文件

我们需要处理的数据，一定是很多，所以才必须由计算机帮我们处理 —— 大量的数据保存、读取、写入，需要的就是文件（Files）。在这一章里，我们只介绍最简单的文本文件。

创建文件

创建一个文件，最简单的方法就是用 Python 的内建函数 `open()` 。

`open()` 函数的官方文档很长，以下是个简化版：

```
open(file, mode='r')
```

第二个参数，`mode`，默认值是 `'r'`，可用的 `mode` 有以下几种：

参数字符	意义
`'r'`	只读模式
`'w'`	写入模式（重建）
`'x'`	排他模式 —— 如果文件已存在则打开失败
`'a'`	追加模式 —— 在已有文件末尾追加
`'b'`	二进制文件模式
`'t'`	文本文件模式 (默认)
`'+'`	读写模式（更新）

创建一个新文件，用这样一个语句就可以：

```
open('test-file.txt', 'w')
```

```
<_io.TextIOWrapper name='test-file.txt' mode='w' encoding='UTF-8'>
```

当然，更多的时候，我们会把这个函数的返回值，一个所谓的 file object，保存到一个变量中，以便后面调用这个 file object 的各种方法，比如获取文件名 `file.name`，比如关闭文件 `file.close()`：

```
f = open('test-file.txt', 'w')
print(f.name)
f.close()
```

```
test-file.txt
```

删除文件

删除文件，就得调用 `os` 模块了。删除文件之前，要先确认文件是否存在，否则删除命令会失败。

```
import os

f = open('test-file.txt', 'w')
print(f.name)

if os.path.exists(f.name):
    os.remove(f.name)
    print(f'{f.name} deleted.')
else:
    print(f'{f.name} does not exist')
```

```
test-file.txt
test-file.txt deleted.
```

读写文件

创建文件之后，我们可以用 `f.write()` 方法把数据写入文件，也可以用 `f.read()` 方法读取文件。

```
f = open('test-file.txt', 'w')
f.write('first line\nsecond line\nthird line\n')
f.close()

f = open('test-file.txt', 'r')
s = f.read()
print(s)
f.close()
```

```
first line
second line
third line
```

文件有很多行的时候，我们可以用 `file.readline()` 操作，这个方法每次调用，都会返回文件中的新一行。

```
f = open('test-file.txt', 'w')
f.write('first line\nsecond line\nthird line\n')
f.close()
```

```
f = open('test-file.txt', 'r')
s = f.readline()    # 返回的是 'first line\n'
print(s)
s = f.readline()    # 返回的是 'second line\n'
print(s)
f.close()
```

```
first line

second line
```

注意，返回结果好像跟你想的不太一样。这时候，之前见过的 `str.strip()` 就派上用场了：

```
f = open('test-file.txt', 'w')
f.write('first line\nsecond line\nthird line\n')
f.close()

f = open('test-file.txt', 'r')
s = f.readline().strip()    # 返回的是 'first line', '\n' 被去掉了……
print(s)
s = f.readline().strip()    # 返回的是 'second line', '\n' 被去掉了……
print(s)
f.close()
```

```
first line
second line
```

与之相对的，

```
f = open('test-file.txt', 'w')
f.write('first line\nsecond line\nthird line\n')
f.close()

f = open('test-file.txt', 'r')
s = f.readlines()    # 返回的是一个列表，注意，readlines，最后的 's'
print(s)
f.close()
```

```
['first line\n', 'second line\n', 'third line\n']
```

既然返回的是列表，那么就可以被迭代，逐一访问每一行：

```
f = open('test-file.txt', 'w')
f.write('first line\nsecond line\nthird line\n')
f.close()

f = open('test-file.txt', 'r')
for line in f.readlines():
    print(line)
f.close()
```

```
first line

second line

third line
```

