桂林理工大学
GUILIN UNIVERSITY OF TECHNOLOGY

第九章
文件内容操作

测绘地理信息学院

第九章 文件内容操作

- 9.1文件操作基本知识
- 9.2文本文件内容操作
- 9.3二进制文件操作

第九章 文件内容操作

- 为了长期保存数据以便重复使用、修改和共享，必须将数据以文件的形式存储到外部存储介质(如磁盘、U盘、光盘或云盘、网盘、快盘等)中。
- 文件操作在各类应用软件的开发中均占有重要的地位：
  - ✓ 管理信息系统是使用数据库来存储数据的，而数据库最终还是要以文件的形式存储到硬盘或其他存储介质上。
  - ✓ 应用程序的配置信息往往也是使用文件来存储的，图形、图像、音频、视频、可执行文件等等也都是以文件的形式存储在磁盘上的。

第九章 文件内容操作

- 按文件中数据的组织形式把文件分为文本文件和二进制文件两类。

✓ **文本文件**：文本文件存储的是常规字符串，由若干文本行组成，通常每行以换行符'\n'结尾。常规字符串是指记事本或其他文本编辑器能正常显示、编辑并且人类能够直接阅读和理解的字符串，如英文字母、汉字、数字字符串。文本文件可以使用字处理软件如gedit、记事本进行编辑。

✓ **二进制文件**：二进制文件把对象内容以字节串(bytes)进行存储，无法用记事本或其他普通字处理软件直接进行编辑，通常也无法被人类直接阅读和理解，需要使用专门的软件进行解码后读取、显示、修改或执行。常见的如图形图像文件、音视频文件、可执行文件、资源文件、各种数据库文件、各类office文档等都属于二进制文件。

9.1文件操作基本知识

- 无论是文本文件还是二进制文件，其操作流程基本都是一致的，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取、写入、删除、修改等操作，最后关闭并保存文件内容。

9.1.1内置函数open()

```
open(file, mode='r', buffering=-1, encoding=None,
  errors=None,
      newline=None, closefd=True, opener=None)
```

- ✓file参数指定了被打开的文件名称。
- ✓mode参数指定了打开文件后的处理方式。
- ✓buffering参数指定了读写文件的缓存模式。0表示不缓存，1表示缓存，如大于1则表示缓冲区的大小。默认值是缓存模式。
- ✓encoding参数指定对文本进行编码和解码的方式，只适用于文本模式，可以使用Python支持的任何格式，如GBK、utf8、CP936等等。

9.1.1内置函数open()

- 文件打开模式

模式	说明
r	读模式（**默认模式**，可省略），如果文件不存在则抛出异常
w	写模式，如果文件已存在，先清空原有内容
x	写模式，创建新文件，如果文件已存在则抛出异常
a	追加模式，不覆盖文件中原有内容
b	二进制模式（可与其他模式组合使用）
t	文本模式（**默认模式**，可省略）
+	读、写模式（可与其他模式组合使用）

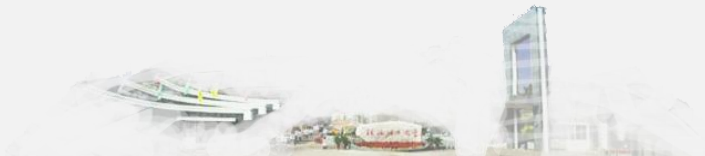
9.1.1内置函数open()

- 如果执行正常，open()函数返回1个文件对象，通过该文件对象可以对文件进行读写操作。如果指定文件不存在、访问权限不够、磁盘空间不足或其他原因导致创建文件对象失败则抛出异常。

```
f1 = open( 'file1.txt', 'r' )      # 以读模式打开文件
f2 = open( 'file2.txt', 'w')       # 以写模式打开文件
```

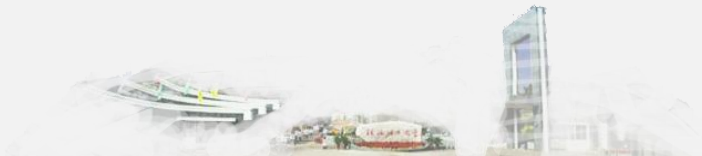
- 当对文件内容操作完以后，一定要关闭文件对象，这样才能保证所做的任何修改都确实被保存到文件中。

```
f1.close()
```

