

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

Python使用内置函数open()打开文件，创建file对象。在系统中，只有存在file对象后，用户才能对文件进行相应的操作。语法格式如下：

```
file object = open(file_name [, access_mode][, buffering]  
                                [, encoding ])
```

各个参数的含义如下：

- file_name：要打开的文件名，必选参数项。
- access_mode：决定了打开文件的模式：只读，写入，追加等，可选参数项。默认访问是只读("r")。且默认以文本方式("t")访问，也可以选择以二进制方式("b")访问。所以默认模式就是"rt"
- buffering：设置文件缓冲区，可选参数项。默认缓冲区大小是4096字节。
- encoding：读写文本时采用的字符编码器或解码器。The default encoding is platform dependent (whatever locale.getpreferredencoding() returns), but any text encoding supported by Python can be used. See the *codecs* module for the list of supported encodings.

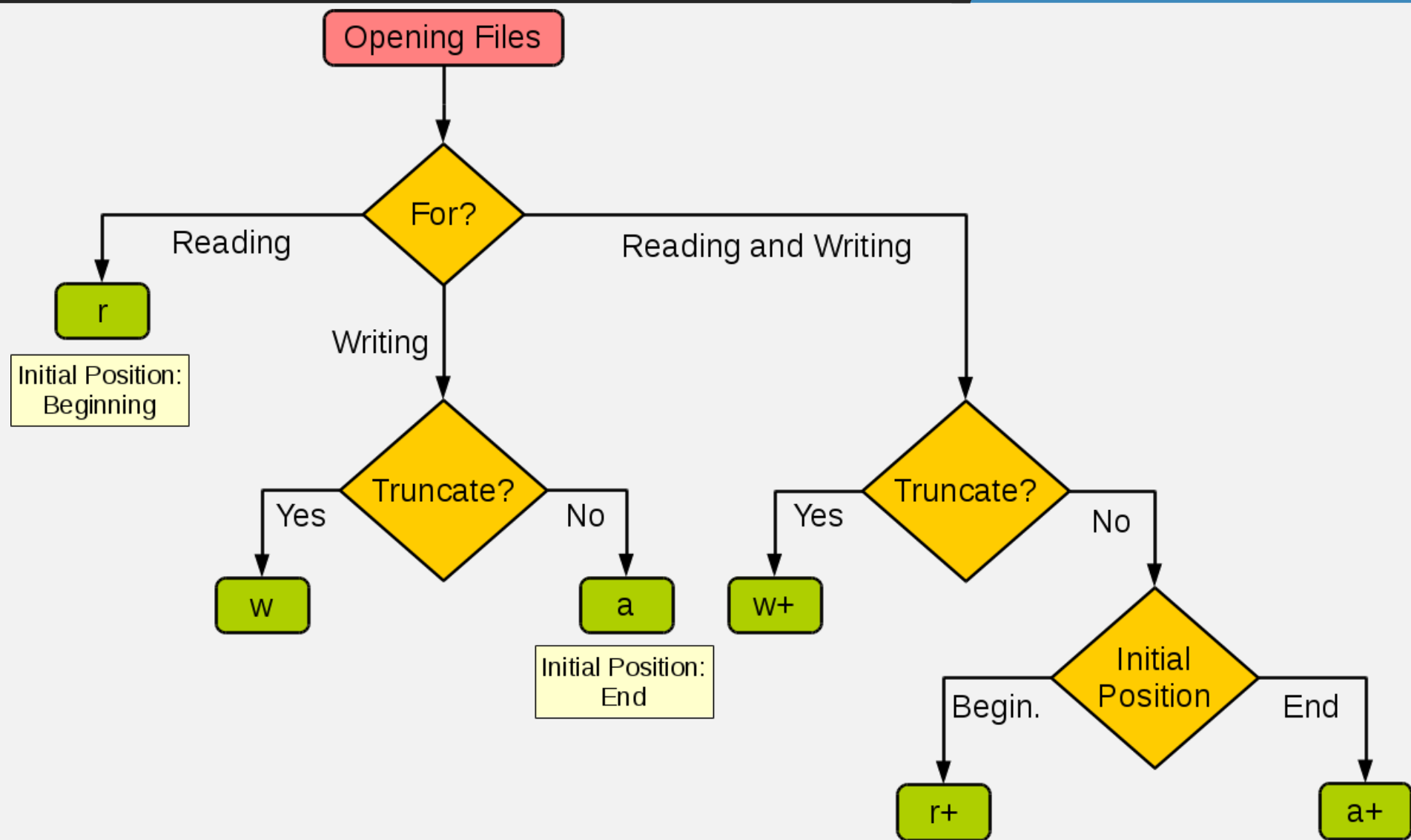
10.1.1 文件模式

访问文件的模式有：读、写、追加等。以不同模式打开文件，详细功能见下表所示。例如，以写模式打开并创建一个文件，如下所示：

```
>>> str_file = open("c:\\test\\file_test.txt","w")
```

模式	描述
t	文本模式 (默认)。
x	写模式，新建一个文件，如果该文件已存在则会报错。
b	二进制模式。
+	打开一个文件进行更新(可读可写)。
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。一般用于非文本文件如图片等。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。一般用于非文本文件如图片等。

模式	描述
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
w+	打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。



10.1.2 文件缓冲区

Python文件缓冲区，一般分为三种模式：全缓冲、无缓冲、行缓冲。

- **全缓冲**：默认情况下，Python文件写入采用全缓冲模式，空间大小为4096字节。前4096个字节的信息都会写在缓冲区中，当第4097个字节写入的时候（即**缓冲区满时**），系统会把先前的4096个字节通过系统调用写入文件。同样，可以用Buffering=n(单位为：字节)自定义缓冲区的大小。
- **行缓冲**：Buffering=1，系统**每遇到一个换行符(' \n')**才进行系统调用，将缓冲区的信息写入文件。
- **无缓冲**：Buffering=0，当需要将系统产生的信息**实时写入**文件时，就须要设置为无缓冲的模式。

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

10.2.1 读和写

1. read()方法

语法格式如下：

```
string = fileobject.read([size])
```

```
bytes = fileobject.read([size])
```

- size——从文件中读取的字节数（二进制方式）或字符数（文本方式），如果未指定则读取文件的全部信息。
- 返回值为从文件中读取的字节或字符串。

例如：从文本文件读取若干字符—以文本方式打开并读取

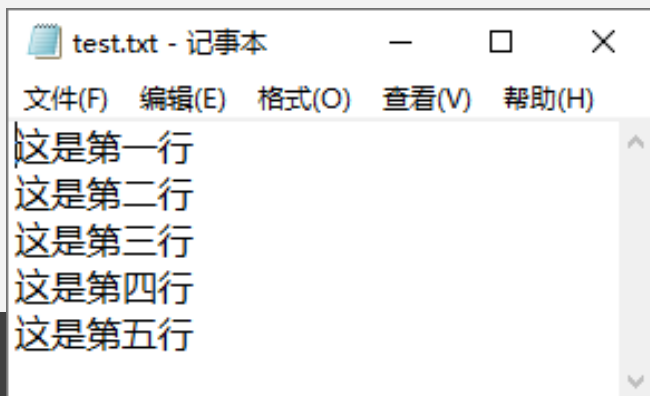
打开文件

```
fo = open("test.txt", "r+", encoding="utf-8") # 以文本方式、可读可写模式打开文件
print("文件名为: ", fo.name)
```

```
text = fo.read(10) # 读取10个字符(包括换行符)
print(f"读取的字符串:\n{text}")
```

