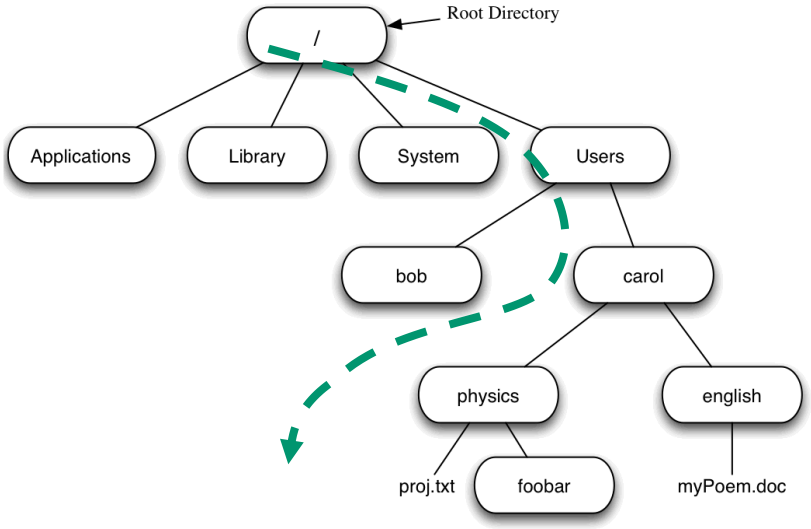
文件：使用外存里的数据

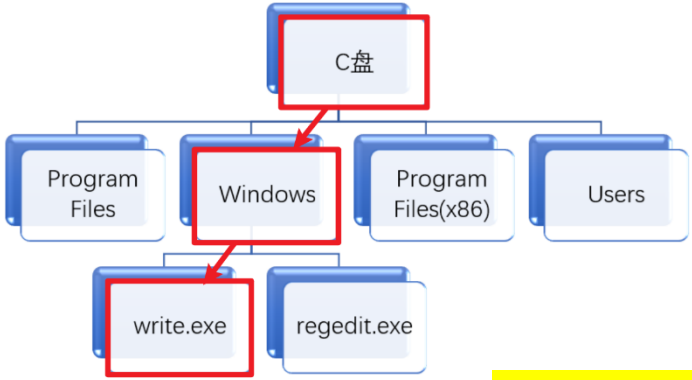
- 程序中能直接使用的数据都在内存，变量是内存存储功能的抽象
  - 程序变量及其关联的数据对象只在程序运行期间存在
  - 如果需要持久地保存数据，则需使用外存文件
- 文件是外部存储器 (磁盘/光盘/U 盘等) 里数据的基本组织和存储单元
  - 文件的要素：文件名、文件内容、文件属性
    - 一个文件可以包含任意多的数据
    - 每个文件有一个名字，程序通过文件名使用文件内容
  - 外存文件由计算机操作系统 (例如 Windows/IOS/Linux 等) 管理
    - 实际计算机系统管理大量文件，一般将文件分门别类，组织在分层次的目录 (也称文件夹) 结构里，以便使用

外存文件的组织

- 目录结构：文件和目录的集合
  - 分层次、树状结构
  - 根目录、父目录、子目录
- 路径 path：描述文件的存储位置
  - 绝对路径：从根目录开始
  - 相对路径：从“当前工作”目录开始
- 路径分隔符："/"、"\"



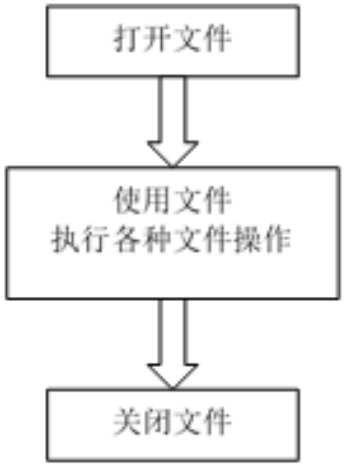
/Users/carol/physics/proj.txt



C:\Windows\write.exe

在程序里使用文件的基本情况

- 外存文件来自程序外部，由操作系统管理，程序里需通过文件名使用
  - 要使用一个文件，必须先要求操作系统找到该文件，并在程序里为该文件建立一个对象 (文件对象)，而后通过该对象使用文件 (上述操作即为打开文件)
  - 把得到的文件对象关联到一个变量，而后通过该变量操作相应的文件 (读取或写入数据)
  - 如果文件不再被使用，程序应撤销与它的联系，称为关闭文件 (一个程序正常结束时，操作系统会关闭它正在用的所有文件)
- 文件可被分为正文文件和二进制文件
  - 正文文件 (text file)：文件内容为某个字符集 (如 Unicode) 中字符的序列，通常划分为一系列正文行
  - 二进制文件 (binary file)：文件内容为任意的二进制编码，其中没有行的概念