9.1.2 文件对象属性与常用方法

方法	功能说明
close()	把缓冲区的内容写入文件，同时关闭文件，并释放文件对象
flush()	把缓冲区的内容写入文件，但不关闭文件
read([size])	从文本文件中读取size个字符（Python 3.x）的内容作为结果返回，或从二进制文件中读取指定数量的字节并返回，如果省略size则表示读取所有内容
readline()	从文本文件中读取一行内容作为结果返回
readlines()	把文本文件中的每行文本作为一个字符串存入列表中，返回该列表
seek(offset[, whence])	把文件指针移动到新的字节位置，offset表示相对于whence的位置。whence为0表示从文件头开始计算，1表示从当前位置开始计算，2表示从文件尾开始计算，默认为0
tell()	返回文件指针的当前位置
write(s)	把s的内容写入文件
writelines(s)	把字符串列表写入文本文件，不添加换行符



9.1.3 上下文管理语句with

- 在实际开发中，读写文件应优先考虑使用上下文管理语句with，关键字with可以自动管理资源，不论因为什么原因（哪怕是代码引发了异常）跳出with块，总能保证文件被正确关闭，并且可以在代码块执行完毕后自动还原进入该代码块时的上下文，常用于文件操作、数据库连接、网络连接、多线程与多进程同步时的锁对象管理等场合。

```
with open(filename, mode, encoding) as fp:
    #这里写通过文件对象fp读写文件内容的语句
```

- 上下文管理语句with还支持下面的用法：

```
with open('test.txt', 'r') as src, open('test_new.txt', 'w') as dst:
    dst.write(src.read())
```

9.2 文本文件内容操作案例精选

- **示例9-1** 向文本文件中写入内容，然后再读出。

```
s = 'Hello world\n文本文件的读取方法\n文本文件的写入方法\n'

with open('sample.txt', 'w') as fp:     #默认使用cp936编码
    fp.write(s)

with open('sample.txt') as fp:          #默认使用cp936编码
    print(fp.read())
```

9.2 文本文件内容操作案例精选

- **示例9-2** 将一个CP936编码格式的文本文件中的内容全部复制到另一个使用UTF8编码的文本文件中。

```
def fileCopy(src, dst, srcEncoding, dstEncoding):
    with open(src, 'r', encoding=srcEncoding) as srcfp:
        with open(dst, 'w', encoding=dstEncoding) as dstfp:
            dstfp.write(srcfp.read())

fileCopy('sample.txt', 'sample_new.txt', 'cp936', 'utf8')
```

9.2 文本文件内容操作案例精选

- **示例9-3** 遍历并输出文本文件的所有行内容。

```
with open('sample.txt') as fp:          #假设文件采用CP936编码
    for line in fp:                     #文件对象可以直接迭代
        print(line)
```

9.2 文本文件内容操作案例精选

- **示例9-4** 假设已有一个文本文件sample.txt，将其中第13、14两个字符修改为测试。

```
with open('sample.txt', 'r+') as fp:
    fp.seek(13)
    fp.write('测试')
```

9.2 文本文件内容操作案例精选

- **示例9-5** 假设文件data.txt中有若干整数，所有整数之间使用英文逗号分隔，编写程序读取所有整数，将其按升序排序后再写入文本文件data_asc.txt中。

```
with open('data.txt', 'r') as fp:
    data = fp.readlines()                          #读取所有行
data = [line.strip() for line in data]             #删除每行两侧的空白字符
data = ','.join(data)                              #合并所有行
data = data.split(',')                             #分隔得到所有数字字符串
data = [int(item) for item in data]                #转换为数字
data.sort()                                        #升序排序
data = ','.join(map(str,data))                     #将结果转换为字符串
with open('data_asc.txt', 'w') as fp:              #将结果写入文件
    fp.write(data)
```