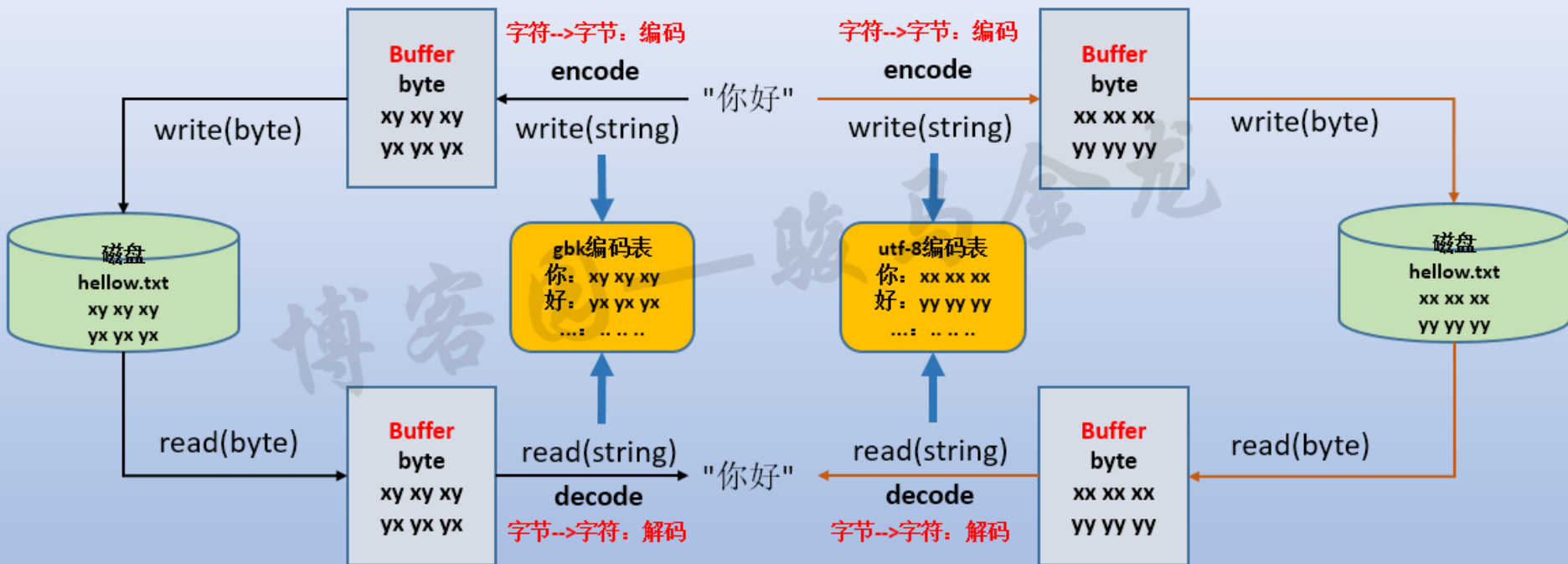
关闭文件

```
fo.close()
```



文件名为: test.txt
读取的字符串:
这是第一行
这是第二

关于字符编码与解码



例如：从文本文件读取所有字符---以字节方式打开并读取

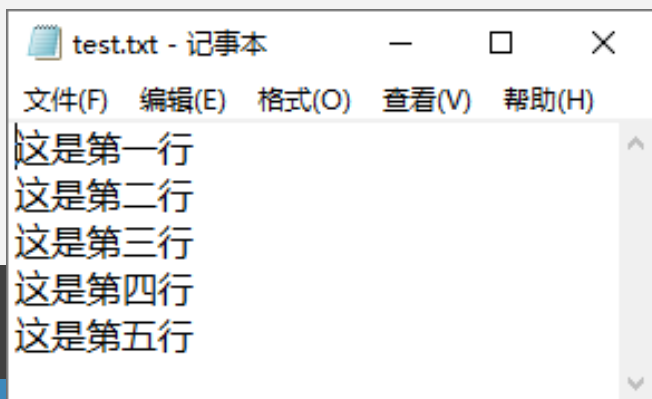
打开文件

```
fo = open("test.txt", "br+") # 以二进制方式、可读可写模式打开文件
print("文件名为: ", fo.name)
```

```
some_byte = fo.read() # 读取所有字节，返回字节序列对象，bytes类型
text = some_byte.decode(encoding='utf-8') # 对字节序列解码，返回字符串
print(f"读取的字符串:\n{text}")
```

关闭文件

```
fo.close()
```



文件名为: test.txt

读取的字符串:

这是第一行

这是第二行

这是第三行

这是第四行

这是第五行

例如：从文本文件读取所有字符---以字节方式打开并读取，改变解码方式

打开文件

```
fo = open("test.txt", "br+") # 以二进制方式、可读可写模式打开文件
print("文件名为:", fo.name)
```

```
some_byte = fo.read() # 读取所有字节，返回字节序列对象，bytes类型
text = some_byte.decode(encoding='ASCII') # 对字节序列解码（错误则产生异常），返回字符串
```

```
print(f"读取的字符串:\n{text}")
```

关闭文件

```
fo.close()
```

UnicodeDecodeError: 'ascii' codec can't decode byte 0xe8 in position 0: ordinal not in range(128).

原因：

`bytes.decode(encoding='utf-8', errors='strict')`
error参数默认为'strict'，则解码错误将产生异常。该参数也可以设置为'ignore', 'replace' 等值，表示忽略、符号替代等含义。

例如：从文本文件读取所有字符---以字节方式打开并读取，改变解码方式

打开文件

```
fo = open("test.txt", "br+") # 以二进制方式、可读可写模式打开文件
print("文件名为: ", fo.name)
```

```
some_byte = fo.read() # 读取所有字节，返回字节序列对象，bytes类型
text = some_byte.decode(encoding='ASCII', errors='replace') # 对字节
序列解码（错误则替换），返回字符串
```

```
print(f"读取的字符串:\n{text}")
```

关闭文件

```
fo.close()
```

文件名为: test.txt

读取的字符串:

?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

10.2.1 读和写

2. write()方法

write()方法将字符串或字节序列写入一个打开的文件。语法格式如下：

fileobject.write(string)

fileobject.write(bytes)

如果以二进制方式写入字符串，则需要先将字符串按指定规则编码为字节序列，然后写入。

文本方式访问时，write()方法不会自动在字符串的末尾添加换行符('\n')，需要人为在字符串末尾添加换行符。

例如：分别以文本方式和二进制方式，向文本文件写入字符串

打开文件

```
fo = open("out.txt", "w", encoding='utf-8')
```

```
fo.write('人生苦短，\n') # 注意手工换行  
fo.write('I love python!')
```

