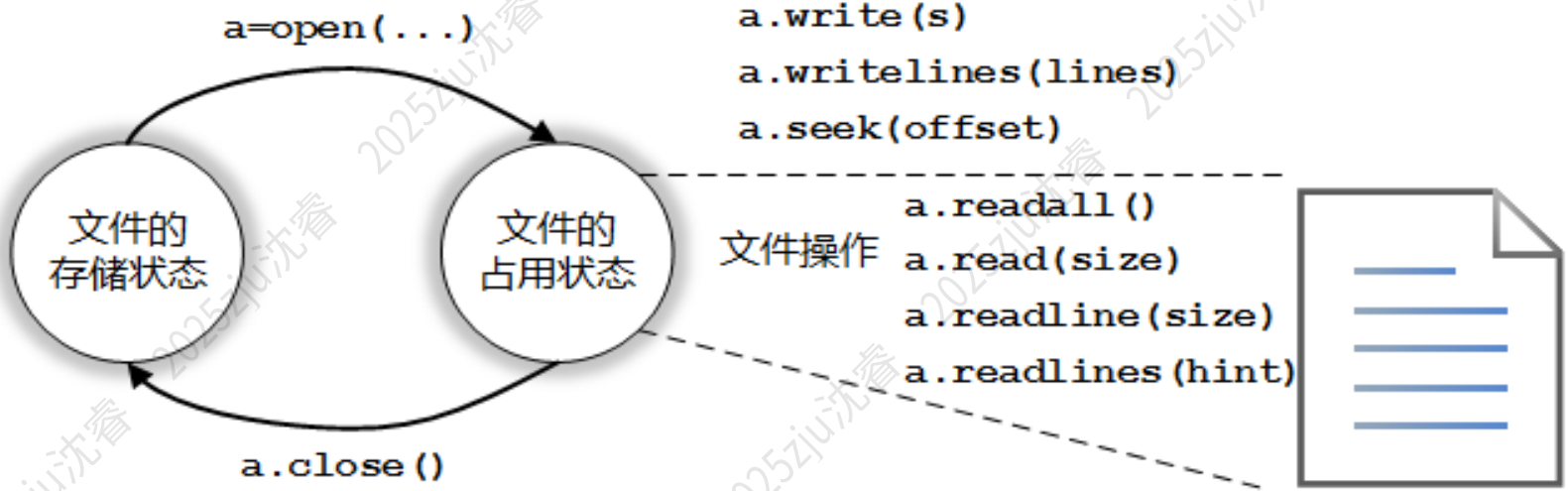
- **问题：你觉得Excel文件属于哪一类？Word文件又属于哪一类？**

# 文件基本操作

# 文件操作核心流程

打开 -> 读写操作 -> 关闭

1. **打开文件 (Open):** 建立与文件的连接，创建一个文件对象。
2. **执行操作 (Read/Write):** 通过文件对象的方法进行数据的读取或写入。
3. **关闭文件 (Close):** 断开与文件的连接，释放系统资源。



- Python通过解释器内置的open()函数打开一个文件，如果打开成功，该函数会返回一个文件对象。这里的fp就是一个变量，用来接收**文件对象**。后续所有对文件的操作，都要通过这个文件对象来完成。

  fp = open(filename, mode[,…])

  - open()函数至少两个参数：**文件名(filename)和打开模式(mode)**。文件名可以是文件的实际名字，也可以是包含完整路径的名字
  - encoding参数可以指定文本文件的字符编码格式，不写就用默认值None，表示 使用平台相关的默认编码。若跨平台，建议显式指定编码，以避免跨平台或处理不同字符集文件时出现乱码问题。

# 打开模式

- open()函数提供7种基本的打开模式

| 打开模式 | 含义 |
| --- | --- |
| 'r' | 只读模式，如果文件不存在，返回异常FileNotFoundError，**默认值** |
| 'w' | 覆盖写模式，文件不存在则创建，存在则完全覆盖原文件 |
| 'x' | 创建写模式，文件不存在则创建，存在则返回异常FileExistsError |
| 'a' | 追加写模式，文件不存在则创建，存在则在原文件最后追加内容 |
| 'b' | 二进制文件模式 |
| 't' | 文本文件模式，**默认值** |
| '+' | 与r/w/x/a一同使用，在原功能基础上增加同时读写功能 |