与之相对的，我们也可以用 `file.writelines()` 把一个列表写入到一个文件中，按顺序每一行写入列表的对应元素：

```
a_list = ['first line\n', 'second line\n', 'third line\n']
f = open('test-file.txt', 'w')
f.writelines(a_list)
f.close()

f = open('test-file.txt', 'r')
for line in f.readlines():
    print(line)
f.close()
```

```
first line

second line

third line
```

with 语句块

针对文件操作，Python 有个另外的语句块写法，更便于阅读：

```
with open(...) as f:
    f.write(...)
    ...
```

这样，就可以把针对当前以特定模式打开的某个文件的各种操作都写入同一个语句块了：

```
import os

with open('test-file.txt', 'w') as f:
    f.write('first line\nsecond line\nthird line\n')

with open('test-file.txt', 'r') as f:
    for line in f.readlines():
        print(line)

if os.path.exists(f.name):
    os.remove(f.name)
    print(f'{f.name} deleted.')
else:
    print(f'{f.name} does not exist')
```

```
first line

second line

third line

test-file.txt deleted.
```

另外，用 `with` 语句块的另外一个附加好处就是不用写 `file.close()` 了……

另一个完整的程序

若干年前，我在写某本书的时候，需要一个例子 —— 用来说明 “**即便是结论正确，论证过程乱七八糟也不行！**”

作者就是这样，主要任务之一就是给论点找例子找论据。找得到不仅_恰当_且又_精彩_的例子和论据的，就是好作者。后面这个 “*精彩*” 二字要耗费很多时间精力，因为它意味着说 “要找到_很多_例子而后在里面选出_最精彩_的那个！” —— 根本不像很多人以为的那样，是所谓的 “信手拈来”。

找了很多例子都不满意…… 终于有一天，我看到这么个说法：

> 如果把字母 `a` 计为 `1`、`b` 计为 `2`、`c` 计为 `3` …… `z` 计为 `26`，那么：
>
> - knowledge = 96
> - hardwork = 98
> - attitude = 100
>
> 所以结论是：
>
> - 知识（*knowledge*）与勤奋（*hardwork*）固然都很重要；
> - 但是，决定成败的却是态度（**attitude**）！

结论虽然有道理 —— 可这论证过程实在是太过分了罢.....

我很高兴，觉得这就是个_好例子_！并且，加工一下，会让读者觉得很精彩 —— 如果能找到一些按照同样的计算方法能得到 100 的单词，并且还是那种一看就是 “反例” 的单词.....

凭直觉，英文单词几十万，如此这般等于 100 的单词岂不是数不胜数？并且，一定会有很多负面意义的单词如此计算也等于 100 罢？然而，这种事情凭直觉是不够的，手工计算又会被累死..... 于是，面对如此荒谬的论证过程，我们竟然 “无话可说”。

幸亏我是会写程序的人。所以，不会 “干着急没办法”，我有能力让计算机帮我把活干了。

很快就搞定了，找到很多很多个如此计算加起来等于 100 的英文单词，其中包括：

> - connivance（纵容）
> - coyness（羞怯）
> - flurry (慌张)
> - impotence（阳痿）
> - stress（压力）
> - tuppence（微不足道的东西）
> - .....

所以，决定成败的可以是 “慌张”（flurry），甚至是 “阳痿”（impotence）？这不明显是胡说八道嘛！

—— 精彩例子制作完毕，我把它放进了书里。

那，具体的过程是什么样的呢？

首先我得找到一个英文单词列表，很全的那种。这事用不着写程序，Google 一下就可以了。我搜索的关键字是 “english word list”，很直观吧？然后就找到一个：https://github.com/dwyl/english-words；这个链接里有一个 words-alpha.txt 文件，其中包含接近 370,101 个单词，应该够用了！下载下来用程序处理就可以了！

因为文件里每行一个单词，所以，就让程序打开文件，将文件读入一个列表，而后迭代这个列表，逐一计算那个单词每个字母所代表的数字，并加起来看看是否等于 100？如果是，就将它们输出到屏幕..... 好像不是很难。

```
with open('words_alpha.txt', 'r') as file:
    for word in file.readlines():
        pass # 先用 pass 占个位，一会儿再写计算过程
```