关闭文件

```
fo.close()
```



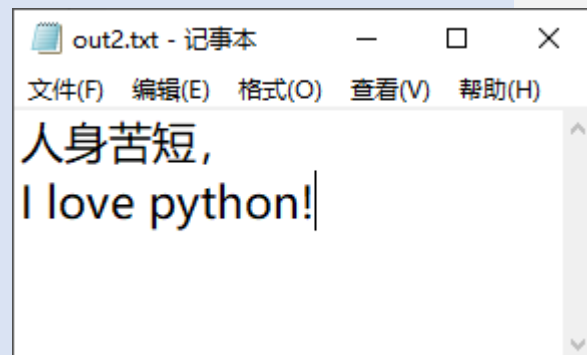
打开文件

```
fo = open("out2.txt", "bw")
```

```
fo.write('人生苦短，\n'.encode(encoding='utf-8'))  
fo.write('I love python!'.encode(encoding='utf-8'))
```

关闭文件

```
fo.close()
```



例如：以二进制方式向数据文件写入整数序列

```
fo = open("out3.dat", "bw") # 打开文件
```

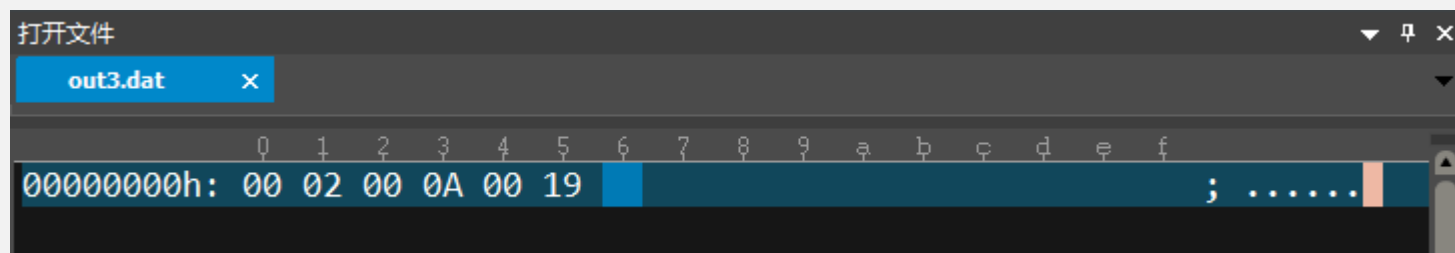
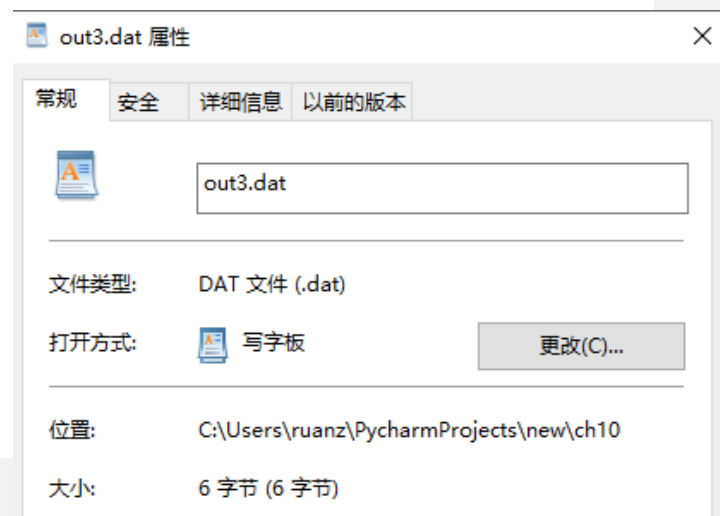
```
a = [2, 10, 25]
```

```
for x in a:
```

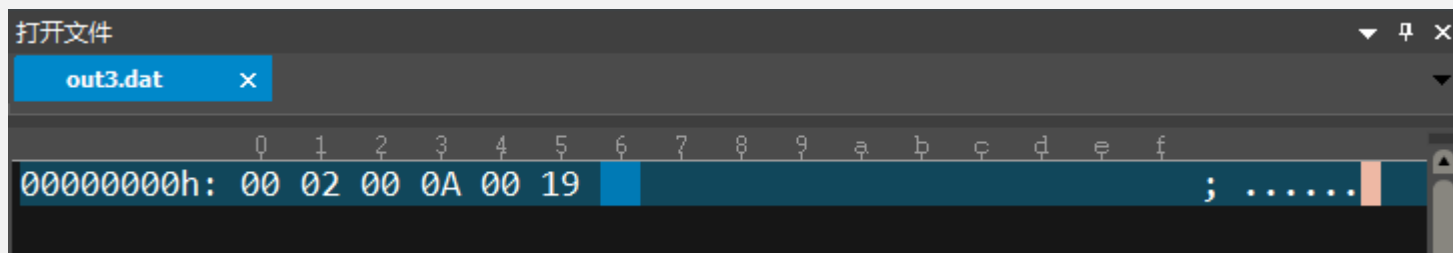
```
    # 两个字节表示的整型, 且为大端优先
```

```
    fo.write(int(x).to_bytes(2, byteorder='big'))
```

```
fo.close() # 关闭文件
```



例如：以二进制方式从数据文件读取整数序列



```
fo = open("out3.dat", "br") # 打开文件
```

```
buf = fo.read() # 读取所有字节
```

```
print(len(buf), '字节')
```

```
a = []
```

```
for i in range(0, int(len(buf) / 2)):
```

```
    x = int.from_bytes(buf[i * 2:i * 2 + 2], byteorder='big')
```

```
    a.append(x)
```

```
print(a)
```

```
fo.close() # 关闭文件
```

6 字节

[2, 10, 25]

10.2.2 读取行

1.readline()方法

用于从文本文件中读取整行，包括“\n”字符。语法如下：

string = fileObject.readline([size])

- size -- 从文件中读取的字符数，如果参数为正整数，则返回指定长度的字符串数据。
- 注意，读到文件末尾返回空字符串，而空行则返回换行符'\n'

2. readlines()方法

用于读取文本文件中所有行，直到结束符 EOF，并返回列表，包括所有行的信息。该列表可以由Python 的“for... in ...”结构进行处理。语法如下：

list=fileObject.readlines()

例如：按行读取文本文件的部分行

打开文件

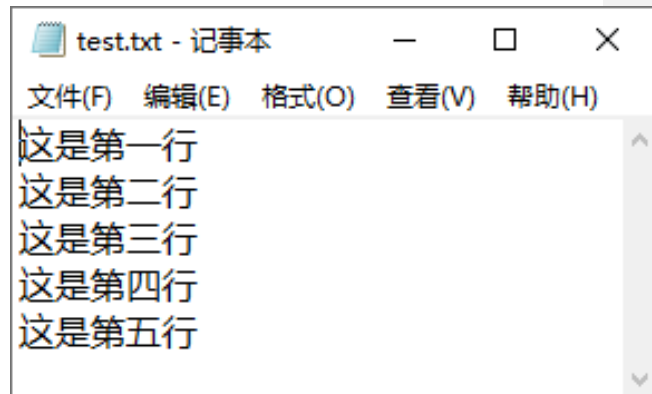
```
fo = open("test.txt", "r+", encoding='utf-8')  
print("文件名为: ", fo.name)
```

```
line = fo.readline()  
print("读取第一行 %s" % (line))
```

```
line = fo.readline(4)  
print("读取第二行前4个字符为: %s" % (line))
```

