1. 文件操作

计算机的文件，就是存储在某种 长期储存设备 上的一段 数据

长期存储设备包括：硬盘、U 盘、移动硬盘、光盘...

文本文件和二进制文件

- 文本文件
  - 可以使用 文本编辑软件 查看
  - 本质上还是二进制文件
  - 例如：python 的源程序
- 二进制文件
  - 保存的内容 不是给人直接阅读的，而是 提供给其他软件使用的
  - 例如：图片文件、音频文件、视频文件等等
  - 二进制文件不能使用 文本编辑软件 查看

1、1 文件的基本操作

操作文件的函数/方法

在 `Python` 中要操作文件需要记住 1 个函数和 3 个方法

序号	函数/方法	说明
01	open	打开文件，并且返回文件操作对象
02	read	将文件内容读取到内存
03	write	将指定内容写入文件
04	close	关闭文件

- `open` 函数负责打开文件，并且返回文件对象

- `read`/`write`/`close` 三个方法都需要通过 文件对象 来调用

1、1、1 读取文件

```
open
```

函数的第一个参数是要打开的文件名（文件名区分大小写）

- 如果文件 存在，返回 文件操作对象
- 如果文件 不存在，会 抛出异常
- `read` 方法可以一次性 读入 并 返回 文件的 所有内容
- `close`方法负责关闭文件
  - 如果 忘记关闭文件，会造成系统资源消耗，而且会影响到后续对文件的访问
- 注意：`read` 方法执行后，会把 文件指针 移动到 文件的末尾

```
# 1. 打开 - 文件名需要注意大小写
file = open("README")

# 2. 读取
text = file.read()
print(text)

# 3. 关闭
file.close()
```

1、1、2 打开文件的方式

- `open` 函数默认以 只读方式 打开文件，并且返回文件对象

语法如下：

```
f = open("文件名", "访问方式")
```

访问方式	说明
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。如果文件不存在，抛出异常
w	以只写方式打开文件。如果文件存在会被覆盖。如果文件不存在，创建新文件
a	以追加方式打开文件。如果该文件已存在，文件指针将会放在文件的结尾。如果文件不存在，创建新文件进行写入
r+	以读写方式打开文件。文件的指针将会放在文件的开头。如果文件不存在，抛出异常
w+	以读写方式打开文件。如果文件存在会被覆盖。如果文件不存在，创建新文件
a+	以读写方式打开文件。如果该文件已存在，文件指针将会放在文件的结尾。如果文件不存在，创建新文件进行写入

提示

- 频繁的移动文件指针，会影响文件的读写效率，开发中更多的时候会以 只读、只写 的方式来操作文件

写入文件示例

```python
# 打开文件
f = open("README", "w")

f.write("hello python! \n")
f.write("今天天气真好")

# 关闭文件
f.close()
```

1、1、3 按行读取文件内容

`read` 方法默认会把文件的 所有内容 一次性读取到内存

如果文件太大，对内存的占用会非常严重

`readline` 方法

`readline` 方法可以一次读取一行内容

方法执行后，会把 文件指针 移动到下一行，准备再次读取

读取大文件的正确姿势

1、2 目录操作

- 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

1、2、1 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	`os.remove(文件名)`

1、2、2 目录操作

序号	方法名	说明	示例
01	listdir	目录列表	`os.listdir(目录名)`
02	mkdir	创建目录	`os.mkdir(目录名)`
03	rmdir	删除目录	`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	`os.path.isdir(文件路径)`

> 提示：文件或者目录操作都支持 相对路径 和 绝对路径

2、excel

> openpyxl 是一个Python库，用于读取/写入Excel 2010 xlsx / xlsm / xltx / xltm文件。
>
> 它的诞生是因为缺少可从Python本地读取/写入Office Open XML格式的库。

> 官方文档:
> https://openpyxl.readthedocs.io/en/stable/

Excel 文件三个对象

```
workbook：  工作簿，一个excel文件包含多个sheet。
sheet：工作表，一个workbook有多个，表名识别，如“sheet1”，“sheet2”等。
cell：  单元格，存储数据对象
```

1、openpyxl 读写单元格时，单元格的坐标位置起始值是（1，1），即下标最小值为1，否则报错!

