第 5 章

文件读写

很多时候我们希望程序可以保存一些数据，例如日志、计算的结果等等。例如用 Python 来处理实验数据，如果能把各种结果保存到一个文件中，即使关闭了终端或者 IDE 下次不用再完全跑一遍也可以直接查看结果，这时候就需要 Python 中相关文件的操作了。

本章会详细讲解在 Python 中文件操作和文件系统相关知识。

5.1 打开文件

用 Python 打开一个文件需要用到内建的 open 函数。这个函数的原型是：

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

其中，file、mode、encoding 三个参数比较重要。

5.1.1 file

file 参数就是文件名，文件名可以是相对路径，也可以是绝对路径，总之可以定位到这个文件就行。

绝对路径非常好理解，例如一个文件的完整路径是 C:\Users\user1\file.txt，那么它的绝对路径就是 C:\Users\user1\file.txt。

这就好比在二维坐标系上，一旦 x 和 y 值确定了，那么这个点的位置就确定了。

而要介绍相对路径需要引入工作路径的概念。事实上任何一个程序在运行的时候都会有一个工作路径，所有的相对路径都是相对这个工作路径而言的，在 Python 中我们可以通过这样查看当前工作路径：

```
import os
print(os.getcwd())
```

这段代码一个可能的输出是：

```
/Users/jiangjiao/PycharmProjects/LearnPythonWithPractice/Chapter    12
```

不难发现，这个路径就是文件所在的文件夹。但是要注意的是，工作路径不一定总是这样，如图 5-1 所示。

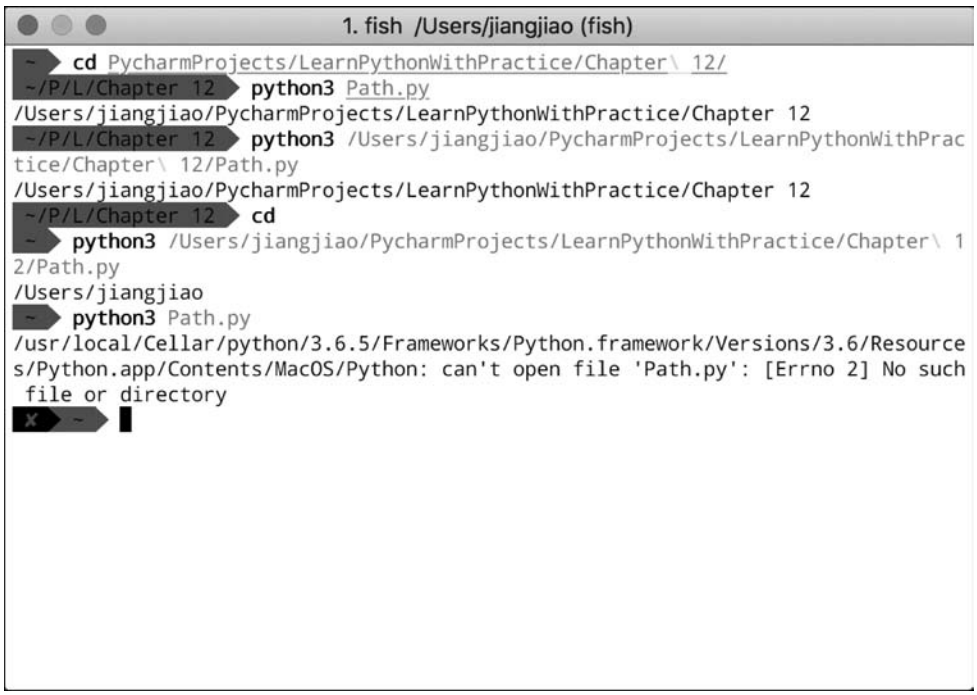

图 5-1　相对路径

要注意的是有蓝色条开头的是用户输入，没有蓝色条开头的是程序的输出，这里解释一下上面终端中发生了什么。

① 第一行：cd 命令用于切换工作路径，这里是切换到了 Path.py 所在的目录，注意这时候工作路径就是 Path.py 所在的目录。

② 第二行：使用 Python 解释器启动了工作路径下的 Path.py，注意这里使用的就是相对路径。

③ 第三行：Path.py 输出了工作路径为当前目录。

④ 第四行：使用 Python 解释器启动了 Path.py，但是这次使用了绝对路径。

⑤ 第五行：将工作路径转到了当前用户根目录下，这是 mac osx 或者 linux 在 cd 没有参数的时候默认操作。在 Windows 下可以使用 cd/来切换到当前驱动器的根目录。

