Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第十章　文件操作

- 通过标准输入/输出设备可以实现的数据输入/输出。
- 但在数据量大、数据访问频繁以及数据处理结果需长期保存的情况下，一般将数据以文件的形式保存。
- 数据文件是存储在外部介质（如磁盘）上的用文件名标识的数据集合。

- 数据以文件的形式进行存储，操作系统以文件为单位对数据进行管理，文件系统仍是高级语言普遍采用的数据管理方式。

10.1 文件的概念

1. 文件格式

- 文件（file）是存储在外部介质上一组相关信息的集合。
- 例：程序文件是程序代码的集合，数据文件是数据的集合。
- 每个文件都有一个名字，称为文件名。

- 操作系统把每一个与主机相连的输入/输出设备都作为文件来管理，称为标准输入/输出文件。
- 例：键盘是标准输入文件，显示器和打印机是标准输出文件。

- Python的文件可分为文本文件和二进制文件。
- 文本文件的每一个字节代表一个ASCII代码，即一个字符。
- 二进制文件是把数据按其在内存中的存储形式原样输出到磁盘上存放。
- 例：图形图像文件、音频视频文件、可执行文件等都是常见的二进制文件。

- 文本文件便于对字符进行逐个处理，也便于输出字符，但一般占用存储空间较多，而且要花费时间转换（二进制形式与ASCII码间的转换）。
- 用二进制形式输出数值，可以节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。
- 一般中间结果数据需要暂时保存在外存中且以后又需要读入到内存的，常用二进制文件保存。

2.文件操作

- 无论是文本文件还是二进制文件，其操作过程是相同的，即首先打开文件并创建文件对象，然后通过该文件对象对文件内容进行读/写操作，最后关闭文件。

- 文件的读（read）操作就是从文件中取出数据，再输入到计算机内存储器；
- 文件的写（write）操作是向文件写入数据，即将内存数据输出到磁盘文件。
- 读/写操作是相对于磁盘文件而言的，而输入/输出操作是相对于内存储器而言的。
- 文件的读/写过程就是数据的输入/输出过程。
- “读”与“输入”、“写”与“输出”指的是同一过程。

10.2 文件的打开与关闭

- 在对文件进行读/写操作之前，首先要打开文件，操作结束后应该关闭文件。
- Python提供了文件对象，通过open()函数可以按指定方式打开指定文件并创建文件对象。

10.2.1 打开文件

- 所谓打开文件是指在程序和操作系统之间建立起联系，程序把所要操作文件的一些信息通知给操作系统。
- 这些信息中除包括文件名外，还要指出读/写方式及读/写位置。

- 如果是读操作，则需要先确认此文件是否已存在；
- 如果是写操作，则检查原来是否有同名文件，如有则先将该文件删除，然后新建一个文件，并将读/写位置设定于文件开头，准备写入数据。

1.open()函数

- 要读取或写入文件，必须使用内置的open()函数来打开它。
- 该函数创建一个文件对象，可以使用文件对象来完成各种文件操作。
- open()函数的一般调用格式为：
- 文件对象=open(文件说明符[,打开方式][,缓冲区])
- 其中，文件说明符指定打开的文件名，可以包含盘符、路径和文件名，它是一个字符串。

- 注意：文件路径中的“\”要写成“\\”。
- 例：要打开e:\my中的test.dat文件，文件说明符要写成“e:\\my\\test.dat”。
- 打开方式指定打开文件后的操作方式，该参数是字符串，必须小写。
- 文件操作方式是可选参数，默认为r（只读操作）。
- 文件操作方式用具有特定含义的符号表示。

文件操作方式

打开方式	含　义	打开方式	含　义
r（只读）	为输入打开一个文本文件	r+（读/写）	为读/写打开一个文本文件
w（只写）	为输出打开一个文本文件	w+（读/写）	为读/写建立一个新的文本文件
a（追加）	向文本文件尾增加数据	a+（读/写）	为读/写打开一个文本文件
rb（只读）	为输入打开一个二进制文件	rb+（读/写）	为读/写打开一个二进制文件
wb（只写）	为输出打开一个二进制文件	wb+（读/写）	为读/写建立一个新的二进制文件
ab（追加）	向二进制文件尾增加数据	ab+（读/写）	为读/写打开一个二进制文件

- 缓冲区设置表示文件操作是否使用缓冲存储方式。
- 如果缓冲区参数被设置为0，则表示不使用缓冲存储。
- 如果该参数设置为1，则表示使用缓冲存储。
- 如果指定的缓冲区参数为大于1的整数，则使用缓冲存储，并且该参数指定了缓冲区的大小。
- 如果缓冲区参数指定为-1，则使用缓冲存储，并且使用系统默认缓冲区的大小，这也是缓冲区参数的默认设置。

- open()函数以指定的方式打开指定的文件，文件操作方式符的含义是：
- （1）用“r”方式打开文件时，只能从文件向内存输入数据，而不能从内存向该文件写数据。
- 以“r”方式打开的文件应该已经存在，不能用“r”方式打开一个并不存在的文件，否则将出现FileNotfoundError错误。这是默认打开方式。