关闭文件

```
fo.close()
```



文件名为: test.txt

读取第一行 这是第一行

读取第二行前4个字符为: 这是第二

例如：按行读取文本文件的所有行---循环读取行

打开文件

```
fo = open("test.txt", "r+", encoding='utf-8')
```

```
i = 0
```

```
line = fo.readline()
```

```
while line != "": # 每一行都包括了一个换行符'\n', 除了文件末尾
```

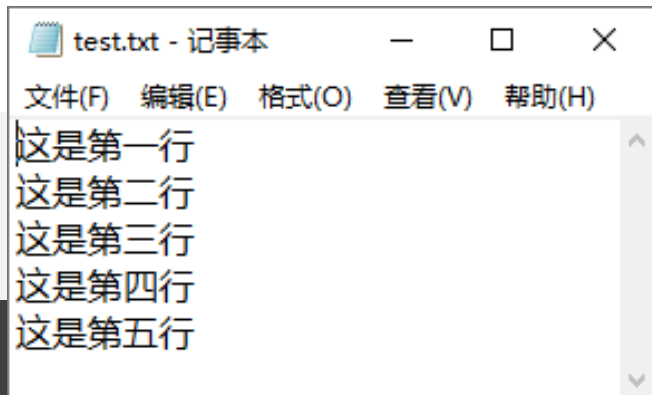
```
    i += 1
```

```
    print("读取第%d行: %s" % (i, line))
```

```
    line = fo.readline() # 读取下一行
```

关闭文件

```
fo.close()
```



读取第1行: 这是第一行

读取第2行: 这是第二行

读取第3行: 这是第三行

读取第4行: 这是第四行

读取第5行: 这是第五行

例如：按行读取文本文件的所有行---用readlines()方法

打开文件

```
fo = open("test.txt", "r+", encoding='utf-8')
```

```
i = 0
```

```
lines = fo.readlines() # 读取所有行
```

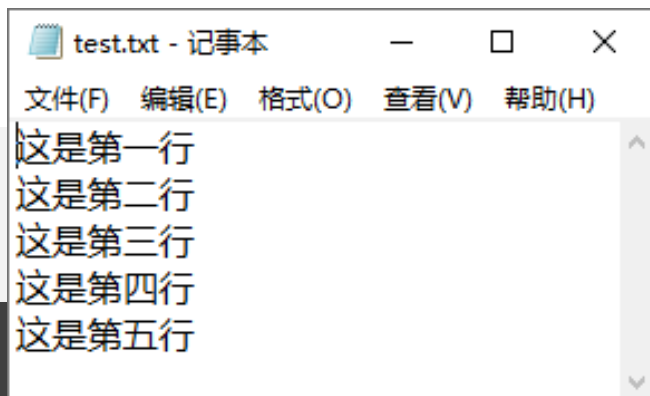
```
for line in lines:
```

```
    i += 1
```

```
    print("读取第%d行: %s" % (i, line))
```

关闭文件

```
fo.close()
```



读取第1行: 这是第一行

读取第2行: 这是第二行

读取第3行: 这是第三行

读取第4行: 这是第四行

读取第5行: 这是第五行

10.2.3 写入一行或多行

writelines()方法:

向文本文件写入文本行。语法格式如下:

fileObject. writelines(lines)

lines为字符串列表。

注意, 该方法不会自动写入换行符, 因此如果lines中的每一个字符串需要换行, 则需要在字符串的结尾提供换行符, 例如:

```
lines = ['众志成城, 共克时艰\n', '亿万人民同舟共济、守望相助\n',  
        '筑起一道抗击疫情的钢铁长城, 铸就一座令人瞩目的精神丰碑']
```

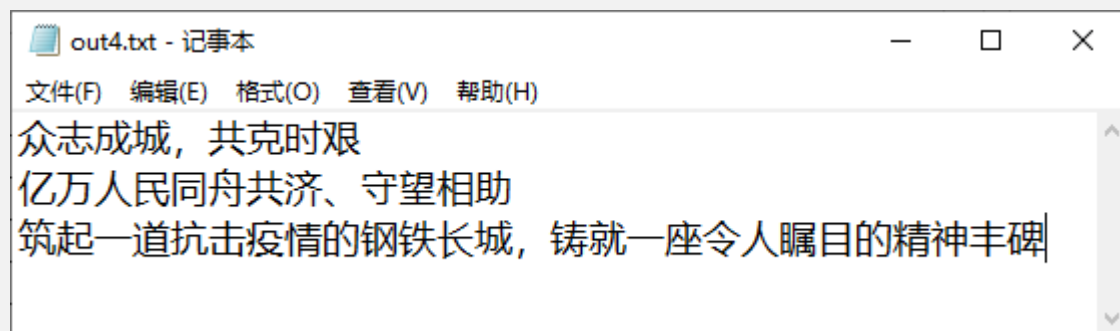
例如：使用writelines()方法向文本文件写入多行字符串

```
fo = open("out4.txt", "w", encoding='utf-8') # 打开文件
```

```
lines = ['众志成城，共克时艰\n', '亿万人民同舟共济、守望相助\n',  
        '筑起一道抗击疫情的钢铁长城，铸就一座令人瞩目的精神丰碑']
```

```
fo.writelines(lines)
```

```
fo.close() # 关闭文件
```



10.2.4 关闭文件

close()方法:

用于关闭该文件，并清除文件缓冲区里的信息，关闭文件后不能再进行写入。语法格式如下：

fileObject.close()

当一个文件对象的引用被重新指定给另一个文件时，系统会关闭先前打开的文件。

10.2.5 文件重命名

rename()方法：

用于将当前文件名称重新命名为一个新文件名称。

语法格式如下：

os.rename(current_filename, new_filename)

current_filename：当前文件的名称；new_filename：重新命名后的文件名称。

注意：要使用这个内置函数rename()，你必须先导入**os模块**，然后才可以调用相关的功能。

该方法也用于更改名录名称。

10.2.6 删除文件

remove()方法：

用于删除系统中已经存在的文件。

语法格式如下：

os.remove(file_name)

file_name——系统中已经存在的文件名称，即将删除的文件名称。

注意：要使用这个内置函数remove()，你必须先导入os模块，然后才可以调用相关的功能。

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

10.3.1 创建目录

(1) 使用os.mkdir()方法创建目录

语法格式如下：

os.mkdir("newdir")

newdir——新建的目录名称，必须要带目录的完整路径。

注意：要使用目录操作相关的内置函数，必须先导入os模块，然后才可以调用相关的功能。

例如：

```
>>> import os
```

```
>>> os.mkdir("C:\\test\\test_dir")
```

10.3.2 显示当前工作目录

(2) 使用getcwd()方法显示当前工作目录

语法格式如下：

os.getcwd()

显示当前的工作目录。例如：

```
>>> os.getcwd()
```

```
'C:\\Users\\ruanz\\PycharmProjects\\classTest'
```

