计算机学院 数据科学概论 课程实验报告

实验题目：数据结构和 python 扩展库练习		学号：202200130041
日期：2023/3/8	班级： 22 级数据班	姓名： 左景萱
Email：zuojingxuan1130@mail.sdu.edu.cn		

实验目的：

本实验首先安装配置好 Anaconda 以及 Python 环境，通过代码编写，从而使我们熟练掌握 python 的各种数据结构以及库的调用，并学会将其运用到实践中。

实验软件和硬件环境：

操作系统：Windows11；

Anaconda 版本：5.3.0；

Python 版本：3.10.5；

实验步骤与内容：

1. 首先我们认识了 python 自带的数据结构中的 list，dict 和 tuple，了解了可变类型和不可变类型。对于 list，dict，我们了解了其对应的增删改查的操作，了解了浅复制(copy)和深复制(deepcopy)的区别：浅复制只会拷贝父级的目录（根目录）的数据，但不会拷贝子级的数据，而深复制递归地拷贝所有目录的数据，完全在另外内存中复制了一份原字典。
2. 然后我们了解了 collections，其中总共有五种数据类型。collections 是 python 的一个标准库模块,它提供了不同类型的容器 1。容器是用来存储和访问不同对象的对象，可以对它们进行迭代。一些内置的容器有元组，列表，字典等。collections 模块中有一些常用的容器类，例如：Counter：一个用来计数可哈希对象的字典子类。defaultdict：一个在访问不存在的键时返回默认值的字典子类。OrderedDict：一个记住元素插入顺序的字典子类。namedtuple：一个创建具有字段名和字段值的元组子类的工厂函数。Deque 和 C++中一样是一个双端队列。
3. 接着我们通过实例了解了 counter 类。使用 Counter 类，我们需要先导入 collections 模块，然后实例化一个 Counter 对象，可以传入一个可迭代的对象或者一个映射对象。

```
1.      from collections import Counter
2. c1 = Counter() # 空的Counter 对象
3. c2 = Counter('hello') # 从字符串中统计字符出现次数
4. c3 = Counter([1, 2, 3, 4, 5]) # 从列表中统计元素出现次数
5. c4 = Counter({'a': 1, 'b': 2}) # 从字典中统计键值对
```

Counter 对象是一个字典子类，它的键是可哈希的对象，它的值是整数，表示出现的次数 1。我们可以像访问字典一样访问 Counter 对象，也可以使用一些特有的方法：

```
1. print(c2['h']) # 访问'h'字符出现的次数，输出 1
2. print(c3.most_common(3)) # 返回出现次数最多的三个元素及其次数，输出
```

```
    [(5, 1), (4, 1), (3, 1)]
3. c4.update({'a': 3}) # 更新'a'键对应的值，输出{'a': 4, 'b': 2}
```

counter 类还有一些交并补，还有访问所有元素的 elements 方法。

4. 统计词出现频率的实例：

```python
import os
import re
import string
from collections import Counter

def initial_text_file(file_dir='./input.txt'):
    TEXT = '''
        Stray birds of summer come to my window to sing and fly away. And yellow leaves of autumn which have no songs, flutter and fall
         there with a sign.
        The world puts off its mask of vastness to its lover. It becomes small as one song, as one kiss of the eternal.
        The mighty desert is burning for the love of a blade of grass who shakes her head and laughs and flies away.
    '''

    with open(file_dir, 'w', encoding='utf-8') as f:
        f.write(TEXT)

file_dir = './input.txt'
if not os.path.exists(file_dir):
    initial_text_file()

lines = open(file_dir, "r").read().splitlines()
pattern=re.compile(re.escape(string.punctuation))
lines = [pattern.sub('', line).lower().strip().split(" ") for line in lines]

words = []
for line in lines:
    words.extend(line)
result = Counter(words)
print(result.most_common(10))
```

对于这段代码的解释如下：

首先，导入 os， re， string 和 collections 模块，这些模块提供了操作文件、正则表达式、字符串和计数器等功能。

然后，定义一个函数 initial_text_file，它的作用是创建一个文本文件，并写入一段英文文本。这个函数接受一个参数 file_dir，表示文件的路径，默认为’./input.txt’。

接着，定义一个变量 file_dir，也是文件的路径，并判断该文件是否存在。如果不存在，则调用 initial_text_file 函数创建该文件。

然后，使用 open 函数打开该文件，并使用 read 方法读取其内容。使用 splitlines 方法将内容按照换行符分割成一个列表，并赋值给变量 lines。

接着，使用列表推导式对 lines 中的每一行进行处理。使用 re.sub 函数将每一行中的标点符号替换为空字符串，使用 lower 方法将每一行转换为小写字母，使用 strip 方法去除每一行两端的空白字符，使用 split 方法将每一行按照空格分割成单词列表。最后得到一个嵌套列表，并赋值给变量 lines。

然后，创建一个空列表 words，并使用 for 循环遍历 lines 中的每个子列表。使用 extend 方法将每个子列表中的单词添加到 words 中。

最后，使用 collections 模块中的 Counter 类创建一个计数器对象 result，并传入 words 作为参数。使用 most_common 方法获取出现频率最高的 10 个单词及其次数，并打印出来。

5. 利用 jieba 统计中文文本中词出现的频率：

```python
import collections
import jieba
parameter_template = "\n{:>10}: {}"
with open('梦里花落知多少.txt', 'r', encoding='utf-8') as f:
    txt = f.read()
words = list(jieba.cut(txt,cut_all=True))
words_list=[]
for i in range(len(words)):
    if len(words[i])!=1 and words[i] not in ', 。！”“ \n':
        words_list.append(words[i])
words_dict=collections.Counter(words_list)
for (k,v) in words_dict.most_common(10):
    print(parameter_template.format(k,v))
```

由题意得出代码。使用 for 循环复制符合要求的单词是为了避免使用.remove()方法进行遍历删除，否则复杂度将会是 O(N^2)，在 80000 多单词的情况下使用 python 进行运算将会是缓慢的（个人觉得）。

得出结果如下：

```
火柴: 620
时候: 612
觉得: 601
什么: 549
知道: 529
可是: 513
微微: 498
怎么: 400
一个: 390
自己: 361
```

结论分析与体会：

使用列表操作进行 remove 虽然可以简化代码的书写，但是由于 remove（）方法的时间复杂度是 O（n）的将会拖慢程序的运行。在不同的数据大小情况下我们要选择合适的方法进行运算。使用 for 循环可以在节约时间的情况下不更多浪费内存。以后在进行更深层次地学习的时候也要根据数据量大小选择合适的算法，根据 GPU 显存的大小选择合适的 batch size。一切从实际出发才是最优的。

对于 jieba 库还有一些额外的操作可以学习，如添加词条，选择不同模式（精确模式、全模式、搜索引擎模式和 paddle 模式）等。

就实验过程中遇到的问题及解决处理方法，自拟 1－3 道问答题：

1. 为什么要避免迭代时修改迭代对象本身？

答：对于这个问题，我通过尝试发现如果在迭代中修改了迭代对象本身，那么每次删除元素后，索引都会发生变化，导致遍历出现漏洞。比如：

```
1. n = [1, 2, 3]
2. for i in n:
3.     n.remove(i)
4.     print(n)
```

输出的结果如下：

```
1. [2, 3]
2. [2]
3. []
```

2. 为什么使用 jieba 分词得到的词频会和使用正则表达式得到的答案不一样？

答：通过询问助教，我了解到这是 cut_all 参数设置的问题。将 cut_all 设置为 True 就可以进行全模式匹配就可以了。