- （2）用“w”方式打开文件时，只能向该文件写数据。
- 如果该文件原来不存在，则打开时建立一个以指定文件名命名的文件。
- 如果原来的文件已经存在，则打开时将文件删空，然后重新建立一个新文件。

- （3）如果希望向一个已经存在的文件的尾部添加新数据（保留原文件中已有的数据），则应用“a”方式打开。
- 如果该文件不存在，则创建并写入新的文件。
- 打开文件时，文件的位置指针在文件末尾。

- （4）用“r+”，“w+”，“a+”方式打开的文件可以写入和读取数据。
- 用“r+”方式打开文件时，该文件应该已经存在，这样才能对文件进行读/写操作；
- 用“w+”方式打开文件时，如果文件存在，则覆盖现有的文件。如果文件不存在，则创建新的文件并可进行读取和写入操作；

- 用“a+”方式打开的文件，则保留文件中原有的数据，文件的位置指针在文件末尾，此时，可以进行追加或读取文件操作。
- 如果该文件不存在，它创建新文件并可进行读取和写入操作。

2.文件对象属性

- 文件一旦被打开，通过文件对象的属性可以得到有关该文件的各种信息。
- 文件属性的引用方法为：
- 文件对象名.属性名

文件对象属性

属 性	含　　义
closed	如果文件被关闭则返回 True，否则返回 False
mode	返回该文件的打开方式
name	返回文件的名称

- 例：
- fo=open("f.txt","wb")
- print("Name of the file:",fo.name)
- print("Closed or not:",fo.closed)
- print("Opening mode:",fo.mode)

```
Name of the file: f.txt
Closed or not: False
Opening mode: wb
```

3.文件对象方法

文件对象常用方法

方　法	含　义
close()	把缓冲区的内容写入磁盘，关闭文件，释放文件对象
flush()	把缓冲区的内容写入磁盘，不关闭文件
read([count])	如果有 count 参数，则从文件中读取 count 个字节。如果省略 count，则读取整个文件的内容
readline()	从文本文件中读取一行内容
readlines()	从文本文件中读取所有行，也就是读取整个文件的内容。把文件每一行作为列表的成员，并返回这个列表
seek(offset[, where])	把文件指针移动到相对于 where 的 offset 位置。where 为 0 表示文件开始处，这是默认值；1 表示当前位置；2 表示文件结尾
tell()	获得当前文件指针位置
truncate([size])	删除从当前指针位置到文件末尾的内容。如果指定了 size，则不论指针在什么位置都留下前 size 个字节，其余的被删除
write(string)	把 string 字符串写入文件（文本文件或二进制文件）
writelines(list)	把 list 列表中的字符串一行一行地写入文本文件，是连续写入文件，没有换行
next()	返回文件的下一行，并将文件操作标记移到下一行

- **close()**：把缓冲区的内容写入磁盘，关闭文件，释放文件对象。
- **flush()**：把缓冲区的内容写入磁盘，不关闭文件。
- **read([count])**：如果有count参数，则从文件中读取count个字节。如果省略count，则读取整个文件的内容。

- **readline()**：从文本文件中读取一行内容。
- **readlines()**：从文本文件中读取所有行，也就是读取整个文件的内容。把文件每一行作为列表的成员，并返回这个列表。
- **seek(offset[,where])**：把文件指针移动到相对于where的offset位置。where为0表示文件开始处，这是默认值；1表示当前位置；2表示文件结尾。

- **tell()**：获得当前文件指针位置。
- **truncate([size])**：删除从当前指针位置到文件末尾的内容。如果指定了size，则不论指针在什么位置都留下前size个字节，其余的被删除。
- **write(string)**：把string字符串写入文件（文本文件或二进制文件）

- **writelines(list)**：把list列表中的字符串一行一行地写入文本文件，是连续写入文件，没有换行。
- **next()**：返回文件的下一行，并将文件操作标记移到下一行。

10.2.2 关闭文件

- 文件使用完毕后，应当关闭，这意味着释放文件对象以供别的程序使用，同时也可以避免文件中数据的丢失。
- 用文件对象的close()方法关闭文件，其调用格式为：
- close()

- **close()**方法用于关闭已打开的文件，将缓冲区中尚未存盘的数据写入磁盘，并释放文件对象。
- 此后，如果再想使用刚才的文件，则必须重新打开。
- 应该养成在文件访问完之后及时关闭的习惯，一方面是避免数据丢失，另一方面是及时释放内存，减少系统资源的占用。

- 例：
- fo=open("f.txt","wb")
- print("Opening mode:",fo.mode)
- fo.close()

10.3 文本文件的操作

- 文本文件是指以ASCII码方式存储的文件：英文、数字等字符存储的是ASCII码，而汉字存储的是机内码。
- 文本文件中除了存储文件有效字符信息（包括能用ASCII码字符表示的回车、换行等信息）外，不能存储其他任何信息。

- 文本文件的优点：方便阅读和理解，使用常用的文本编辑器或文字处理器就可以对其创建和修改。

10.3.1 文本文件的读取

- Python对文件的操作都是通过调用文件对象的方法来实现的，文件对象提供了read()、readline()和readlines()方法用于读取文本文件的内容。