⑥ 第六行：再次执行 Path.py，但是这里使用了绝对路径，可以看到工作路径并不是文件所在路径了，而是当前终端的工作路径。

⑦ 第七行：如果这时候使用相对路径访问 Path.py，会提示 No such file or directory，意味着用相对路径找不到这个文件或目录。

从这个例子中我们可以看到相对路径和绝对路径的关系，那就是绝对路径＝工作路径＋相对路径。例如，工作路径是 C:\Users\user1，这时候用相对路径 file.txt 去定位文件，实际上是跟绝对路径 C:\Users\user1\file.txt 是等价的，也就是说相对路径是相对工作路径而言的。

特殊地，我们可以用. 表示当前目录和 .. 表示父目录，例如在工作路径 C:\Users\user1 下用 . 就表示 C:\Users\user1，而用 .. 就表示 C:\User。

如果还用之前二维坐标系的例子来描述的话，相对路径就好比是一个点相对另一个点的偏移 Δx 和 Δy，一旦相对的点和偏移确定了，这个点就确定了。

5.1.2 mode

mode 参数表示我们打开这个文件的时候采取的行为，有如表 5-1 所示的不同模式。

表 5-1 模式

模式	解　　释
'r'	r 表示读，即以只读方式打开文件。这是默认模式，所以如果用只读方式打开文件，这个参数可以省略
'w'	w 表示写，新建一个文件只用于写入。如文件已存在则会覆盖旧文件
'x'	x 表示创建新文件，如果文件已存在则报错
'a'	a 表示追加，打开一个文件用于追加，后续的写入会从文件的结尾开始。如果该文件不存在，则创建新文件
'b'	二进制读写模式
't'	文本模式
'＋'	以更新的方式打开一个文件

这些开关可以自由组合，但是需要注意的是前四种至少要选择一个，同时默认情况下是文本模式读写，如果需要二进制读写必须单独指明。

一些常用的模式组合如表 5-2 所示。

表 5-2 常用模式组合

模式	解　　释
rb	b 表示二进制读写模式，配合 r 的意思就是二进制只读方式打开
r＋	＋ 表示更新，打开一个文件用于更新。文件指针将会放在文件的开头。如文件不存在则报错。r＋ 会覆盖写原来的文件，覆盖位置取决于文件指针的位置
rb＋	相比 r＋ 不同之处在于是二进制读写
wb	二进制写入
w＋	新建一个文件用于写入，如果文件已经存在则会清空文件内容
wb＋	相比 w＋ 不同之处在于是二进制写入
ab	相比 a 不同之处在于是二进制追加
a＋	相比 a 不同之处是可以读写
ab＋	相比 a＋ 不同之处是二进制读写

这里出现了一个新名词：文件指针，实际上只要把它理解为 word 中的光标就好了，它代表了我们下次写入或者读取的起始位置。

5.1.3　encoding

这个单词的意思是编码，在这里指的是文件编码，例如 GB18030，UTF-8 等。有的时候我们打开一个文件乱码，就可以尝试修改这个参数。一般来说推荐无论读写都使用 UTF-8 来避免乱码问题。

5.2 关闭文件

对文件操作后应该关闭文件，否则可能会丢失写入的内容，同时如果是写模式打开一个文件却不关闭，那么这个文件会一直被占用，所以一定要养成关闭文件的好习惯。

文件的关闭非常简单，只需要调用 close 方法即可：

```
file = open('file.txt', 'r')
file.close()  #别忘记关闭文件
```

5.3 读文件

读文件一般有四种方式，即 read、readline、readlines 和迭代。

下面要读取的 file.txt 中的内容为：

```
Hello, this is a test file.
Let's read some lines from The Matrix.
This is your last chance.
After this, there is no turning back.
You take the blue pill—the story ends, you wake up in your bed and believe whatever you want to
believe.
You take the red pill—you stay in Wonderland, and I show you how deep the rabbit hole goes.
Remember: all I'm offering is the truth.
Nothing more.
```