2、openpyxl 支持直接横纵坐标访问，如 A1，B2...

```
# 模块安装
pip install openpyxl
```

2、1 创建工作簿

只需导入`workbook`对象就可以创建工作簿

```
from openpyxl import Workbook

# 创建工作簿
wb = Workbook()
```

如果想写入数据就需要获取一张表

表明创建时会自动命名。它们按顺序编号（Sheet，Sheet1，Sheet2等）。可以随时通过`worksheet.title`属性更改此名称：

```
# 创建一张表
sheet = wb.active
sheet.title = '表1'

# 创建新表
sheet2 = wb.create_sheet('表2')

# 通过表名获取表
sheet1 = wb['表1']
```

2、2 写入操作

现在我们知道了如何获取工作表，可以开始修改单元格内容了。单元格可以直接作为工作表的键进行访问：

这将使单元格返回A4，如果尚未存在，则创建一个单元格。可以直接分配值：

```
# 写入值
sheet['A1'] = 42
```

还有 `Worksheet.cell()` 方法可以更方便操作。

```
sheet.cell(row=2, column=5).value = 99
sheet.cell(row=3, column=5, value=100)
```

逐行写

```
# 行内容可以直接被列表覆盖
# ws.append(iterable)
# 添加一行到当前sheet的最底部（即逐行追加从第一行开始）
# iterable必须是list,tuple,dict,range,generator类型的。
# 1，如果是list，将list从头到尾顺序添加。
# 2，如果是dict，按照相应的键添加相应的键值。
ws.append(['This is A1',  'This is B1',  'This is C1'])
ws.append({'A': 'This is A1',  'C': 'This is C1'})
ws.append({1: 'This is A1',  3: 'This is C1'})
```

同时可以遍历单元格

2、3 案例:

将九九乘法表写入excel表

```
# 写入九九乘法表
ws9 = wb.create_sheet('九九乘法表')
i = 1
while i < 10:
    j = 1
    while j <= i:
        # print('{} * {} = {}'.format(j,  i,  j * i),  end='\t')
        ws9.cell(row=i,  column=j).value = '{} * {} = {}'.format(j,  i,  j * i)
        j += 1
    i += 1
```

2、4 读表操作

```
# 打开文件:
from openpyxl import load_workbook
wb = load_workbook('sample.xlsx')
```

```
# 通过表名获取获取 sheet：
table = wb['九九乘法表']

# 获取行数和列数：
rows = table.max_row
cols = table.max_column
print(rows, cols)

# 获取单元格值：
# 获取表格内容,是从第一行第一列是从1开始的,注意不要丢掉
.value
Data = table.cell(row=1, column=1).value
print(Data)

# 获取所有表名
sheet_names = wb.sheetnames
print(sheet_names[0])
ws = wb[(wb.sheetnames[0])]  # index为0为第一张表
# 活动表表名
print(wb.active.title)
```

逐行读取

```
# 逐行读 ws9['A1:I9']: 例如('A1:C4')
# 返回一个生成器， 注意取值时要用value，例如：
for row in table['A1:I9']:
    for cell in row:
        print(cell.value)
```

显示有多少张表

```
print(wb.sheetnames)
# 显示表名，表行数，表列数
print(ws9.title)
print(ws9.max_row)
print(ws9.max_column)
```

保存文件

```
wb.save("sample1.xlsx")
```

2、5 操作实例

将猫眼爬虫的数据保存到 Excel 表。

3、json

目的： 将 Python 对象编码为 JSON 字符串，并将 JSON 字符串解码为 Python 对象。

`json` 模块提供了 API，将内存中的 Python 对象转换为 JSON 序列。JSON 具有以多种语言（尤其是 JavaScript）实现的优点。它在 REST API 中 Web 服务端和客户端之间的通信被广泛应用，同时对于应用程序间通信需求也很有用。 下面演示如何将一个Python数据结构转换为JSON：

3、1 编码和解码

Python 的默认原生类型（`str`，`int`，`float`，`list`，`tuple`，和`dict`）。

```
import json

data = {
    'name': 'ACME',
    'shares': 100,
    'price': 542.23
}

json_str = json.dumps(data)
print(json_str)
```

表面上看，类似于 Python `repr()` 的输出。虽然内容看似是一样，但是类型却已经发生改变

```
print(type(json_str))
```

从无序的字典到有序的字符串，这个过程被称之为序列化。

最终我们将json保存到文件

```
with open('data.json', mode='w', encoding='utf-8')
as f:
    f.write(json_str)
```