1. read()方法

- read()方法的用法：
- 变量=文件对象.read()
- 其功能是读取从当前位置直到文件末尾的内容，并作为字符串返回，赋给变量。
- 如果是刚打开的文件对象，则读取整个文件。
- read()方法通常将读取的文件内容存放到一个字符串变量中。

- read()方法也可以带有参数：
- 变量=文件对象.read(count)
- 其功能是读取从文件当前位置开始的count个字符，并作为字符串返回，赋给变量。
- 如果文件结束，就读取到文件结束为止。
- 如果count大于文件从当前位置到末尾的字符数，则仅返回这些字符。

- 用Python解释器或Windows记事本建立文本文件my.txt，其内容如下：
- Good good study, day day up.
- Open yellow gun.
- 例：
- >>> fo=open("c:\\user\\my.txt","r")
- >>> fo.read()
- 'Good good study, day day up.\nOpen yellow gun.\n'
- >>> fo=open("c:\\user\\my.txt","r")
- >>> fo.read(4)
- 'Good'

- 例：已经建立文本文件my.txt，统计文件中元音字母出现的次数。
- 分析：先读取文件的全部内容，得到一个字符串，然后遍历字符串，统计元音字母的个数。
- 程序如下：

- infile=open("my.txt","r")
- #打开文件，准备输出文本文件
- s=infile.read()　　#读取文件全部字符
- print(s)　　#显示文件内容
- n=0
- for c in s:　　#遍历读取的字符串
- 　if c in 'aeiouAEIOU':n+=1
- print(n)
- infile.close() #关闭文件

```
Good good study, day day up.
Open yellow gun.

13
```

2. readline()方法

- readline()方法的用法如下：
- 变量=文件对象.readline()
- 其功能是读取从当前位置到行末（即下一个换行符）的所有字符，并作为字符串返回，赋给变量。
- 通常用此方法来读取文件的当前行，包括行结束符。
- 如果当前处于文件末尾，则返回空串。

- 例：
- >>> fo=open("my.txt","r")
- >>> fo.readline()
- 'Good good study, day day up.\n'
- >>> fo.readline()
- 'Open yellow gun.\n'
- >>> fo.readline()
- ''

- 例：已经建立文本文件my.txt，统计文件中元音字母出现的次数。用readline()方法实现。
- 分析：逐行读取文件，得到一个字符串，然后遍历字符串，统计元音字母的个数。当文件读取完毕，得到一个空串，控制循环结束。
- 程序如下：

```
infile=open("my.txt","r")
#打开文件，准备输出文本文件
s=infile.readline()          #读取一行
n=0
while s!='':                 #还没有读完时继续循环
    print(s[:-1])            #显示文件内容
    for c in s:              #遍历读取的字符串
        if c in 'aeiouAEIOU':n+=1
    s=infile.readline()      #读取下一行
print(n)
infile.close()  #关闭文件
```

```
Good good study, day day up.
Open yellow gun.
13
```

- 程序中“print(s[:-1])”一句用“[:-1]”去掉每行读入的换行符。
- 如果输出的字符串末尾带有换行符，输出会自动跳到下一行，再加上print()函数输出完后换行，这样各行之间会输出一个空行。
- 也可以用字符串的strip()方法去掉最后的换行符，即用语句“print(s.strip())”替换语句“print(s[:-1])”。

3. readlines()方法

- readlines()方法的用法如下：
- 变量=文件对象.readlines()
- 其功能是读取从当前位置直到文件末尾的所有行，并将这些行构成列表返回，赋给变量。列表中的元素即每一行构成的字符串。如果当前处于文件末尾，则返回空列表。

3. readlines()方法

- readlines()方法的用法如下：
- 变量=文件对象.readlines()
- 其功能是读取从当前位置直到文件末尾的所有行，并将这些行构成列表返回，赋给变量。
- 列表中的元素即每一行构成的字符串。
- 如果当前处于文件末尾，则返回空列表。

- 例：
- >>> fo=open("my.txt","r")
- >>> fo.readlines()
- ['Good good study, day day up.\n', 'Open yellow gun.\n']

- 例：已经建立文本文件my.txt，统计文件中元音字母出现的次数。用readlines()方法实现。
- 分析：读取文件所有行，得到一个字符串列表，然后遍历列表，统计元音字母的个数。
- 程序如下：

```
infile=open("my.txt","r")
#打开文件，准备输出文本文件
ls=infile.readlines()  #读取各行，得到一个列表
n=0
for s in ls:                #遍历列表
    print(s[:-1])           #显示文件内容
    for c in s:             #遍历列表的字符串元素
        if c in 'aeiouAEIOU':n+=1
print(n)
infile.close()  #关闭文件
```

```
Good good study, day day up.
Open yellow gun.
13
```

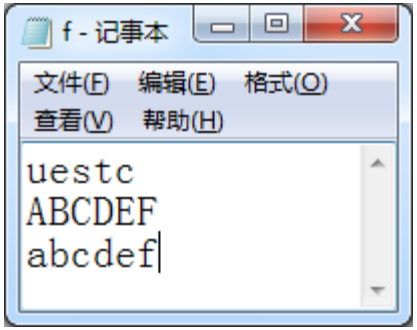
10.3.2 文本文件的写入

- 当文件以写方式打开时，可以向文件写入文本内容。
- Python文件对象提供两种写文件的方法：
- （1）write方法
- （2）writelines()方法