5.3.1　read

read 方法的原型是：

```
read(size = -1)
```

它用于读取指定数量的字符，默认参数－1 表示读取文件中的全部内容。注意如果直到文件末尾还没有读取够 size 个字符，那么会直接返回，也就是说 size 只表示最多读取的

字符数量。

例如，读取前 10 个字符可以这么写：

```
file = open('file.txt', 'r')
result = file.read(10)
print(result)
file.close()  #别忘记关闭文件
```

这段代码会输出：

```
Hello, thi
```

5.3.2 readline

readline 的原型是：

```
readline(size = -1)
```

和 read 类似，size 指定了最多读入的字符数量，但是 readline 一次会读入一整行，也就是说遇到换行符\n 会返回一次，例如希望读第一行可以这么写：

```
file = open('file.txt', 'r')
result = file.readline()
print(result)
file.close()  #别忘记关闭文件
```

这段代码会输出：

```
Hello, this is a test file.
```

5.3.3 readlines

readlines 的原型是：

```
readlines(hint = -1)
```

它表示一次读取多行，如果没有指定参数则默认读到最后一行，例如如果想读取文件中所有行可以这么写：

```
file = open('file.txt', 'r')
result = file.readlines()
print(result)
file.close()#别忘记关闭文件
```

这段代码会输出：

```
['Hello, this is a test file.\n', "Let's read some lines from The Matrix.\n", 'This is your last chance.\n', 'After this, there is no turning back.\n', 'You take the blue pill—the story ends, you wake up in your bed and believe whatever you want to believe.\n', 'You take the red pill—you stay in Wonderland, and I show you how deep the rabbit hole goes.\n', "Remember: all I'm offering is the truth.\n", 'Nothing more.']
```

这里可以看到返回的 List 中每个元素就代表文件中的一行。

5.3.4　迭代

此外其实文件对象本身也是一个可迭代对象，也就是说我们可以用 for 循环来遍历每一行，例如：

```
file = open('file.txt', 'r')
for line in file:
    print(line, end = "")#文件中每一行本身有一个换行所以用 end = "" 让 print 不换行
file.close()             #别忘记关闭文件
```

这段代码会输出：

```
Hello, this is a test file.
Let's read some lines from The Matrix.
This is your last chance.
After this, there is no turning back.
You take the blue pill—the story ends, you wake up in your bed and believe whatever you want to believe.
You take the red pill—you stay in Wonderland, and I show you how deep the rabbit hole goes.
Remember: all I'm offering is the truth.
Nothing more.
```

5.4 写文件

5.4.1　write 和 writelines

写文件有两种方法，write 和 writelines，例如：

```
file2 = open('file2.txt', 'w')
file2.write('hello world!\n')
file2.writelines(('this ', 'is ', 'a\n', 'file!'))
file2.close()  #别忘记关闭文件
```

会得到这样一个文件：

```
hello world!
this is a
file!
```

要注意的是，写入的时候不会像 print 那样自动在最后添加一个换行符，因此如果想换行的话需要自己添加换行符。

5.4.2 flush

另外如果想在不关闭文件的前提下把内容写入到文件中，可以使用 flush，例如：

```
from time import sleep
file2 = open('file3.txt', 'w')
file2.write('hello world!\n')
file2.writelines(('this ', 'is ', 'a\n', 'file!'))
file2.flush()
sleep(60)     #这时候去查看文件，已经有写入的内容
file2.close() #但是文件依旧需要正常关闭
```

这个函数的作用就是立即把刚才要写入的内容写到文件中。

5.5 定位读写

刚才在讲模式的时候提到过“文件指针”的概念，实际上还可以像在 Word 里移动光标一样定位或者移动这个指针来为读写做准备。

5.5.1 tell

tell 用来返回光标的位置，或者说是相对文件起始的偏移，例如：

```
file = open('file.txt', 'a')
print(file.tell())
file.close()  #别忘记关闭文件
```

这段代码会输出：

```
391
```

因为我们使用了'a'模式，打开的时候指针在文件的末尾。

5.5.2 seek

seek 的原型是：

```
seek(offset[, whence])
```

offset 表示要设置的偏移量，以字节为单位，正数表示正向偏移，负数表示反向偏移。whence 表示偏移的基准，0 表示相对文件起始，1 表示相对当前文件指针位置，2 表示相对文件结尾。如果导入了 io 模块的话还可以相应的使用 io. SEEK_SET、io. SEEK_CUR 和 io. SEEK_END 表示偏移的基准来提高可读性。