10.3.3 改变目录

(3) 使用chdir()方法改变当前工作目录

语法格式如下：

os.chdir("newdir")

newdir——要改变的新的工作目录名称，需要带目录的完整路径。

os.chdir()方法应用如下所示：

```
>>> os.getcwd()
```

```
'C:\\Users\\ruanz\\PycharmProjects\\classTest'
```

```
>>> os.chdir('C:\\Users\\ruanz\\PycharmProjects\\new')
```

```
>>> os.getcwd()
```

```
'C:\\Users\\ruanz\\PycharmProjects\\new'
```

10.3.4 删除目录

(4) 使用`rmdir()`方法删除目录

语法格式如下：

`os.rmdir(path)`

`path`——要删除的目录名称，需要带目录的完整路径。

删除`path`指定的空目录，如果目录非空，则抛出一个`OSError`异常。例如：

```
>>> os.rmdir("C:\\test\\test_dir")
```

`os.removedirs(path)`方法则用于递归删除目录。
也要求个目录必须为空，否则抛出`OSError`异常。

10.3.5 列出目录中的所有文件和子目录名称

(5) 使用listdir() 方法

语法格式如下：

os.listdir(path)

path——需要列出的目录路径。

返回指定路径下的文件和文件夹列表。例如：

```
>>> os.listdir("C:\\test")
```

```
['a.txt', 'a_1.txt', 'b.txt', ..... , 'workspace', ..... , '随钻曲线.jpg']
```


示例：递归删除当前目录下的某个子目录

```
import os

# 列出目录和文件
print("目录为: %s" % os.listdir(os.getcwd()))
# 递归移除目录
os.removedirs("/test")
# 列出移除后的目录
print("移除后目录为:" % os.listdir(os.getcwd()))
```

目录为: ['demo_10_02_01.py', 'demo_10_02_02.py', 'demo_10_02_10.py', 'out.txt', 'out2.txt', 'out3.dat', 'out4.txt', 'test', 'test.txt', '__init__.py']

移除后目录为: ['demo_10_02_01.py', 'demo_10_02_02.py', 'demo_10_02_10.py', 'out.txt', 'out2.txt', 'out3.dat', 'out4.txt', 'test.txt', '__init__.py']

10.3.6 os.path模块

os.path 模块主要用于获取文件的属性以及判断某个路径是否为文件或目录

方法	说明
os.path. abspath (path)	返回绝对路径
os.path. basename (path)	返回文件名
os.path.dirname(path)	返回文件路径
os.path. exists (path)	路径存在则返回True,路径损坏返回False
os.path.getatime(path)	返回最近访问时间（浮点型秒数）
os.path.getmtime(path)	返回最近文件修改时间
os.path.getctime(path)	返回文件 path 创建时间
os.path. getsize (path)	返回文件大小，如果文件不存在就返回错误

10.3.6 os.path模块

方法	说明
os.path.isabs(path)	判断是否为绝对路径
os.path.isfile(path)	判断路径是否为文件
os.path.isdir(path)	判断路径是否为目录
os.path.join(path1[, path2[, ...]])	把目录和文件名合成一个路径
os.path.abspath(path)	返回一个目录或文件的绝对路径
os.path.realpath(filename)	返回指定文件的标准路径，而非软链接所在的路径
os.path.samefile(path1, path2)	判断目录或文件是否相同
os.path.sameopenfile(fp1, fp2)	判断fp1和fp2是否指向同一文件
os.path.split(path)	把路径分割成 dirname 和 basename，返回一个元组
os.path.splitdrive(path)	返回驱动器名和路径组成的元组
os.path.splitext(path)	分割路径中的文件名与扩展名

示例：输出指定目录下的所有文件名

```
import os
import pprint

my_dir = r"c:\test"
paths = os.listdir(my_dir) # 获取所有子目录和文件
files = [x for x in paths
         if os.path.isfile(my_dir + '\\' + x)]

# 列出所有文件
pp = pprint.PrettyPrinter()
print(f"{my_dir} 目录中文件为:")
pp.pprint(files)
```

c:\test目录中文件为:

- 'a.txt',
- 'a_1.txt',
- 'b.txt',
- 'b1.txt',
- 'Hello.txt',
- 'Hello2.txt',
-
- 'hello_FOS.txt',
- 'json_test.json',
- 'note.txt',
- 'top10.CSV',
- 'top10.html',
- 'XingLL_sinx2.gif',
- '随钻曲线.jpg']

示例：输出指定目录下的所有后缀为txt文件的文件名

```
import os
import pprint

my_dir = r"c:\test"
paths = os.listdir(my_dir) # 获取所有子目录和文件
# 所有txt文件
files = [x for x in paths
         if os.path.splitext(x)[1] == '.txt']
# 列出所有文件
pp = pprint.PrettyPrinter()
print(f"{my_dir}目录中txt文件为:")
pp.pprint(files)
```

c:\test目录中txt文件为:

- 'a.txt',
- 'a_1.txt',
- 'b.txt',
- 'b1.txt',
- 'bookinf.txt',
- 'Hello.txt',
- 'Hello2.txt',
- 'hello_FOS.txt',
- 'note.txt',
- 'top10.txt']

示例：输出指定目录下的所有大于1字节txt文件的文件名

```
import os
import pprint

my_dir = r"c:\test"
paths = os.listdir(my_dir) # 获取所有子目录和文件
files = [x for x in paths
         if os.path.splitext(x)[1] == '.txt' # 所有txt文件
         and os.path.getsize(my_dir + "\\ " + x) > 1024] # 文件>1k字节

# 列出所有文件
pp = pprint.PrettyPrinter()
print(f"{my_dir}目录中所有大于1k字节txt文件为:")
pp.pprint(files)
```

c:\test目录中所有大于1k字节txt文件为:
['bookinf.txt']

示例：输出本程序的绝对路径和所在目录

```
import os
print('__file__:', __file__) # __file__ 是用来获得模块（本程序）所在路径
abs_path = os.path.abspath(__file__)
print('abspath:', abs_path)
real_path = os.path.realpath(__file__)
print('realpath:', real_path)
print('split:', os.path.split(real_path)[0]) # 获取本程序所在目录
```

```
__file__: D:/PycharmProjects/new/ch10/demo_10_02_14.py
abspath: D:\PycharmProjects\new\ch10\demo_10_02_14.py
realpath: D:\PycharmProjects\new\ch10\demo_10_02_14.py
split: D:\PycharmProjects\new\ch10
```

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

10.4.1 什么是 CSV 文件?