1. write()方法

- write()方法的用法如下：
- 文件对象.write(字符串)
- 其功能是在文件当前位置写入字符串，并返回字符的个数。
- 每次write()方法执行完后并不换行，如果需要换行则在字符串最后加换行符“\n”。

- 例：
- >>> fo=open("f.dat","w")
- >>> fo.write("uestc")
- 5
- >>> fo.write("\nABCDEF\n")
- 8
- >>> fo.write("abcdef")
- 6
- >>> fo.close()



- 例：从键盘输入若干字符串，逐个将它们写入文件f.txt中，直到输入“*”时结束。然后从该文件中逐个读出字符串，并在屏幕上显示出来。
- 分析：输入一个字符串，如果不等于“*”则写入文件，然后再输入一个字符串，进行循环判断，直到输入“*”结束循环。
- 程序如下：

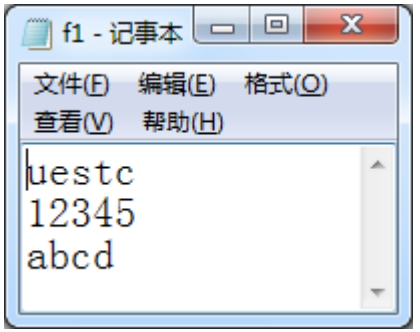
- fo=open("f.txt","w") #打开文件,准备建立文本文件
- print("输入多行字符串(输入“*”结束):")
- s=input()　#从键盘输入一个字符串
- while s!="*":　#不断输入，直到输入结束标志“*”
- 　fo.write(s+'\n')　#向文件写入一个字符串
- 　s=input()　#从键盘输入一个字符串
- fo.close()
- fo=open("f.txt","r")　#打开文件,准备读取文本文件
- s=fo.read()
- print("输出文本文件:")
- print(s.strip())

```
输入多行字符串(输入“*”结束):
Good good study, day day up.
Open yellow gun.
uestc*
*
输出文本文件:
Good good study, day day up.
Open yellow gun.
uestc*
```

2. writelines()方法

- writelines()方法的用法如下：
- 文件对象.writelines(字符串元素的列表)
- 其功能是在文件当前位置处依次写入列表中的所有字符串。
- writelines()方法接收一个字符串列表作为参数，将它们写入文件，它并不会自动加入换行符，如果需要，必须在每一行字符串结尾加上换行符。

- 例：
- >>> fo=open("f1.dat","w")
- >>> fo.writelines(["uestc","\n12345\n","abcd"])
- >>> fo.close()



- 例：从键盘输入若干字符串，逐个将它们写入文件f1.txt的尾部，直到输入“*”时结束。然后从该文件中逐个读出字符串，并在屏幕上显示出来。
- 分析：首先以“a”方式打开文件，当前位置定位在文件末尾，可以继续写入文本而不改变原有的文件内容。本例考虑先输入若干个字符串，并将字符串存入一个列表中，然后通过writelines()方法将全部字符串写入文件。
- 程序如下：

```
print("输入多行字符串(输入“*”结束):")
lst=[]
while True:  #不断输入，直到输入结束标志“*”
    s=input()   #从键盘输入一个字符串
    if s=="*":break
    lst.append(s+'\n')  #将字符串附加在列表末尾
fo=open("f.txt","a")  #打开文件,准备追加文本文件
fo.writelines(lst) #向文件写入一个字符串
fo.close()
fo=open("f.txt","r") #打开文件,准备读取文本文件
s=fo.read()
print("输出文本文件:")
print(s.strip())
```

```
def main():
    f=open("d:\\f.txt","w")
    for i in range(1,10):
        for j in range(1,i+1):
            k=i*j
            s=str(j)+'x'+str(i)+'='+str(k)+' '+('' if k>9 else ' ')
            f.write(s)
        f.write('\n')
    f.close()
    f=open("d:\\f.txt","r")
    s=f.read()
    print(s)
    f.close()
main()
```

10.4 二进制文件的操作

- 从文本文件的第一个数据开始，依次进行读/写，称为顺序文件（Sequential File）。
- 如需对文件中某个特定的数据进行处理，这就要求对文件具有随机读/写的功能，也就是强制将文件的指针指向用户所希望的指定位置。
- 这类可以任意读/写的文件称为随机文件（Random File）。

- 文本文件的信息表示单位至少是一个字符，有些字符用一个字节，有些字符可能用多个字节。
- 二进制文件的存储内容为字节码，甚至可以用一个二进制位来代表一个信息表示单位（位操作）。
- 二进制文件一般采用随机存取。

10.4.1 文件的定位

- 文件中有一个位置指针，指向当前的读/写位置，读/写一次指针向后移动一次（一次移动多少字节，由读/写方式而定）。
- 若要主动调整指针位置，可用系统提供的文件指针定位函数。