例如可以这样使用：

```
import io

file3 = open('file3.txt', 'w+')
file3.write('congratulations, you mastered this skill!')
print(file3.tell())
file3.seek(35)
print(file3.tell())
file3.write('tool!')
file3.close()
```

会输出一个这样的文本文件：

```
congratulations, you mastered this tool!!
```

可以看到我们定位到 skill 这个单词的位置，然后修改了它。

5.6 数据序列化

有时候我们除了希望把变量的值存起来，还希望下次读取的时候可以用这些数据直接恢复当时变量的状态，这时候就需要用到序列化的技术。

5.6.1 Pickle

Pickle 是 Python 内建的序列化工具。它有序列化和反序列化两个过程，对应的就是变量的存储和读取。

我们直接看一个完整的例子：

```
import pickle
import datetime

list1 = ['hello', 1, 'world!']
dict1 = {'key': 'random value'}

time = datetime.datetime.now()
```

```
file = open('pickle.pkl', 'wb+')

#序列化
pickle.dump(list1, file)
pickle.dump(dict1, file)
pickle.dump(time, file)

file.close()

file = open('pickle.pkl', 'rb+')

#反序列化
data = pickle.load(file)
print(data)
print(type(data))
data = pickle.load(file)
print(data)
print(type(data))
data = pickle.load(file)
print(data)
print(type(data))

file.close()
```

这段代码会输出：

```
['hello', 1, 'world!']
<class 'list'>
{'key': 'random value'}
<class 'dict'>
2018-02-24 11:50:31.931213
<class 'datetime.datetime'>
```

可以看到这里核心方法是 pickle.dump 和 pickle.load，前者用于把数据序列化到文件中，后者用于把数据从文件中反序列化赋值给变量。

要注意的是由于 pickle 使用的协议是使用二进制来序列化，因此生成的文件用普通的编辑器是不可读的，而且在 dump 方法中传入的文件对象应该是以'b'模式打开的。

5.6.2 JSON

JSON 是一种轻量化的数据交换格式，它并不是专门为 Python 服务的。但是由于 JSON 数据格式跟 Python 中的 List，Dict 非常相近，因此 JSON 和 Python 的亲和度相当高，所以也常用 JSON 来序列化数据。而且相比之前的 Pickle，JSON 序列化产生的是文本文件，也就是说依旧是可读可编辑的。

例如我们可以轻松地序列化和反序列化这种嵌套式的变量：

```
import json

dict1 = {
   'Name': 'Steve Jobs',
   'Birth Year': 1955,
   'Company Owned': [
      'Apple',
      'Pixar',
      'NeXT'
   ]
}

file = open('data.json', 'w+')

#序列化
json.dump(dict1, file)

file.close()

file = open('data.json', 'r+')

#反序列化
data = json.load(file)
print(data)
print(type(data))

file.close()
```

这段代码可以输出：

```
{'Name': 'Steve Jobs', 'Birth Year': 1955, 'Company Owned': ['Apple', 'Pixar', 'NeXT']}
<class 'dict'>
```

用任意文本编辑器打开刚刚生成的 JSON 文件可以看到文件内容是：

```
{"Name": "Steve Jobs", "Birth Year": 1955, "Company Owned": ["Apple", "Pixar", "NeXT"]}
```

可以发现看出数据的格式基本是跟 Python 中的表示方法是一样的。
如果想进一步提高可读性，可以简单修改一下序列化时候的参数：

```
#把 json.dump(dict1, file) 修改为
json.dump(dict1, file, indent = 4)
```

这样序列化的数据就会变成：

```
{
   "Name": "Steve Jobs",
```

```
    "Birth Year": 1955,
    "Company Owned": [
        "Apple",
        "Pixar",
        "NeXT"
    ]
}
```

但是在 Python 中用 JSON 序列化数据也是有缺陷的，如果我们想序列化一个自己写的类，还需要自己写一个 Encoder 和 Decoder 用于编码和解码对象，相比 Pickle 来说就复杂得多了。

5.7 文件系统操作

对于文件系统 Python 提供了一个专门的库 os，其中封装了许多跟操作系统相关的操作，但是其中有的函数只能在特定的平台上使用，例如 chmod 只能在 Linux/OSX 上获得完整的支持，在 Windows 上只能用于设置只读，虽然 Python 是跨平台的，但是毕竟不同平台的特性相差太多，os 中的很多方法都有这样的平台依赖性。

接下来会介绍一些和文件系统相关的方法。

5.7.1 os.listdir(path='.')