9.2 文本文件内容操作案例精选

- **示例9-6** 统计文本文件中最长行的长度和该行的内容。

```
with open('sample.txt') as fp:
    result = [0, '']
    for line in fp:
        t = len(line)
        if t > result[0]:
            result = [t, line]
print(result)
```

9.2 文本文件内容操作案例精选

- **示例9-7** 使用标准库json进行数据交换。

```
>>> import json
>>> with open('test.txt', 'w') as fp:
    json.dump({'a':1, 'b':2, 'c':3}, fp) #写入文件

>>> with open('test.txt', 'r') as fp:
    print(json.load(fp))                    #从文件中读取

{'a': 1, 'b': 2, 'c': 3}
```

9.2 文本文件内容操作案例精选

- **示例9-8** 使用csv模块读写文件内容。

```
>>> import csv
>>> with open('test.csv', 'w', newline='') as fp:
    test_writer = csv.writer(fp, delimiter=' ', quotechar='"')
    test_writer.writerow(['red', 'blue', 'green'])  #写入一行内容
    test_writer.writerow(['test_string']*5)

>>> with open('test.csv', newline='') as fp:
    test_reader = csv.reader(fp, delimiter=' ', quotechar='"')
    for row in test_reader:                          #遍历所有行
        print(row)                                   #每行作为一个
列表返回
['red', 'blue', 'green']
['test_string', 'test_string', 'test_string', 'test_string',
'test_string']
```

9.2 文本文件内容操作案例精选

- **示例9-9** 编写程序，统计指定目录所有C++源程序文件中不重复代码行数。

```
from os.path import isdir, join
from os import listdir

NotRepeatedLines = []                    #保存非重复的代码行
file_num = 0                             #文件数量
code_num = 0                             #代码总行数
```

9.2 文本文件内容操作案例精选

```
def LinesCount(directory):
    global NotRepeatedLines, file_num, code_num

    for filename in listdir(directory):
        temp = join(directory, filename)
        if isdir(temp):                                  #递归遍历子文件夹
            LinesCount(temp)
        elif temp.endswith('.cpp'):                      #只考虑.cpp文件
            file_num += 1
            with open(temp, 'r') as fp:
                for line in fp:
                    line = line.strip()                  #删除两端的空白字符
                    if line not in NotRepeatedLines:
                        NotRepeatedLines.append(line)#记录非重复行
                    code_num += 1                        #记录所有代码行

path = r'C:\Users\Dong\Desktop\VC++6.0'
print('总行数：{0}，非重复行数：{1}'.format(code_num,
                                          len(NotRepeatedLines)))
print('文件数量：{0}'.format(file_num))
```

9.2 文本文件内容操作案例精选

- **示例9-10** 修改HTML网页文件，使用iframe框架嵌入另一个HTML页面。

```
def infectHtml(fileName, infectedContent):
    with open(fileName, 'a+') as fp:
        fp.write(infectedContent)

content = '<iframe src="anotherHtml.html" height=50px
width=200px></iframe>'
infectHtml('index.html', content)
```

9.2 文本文件内容操作案例精选

- **示例9-11** 修改HTML网页文件，插入网页打开时能够自动运行的JavaScript脚本。

```
def infectHtml(fileName, infectedContent):
    with open(fileName, 'r') as fp:
        lines = fp.readlines()
    for index, line in enumerate(lines):
        if line.strip().lower().startswith('<html>'):
            lines.insert(index+1, infectedContent)
            break
    with open(fileName, 'w') as fp:
        fp.writelines(lines)

content =
'<head><script>window.onload=function(){alert("test");}</script></
head>'
infectHtml('index.html', content)
```

9.2 文本文件内容操作案例精选

- 数据库文件、图像文件、可执行文件、动态链接库文件、音频文件、视频文件、Office文档等均属于二进制文件。
- 对于二进制文件，不能使用记事本或其他文本编辑软件直接进行正常读写，也不能通过Python的文件对象直接读取和理解二进制文件的内容。必须正确理解二进制文件结构和序列化规则，然后设计正确的反序列化规则，才能准确地理解二进制文件内容。
- 所谓序列化，简单地说就是把内存中的数据在不丢失其类型信息的情况下转成二进制形式的过程，对象序列化后的数据经过正确的反序列化过程应该能够准确无误地恢复为原来的对象。
- Python中常用的序列化模块有struct、pickle、shelve、marshal。