1. tell()方法

- tell()方法的用法如下：
- 文件对象.tell()
- 其功能是告诉文件位置指针的当前位置，即相对于文件开始位置的字节数，下一个读取或写入操作将发生在当前位置。
- 文件刚打开时的文件指针指向第一个字符的位置，即为0。

- 例：
- >>> fo=open("f.txt","r")
- >>> fo.tell()
- 0
- >>> fo.read(4)
- 'Good'
- >>> fo.tell()
- 4 #这是读取4个字符以后的文件指针位置。

2. seek()方法

- seek()方法的用法如下：
- 文件对象.seek(偏移[,参考点])
- 其功能是更改当前的文件指针位置。
- 偏移参数指示要移动的字节数，移动时以设定的参考点为基准。
- 偏移为正数表示朝文件尾方向移动，偏移为负数表示朝文件头方向移动；
- 参考点指定移动的基准位置。

- 文件对象.seek(偏移[,参考点])
- 如果参考点被设置为0，则意味着使用该文件的开始处作为基准位置（这是默认的情况）；
- 设置为1，则是使用当前位置作为基准位置；
- 如果它被设置为2，则该文件的末尾将被作为基准位置。

- 例：
- >>> fo=open("f.txt","rb") #以二进制读方式打开文件
- >>> fo.read()
- b'Good good study, day day up.\r\nOpen yellow gun.\r\nuestc*\r\n'
- >>> fo.read()
- b''

- “f.txt”是一个文本文件，可以用文本方式读取，也可以用二进制方式读取，两者的差别仅仅体现在回车换行符的处理上。
- 二进制读取时需要将“\n”转换成“\r\n”，即多出一个字符。
- 当文件中不存在回车换行符时，文本读与二进制读的结果是一样的。
- 文件所有字符被读完后，文件读/写位置位于文件末尾，再读则读出空串。

- 例：从文件开始处移动10个字节后读取文件全部字符。
- **>>> fo.seek(10,0)**
- **10**
- **>>> fo.read()**
- **b'study, day day up.\r\nOpen yellow gun.\r\nuestc*\r\n'**

- 例：
- >>> fo.seek(10,0)
- 10
- >>> fo.seek(10,1)
- 20
- >>> fo.seek(-10,1)
- 10
- >>> fo.seek(0,2)
- 56
- >>> fo.seek(-10,2)
- 46
- >>> fo.seek(10,2)
- 66

- 注意：
- 文本文件也可以使用seek()方法，但Python3.x限制文本文件只能相对于文件起始位置进行位置移动。
- 当相对于当前位置和文件末尾进行位置移动时，偏移量只能取0，seek(0,1)和seek(0,2)分别定位于当前位置和文件末尾。

- 例：
- >>> fo=open("f.txt","r") #以文本读的方式打开文件
- >>> fo.read()
- 'Good good study, day day up.\nOpen yellow gun.\nuestc*\n'
- >>> fo.seek(10,0)
- 10
- >>> fo.seek(0,1)
- 10
- >>> fo.seek(0,2)
- 56

10.4.2 二进制文件的读写

- 使用open()函数打开文件时，在打开方式中加上“b”，如“rb”、“wb”、“ab”等，以打开二进制文件。
- 文本文件存放的是与编码对应的字符，而二进制文件直接存储字节编码。

1. read()方法和write()方法

- 二进制文件的读取与写入可以使用文件对象的read()和write()方法。

- 例：从键盘输入一个字符串，以字节数据写入二进制文件；从文件末尾到文件头依次读取一个字符，对其加密后反向输出全部字符。加密规则是，对字符编码的中间两个二进制位取反。
- 分析：对中间两个二进制位取反的办法是，将读出的字符编码与二进制数**00011000**（也就是十进制数**24**）进行异或运算，将异或后的结果写回原位置。
- 程序如下：

```
s=input('输入一个字符串:')
s=s.encode()  #变成字节数据
fo=open("f1.txt","wb") #建立二进制文件
fo.write(s)
fo.close()
fo=open("f1.txt","rb") #读二进制文件
lst=[]
for n in range(1,len(s)+1):
    fo.seek(-n,2)  #文件定位从最后一个字符到第一个字符
    s=fo.read(1)  #读一个字节
    s=chr(ord(s.decode())^24)  #加密处理
    lst.append(s)
lst="".join(lst)  #将序列元素组合成字符串
print(lst)
fo.close()
```

```
输入一个字符串:uestc
{1k}m
```

2. struct模块

- read()和write()方法以字符串为参数，对于其他类型数据需要进行转换。
- Python没有二进制类型，但可以存储二进制类型的数据，就是用字符串类型来存储二进制数据。
- Python中struct模块的pack()和unpack()方法可以处理这种情况。

- **pack()**函数可以把整型（或者浮点型）打包成二进制的字符串（**Python**中的字符串可以是任意字节）。

- 例：
- >>> import struct
- >>> a=65
- >>> bytes=struct.pack('i',a) #将a变为二进制字符串
- >>> bytes
- b'A\x00\x00\x00'
- #此时，bytes就是一个4字节字符串

- 例：写文件
- **>>> fo=open("f1.txt","wb")**
- **>>> fo.write(bytes)**
- **4**
- **>>> fo.close()**

