Python 程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">9.1 文件的概念及分类
9.2 文件操作基本知识
9.3 文本文件操作案例
9.4 二进制文件操作</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">了解文件的概念及分类
掌握内置函数 open()的用法
熟练运用 with 关键字
了解 pickle、struct、shelve、marshal 等模块的用法</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">文件打开模式
二进制文件与文本文件操作的区别
序列化与反序列化</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">首先介绍文件操作的基本概念与流程，然后介绍文本文件操作的方法，最后介绍二进制文件操作的原理以及序列化和反序列化的应用。</td></tr>
<tr><td>实　验</td><td colspan="2">教材中 9.4 节之前的例题。</td></tr>
<tr><td>思考题及作业题</td><td colspan="2">课后习题 9.1、9.2</td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2">第　一　节　课</td></tr>
</table>

课堂重点内容详解

Python 内置函数 open()可以指定模式打开指定文件并创建文件对象，该函数完整的用法如下：

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

内置函数 open()的主要参数含义如下：

- 参数 file 指定要打开或创建的文件名称。
- 参数 mode（取值范围见表 9-1）指定打开文件后的处理方式。
- 参数 encoding 指定对文本进行编码和解码的方式。

如果执行正常，open()函数返回 1 个可迭代的文件对象，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、磁盘空间不够或其他原因导致创建文件对象失败则抛出异常。下面的代码分别以读、写方式打开了两个文件并创建了与之对应的文件对象。

```
fp = open('file1.txt', 'r')
fp = open('file2.txt', 'w')
```

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。

```
fp.close()
```

另外需要注意的是，即使我们写了关闭文件的代码，也无法保证文件一定能够正常关闭。例如，如果在打开文件之后和关闭文件之前发生了错误导致程序崩溃，这时文件就无法正常关闭。在管理文件对象时推荐使用 with 关键字，可以避免这个问题（具体参见 9.2.3 内容）。

表 9-1 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），使用二进制模式打开文件时不允许指定 encoding 参数
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）

表 9-2 文件对象常用方法

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
read([size])	从文本文件中读取 size 个字符作为结果返回，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，对于大文件会占用较多内存，不建议使用
write(s)	把字符串 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

例 9-1 将字符串写入文本文件，然后再读取并输出。

```
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())
```

例 9-2 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:                       #文本文件对象可以直接迭代
        print(line)
```

例 9-3 假设文件 data.txt 中有若干行整数，每行有 1 个整数。编写程序读取所有整数，将其按降序排序后再写入文本文件 data_desc.txt 中，每行 1 个整数。

```
with open('data.txt', 'r') as fp:
    data = fp.readlines()                          #读取所有行，存入列表
data = [int(item) for item in data]                #列表推导式，转换为数字
data.sort(reverse=True)                            #降序排序
data = [str(item)+'\n' for item in data]           #将结果转换为字符串
with open('data_desc.txt', 'w') as fp:             #将结果写入文件
    fp.writelines(data)
```

例 9-4 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, '']           #使用列表保存某行长度和内容
    for line in fp:            #遍历文件中每一行的内容
        t = len(line)          #该行文本的长度
        if t > result[0]:      #发现更长的行，保存最新结果
            result = [t, line]
print(result)
```

第 二 节 课

例 9-5 使用 pickle 模块写入二进制文件。

```
import pickle
```

```
#实际要写入的数据，有整数、实数、字符串、列表、元组、集合、字典
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'}
#把所有要写入的数据变量放入一个元组，以便编写循环代码
data = (i, a, s, lst, tu, coll, dic)

#wb 表示打开文件用来写入二进制数据
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')
```

例 9-6 使用 pickle 模块读取上例中二进制文件的内容。

```
import pickle

with open('sample_pickle.dat', 'rb') as f:
    n = pickle.load(f)                  #读出文件中的数据个数
    for i in range(n):
        x = pickle.load(f)              #读取并反序列化每个数据
        print(x)
```