补充说明：对于简单的二进制数据流，可以使用 b 模式进行基本的读写。 对于具有复杂内部结构的二进制文件格式，通常使用专门的 Python 库来处理，如图像： Pillow；声音： pydub或wave；office：python-docx 、 openpyxl 等。

# 文件路径

- 绝对路径：文件在计算机文件系统中的完整、唯一的地址。它从根目录开始，一直指明到目标文件或目录。

例如：打开d:\python\test.txt的文件（windows系统）

```
infile=open('d:\python\test.txt','r')    X
infile=open(r'd:\python\test.txt','r')   √
infile=open('d:\\python\\test.txt','r')  √
infile=open('d:/python/test.txt','r')    √
```

注：macOS/Linux：以根目录 / 开始

- 相对路径：相对于当前工作目录 的文件或目录的路径。

例如：打开当前目录下名为test.txt的文件进行读操作：

```
infile=open('test.txt','r')
```

deepseek

拓展任务：

了解 os.path.join()使用

# 文件关闭

- 必要性：
  - 防止数据可能的丢失与损坏
  - 防止资源泄漏
  - 避免因文件被锁定而导致其他操作无法进行
- 关闭文件：

方法一： 使用file.close()方法

```
file = open("my_file.txt", "w")
try:
    file.write("一些数据写入文件。\n")
finally:
    file.close()
    print("文件已关闭。")
```

方法二：with open （推荐）

```
with open("my_file.txt", "w") as file:
    file.write("使用 with 语句写入数据。\n")
print("文件已自动关闭。")
```

**简洁：** 代码更简洁，不需要显式编写 close() 。

**安全：** 保证文件总是会被关闭，即使在 with 块中发生异常，也能避免资源泄露和数据丢失的风险。

# 文件的读取操作

- Python提供3个常用的文件内容读取方法

| 方法 | 含义 |
| --- | --- |
| file.read( ) | 读入整个文件内容，如果给出参数，读入前size长度的字符串或字节流。适合小文件，大文件会消耗大量内存。 |
| file.readline( ) | 读入一行内容，如果给出参数，读入该行前size长度的字符串或字节流。适用于逐行读取文件，但需手动控制循环。 |
| file.readlines( ) | 读入所有行，返回以每行为元素形成的列表。 |

- **迭代文件对象:** 读取文本文件内容最常用和内存效率最高的方法。尤其适用于处理大文件。

# 文件的写入操作

- Python提供2个常用的文件写入方法。

| 方法（函数） | 含义 |
| --- | --- |
| file.write(s) | 向文件写入一个字符串或字节流。 |
| file.writelines(lines) | 将存放字符串的可迭代对象（如：列表）写入文件 |

# 控制文件读写位置(*)示例

| 方法（函数） | 含义 |
| --- | --- |
| file.seek(offset,whence) | 改变当前文件操作指针的位置，<br>Offset: 从该值开始<br>whence：0：文件开头； 1: 当前位置； 2: 文件结尾<br>（文本文件，只有0，二进制文件可以是0，1，2） |
| file.tell() | 返回文件对象当前的文件指针位置（以字节为单位）。通常与 seek() 方法一起使用，用于记录和恢复文件中的位置。 |

# 示例程序

# 迭代文件对象

- 文件对象本身是可迭代的。可以直接在 for 循环中迭代文件对象，每次迭代会返回文件的一行（包含行尾换行符）。

```
with open('file.txt', 'r') as f:
    for line in f:
        # 处理每一行
        print(line.strip()) # 去掉行尾的换行符
```