- 读文件的时候，可以一次读出4个字节，然后用unpack()方法转换成Python的整数。
- 注意：unpack()方法执行后得到的结果是一个元组。

- 例：
- >>> fo=open("f1.txt","rb")
- >>> bytes=fo.read(4)
- >>> a=struct.unpack('i',bytes)
- >>> a
- (65,)

- 如果写入的数据是由多个数据构成的，则需要在pack()方法中使用格式串。

- 例：
- >>> a=b'hello'
- >>> b=b'uestc!'
- >>> c=2
- >>> d=45.123
- >>> bytes=struct.pack('5s6sif',a,b,c,d)
- >>> bytes
- b'hellouestc!\x00\x02\x00\x00\x00\xf4}4B'

- 此时的bytes就是二进制形式的数据了，可以直接写入二进制文件。
- 例：
- >>> fo=open("f2.txt","wb")
- >>> fo.write(bytes)
- 20
- >>> fo.close()

- 当需要时可以读出来，再通过struct.unpack()方法解码成Python变量。
- 例：
- >>> fo=open("f2.txt","rb")
- >>> bytes=fo.read(4)
- >>> a,b,c,d=struct.unpack('5s6sif',bytes)
- >>> a,b,c,d
- (b'hello', b'uestc!', 2, 45.123)
- #在unpack()方法中，“5s6sif”称为格式化字符串，由数字加字符构成，5s表示占5个字符的字符串，2i表示2个整数等。

unpack()方法的可用的格式符及对应的 Python 类型

格式符	Python 类型	字节数	格式符	Python 类型	字节数
b	整型	1	B	整型	1
h	整型	2	H	整型	2
i	整型		I	整型	4
l	整型	4	L	整型	4
q	整型	8	Q	整型	8
p	字符串	1	P	整型	
f	浮点型	4	d	浮点型	8
s	字符串	1	?	布尔型	1
c	单个字符	1			

3. pickle模块

- 字符串很容易从文件中读/写，数值则需要更多的转换，当处理更复杂的数据类型时，例如列表、字典等，这些转换更加复杂。
- **Python带有一个pickle模块，用于把Python的对象**（包括内置类型和自定义类型）直接写入到文件中，而不需要先把它们转化为字符串再保存，也不需要用底层的文件访问操作把它们写入到一个二进制文件里。

- 在pickle模块中有2个常用的方法：dump()和load()。
- dump()方法的用法如下：
- pickle.dump(数据,文件对象)
- 其功能是直接把数据对象转换为字节字符串，并保存到文件中。

- 例：创建二进制文件ff。
- import pickle
- info={'one':1,'two':2,'three':3}
- f=open('ff','wb')
- pickle.dump(info,f)
- f.close()

- load()方法的用法如下：
- 变量=pickle.load(文件对象)
- 其功能与dump()方法相反。
- load()方法从文件中读取字符串，将它们转换为Python的数据对象，可以像使用通常的数据一样来使用它们。

- 例：显示二进制文件ff的内容。
- import pickle
- f2=open("ff","rb")
- s=pickle.load(f2)
- f2.close()
- print(s)

{'one': 1, 'two': 2, 'three': 3}

10.5 文件管理方法

- Python的os模块提供了类似于操作系统级的文件管理功能，如文件重命名、文件删除、目录管理等。
- 要使用这个模块，需要先导入它，然后调用相关的方法。

1. 文件重命名

- rename()方法实现文件重命名，它的一般格式为：
- os.rename("当前文件名","新文件名")
- 例：将文件t.txt重命名为tt.txt。
- >>> import os
- >>> os.rename("t.txt","tt.txt")

2. 文件删除

- 可以使用remove()方法来删除文件，一般格式为：
- os.remove("文件名")
- 例：删除现有文件t.txt。
- >>> import os
- >>> os.remove("t.txt")

3. Python中的目录操作

（1）mkdir()方法

- mkdir()方法在当前目录下创建目录。
- 一般格式为：
- os.mkdir("新目录名")
- 例：在当前盘当前目录下创建test目录。
- >>> import os
- >>> os.mkdir("test")

（2）chdir()方法

- 可以使用chdir()方法来改变当前目录。
- 一般格式为：
- os.chdir("要成为当前目录的目录名")
- 例：将“c:\user\test”目录设定为当前目录。
- >>> import os
- >>> os.chdir("c:\\user\\test")

（3）getcwd()方法

- getcwd()方法显示当前的工作目录。
- 一般格式为：
- os.getcwd()
- 例：显示当前目录。
- >>> import os
- >>> os.getcwd()

（4）rmdir()方法

- rmdir()方法删除空目录。
- 一般格式为：
- os.rmdir("待删除目录名")
- 在用rmdir()方法删除一个目录前，先要删除目录中的所有内容。
- 例：删除空目录“c:\test”。
- >>> import os
- >>> os.rmdir("c:\\test")