CSV 全称是 “Comma-Separated Values” ，中文叫做【逗号分割的值】。

- CSV文件以**纯文本**的形式存储表格数据，后缀为csv
- 每行中的各数据项用逗号分隔。通常，第一行标识每个数据列的名称，之后的每一行都是实际数据，仅受文件大小限制。
- 比Excel文件更加简洁，XLS文本是电子表格，包含文本、数值和公式格式等内容，CSV则不包含这些。
- CSV 文件也可以通过 Excel 打开，数据以电子表格的样式进行显示，因此 CSV 文件也被视为一种简化版的电子表格。

学生体检表.csv	
1	Name, Height, Weight
2	Mike, 180, 76
3	Bob, 160, 65
4	Andy, 170, 72

学生体检表.csv - Excel					
文件 开始 插入 绘图 页面布局 公式 数据 审阅 视图					
C9					
	A	B	C	D	E
1	Name	Height	Weight		
2	Mike	180	76		
3	Bob	160	65		
4	Andy	170	72		
5					

10.4.2 CSV 文件读写的常用方法


- 方法1：使用文件对象直接读写
 - 内置函数open打开文件文件后，使用其readline/readlines和writelines方法直接读写
 - 分隔符和换行符都需要处理，数据行需要程序分割成数据项，或数据项合并成数据行
- 方法2：使用文件对象 + csv 模块完成读写
 - 程序中数据行由嵌套列表组成：使用csv模块的reader和writer对象完成
 - 程序中数据行由字典列表组成：使用csv模块的DictReader和DictWriter对象完成
- 方法3：使用Pandas库中的 read_csv 方法和 to_csv 方法完成读写
 - 讲解Pandas库时再介绍

(1) 方法1：使用文件对象直接读写

示例：将嵌套列表数据写入csv文件

```
header = ['Name', 'Height', 'Weight']  
rows = [['Mike', '180', '76'],  
        ['Bob', '160', '65'],  
        ['Andy', '170', '72']]
```

```
with open('体检1.csv', 'wt', encoding='utf-8-sig') as fp:  
    fp.write(','.join(header) + '\n') # 写入标题行（表头）  
    # 循环写入数据行  
    for row in rows:  
        fp.write(','.join(row) + '\n') # 写入一行数据
```



The screenshot shows a code editor with two tabs: 'demo_10_04_01.py' and '体检1.csv'. The Python script in the first tab defines a header and rows list. The second tab shows the resulting CSV file content, which matches the script's output. The CSV file has a header row and three data rows, with the last row highlighted in yellow.

1	Name, Height, Weight
2	Mike, 180, 76
3	Bob, 160, 65
4	Andy, 170, 72
5	

注意数据项用逗号连接合并，并在末尾的换行符

(1) 方法1：使用文件对象直接读写

示例：从csv文件读取数据到嵌套列表

```
rows = []  
with open('体检1.csv', 'tr', encoding='utf-8-sig') as fp:  
    line = fp.readline() # 读取标题行（表头）  
    line = line[0:-1] # 去掉末尾的换行符  
    header = line.split(',') # 分割字符串  
    for line in fp.readlines(): # 读取剩余的所有行（即数据行）  
        line = line[0:-1] # 去掉末尾的换行符  
        rows.append(line.split(',')) # 追加到嵌套列表  
print(header)  
print(rows)
```

```
['Name', 'Height', 'Weight']
```

```
[['Mike', '180', '76'], ['Bob', '160', '65'], ['Andy', '170', '72']]
```

demo_10_04_01.py	体检1.csv
1	Name, Height, Weight
2	Mike, 180, 76
3	Bob, 160, 65
4	Andy, 170, 72
5	

注意

- 解码与编码一致
- 去除末尾换行符
- 分割字符串

(2) 方法2：使用文件对象 + csv 模块完成读写

csv 模块是 Python 标准库中的内置模块，作用是用于处理 csv 文件。

- 程序中数据行由**嵌套列表**组成：使用csv模块的reader和writer对象完成
 - 首先，构造reader或writer必须传入一个已经用内置函数open打开的csv文件对象
 - 然后，调用writerow方法写入一行或writerows方法写入多行，或遍历reader对象获取数据行（表头行用next(reader)单独获取）
- 数据行由**字典列表**组成：使用csv模块的DictReader和DictWriter对象完成
 - 首先，先构造DictReader或DictWriter必须传入一个已经用内置函数open打开的csv文件对象
 - 然后，调用writeheader方法写入表头，或filenames属性获取表头
 - 最后，调用writerow/writerows方法写入一行/多行，或遍历变量DictReader对象获取数据行（表头行用next(DictReader)单独获取）

(2) 方法2: 使用文件对象 + csv 模块完成读写

示例: 用csv.writer对象将嵌套列表数据写入csv文件

```
import csv
header = ['Name', 'Height', 'Weight']
rows = [['Mike', '180', '76'],
        ['Bob', '160', '65'],
        ['Andy', '170', '72']]
with open('体检2.csv', 'tw', encoding='utf-8-sig', newline='') as fp:
    csv_writer = csv.writer(fp) # 构造csv的writer对象
    csv_writer.writerow(header) # 写入一行数据, 此处是标题行 (表头)
    csv_writer.writerows(rows) # 写入多行数据
```

1	Name, Height, Weight
2	
3	Mike, 180, 76
4	
5	Bob, 160, 65
6	
7	Andy, 170, 72
8	
9	

1	Name, Height, Weight
2	Mike, 180, 76
3	Bob, 160, 65
4	Andy, 170, 72
5	

注意为 open() 函数指定关键字参数 **newline=""**, 否则没写入一行, 后面会添加一个空白行

(2) 方法2: 使用文件对象 + csv 模块完成读写

示例: 用csv.reader对象从csv文件读取数据到
嵌套列表

```
import csv
rows = []
with open('体检2.csv', 'tr', encoding='utf-8-sig') as fp:
    csv_reader = csv.reader(fp) # 构造csv的reader对象
    header = next(csv_reader) # 得到表头行
    for row in csv_reader: # 遍历剩余的数据行
        rows.append(row)
print(header)
print(rows)
```

demo_10_04_03.py ×	体检2.csv ×
1	Name, Height, Weight
2	Mike, 180, 76
3	Bob, 160, 65
4	Andy, 170, 72
5	

注意csv.reader是一个生成器，
可以用next()函数得到下一个
元素，也可以迭代。

```
['Name', 'Height', 'Weight']
```

```
[['Mike', '180', '76'], ['Bob', '160', '65'], ['Andy', '170', '72']]
```

(2) 方法2: 使用文件对象 + csv 模块完成读写

示例: 用csv.DictWriter对象将字典列表数据写入csv文件

```
import csv
header = ['Name', 'Height', 'Weight']
rows = [{ 'Name': 'Mike', 'Height': '180', 'Weight': '76'},
        { 'Name': 'Bob', 'Height': '160', 'Weight': '65'},
        { 'Name': 'Andy', 'Height': '170', 'Weight': '72'}]
with open('体检3.csv', 'tw', encoding='utf-8-sig', newline='') as fp:
    dict_writer = csv.DictWriter(fp, fieldnames=header) # 构造csv的
    DictWriter对象, 注意传入表头
    dict_writer.writeheader() # 写入标题行 (表头)
    dict_writer.writerows(rows) # 写入多行数据
```

	demo_10_04_05.py ×	体检3.csv ×
1		Name, Height, Weight
2		Mike, 180, 76
3		Bob, 160, 65
4		Andy, 170, 72
5		