# 文件复制

cj.txt 文件是学生一门课的成绩，它的内容如下：

97 80 93 69 87 90 84 94 75 76 89 83 83 33 72 48 66 86 98

89 89 88 87 63 87 81 100 80 37 68 71 77 98 66 47 29 87 93

96 100 70 85 83 35

需要把这个文件的内容复制到“cjback.txt”文件中。复制文件不需要考虑行结构，用read函数就可以了。

程序：

```
source = open("cj.txt","r")
back = open("cjback.txt","w")
s = source.read()
back.write(s)
source.close()
back.close()
```

# 文件合并

编写程序，有两个文本文件file1.txt和file2.txt，把这两个文本文件的每一行交替合并到file.txt文件中，也即file.txt一行file2.txt一行，最后把行数较多的文件剩余内容全部写入file.txt文件。

思路：

- 读打开file1.txt和file2.txt；创建file.txt文件，写打开。
- 循环交替读fiel1.txt和file2.txt的每一行，写入到file.txt中
- 判断哪个文件先全部读入，另一个文件全部内容写入file.txt

# 代码

```
f1 = open('file1.txt',"r")
f2 = open('file2.txt',"r")
f = open('file.txt', "w")
flag=1
while True:
      line1=f1.readline()
      if line1:
            f.writelines(line1)
      else:
            flag=1   #交替读文件，标记哪个文件先结束
            break
      line2=f2.readline()
      if line2:
            f.writelines(line2)
      else:
            flag=2
            break
if flag==1:   #没有读完的文件继续读出
    f.writelines(f2.readlines()) #file1文件先全部读入
else:
    f.writelines(f1.readlines()) #file2文件先全部读入
f1.close()
f2.close()
f.close()
```

# 计算平均分

文件score.txt是学生一学期的成绩，每一行代表一个学生的成绩，由笔试、平时和实验三部分构成。编写程序计算每一个学生的总评成绩，并写入out_score.txt文件。（总评=笔试*50%+平时* 25%+实验*25%）

```
学号 姓名 专业 笔试 平时 实验
2050921018 詹延峰 计算数学 65 85 76
2050921036 李小鹏 金融学类 86 95 85
2050921039 裴凡法 经济学类 86 95 65
2040912116 茅舒瑶 社会保障 90 95 100
2050912017 陈见影 化学工程 62 75 92
2050912064 梅钦钦 材料科学 87 95 80
2050109153 王影平 大气科学 86 89 72
050151003 韩平医 化学工程 82 99 60
```

score.txt

| 学号 | 姓名 | 专业 | 笔试 | 平时 | 总评成绩 |
|---|---|---|---|---|---|
| 2050921018 | 詹延峰 | 计算数学 | 65 | 85 | 72.75 |
| 2050921036 | 李小鹏 | 金融学类 | 86 | 95 | 88.00 |
| 2050921039 | 裴凡法 | 经济学类 | 86 | 95 | 83.00 |
| 2040912116 | 茅舒瑶 | 社会保障 | 90 | 95 | 93.75 |
| 2050912017 | 陈见影 | 化学工程 | 62 | 75 | 72.75 |
| 2050912064 | 梅钦钦 | 材料科学 | 87 | 95 | 87.25 |
| 2050109153 | 王影平 | 大气科学 | 86 | 89 | 83.25 |
| 050151003 | 韩平医 | 化学工程 | 82 | 99 | 80.75 |

out_score.txt

# 步骤1:读入多行文本

```
with open("score.txt", "r") as infile:
    lines = infile.readlines()
```

['学号 姓名 专业 笔试 平时 实验\n', '2050921018 詹延峰 计算数学 65 85 76\n', '2050921036 李 小鹏 金融学类 86 95 85\n', '2050921039 裴凡法 经济学类 86 95 65\n', '2040912116 茅舒瑶 社会保障 90 95 100\n', '2050912017 陈 见影 化学 工程 62 75 92\n', '2050912064 梅 钦钦 材料科学 87 95 80\n', '2050109153 王影平 大 气科学 86 89 72\n', '2050151003 韩平医 化学 工程 82 99 60\n']