10.6 文件操作应用举例

- 例：有两个文件f1.txt和f2.txt，各存放一行已经按升序排列的字母，要求依然按字母升序排列，将两个文件中的内容合并，输出到一个新文件f.txt中去。

- 分析：分别从两个有序的文件读出一个字符，将ASCII值小的字符写到f.txt文件，直到其中一个文件结束而终止。最后将未结束文件复制到f.txt文件，直到该文件结束而终止。

```
def ftcomb(fname1,fname2,fname3):     #文件合并
    fo1=open(fname1,"r")
    fo2=open(fname2,"r")
    fo3=open(fname3,"w")
    c1=fo1.read(1)
    c2=fo2.read(1)
    while c1!="" and c2!="":
        if c1<c2:
            fo3.write(c1)
            c1=fo1.read(1)
        elif c1==c2:
            fo3.write(c1)
            c1=fo1.read(1)
            fo3.write(c2)
            c2=fo2.read(1)
        else:
            fo3.write(c2)
            c2=fo2.read(1)
    while c1!="":           #文件1复制未结束
        fo3.write(c1)
        c1=fo1.read(1)
    while c2!="":           #文件2复制未结束
        fo3.write(c2)
        c2=fo2.read(1)
    fo1.close()
    fo2.close()
    fo3.close()
def ftshow(fname):      # 输出文本文件
    fo=open(fname,"r")
    s=fo.read()
    print(s.replace('\n',''))   #去掉字符串中的换行符后输出
    fo.close()
def main():
    ftcomb("f1.txt","f2.txt","f.txt")
    ftshow("f.txt")
main()
```

- 假设f1.txt的内容为：ACEFb
- f2.txt的内容为：BDGHacd
- 则运行后f.txt的内容为：ABCDEFGHabcd
- 屏幕显示内容为：ABCDEFGHabcd

- 例：在number.dat文件中放有若干个不小于2的正整数（数据间以逗号分隔），编写程序实现：
- （1）在prime()函数中判断和统计这些整数中的素数以及个数。
- （2）在主函数中将number.dat中的全部素数以及素数个数输出到屏幕上。

```
def prime(a,n):    #判断列表a中的n个元素是否素数
    k=0
    for i in range(0,n):
        flag=1    #素数标志
        for j in range(2,a[i]):
            if a[i]%j==0:
                flag=0
                break
        if flag:
            a[k]=a[i]    #将素数存入列表
            k+=1         #统计素数个数
    return k
def main():
    fo=open("number.dat","r")
    s=fo.read()
    fo.close()
    x=s.split(sep=',')          #以“,”为分隔符将字符串分割为列表
    for i in range(0,len(x)):   #将列表元素转换成整型
        x[i]=int(x[i])
    m=prime(x,len(x))
    print('全部素数为:',end=' ')
    for i in range(0,m):
        print(x[i],end=' ')       #输出全部素数
    print()                      #换行
    print('素数的个数为:',end=' ')
    print(m)                     #输出素数个数
main()
```

- 假设number.dat的文件内容为：
- **2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23**
- 程序输出结果为：
- 全部素数为：**2 3 5 7 11 13 17 19 23**
- 素数的个数为：**9**

- 例：对一个BMP图像文件进行操作，使其高度缩为原来的一半。
- 分析：图像文件是一种典型的二进制文件，利用二进制文件的操作方法可以实现图像文件的读写。
- BMP（bitmap）图像文件格式采用位映射存储格式，除了图像深度可选以外，不采用其他任何压缩，因此，BMP文件包含的图像信息较丰富，但所占用的存储空间较大。

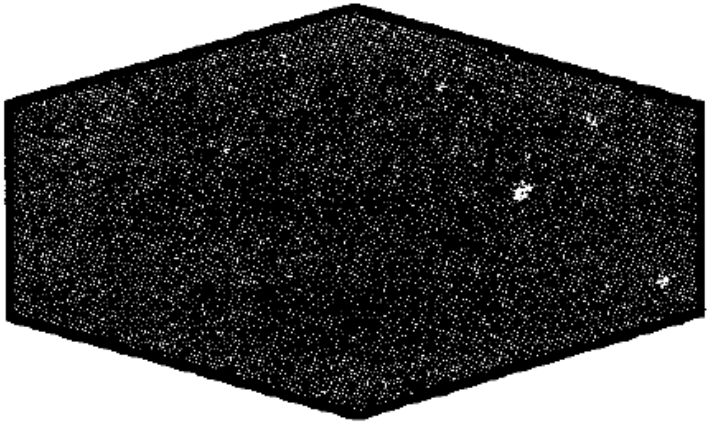
- BMP文件的图像深度可选1位（双色）、4位（16色）、8位（256色）或24位（真彩色）。BMP文件存储数据时，图像的扫描方式是按从左到右、从下到上的顺序。
- 典型的BMP图像文件由文件头、信息头、调色板和位图数据内容4部分组成。

- 文件头包含图像类型、图像大小和图像数据存放起始位置等信息，有14字节，即文件的第0~13字节。其中，第2-5字节表示文件的大小（以字节为单位）；
- 信息头包含图像的宽、高、压缩方法以及定义颜色等信息，有40字节，即文件的第14~53字节。
- 其中第14~17字节表示信息头长度，第18~21字节表示图像的宽度（以像素为单位），第22~25字节表示图像的高度(以像素为单位)，第34~37字节表示图像的大小（以字节为单位）；