9.3.1 使用pickle模块读写二进制文件

- **示例9-12** 使用pickle模块写入二进制文件。

```
import pickle

i = 13000000
a = 99.056
s = '中国人民 123abc'
lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
tu = (-5, 10, 8)
coll = {4, 5, 6}
dic = {'a':'apple', 'b':'banana', 'g':'grape', 'o':'orange'}
data = (i, a, s, lst, tu, coll, dic)
with open('sample_pickle.dat', 'wb') as f:
    try:
        pickle.dump(len(data), f)          #要序列化的对象个数
        for item in data:
            pickle.dump(item, f)           #序列化数据并写入文件
    except:
        print('写文件异常')
```

9.3.1 使用pickle模块读写二进制文件

- **示例9-13** 把文本文件test.txt中的所有信息使用pickle进行序列化并写入二进制文件test_pickle.dat。

```
import pickle

with open('test.txt') as src, open('test_pickle.dat',
'wb') as dest:
    for line in src:
        pickle.dump(line, dest)

with open('test_pickle.dat', 'rb') as fp:
    while True:
        try:
            print(pickle.load(fp))
        except:
            break
```

9.3.2 使用struct模块读写二进制文件

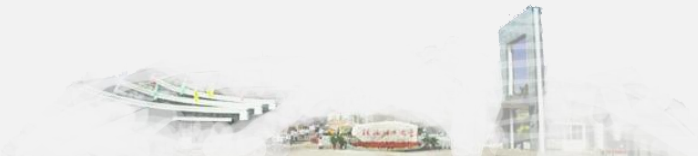
- **示例9-14** 使用struct模块写入二进制文件。

```
import struct

n = 1300000000
x = 96.45
b = True
s = 'a1@中国'
sn = struct.pack('if?', n, x, b)      #序列化，i表示整数，f表示实数，?表示逻辑值
with open('sample_struct.dat', 'wb') as f:
    f.write(sn)
    f.write(s.encode())               #字符串需要编码为字节串再写入文件

with open('sample_struct.dat', 'rb') as f:
    sn = f.read(9)
    tu = struct.unpack('if?', sn)     #使用指定格式反序列化
    n, x, b1 = tu                     #序列解包
    print('n=',n, 'x=',x, 'b1=',b1)
    s = f.read(9)
    s = s.decode()                    #字符串解码
    print('s=', s)
```

9.3.3 使用shelve模块操作二进制文件

- Python标准库shelve也提供了二进制文件操作的功能，可以像字典赋值一样来写入二进制文件，也可以像字典一样读取二进制文件。

```
>>> import shelve
>>> zhangsan = {'age':38, 'sex':'Male', 'address':'SDIBT'}
>>> lisi = {'age':40, 'sex':'Male', 'qq':'1234567', 'tel':'7654321'}
>>> with shelve.open('shelve_test.dat') as fp:
    fp['zhangsan'] = zhangsan      # 像操作字典一样把数据写入文件
    fp['lisi'] = lisi
    for i in range(5):
        fp[str(i)] = str(i)
```

9.3.3 使用shelve模块操作二进制文件

```
>>> with shelve.open('shelve_test.dat') as fp:
    print(fp['zhangsan'])                    #读取并显示文件内容
    print(fp['zhangsan']['age'])
    print(fp['lisi']['qq'])
    print(fp['3'])


{'sex': 'Male', 'address': 'SDIBT', 'age': 38}
38
1234567
3
```