注意为 open() 函数指定关键字参数 **newline=""**, 否则没写入一行, 后面会添加一个空白行

(2) 方法2: 使用文件对象 + csv 模块完成读写

示例: 用csv.DictReader对象从csv文件读取数据

得到字典列表

注意csv.DictReader是一个生成器, 可以用next()函数得到下一个元素, 也可以迭代。

```
import csv
rows = []
with open('体检3.csv', 'tr', encoding='utf-8-sig') as fp:
    dict_reader = csv.DictReader(fp) # 构造csv的DictReader对象, 若数据有表头行, 它们将作为字典的键
    # header = ['Name', 'Height', 'Weight']
    # dict_reader = csv.DictReader(fp, fieldnames=header) # 若数据无表头行, 则指定表头作为字典的键
    firstRow = next(dict_reader) # 得到第一行数据对应的字典
    rows.append(firstRow)
    for row in dict_reader: # 读取剩余的各行, 并以字典形式加入列表
        rows.append(row)
print(rows)
```

```
[{'Name': 'Mike', 'Height': '180', 'Weight': '76'},
{'Name': 'Bob', 'Height': '160', 'Weight': '65'},
{'Name': 'Andy', 'Height': '170', 'Weight': '72'}]
```

	demo_10_04_05.py	体检3.csv
1		Name, Height, Weight
2		Mike, 180, 76
		Bob, 160, 65
		Andy, 170, 72

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

10.5.1 pickle 模块

对象序列化 (serializing) 是指将对象转换为文本或字节序列；如果对象还以文件的形式存放在磁盘上，便于以后检索，则称对象具有了**持久性**；**反序列化** (de-serializing) 则是相反的过程。

pickle 模块实现了用于序列化和反序列化Python对象结构的二进制协议。

- "Pickling"是将Python对象转换为字节流的过程，"unpickling"是反向操作，从而将字节流（二进制文件或类似字节的对象）转换回对象。
- Python中几乎所有的数据类型（列表，字典，集合，类等）都可以用pickle来序列化与反序列化。
- pickle 数据格式是特定于Python的，不便于在非python程序间数据共享（与文本文件、CSV文件、Excell文件、数据库等不同）。
- pickle 数据格式使用相对紧凑的二进制表示，如果需要最佳尺寸特征，则可以有效地压缩数据。
- pickle序列化后的数据，可读性差，人一般无法识别。

pickling (和 unpickling) 也被称为“序列化”，“编组”或“平面化”。为了避免混乱，此处采用术语“封存 (pickling)”和“解封 (unpickling)”。

pickle协议和JSON (JavaScript Object Notation) 的区别

- JSON是一种文本序列化格式（它输出unicode文本，虽然大部分时间它被编码utf-8），而pickle是二进制序列化格式。
- JSON是人类可读的，而pickle则不是。
- JSON是可互操作的，并且在Python生态系统之外广泛使用；而pickle数据格式是特定于Python的，它的优点是没有外部标准强加的限制，例如JSON；但是这意味着非Python程序可能无法重建pickled Python对象。
- 默认情况下，JSON只能表示Python内置类型的子集，而不能表示自定义类；pickle可以表示极其庞大的Python类型（其中许多是自动的，通过巧妙地使用Python的内省工具；复杂的案例可以通过实现特定的对象API来解决）。
- 与pickle不同，反序列化不受信任的JSON本身不会造成任意代码执行漏洞。

10.5.2 pickle 模块接口

要序列化对象，只需调用该模块的dumps()方法即可。同样，要对数据流进行反序列化，请调用该模块的loads () 方法。但是，如果想要更多地控制序列化和反序列化，则可以分别创建一个**Pickler**或一个**Unpickler**对象。

- `pickle.dump(obj, file, protocol=None, *, fix_imports=True, ...)`
 - 序列化对象obj，并将结果数据流写入到文件对象file中。参数protocol是序列化模式，python3中默认值为4，0表示以文本的形式序列化，为原始版本，还可以是1、2、3、4，表示以二进制的形式序列化，不同Python版本有所区别。如果参数fix_imports为true，则pickle将尝试将旧的Python 2名称映射到Python 3中使用的新名称。
 - 相当于**`Pickler(file, protocol).dump(obj)`**。
- `obj=pickle.load(file, *, fix_imports=True, encoding="ASCII", ...)`
 - 从打开的文件对象file中读取pickle对象表示，并返回其中指定的重构对象obj。
 - 相当于**`obj=Unpickler(file).load()`**。

10.5.2 pickle 模块接口

要序列化对象，调用该模块的dumps()或dump()方法即可；要对数据流进行反序列化，请调用该模块的loads()或load()方法。但是，如果想要更多地控制序列化和反序列化，则可以分别创建一个**Pickler**或一个**Unpickler**对象。

- `data=pickle.dumps(obj, protocol=None, *, fix_imports=True, ...)`
 - 序列化对象obj，并将返回对象的pickle对象表示。
 - 相当于`data=Pickler(protocol).dumps(obj)`。
- `obj=pickle.loads(data, *, fix_imports=True, encoding="ASCII", ...)`
 - 从data中读取pickle对象表示，并返回其中指定的重构对象。
 - 相当于`obj=Unpickler(data).loads()`。

10.5.2 pickle 模块接口

例如：将一个字典序列化与反序列化并保存到文件

```
import pickle
```

```
path = 'pickle_1.dat'
```

```
f = open(path, 'wb')
```

```
data = {'a': 123, 'b': 'python', 'c': [[1, 2], [3, 4]]}
```

```
pickle.dump(data, f, protocol=?) # 序列化对象到磁盘文件
```

```
f.close()
```

```
f1 = open(path, 'rb')
```

```
data1 = pickle.load(f1) # 反序列化
```

```
print(data1)
```

```
f1.close()
```

protocol=0

protocol=4

pickle_1.dat -
文件(F) 编辑(E)
(dp0
Va
p1
l123
sVb
p2
Vpython
p3
sVc
p4
(lp5
(lp6
l1
a12
aa(lp7
l3
a14
aas.

pickle_1.dat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
€? }??a摩{?b敲python敲c撸]?K K e]?K K eeu.

{'a': 123, 'b': 'python', 'c': [[1, 2], [3, 4]]}

例如：将一个Person对象序列化与反序列化并保存到文件

```
import pickle
class Person: # 定义Person类
    def __init__(self, n, a):
        self.name = n
        self.age = a
    def show(self):
        print(self.name, self.age)
```

```
p = Person("Fred", 2)
p.show()
f = open('p.dat', 'wb')
pickle.dump(p, f, protocol=?)
f.close()
```

```
f = open('p.dat', 'rb')
q = pickle.load(f)
f.close()
q.show()
```

Fred 2
Fred 2

protocol=0

protocol=4

p.dat - 记事本

文件(F) 编辑(E) 格式(O)

ccopy_reg
_reconstructor
p0
(c__main__
Person
p1
c_builtin_
object
p2
Ntp3
Rp4
(dp5
Vname
p6
VFred
p7
sVage
p8
l26
sb|

p.dat - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

€□? ?_main_敲□Person战?伙}??name敲□Fred敲□age摩□ub.