图：文件使用流程

外存文件的使用模式

- 要在程序里使用 (或创建) 外存文件，需要使用编程语言提供的文件操作功能
- 程序里使用文件的不同方式
  1. 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
  2. 把程序计算产生的结果存入文件 (例如需要持久地保存)
     - 只向文件写入信息，“写方式”，通常是写一个新文件
     - 向一个已存在文件的末尾添加新内容，“附加方式”
  3. 对于长时间执行的程序，其运行期间可能需要保存大量中间数据，又可能需要把保存的数据重新装入内存使用
     - “读写方式”使用文件，是较复杂的文件使用方式，需要在读、写方式之间转换，可能存在读写位置的问题

Python 的文件功能

- 在 Python 程序里使用文件，首先要建立与被使用的实际文件之间的关联 — 打开文件
- 文件打开的简单形式： **fileobj = open(file, mode='r', 其它参数)**
  - 返回值：与被打开文件关联的文件对象 (file object，某种流式数据)
    - 之后，程序通过 fileobj 来操作所打开的文件
  - 参数：(普通使用时，不需要提供其它参数)
    - file：字符串，描述所要打开的文件 (名)
    - mode：字符串，描述文件打开的方式

```
>>> fin = open('input.txt', 'rt')        # 以读、文本方式打开文件
# 等价于 fin = open('input.txt'), fin = open('input.txt', 'r')
>>> fout = open('output.txt', 'wt')   # 以写、文本方式打开文件
# 等价于 fout = open('output.txt', 'w')
```

打开文件的不同模式

- mode 实参字符串里可以出现的字符：

r	读模式打开文件 (默认方式)
w	写模式打开文件 (如文件存在，则重写新内容；否则创建新文件)
x	排他性地创建文件 (如果所指名的文件已存在，就报 OSError 错误) 通常和 w 结合使用，排他性地创建写文件
a	附加方式的写模式打开，在文件中已有内容之后写入 (如果指定的文件不存在则创建新文件)
+	更新文件，不会单独出现：r+ 表示保留原文件内容，从头开始读或写；w+ 表示清除已有内容 (如果文件存在)，但操作中可以读或写；x+ 与 w+ 类似，但排他性地创建文件，从头开始，操作中可以写或读；a+ 也与 w+ 类似，但不清除已有内容，从已有内容之后开始写或读

文件的打开与关闭

- 打开文件有两种基本方式：正文方式 vs 二进制方式
  - 具体方式应该符合所打开外存文件的实际情况
  - (本课程主要考虑正文方式 Text I/O，即以 str 为基础来操作文件)
- mode 的实参字符串里，有字符 b 表示按二进制方式打开文件
  - 如果其中没有字符 b，或者有字符 t，则表示按正文方式打开
- open 函数的完整形式：(细节见手册)

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
    Open file and return a corresponding file object. If the file cannot be opened, an OSError is raised.
```

- 当所打开的外存文件使用完毕后，程序应该用 close 方法将其关闭：

```
>>> fin.close()
>>> fout.close()
```

文件描述 file 的实参

- **file 的实参字符串**

1. 相对“当前工作目录”路径描述：所打开的文件在“当前目录/文件夹”(默认是正在运行程序所在的目录) 的某个子目录下，需用 "子目录的名字/文件名"，且允许有多层子目录

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

   - 最简单的情况：打开“当前目录/文件夹”下的文件，只需用 "文件名"

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

2. 绝对路径描述：从最上层目录开始描述文件 (以字符 / 开始)；也可以指定盘符 (默认为当前盘，即程序所在的盘)

```
>>> file3 = open(" /Users/gan/Desktop/xiaoli2017-2018.doc")
>>> file4 = open(" C:/CTEX/Readme.txt ")      # 将被正确识别
```

- **Python 统一采用字符 / 作为路径分隔符，以屏蔽操作系统之间差异**

打开文件时的各种情况

- 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？
  - ……
- 如果无法打开文件，**open** 报 **OSError** 或更具体的错误
  - 以 **r** 方式打开，文件不存在时报 **FileNotFoundError**
  - 所打开名字是目录时报 **IsADirectoryError**
  - 以 **x** 方式打开，同名文件已存在时报 **FileExistsError**
  - 打开文件夹或者权限错误时报 **PermissionError**
  - ……

读入正文文件 (1)

文件对象的各种操作详见标准库模块 io

- 以读方式打开正文文件，得到的文件对象能用于读取文件内容：