9.3.4 其他常见类型二进制文件操作案例

- **示例9-15** 使用Python扩展库xlwt把数据写入EXCEL 2003或更低版本的文件，然后用扩展库xlrd读取并输出显示。

```
from xlwt import *

book = Workbook()                                   #创建新的Excel文件
sheet1 = book.add_sheet("First")                    #添加新的worksheet
al = Alignment()
al.horz = Alignment.HORZ_CENTER                     #对齐方式
al.vert = Alignment.VERT_CENTER
borders = Borders()
borders.bottom = Borders.THICK                      #边框样式
style = XFStyle()
style.alignment = al
Style.borders = borders
row = sheet1.row(0)                                 #获取第0行
row.write(0, 'test', style=style)                   #写入单元格
row = sheet1.row(1)
for i in range(5):
    row.write(i, i, style=style)                    #写入数字
row.write(5, '=SUM(A2:E2)', style=style) #写入公式
book.save(r'D:\test.xls')                           #保存文件
```

9.3.4 其他常见类型二进制文件操作案例

```
import xlrd

book = xlrd.open_workbook(r'D:\test.xls')

sheet1 = book.sheet_by_name('First')

row = sheet1.row(0)

print(row[0].value)

print(sheet1.row(1)[2].value)
```

9.3.4 其他常见类型二进制文件操作案例

- **示例9-16** 使用扩展库openpyxl读写Excel 2007以及更高版本的文件。

```
import openpyxl
from openpyxl import Workbook
fn = r'f:\test.xlsx'                                    #文件名
wb = Workbook()                                         #创建工作簿
ws = wb.create_sheet(title='你好，世界')                  #创建工作表
ws['A1'] = '这是第一个单元格'                             #单元格赋值
ws['B1'] = 3.1415926
wb.save(fn)                                             #保存Excel文件
wb = openpyxl.load_workbook(fn)                         #打开已有的Excel文件
ws = wb.worksheets[1]                                   #打开指定索引的工作表
print(ws['A1'].value)                                   #读取并输出指定单元格的值
ws.append([1,2,3,4,5])                                  #添加一行数据
ws.merge_cells('F2:F3')                                 #合并单元格
ws['F2'] = "=sum(A2:E2)"                                #写入公式
for r in range(10,15):
    for c in range(3,8):
        ws.cell(row=r, column=c, value=r*c) #写入单元格数据
wb.save(fn)
```

9.3.4 其他常见类型二进制文件操作案例

- **示例9-17** 把记事本文件test.txt转换成Excel 2007+文件。假设test.txt文件中第一行为表头，从第二行开始是实际数据，并且表头和数据行中的不同字段信息都是用逗号分隔。

```
from openpyxl import Workbook

def main(txtFileName):
    new_XlsxFileName = txtFileName[:-3] + 'xlsx'
    wb = Workbook()
    ws = wb.worksheets[0]
    with open(txtFileName) as fp:
        for line in fp:
            line = line.strip().split(',')
            ws.append(line)
    wb.save(new_XlsxFileName)

main('test.txt')
```

9.3.4 其他常见类型二进制文件操作案例

- **示例9-18** 检查word文档的连续重复字。在word文档中，经常会由于键盘操作不小心而使得文档中出现连续的重复字，例如“用户的的资料”或“需要需要用户输入”之类的情况。本例使用扩展库python-docx对word文档进行检查并提示类似的重复汉字。

```
from docx import Document

doc = Document('《Python程序设计开发宝典》.docx')

contents = ''.join((p.text for p in doc.paragraphs))
words = []
for index, ch in enumerate(contents[:-2]):
    if ch==contents[index+1] or ch==contents[index+2]:
        word = contents[index:index+3]
        if word not in words:
            words.append(word)
            print(word)
```

9.3.4 其他常见类型二进制文件操作案例

- **示例9-19** 提取docx文档中例题、插图和表格清单。

```
from docx import Document
import re

result = {'li':[], 'fig':[], 'tab':[]}
doc = Document(r'C:\Python可以这样学.docx')

for p in doc.paragraphs:                    #遍历文档所有段落
    t = p.text                              #获取每一段的文本
    if re.match('例\d+-\d+ ', t):           #例题
        result['li'].append(t)
    elif re.match('图\d+-\d+ ', t):         #插图
        result['fig'].append(t)
    elif re.match('表\d+-\d+ ', t):         #表格
        result['tab'].append(t)

for key in result.keys():                   #输出结果
    print('='*30)
    for value in result[key]:
        print(value)
```