10.5.2 pickle 模块接口

例如：将一个字典序列化与反序列化

```
import pickle
```

```
data = {'a': 123, 'b': 'python', 'c': [[1, 2], [3, 4]]}
pickle_data = pickle.dumps(data, protocol=?) # 序列化对象
print(type(pickle_data))
print(pickle_data)
```

```
data1 = pickle.loads(pickle_data) # 反序列化
print(data1)
```

```
{'a': 123, 'b': 'python',
'c': [[1, 2], [3, 4]]}
```

<class 'bytes'>

```
b'(dp0\nVa\np1\nl123\nsVb\np2\nVpython\np3\nsVc\np4\n(lp5\n(lp6\nl1\nal2\naa(lp7\nl3\nal4\naas.'
```

protocol=0

protocol=4

<class 'bytes'>

```
b'\x80\x04\x950\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\x01a\x94K{\x8c\x01b\x94\x8c\x06python\x94\x8c\x01c\x94]\x94(\x94(K\x01K\x02e]\x94(K\x03K\x04eeu.'
```

例如：将一个Person对象序列化与反序列化

```
import pickle
class Person: # 定义Person类
    def __init__(self, n, a):
        self.name = n
        self.age = a
    def show(self):
        print(self.name, self.age)
```

```
p = Person("Fred", 2)
pickle_data = pickle.dumps(p, protocol=?)
print(pickle_data)
```

```
q = pickle.loads(pickle_data)
q.show()
```

Fred 2

b'ccopy_reg\n_reconstructor\np0\n(c_main_\nPerson\np1\nc_builtin_\nobject\np2\nNtp3\nRp4\n(dp5\nVname\np6\nVFred\np7\nsVage\np8\nl26\nsb.'

protocol=0

protocol=4

b'\x80\x04\x955\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\x06Person\x94\x93\x94)\x81\x94}\x94(\x8c\x04name\x94\x8c\x04Fred\x94\x8c\x03age\x94K\x1aub.'

例如：从CIFAR-10 数据集（序列化存储）文件中读取图片数据并显示图片

CIFAR-10 是由 Hinton 的学生整理的一个用于识别普适物体的小型数据集。

该数据集共有60000张彩色图像，这些图像是32*32，分为10个类，每类6000张图。

- 其中有50000张用于训练，构成了5个训练批，每一批含10000张图；
- 另外10000张图用于测试，单独构成一批。

airplane



automobile



bird



cat



deer



dog



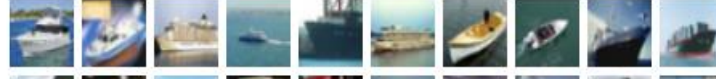
frog



horse



ship



truck



例如：从CIFAR-10 数据集（序列化存储）文件中读取图片数据并显示图片

```
import pickle
import numpy as np
import matplotlib.pyplot as plt

path1 = r'C:\cifar-10-python\cifar-10-batches-py\data_batch_1'

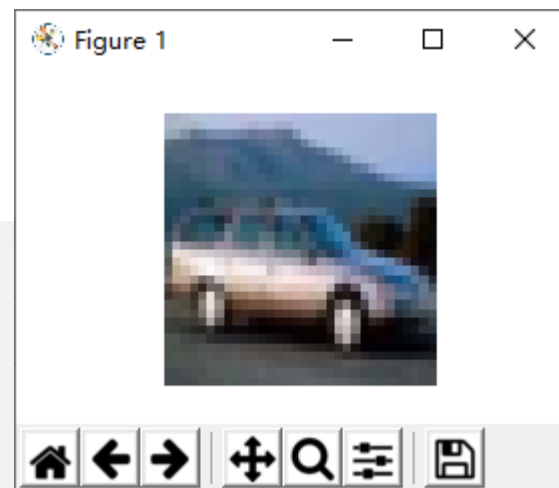
with open(path1, 'rb') as fo:
    data = pickle.load(fo, encoding='bytes')

    print(data[b'batch_label']) # b'training batch 1 of 5'
    print(data[b'labels']) # [6, 9, 9, 4, 1, 1, 2, 7...] 每张图片对应的分类标签
    print(len(data[b'labels'])) # 10000
    print(data[b'filenames']) # [b'leptodactylus_pentadactylus_s_000004.png',
    b'camion_s_000148.png', ...] 每张图片对应的（文件）名称
    print(data[b'data'].shape) # (10000, 3072) 10000张照片，每张一行存
    储，大小为32*32*3=3072
```

例如：从CIFAR-10 数据集（序列化存储）文件中读取图片数据并显示图片

```
images_batch = np.array(data[b'data'])
images = images_batch.reshape([-1, 3, 32, 32])
print(images.shape)
imgs = images[4, :, :, :].reshape([3, 32, 32]) # 第5张照片的数据
img = np.stack((imgs[0, :, :], imgs[1, :, :], imgs[2, :, :]), 2) # 取出红、绿、蓝三色对应的三个矩阵，并按第三个轴堆叠成一个三维数组
print(img.shape) # (32, 32, 3)彩色图像是三维数组，
```

```
plt.imshow(img)
plt.axis('off')
plt.show()
```



第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

10.6.1 文件操作

10.6.2 目录操作

10.6.3 I/O函数的使用

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

本章系统学习了文件的读写操作、CSV文件的读写、文件的各种系统操作以及存储对象等。

保存文件时，如果遇到是列表、字典、集合，甚至是类的实例这些更加复杂的数据类型的时候，我们就变得不知所措了，也许我们会把这些数据类型转换成字符串再保存到一个文本文件里，但是我们发现把这个过程反过来，从文本文件恢复数据对象，把一个字符串恢复成列表，恢复成字典，甚至恢复成集合，类，类的实例，我们发现会是一件异常困难的事情，庆幸的是**Python**提供了一个功能强大的标准模块“**pickle**”，使我们非常复杂的数据类型(比如列表，字典等)转换为二进制文件。

注意，Python的json模块用于处理**JSON文件**与**字典**的相互转化，第七章模块中已经介绍了其使用方法。

第十章 文件操作

10.1 打开文件

10.2 基本的文件方法

10.3 基本的目录方法

10.4 CSV文件读写

10.5 序列化与反序列化对象

10.6 实验

10.7 小结

10.8 习题

习题：

1. 二进制文件与文本文件有什么区别？
2. 从给定的一段英文文本中选出以'er'结尾的单词，并将结果写入文本文件，每个单词单独占用一行。
3. 写函数，函数接收四个参数分别是：姓名、性别、年龄和学历，其中性别默认为男。该函数的功能是将传入的这四项内容（逗号分割）作为一行**追加**到一个student.txt文件中。测试代码则支持用户持续输入，Q或者q退出，输入一个学生信息前给出提示“是否继续录入学生信息(输入Q退出):”，学生的各项信息也分别提示，例如“请输入姓名:”。
4. 给定若干表示学生信息数据（姓名、性别、年龄和学历）的嵌套列表，然后写入csv文件。
5. 从国家海洋科学数据中心爬取到相关数据存放在文件“全球海平面观测数据文件信息.csv”内，请读取其数据到字典列表中。

习题：

6. 文本文件depth.xyz中存储了用Tab分割的3列数据，分别表示某区域的经度、纬度和水深值。尝试使用正则表达式提取其中的水深为负值的各行数据，并将其中的负号（-）去除，并将结果保存至文件depth2.xyz。
7. 编程实现jpg、pdf、word、txt、zip等任意文件的复制。
8. 编写一个梯形类，然后创建一个对象并将其序列化至磁盘文件，在另一个python程序中对其进行反序列化。
9. 从CIFAR-10 数据集文件中随机读取两张图片，然后将两张图片像素点按一定比例融合，最后显示这两张原图及其融合图。

感谢聆听