```
>>> inf = open('file1.dat', 'r')    # 以正文读方式打开
>>> data = inf.read()               # data 得到文件全部内容的字符串
>>> data2 = inf.read()              # 已读完内容再读，data2 将得到空串
>>> f.read(n)                       # 从文件对象 f 读入 n 个字符
```

- readline 方法实现按正文行读入文件内容

```
>>> inf = open('file2.dat')
>>> line1 = inf.readline()          # 读入一行并做成字符串
>>> line2 = inf.readline()          # 读入下一行
```

  - readline 读到下一个换行符，该换行符作为结果字符串的末尾字符
  - 读入空行时得到只包含换行符的串 '\n'
  - 文件已读完再读则得到空串 ''

读入正文文件 (2)

- 按行读入时，“所读的串是否为空”常作为逻辑条件来控制读入循环

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

- 带参数的 f.readline(n)：最多读入 n 个字符，遇到换行符也结束
- readlines：与 read 方法类似，一次性读入整个文件的内容，返回一个字符串为元素的表，其中的每个元素对应文件里的一个字符行 (包括最后的换行符)

```
inf1 = open('file1.dat', 'r')
text1 = inf1.readlines()      # text1 是元素为各正文行的表
```

  - 带参数的调用形式控制读入的行数

读入正文文件 (3)

- 正文读文件对象是一种可迭代对象 (iterable)
  - 允许直接作为 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 是正文读文件对象
    … ... line ... …             # 处理一行文本
inf.close()                      # 常用的简洁写法
```

  - 也可以直接 (整体) 按行转换到 list 或 tuple

```
inf1 = open('file1.dat', 'r')
inf2 = open('file2.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

输出正文文件

- 基本的正文文件输出：

  f.write(...)　　*# f 应是一个正文写对象，实参是一个字符串*

  - 把参数字符串写入文件 f，返回实际写入文件的字符个数
  - 如输出的内容需要换行，必须在输出串里实际包含换行符
  - 如果要输出非字符串的数据，需要显示地用 str() 或 repr() 将其转换为字符串 (也可根据需求使用字符串格式化功能)

```
outf = open("outfile1.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')
```

- **f.writelines(lines)** 以一个字符串的表为参数，把一组字符串输出到文件 (Note：不会自动地添加行分隔符)

缓冲式输入输出

- 文件输入和输出默认采用缓冲方式
  - 调用 **open** 函数打开一个文件时，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
  - 对文件输入，程序不断从缓冲区读取数据，一旦缓冲区被读完，系统自动从文件里搬一批新数据填满缓冲区
  - 对文件输出，程序产生的输出被存入缓冲区，直到缓冲区被装满，系统才把这一批输出实际写入文件
  - 可在打开文件时通过 **open** 函数的 **buffering 参数**控制缓冲方式
- 调用 **close** 方法关闭文件时，系统会把当时缓冲区中 **(**尚未实际写入**)** 的内容全部写入外存文件
  - 如果程序中没有关闭输出文件，但又非正常地结束执行，输出文件内容的完整性没有保证
- 调用 **f.flush()** 可把当时缓冲区内容写入文件 **(**“冲刷”缓冲区**)**

文件处理实例 1：文件内容统计

- 问题：定义函数统计一个文件里各类字符出现的次数
  - 根据文件名 (字符串参数) 打开一个文件
  - 依次处理文件里的每个字符
    - 判别每个字符的类别 (字母、数字、空白和其他字符等)，并进行统计
  - 关闭文件，并输出统计结果
- 细节：文件里的字符可能超出基本的 ASCII 编码字符集 的范围
  - 打开文件时，可能需要通过函数 open 的 encoding 参数说明被读文本文件的编码方式 (编解码器 codec, encoder/decoder)
  - 例如，当读入文件里有中文，可以给 open 增加关键字实参 encoding="utf_8" (注：'utf8', 'utf-8', 'U8' 等是 'utf_8' 的别名)
- (演示)

文件处理实例 2：词频统计

- 需求：实现函数统计一个文本文件 (如一部小说或其他文献) 中各个单词出现的频率，并把统计结果输出至文件
  - 事先不知道文件包含哪些单词，遇到任意单词都需要记录
  - 单词可能重复出现，需要找到已经记录的单词更新统计
- 用字典记录统计结果
  - 关键字 → 单词，关联值 → 相应计数值
- 读文本过程中遇到每一个单词时均计数
  - 如果单词不在字典里，就加入字典，计数值设定为 1
  - 如果已经在字典里，计数值加 1
- 文本形式：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字符序列 (非常粗略地处理)
- (演示)