其他问题-2，文件

- 文件（**file**）：概念和用途
- 打开/关闭（**open/close**）
- 文件和文件对象
- **Python**文件操作
- 文件输入和输出
- 文件处理程序实例

文件（file）

- 程序中能直接使用的数据都在内存，变量是内存地址的抽象
  - 程序变量及其记录数据只在程序运行期间存在
  - 如果需要长期保存数据，就要使用文件
- 文件是计算机系统组织和保存位于外存的数据的基本单元，一个文件可以包含任意多的大量数据，由计算机的操作系统管理
  - 文件保存在外部存储器（磁盘/光盘/U盘等）里，由计算机上运行的操作系统（例如 Windos/IOS/Linux等）管理
  - 每个文件有一个名字，通过文件名使用文件内容
  - 由于实际计算机系统里的文件很多，通常将它们分门别类，组织在分层次的目录（文件夹）结构里，以方便使用
  - 如果程序生成的数据需要长期保存，或需要保存大量的数据，都应该将它们以文件形式存入外存储器

文件

- 要在自己编写的程序里使用（或创建）操作系统管理下的文件，就需要使用编程语言的文件操作功能

  编写 **Python** 程序时也一样

- 使用文件的一些基本情况：
  - 从已有的文件读入数据，在程序里使用

    程序只读取文件里保存的信息，以“读”方式使用文件
  - 把程序计算产生的结果存入文件（例如需要长期保存）

    只向文件写入信息，“写”方式，通常是写一个新文件
  - 在长时间执行的程序的运行中，可能需要保存大量的间数据，有时又需要把保存的数据重新装入内存使用

    “读写”一个文件，工作中还需要在两个方式间转换，是比较复杂的文件使用方式，可能有读写位置的问题

文件

- 人们通常把文件分为正文文件和二进制文件
  - 正文文件 **(text file)** 是某个字符集中字符的序列，可能分行
  - 二进制文件 **(binary file)** 是任意的编码文件
- 文件是位于（我们编写的）程序之外的一种数据对象，由操作系统管理。文件有名字，需要通过文件名使用
  - 要使用一个文件，必须先要求操作系统找到文件，并在程序里为该文件建立一个（代表文件的）程序对象，而后通过该对象使用文件。这一套操作称为打开文件
  - 打开文件后把得到的文件对象赋给一个变量，而后就可以通过该变量（经由相应对象）使用文件了（读或写数据）
  - 如果文件不再使用，应撤销与它的联系，称为关闭文件。一个程序结束时，操作系统通常会关闭它正在用的所有文件

Python 的文件功能

- 现在介绍 **Python** 语言提供的文件操作机制
- 在 **Python** 程序里使用文件
  - 需要先建立程序与被使用的实际文件之间的关联
  - **Python** 为此专门设置了一种文件类型的对象
- 使用文件之前需要先“打开”它
  - 打开就是建立文件关联
  - 打开操作的实际效果是建立一个文件对象，代表被用的文件
- 函数 **open(...)** 打开文件，正常完成时返回一个新建的文件对象
  - 通过变量掌握文件对象，就可以操作与之相关的文件
  - 打开文件有两种基本模式：正文模式和二进制模式。打开正文文件时应该用正文模式，本课程不考虑二进制模式

文件基本操作

- open 函数的基本使用
  - open(文件名) 以正文模式打开一个用于读数据的文件（不加说明时的默认使用方式为读方式）
  - open(文件名, 'w') 以正文模式打开用于写数据的文件
  - 文件名用字符串表示。open 返回与被打开文件关联的 file 对象；如文件无法打开将报 OSError
  - 还有一些操作模式，见标准库手册有关 open 的介绍
- 用完的文件应该用 close 操作“关闭”

  假如 file1 的值是前面打开文件得到的文件对象

  file1.close() 关闭该文件

文件的描述

- 要打开一个文件，需要告诉操作系统打开哪个文件，这就牵涉到文件的描述问题，用字符串描述
- 最简单的方式是打开“当前目录（文件夹）”（也就是运行的程序所在的目录）下的文件，只需简单写出描述文件名的字符串

```
file1 = open("data.dat")
```

- 如果要打开的文件在当前目录的某子目录下，要先给出子目录名而后是文件名，/ 分隔（可多层子目录。称为“相对路径描述”）

```
file2 = open("save/data.dat")
```

- 另一种方式是从最上层目录开始，还可以指定盘符（默认为当前盘，即程序所在的盘）。这种形式称为“绝对路径描述”

```
file3 = open("/courses/new-Computing/progs/data.dat")
file4 = open("C:/texlive/2014/release-texlive.txt")
```

从正文文件输入

- 文件对象的各章操作详见标准库 io 包，在 Generic Operating System Services 栏目下
- 以读方式打开正文文件，得到对象能用于读文件内容：

```
inf = open('file1.dat', 'r') # 以正文读方式打开
data = inf.read()          # data 得到整个文件内容的字符串
data2 = inf.read()         # 读完再读，data2 得到空串
```

f.read(n) 要求读入 n 个字符

- 按行读入用 readline()，例：

```
inf = open('file2.dat', 'r')
line1 = inf.readline()     # 读入一行做成字符串
line2 = inf.readline()     # 读入下一行
```

readline 读到换行符，换行符作为结果字符串的最后一个字符

读空行得到只包含换行符的串，文件读完再读得到空串

从正文文件输入

- 得到的串为空，可以作为逻辑值控制读入循环
- 打开的正文读文件对象也是一种可迭代对象，可以直接作为 for 语句循环变量的数据源，每次迭代得到一行

```
for line in inf :  # 设 inf 的值是正文读文件对象
    ... line ...      # 处理一行文本
```