3、1、1 中文字符问题

```
import json

data = {
    'name': '青灯',
    'shares': 100,
    'price': 542.23
}

# 将字典序列化为json
json_str = json.dumps(data)
```

```
# 写入 json 数据
with open('data.json', mode='w', encoding='utf-8')
as f:
    f.write(json_str)
```

```
# filename:data.json
{"name": "\u9752\u706f", "shares": 100, "price":
542.23}
```

解决办法: `json_str = json.dumps(data, ensure_ascii=False)`

3、2 读取数据

将json数据变为字典类型的这个过程被称之为反序列化

```
# 读取 json 数据
with open('data.json', 'r', encoding='utf-8') as f:
    # 反序列化
    data = json.load(f)

# 打印数据
print(data)
print(data['name'])
```

3、3 格式化输出

JSON 的结果是更易于阅读的。`dumps()` 函数接受几个参数以使输出更易读结果。

```
import json

data = {'a': 'A', 'b': (2, 4), 'c': 3.0}
print('DATA:', repr(data))  # DATA: {'a': 'A', 'b':
(2, 4), 'c': 3.0}

unsorted = json.dumps(data)
print('JSON:', json.dumps(data))    # JSON: {"a":
"A", "b": [2, 4], "c": 3.0}
```

编码，然后重新解码可能不会给出完全相同类型的对象。

特别是，元组成为了列表。

JSON跟Python中的字典其实是一样一样的，事实上JSON的数据类型和Python的数据类型是很容易找到对应关系的，如下面两张表所示。

Python	JSON
dict	object
list， tuple	array
str	string
int， float， int- & float-derived Enums	number
True / False	true / false
None	null

4、CSV

csv文件格式是一种通用的电子表格和数据库导入导出格式。最近我调用RPC处理服务器数据时，经常需要将数据做个存档便使用了这一方便的格式。

python中有一个读写csv文件的包，直接import csv即可。利用这个python包可以很方便对csv文件进行操作，一些简单的用法如下。

4、1 写入文件

我们把需要写入的数据放到列表中，写文件时会把列表中的元素写入到csv文件中。

```
import csv

ll = [[1, 2, 3, 4],
      [1, 2, 3, 4],
      [5, 6, 7, 8],
      [5, 6, 7, 8]]

with open('example1.csv', 'w', newline='') as
csvfile:
    """
    delimiter: 分割符
    """
    spamwriter = csv.writer(csvfile, delimiter=',')
    for l in ll:
        spamwriter.writerow([1, 2, 3, 4])
```

> 可能遇到的问题：直接使用这种写法会导致文件每一行后面会多一个空行。使用 `newline=''` 解决

使用 open 直接写入

```
with open('example2.csv', 'w') as csvfile:
    """
    delimiter: 分割符
    """
    for l in ll:
        csvfile.write(",".join(map(str, l)))
        csvfile.write('\n')
```

4、2 读取文件

```
import csv

with open('example.csv', encoding='utf-8') as f:
    csv_reader = csv.reader(f)
    for row in csv_reader:
        print(row)
```

`file:example.csv` `csv` 数据

```
['1', '2', '3', '4']
['1', '2', '3', '4']
['1', '2', '3', '4']
['1', '2', '3', '4']
```

默认的情况下， 读和写使用逗号做分隔符(delimiter)，当遇到特殊情况是，可以根据需要手动指定字符， 例如:

```
with open('example.csv', encoding='utf-8') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        print(row)
```

上述示例指定冒号作为分隔符

有点需要注意的是，当用writer写数据时，`None`会被写成空字符串，浮点类型会被调用`repr()`方法转化成字符串。所以非字符串类型的数据会被`str()`成字符串存储。所以当涉及到`unicode`字符串时，可以自己手动编码后存储或者使用csv提供的`UnicodeWriter`。

4、3 写入与读取字典

`csv`还提供了一种类似于字典方式的读写，方式如下:

```
class csv.DictReader(csvfile, fieldnames=None,
restkey=None, restval=None, dialect='excel',
*args, **kwds)

class csv.DictWriter(csvfile, fieldnames,
restval='', extrasaction='raise', dialect='excel',
*args, **kwds)
```

其中 fieldnames 指定字典的 key 值，如果 reader 里没有指定那么默认第一行的元素，在 writer 里一定要指定这个。

```python
# %% 写

import csv

with open('names.csv', 'w') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile,
fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name': 'Baked',
'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely'})
    writer.writerow({'first_name': 'Wonderful',
'last_name': 'Spam'})
```

```python
# %% 读
import csv

with open('names.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['first_name'], row['last_name'])
```