# 步骤2:计算平均分

```
with open("score.txt", "r") as infile:
    lines = infile.readlines()
```

```
# 处理每一行数据
for line in lines[1:]:
    data = line.split()
    total_score = int(data[3]) * 0.5 + int(data[4]) * 0.25 + int(data[5]) * 0.25
    formatted_line = f"{data[0]:^12}{data[1]:^7}{data[2]:^7}{data[3]:^6}{data[4]:^7}{total_score:^10.2f}\n"
    outfile.write(formatted_line)
    print(formatted_line,end='')
```

# 词频统计

读入一个英文文本文件，统计其中不同单词的出现次数，并输出词频最高的前10%的单词，将结果以文件的形式保存。

所谓“单词”，是指连续字符串，只包括大小写字母、数字和下划线，其它字符均被认为是单词分隔符。“单词”不区分英文大小写，例如“PAT”和“pat”被认为是同一个单词。

输入要求：

读入文本文件

输出要求：

按照词频递减的顺序，按照“词频:单词”的格式输出词频最大的前10%的单词。若有并列，则按递增字典序输出，将结果写入文本文件

# 实现过程

1. 读入文本文件

```
with open("Potter.txt","r") as fin:
    strs = fin.read()
```

2. 处理文本（剔除非单词字符，不区分大小写处理）
3. 词频统计
4. 将结果写入文件
5. 可视化展示（*）

# 分割单词

```
for k in set([i for i in strs if i.isalnum()==False and i!='_']):
    strs = strs.replace(k, ' ')    #其它字符均认为是单词分隔符
strs = strs.rstrip(' ').lower().split()    #全部变小写
```

```
words = []
current_word = []
for char in strs:
    if char.isalnum() or char == '_':
        current_word.append(char.lower())  # 合法字符，加入当前单词
    else:
        if current_word:  # 遇到非法字符，结束当前单词
            words.append(''.join(current_word))
            current_word = []

if current_word:  # 添加最后一个单词
    words.append(''.join(current_word))
```

```
strs = re.sub(r'[^a-zA-Z0-9\s_]', ' ', strs).lower().split()
```

# 产生词频字典

```
counts = dict()
for i in strs:
    if k not in counts:
        counts[k] = 1
    else:
        counts[k] += 1
```

```
counts[k]=counts.get(k,0)+ 1
```

# 排序及输出

词频递减的顺序，若有并列，则按递增字典序

```
ans = sorted(counts.items(),key = lambda x:(-x[1],x[0]))
```

若改成以下代码有什么不同？

ans=sorted(counts.items(), key=lambda x:(x[1], x[0]),reverse=True)

# 结果输出及写入文本文件

选取词频最高的前10%的单词作为输出结果

```
for i in range(0,int(0.1*len(counts))):
    print(str(ans[i][1])+':'+ans[i][0])
```

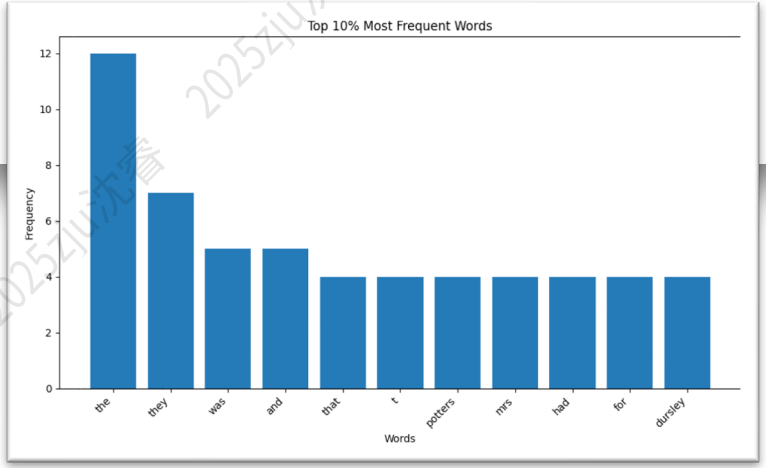
结果写入文本文件

```
with open("Potter_result.txt","w") as fout:
    for i in range(0,int(0.1*len(counts))):
        fout.write(str(ans[i][1])+':'+str(ans[i][0]+'\n'))
        fout.writelines([str(ans[i][1]),':',str(ans[i][0]),'\n'])
```