例 9-7 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)
with open('sample_struct.dat', 'wb') as f:
    f.write(sn)
    f.write(s.encode())
```

例 9-8 使用 struct 模块读取上例中二进制文件的内容。

```
import struct

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

例 9-9 使用 shelve 模块读写二进制文件。

```
>>> 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              #以字典形式把数据写入文件
    fp['lisi'] = lisi
    for i in range(5):
        fp[str(i)] = str(i)

>>> 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'])
```

例 9-10 使用 marshal 模块读写二进制文件，并对对象进行序列化和反序列化操作。

```
>>> import marshal                      #导入模块
>>> x1 = 30                              #待序列化的对象
>>> x2 = 5.0
>>> x3 = [1, 2, 3]
>>> x4 = (4, 5, 6)
>>> x5 = {'a':1, 'b':2, 'c':3}
>>> x6 = {7, 8, 9}
>>> x = [eval('x'+str(i)) for i in range(1,7)]  #把所有数据放入列表
>>> with open('test.dat', 'wb') as fp: #创建二进制文件
    marshal.dump(len(x), fp)             #先写入对象个数
    for item in x:
        marshal.dump(item,fp)  #把列表中的对象依次序列化并写入文件

>>> with open('test.dat', 'rb') as fp:  #打开二进制文件
    n = marshal.load(fp)                  #获取对象个数
    for i in range(n):
        print(marshal.load(fp))          #反序列化，输出结果
```

Python 程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">9.5 Excel 与 Word 文件操作案例</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">了解 python-docx、openpyxl 扩展库的安装与使用
了解 Excel、Word 文件操作</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">Excel、Word 文档结构与相关操作</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">首先介绍扩展库 python-docx 和 openpyxl 的安装，然后介绍 Excel 文件与 Word 文档的结构，最后通过几个例题演示相关操作。</td></tr>
<tr><td>实　验</td><td colspan="2">例题 9-11 至 9-16</td></tr>
<tr><td>思考题及作业题</td><td colspan="2">课后习题 3、4、5</td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2">第　一　节　课</td></tr>
</table>

课堂重点内容详解

例 9-11 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)                                    #保存文件
```

例 9-12 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):
    #得到对应的 Excel 文件名
    new_XlsxFileName = txtFileName[:-3] + 'xlsx'
    #创建工作簿，并获取其中第一个工作表
    wb = Workbook()
    ws = wb.worksheets[0]
    #打开原始的记事本文件，依次读取每行内容，切分后写入 Excel 文件
    with open(txtFileName) as fp:
        for line in fp:
            #切分后得到列表，可以直接追加到工作表中
            #每个元素写入一个单元格
            line = line.strip().split(',')
            ws.append(line)
    #保存 Excel 文件
    wb.save(new_XlsxFileName)

main('test.txt')
```

例 9-13 输出 Excel 文件中单元格中公式计算结果。

```
import openpyxl
```

```
#打开 Excel 文件
wb = openpyxl.load_workbook('data.xlsx', data_only=True)
ws = wb.worksheets[1]
for row in ws.rows:
    print(row[3].value)
```

第 二 节 课

例 9-14 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')

#把所有段落的文本连接成为一个长字符串
contents = ''.join((p.text for p in doc.paragraphs))
#使用列表来存储可疑的子字符串
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)
```

例 9-15 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('《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)
```

例 9-16 查找 Word 文件中所有红色字体和加粗的文字。

```
from docx import Document
from docx.shared import RGBColor

boldText = []
redText = []
#打开 Word 文件，遍历所有段落
doc = Document('test.docx')
for p in doc.paragraphs:
    for r in p.runs:
        #加粗字体
        if r.bold:
            boldText.append(r.text)
        #红色字体
        if r.font.color.rgb == RGBColor(255,0,0):
            redText.append(r.text)

result = {'red text': redText,
          'bold text': boldText,
          'both': set(redText) & set(boldText)}

# 输出结果
for title in result.keys():
    print(title.center(30, '='))
    for text in result[title]:
        print(text)
```