按照上面那说法，把 `a` 记为 `1`，直至把 `z` 记为 `26`，这事并不难，因为有 `ord()` 函数啊 —— 这个函数返回字符的 Unicode 编码：`ord('a')` 的值是 `97`，那按上面的说法，用 `ord('a') - 96` 就相当于得到了 `1` 这个数值..... 而 `ord('z') - 96` 就会得到 `26` 这个数值。

```
ord('a')
```

```
97
```

那么，计算 'knowledge' 这个字符串的代码很简单：

```
word = 'knowledge'
sum = 0
for char in word:
    sum += ord(char) - 96
print(sum)
```

```
96
```

果然，得到的数值等于 96 —— 不错。把它写成一个函数罢：sum_of_word(word):

```
def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum

sum_of_word('attitude')
```

```
100
```

那让程序就算把几十万行都算一遍也好像很简单了：

```
def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum

with open('words_alpha.txt', 'r') as file:
    for word in file.readlines():
        if sum_of_word(word) == 100:
            print(word)
```

```
abstrusenesses
acupuncturist
adenochondrosarcoma

...
```

```
worshipability
zeuctocoelomatic
zygapophysis
```

嗯？怎么输出结果跟想得不一样？找到的词怎么都“奇形怪状”的..... 而且，输出结果中也没有 `attitude` 这个词。

插入个中止语句，`break`，把找到的第一个词中的每个字符和它所对应的值都拿出来看看？

```
def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum

with open('words_alpha.txt', 'r') as file:
    for word in file.readlines():
        if sum_of_word(word) == 100:
            print(word)
            for c in word:          # 把字母和值都打出来，看看对不对？
                print(c, ord(c) - 96)
            break                   # 找到一个之后就停下来。
```

```
abstrusenesses

a 1
b 2
s 19
t 20
r 18
u 21
s 19
e 5
n 14
e 5
s 19
s 19
e 5
s 19

 -86
```

怎么有个 `-86`？！仔细看看输出结果，看到每一行之间都被插入了一个空行，想到应该是从文件里读出的行中，包含 `\n` 这种换行符..... 如果是那样的话，那么 `ord('\n') -96` 返回的结果是 `-86` 呢，怪不得找到的词都“奇形怪状”的.....

```
ord('\n') -96
```

```
-86
```

改进一下呗 —— 倒也简单，在计算前把读入字符串前后的空白字符都给删掉就好了，用 `str.strip()` 就可以了：

```
def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum

with open('words_alpha.txt', 'r') as file:
    for word in file.readlines():
        if sum_of_word(word.strip()) == 100:
            print(word)
```

```
abactinally
abatements
abbreviatable

...

zithern
zoogleas
zorgite
```

如果想把符合条件的词保存到一个文件 `result.txt` 里的话，那么：

```
def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum

with open('results.txt', 'w') as result:
    with open('words_alpha.txt', 'r') as file:
        for word in file.readlines():
            if sum_of_word(word.strip()) == 100:
                result.write(word)
```

竟然这么简单就搞定了？！

这 12 行的代码，在几秒钟内从接近 370,101 个英文单词中找到 3,771 个如此计算等于 100 的词汇。

喝着咖啡翻一翻 `result.txt`，很快就找到了那些可以用来做反例格外恰当的词汇。

真无法想象当年的自己若是不懂编程的话现在会是什么样子.....

总结

这一章我们介绍了文本文件的基本操作：

> - 打开文件，直接用内建函数，`open()`，基本模式有 `r` 和 `w`；
> - 删除文件，得调用 `os` 模块，使用 `os.remove()`，删除文件前最好确认文件确实存在.....
> - 读写文件分别有 `file.read()`、`file.write()`、`file.readline()`、`file.readlines()`、`file.writelines()`；
> - 可以用 `with` 把相关操作都放入同一个语句块.....