# 可视化结果（*）

```
# 结果图形化
plt.figure(figsize=(10, 6))  # 调整图表大小
plt.bar([x[0] for x in ans[:int(0.1 * len(counts))]], [x[1] for x in ans[:int(0.1 * len(counts))]])
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 10% Most Frequent Words')
plt.xticks(rotation=45, ha='right') # 旋转x轴标签
plt.tight_layout()  # 自动调整子图参数
plt.show()

# 以Excel表形式输出结果
df = pd.DataFrame(ans, columns=['Word', 'Quantity'])
df.to_excel("result.xlsx", index=False)
```



# 输出结果为excel文件（*）

import pandas as pd

```
df = pd.DataFrame(ans, columns=['Word', 'Quantity'])
df.to_excel("result.xlsx", index=False)
```

# JSON （*）

# JSON 简介

- JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，可用于在不同的系统和应用程序之间传输和存储数据。
- JSON具有可读性强、易于解析和处理的特点，使其成为人工智能领域中广泛使用的数据格式。
- JSON文件主要有两种结构：
  - “键/值”对的集合：不同的语言中，它被理解为对象，记录，结构，字典，哈希表，有键列表，或者关联数组。Python中对应**字典**类型。
  - 值的有序列表：在大部分语言中，它被理解为数组。Python中对应列表类型。

# 各种json对象

## 简单的JSON对象

```
{
  "name": "Lihua",
  "age": 18,
  "skills": ["Python", "Machine Learning"]
}
```

## 嵌套结构的JSON对象

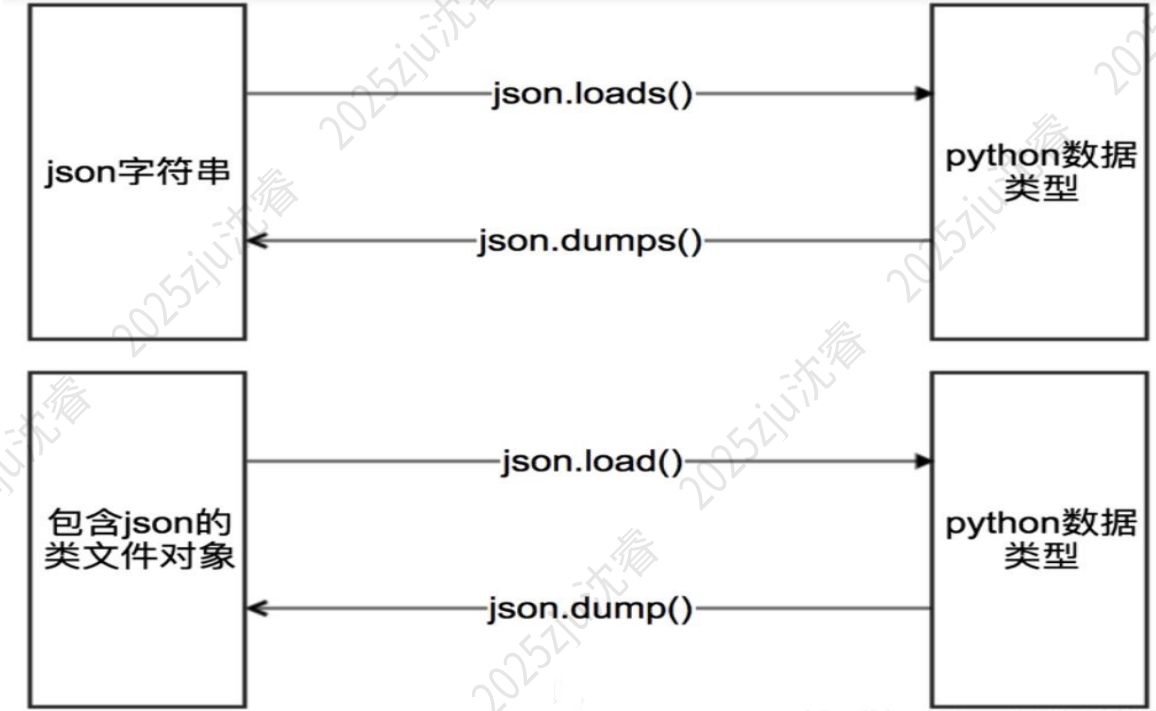
```
{
  "person": {
    "name": "Lihua",
    "age": 18,
    "address": {
      "street": "Jiefang Street",
      "city": "Hangzhou",
      "zip": "310010"
    }
  },
  "isGraduate": false
}
```