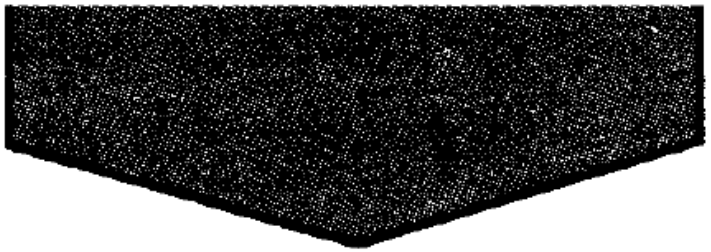
- 调色板用于说明图像的颜色；
- 位图数据内容记录了图像的每一个像素值，记录顺序是，在扫描行内是从左到右，扫描行之间是从下到上。
- 对图像文件进行压缩裁剪，首先要打开文件，然后读取文件大小信息，对其进行修改后写入新的文件中，不修改的信息原样写入新的文件，最后关闭文件。

```
from struct import *
fi=open('image.bmp','rb')
fo=open('image_new.bmp','wb+')
fi.read(14)                  #读文件头
headl=fi.read(4)             #读信息头长度
headl=unpack('i',headl)[0]
fi.read(4)                   #读图像宽度
h=fi.read(4)                 #读图像高度
h=unpack('i',h)[0]//2        #新的图像高度
fi.read(8)                   #跳过8字节
imsize=fi.read(4)            #读图像大小
imsize=unpack('i',imsize)[0]//2   #新的图像大小
#开始写入新的BMP文件
fi.seek(0,0)
fo.write(fi.read(2))         #原样写入第0~1字节
newfilesize=14+headl+imsize
fo.write(pack('i',newfilesize))  #写入文件大小（占4字节）
fi.read(4)                   #跳过4字节
fo.write(fi.read(8))         #原样写入文件头中余下的8字节
fo.write(fi.read(4))         #原样写入信息头长度
fo.write(fi.read(4))         #原样写入图像宽度
fo.write(pack('i',h))        #写入新的图像高度
fi.read(4)                   #跳过4字节
fo.write(fi.read(8))         #原样写入第26~33字节，即8字节
fo.write(pack('i',imsize))   #写入新的图像大小
fi.read(4)                   #跳过4字节
fo.write(fi.read(headl-24))  #原样写入信息头的余下的字节内容
fo.write(fi.read(imsize))    #原样写入位图数据
fi.close()
fo.close()
```

- 假设原始BMP图像为图（a），则运行程序后得到如图（b）所示的新的BMP图像。
- 由于图像数据的存储是从左到右、从下到上，所以图像裁剪后余下的是下半部分图像。



(a) 原始BMP图像



(b) 裁剪后的BMP图像

BMP 图像裁剪

自测题

- 一、选择题
- 1．在读写文件之前，用于创建文件对象的函数是（　　）。A
- A．open　　B．create
- C．file　　D．folder

- 2．关于语句f=open('demo.txt','r')，下列说法不正确的是（）。C
- A．demo.txt文件必须已经存在
- B．只能从demo.txt文件读数据，而不能向该文件写数据。
- C．只能向demo.txt文件写数据，而不能从该文件读数据。
- D．“r”方式是默认的文件打开方式

- 3．下列程序的输出结果是（　　）。C
- f=open('c:\\out.txt','w+')
- f.write('Python')
- f.seek(0)
- c=f.read(2)
- print(c)
- f.close()
- A．Pyth　　B．Python
- C．Py　　D．th

- 4．下列程序的输出结果是（　　）。B
- f=open('f.txt','w')
- f.writelines(['Python programming.'])
- f.close()
- f=open('f.txt','rb')
- f.seek(10,1)
- print(f.tell())
- A．1　　B．10
- C．gramming　　D．Python

- 5．下列语句的作用是（　　）。D
- >>> import os
- >>> os.mkdir("d:\\ppp")
- A．在D盘当前文件夹下建立ppp文本文件
- B．在D盘根文件夹下建立ppp文本文件
- C．在D盘当前文件夹下建立ppp文件夹
- D．在D盘根文件夹下建立ppp文件夹

- 二、填空题
- 1．根据文件数据的组织形式，Python的文件可分为（　）文件和（　）文件。一个Python程序文件是一个（　）文件，一幅JPG图像文件是一个（　）文件。
- 文本或ASCII，二进制，文本或ASCII，二进制

- **2．Python提供了（　）、（　）和（　）方法用于读取文本文件的内容。**
- **read()，readline()，readlines()**
- **3．二进制文件的读取与写入可以分别使用（　）和（　）方法。**
- **read()，write()**

- **4．seek(0)将文件指针定位于（　），seek(0,1) 将文件指针定位于（　），seek(0,2) 将文件指针定位于（　）。**
- **文件头，当前位置，文件尾**
- **5．Python的（　）模块提供了许多文件管理方法。**

  os

- 三、问答题
- 1. 什么是打开文件？为何要关闭文件？
- 2. 文件的主要操作方式有哪些？
- 3. 文本文件的操作步骤是什么？
- 4. 二进制文件的操作步骤是什么？
- 5. 在Python环境下如何实现文件更名和删除？