相当于（前一写法非常方便，很常用）

```
while True :
    line = inf.readline()
    if not line: # 判断文件是否已经读完
        break
    ... line ...  # 处理一行文本
```

- open 还有一些打开模式，另外有些参数指定文件操作的行为方式，详情见语言手册（后面可能讲到一些）

文件操作实例1

- 考虑简单问题：统计程序里字母、数字、空白和其他字符的个数
- 文件使用模式：
  1. 打开文件，不能正确打开时报错（**FileNotFoundError**等）
  2. 处理文件内容
  3. 关闭文件
- 文件在程序之外，能否正常打开依赖于很多因素：
  - 文件是否存在？文件名（以及路径）描述是否正确？
  - 是否有使用该文件的权限？
  - 操作系统的规则是否允许所指定方式打开这个文件？等等
- 上述程序的实现并不复杂

  如果文件里有中文，可以给 **open** 加 **encoding="utf_8"** 参数

从正文文件输入

- 以正文读方式得到的文件对象是一个可迭代对象（按行迭代），可以直接转换到 list 或 tuple

```
inf1 = open('file1.dat', 'r')
inf2 = open('file1.dat', 'r')
text1 = list(inf1)    # 得到 file1.dat 正文行的表
text2 = tuple(inf2)  # 得到 file2.dat 正文行的元组
```

  - 如果需要修改读入内容，可考虑用 list 保存文件内容
  - 如果只是读入和使用，可以考虑用元组

- readlines(...) 直接得到文件里所有正文行的表

```
inf1 = open('file1.dat', 'r')
text1 = inf1.readlines()
```

向正文文件输出

- 基本的正文文件输出函数是 write
  - 用 f.write(...) 的形式调用
  - 其参数应该是一个字符串
  - 需要换行时，必须用换行符明确写出
  - 如果需要把其他类型的数据输出到正文文件，就需要明确地用 str() 或 repr() 转换为字符串
  - write 返回实际输出的字符个数
- 例：

```
outf = open("resfile1.dat", "w")
outf.write("Generated integers:\n")
for i in range(10) :
    outf.write(", ".join([str(i**2 + j) for j in range(10)]) + '\n')
```

向文件输入输出

- 文件输入和输出默认采用缓冲方式
  - 打开一个文件时，系统自动为其建立一个内部的缓冲存储区（作为程序和文件之间的数据中介）
  - 对于输入文件，程序不断从缓冲区输入数据，一旦缓冲区空，系统自动从文件里搬过一批数据到缓冲区
  - 对输出文件，程序产生的输出存入缓冲区，直到缓冲区满时，习题才把做些输出实际地写入文件
- 对文件输出有实际影响（例如在 IDLE 里运行程序）
  - **close** 关闭文件前将把当时缓冲区内容全部写入文件
  - 调用 **f.flush()** 要求把当时缓冲区内容写入文件（“冲刷”缓冲区）。没做这些就关闭程序可能看不到实际输出

使用文件数据的实例

- 例：假定一个正文文件里保存了一批浮点数形式的数据，用空格/换行分隔，需要读入处理。处理过程
  - 首先打开文件，建立文件对象
  - 从文件对象读入文本内容，转换到浮点数
  - 在转换的同时处理，或者转换后存入表中，以后使用
- 例：考虑定义函数，以便从指定文件读入浮点数，程序里每次调用函数得到文件里的下一个浮点数
  - 考虑定义一个打开文件函数和一个读入数据的函数
  - 一个可能做法：采用缓冲式输入，每次实际输入一行，分解为一个个浮点数，逐次返回
  - 定义两个函数完成工作，第一个打开文件，设置全局变量；第二个实际返回文件里的浮点数，调用一次返回一个

使用文件数据的实例

- 前面实现的缺点是使用了全局变量，其中保存的是内部数据，不安全。现在考虑用局部变量和函数完成这一工作
  - 定义几个局部变量，保存读入过程中的状态
  - 定义一个局部函数，每次调用返回下一个浮点数。再也没有更多浮点数的时候返回 **None**
  - 注意：主函数调用返回局部建立的函数对象后结束，但其局部环境仍然存在（因为这个局部函数对象正使用着该环境）
  - 每次调用这个主函数，建立一个新的函数对象。这样就可以同时用**read_floats(...)**函数打开多个输入文件
- 为完成这一工作，最方便的方式是定义一个生成器函数

文本处理实例

- 现在考虑开发一个函数，它读入一个文本（例如一部小说或其他著作），统计其中各个单词出现频率
- 情况：
  - 不知道文件里出现哪些单词，遇到单词需要记录
  - 单词可能重复出现，需要找到已经记录的单词，统计
  - 文本里的每个单词统计一次
- 基本设计：用一个字典，以单词作为关键码，相应的计数值作为关联值。在读文本遇到每一个单词时计数
  - 如果不在字典里，就加入字典，计数值设定为 1
  - 如果已经在字典里，计数值加一

  简单考虑：文本是空白字符分隔的一系列单词，一个单词就是非空白字符的一段连续序列