## 列表字典组合的JSON对象

```
[
  {
    "title": "Python Programming",
    "author": "Robert",
    "isbn": "97801234567",
    "year": 2022,
    "price": 22.5
  },
  {
    "title": "Math",
    "author": " McDowell",
    "isbn": "9780154782853",
    "year": 2015,
    "price": 30.5
  }
]
```

# 引入json模块

标准模块: import　json



json.dumps()：用于将 Python 对象（通常是字典）编码成 JSON 格式的字符串。

这个函数提供了多个参数来自定义输出，包括美化输出和排序键等。

```python
import json
data = {
    'name': 'Lihua',
    'age': 18,
    'is_student': False,
    'courses': ['Math', 'Science'],
    'city': None
}
json_str = json.dumps(data)
print(json_str)
```

json.dump()：适用于需要将 Python 对象直接写入文件或类似文件对象的场景，无需中间处理字符串

```python
import json

data = {
    'name': 'Lihua',            # 字符串类型
    'age': 18,                  # 整数类型
    'is_student': False,        # 布尔类型
    'courses': ['Math', 'Science'],  # 列表类型，包含字符串
    'city': None                # None类型，表示空值或未知
}

# 使用with语句打开一个文件，模式为'w'（写入），并指定编码为'utf-8'
with open('output.json', 'w', encoding='utf-8') as f:
    # 使用json.dump()方法将字典序列化为JSON格式，并写入文件
    # indent=4参数指定缩进级别为4个空格，使输出的JSON更易于阅读
    # sort_keys=True参数指定在序列化时按键名排序
    json.dump(data, f, indent=4, sort_keys=True)
```

json.loads()：将包含JSON数据的字符串解析成Python数据结构

```
import json

# 假设我们有一个包含 JSON 数据的字符串
json_str = '{"name": "Lihua", "age": 18, "is_student": false}'
data = json.loads(json_str)
print(data)
```

json.load()：从文件对象中读取并解析JSON数据

```
import json

with open('output.json', 'r') as f:
    data = json.load(f)
print(data)
```

当然可以。在Python中，正则表达式（Regular Expressions）是一种强大的文本处理工具，它允许你通过特定的模式来匹配或查找字符串中的特定部分。

让我们分解这个正则表达式 `r'[^a-zA-Z0-9\s_]'`：

- `r`：这个前缀表示这是一个原始字符串（raw string），它告诉Python不要对字符串中的反斜杠进行转义。这在正则表达式中很重要，因为正则表达式经常需要使用反斜杠来表示特殊字符。
- `[]`：方括号用于定义一个字符集，匹配方括号内的任意一个字符。
- `^`：当这个符号出现在字符集（`[]`）的开头时，它表示取反，即匹配不在字符集内的字符。
- `a-zA-Z`：这表示匹配任何小写或大写字母。
- `0-9`：这表示匹配任何数字。
- `\s`：这是一个特殊字符，表示匹配任何空白字符，包括空格、制表符、换行符等。
- `_`：这表示匹配下划线字符。

将这些组合起来，`[^a-zA-Z0-9\s_]` 匹配任何不是字母、数字、空白字符或下划线的字符。

然后，`re.sub(r'[^a-zA-Z0-9\s_]', '', strs)` 这个函数调用的作用是：

- 在字符串 `strs` 中查找所有匹配 `[^a-zA-Z0-9\s_]` 的字符（即所有非字母、非数字、非空白、非下划线的字符）。
- 将这些字符替换为空格 `''`。