这个函数可以列出一个目录下的所有文件，path 是路径，如果不指定则是当前的工作路径，例如：

```
print(os.listdir())
```

会输出：

```
['file2.txt', 'file.txt', 'pickle.pkl', 'file3.txt', 'OS.py', 'data.json', 'File.py', 'Pickle.
py', 'Path.py', 'Json.py']
```

5.7.2 os.mkdir(path, mode=0o777)

这个函数可以创建一个目录，path 是路径，mode 是 Linux/OSX 上的文件权限，在 Windows 中这个参数是不可用的。

5.7.3 os.makedirs(name, mode=0o777, exist_ok=False)

os.mkdir 只能创建一个目录，但是 os.makedirs 可以创建包括子目录在内的多个目录。exist_ok 参数决定了如果目录存在会不会报错，如果设置为 False，那就会报错。

看一个例子就能明白 makedirs 的方便之处：

```
os.mkdir('testdir')
os.makedirs('testdir2/testdir')
```

可以看到创建出了两种目录，如图 5-2 所示。

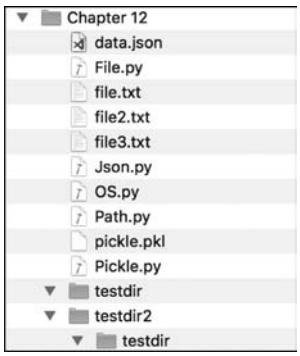

图 5-2　创建目录

其中在创建第二个 testdir 的时候不存在父目录 testdir2 而 makedirs 自动创建了这个目录。

5.7.4　os.remove(path)

删除指定路径的文件，不能用来删除目录。

5.7.5　os.rmdir(path)

删除一个空目录，例如：

```
os.rmdir('testdir')
```

但是如果尝试删除 testdir2 就会报错，因为它非空。

5.7.6　os.removedirs(name)

递归删除一个具有子目录的目录。使用这个函数就可以删除 testdir2 了，例如：

```
os.removedirs('testdir2')
```

5.7.7　os.rename(src, dst)

重命名一个文件。src 是源文件，dst 是目标文件，例如：

```
os.rename('data.json', 'data')
```

5.7.8 os.path.exists(path)

可以判断一个文件是否存在。例如：

```
os.path.exists('./Path.py')
```

5.7.9 os.path.isfile(path)

可以判断一个路径是不是文件，而不是目录或者其他类型。例如：

```
os.path.isfile('./Path.py')
```

5.7.10 os.path.join(path, paths)

这是一个很常用的计算路径的函数，它的作用是将一串 path 按照正确的方式连起来，例如：

```
print(os.path.join('home', 'dir1', 'dir2/dir3', 'something.txt'))
```

这句代码会输出：

```
home/dir1/dir2/dir3/something.txt
```

5.7.11 os.path.split(path)

这个函数用于分离目录和文件名，例如：

```
print(os.path.split('home/dir1/dir2/dir3/something.txt'))
```

这句代码会输出：

```
('home/dir1/dir2/dir3', 'something.txt')
```

至于 os 模块中其他的方法的使用以及不同方法在不同平台上的限制都可以通过查阅文档获知，这里只列出了一些最常用的文件系统操作方法。

本章小结

Python 中与文件的交互是非常简单的，读取文件可以按字节读取也可以按行读取，而写文件的时候可以按字符串写入也可以按行写入，同时 Python 也支持传统的文件指针移动。

本章习题

1. 通过文件操作，写一个记录用户输入的小程序。

2. 写一个给图片按照日期批量重命名的小程序。

3. 写一个文本文件搜索工具，可以在一个文本文件中搜索指定字符串。

4. 通过 Pickle 和 JSON 来序列化学生的信息，学生的信息应该至少包括姓名、学号、班级、年龄、性别。