Python 计算实验报告

组　　号	第 10 组
实验序号	实验一
学　　号	19122206
姓　　名	陈宝润
日　　期	2020 年 4 月 18 日

一、 实验目的与要求

1. 熟悉 Python 的开发调试环境;

2. 熟悉 Python 外部库的调用;

3. 掌握 Python 语言基本语法;

4. 熟悉 Python 的数据结构.

二、 实验环境

1. 操作系统：windows10

2. Python IDLE、VS Code

三、 实验内容

1．Python 代码理解 polygon.py:

（1）运行和阅读代码;

（2）理解代码功能;

（3）修改代码，练习调用文件中其他几个图形函数。

2．输入输出：编写脚本文件，设计友好的用户输入输出提示，用户输入一个时间（24 小时制，包含时、分、秒），输出 1 秒后的时间。

3．反序对：如果一个单词是另一个单词的反向序列，则称这两个单词为“反向对”。编写代码输出 word.txt 中词汇表包含的反向对。

4．文本分析算法设计:

（1）设计 Python 程序读入一个英文单词组成的文本文件，统计该文本文件中各个单词出现的次数。设计测试用例验证代码的正确性。

（2）设计 Python 程序读入一个英文单词组成的文本文件，统计其中包含的某给定关键词列表中各个单词出现的频率。设计测试用例验证代码的正确性。

四、 实验内容的设计与实现

1. 实验内容 1

（1） 安装 swampy 库及验证是否安装成功

a. Pip 安装

Pip 安装十分简洁方便，直接执行命令:

```
pip install swampy
```

b. 下载源码安装

先下载源码，可以使用 git 从 GitHub 上得到源代码

```
git clone https://github.com/AllenDowney/Swampy.git
```

然后在源代码目录下，打开 cmd 窗口，执行命令

```
setup.py install
```

即可安装成功

c. 验证是否安装成功

第一种方法比较简单，直接利用 pip 命令查看已安装的库

```
pip list
```

查看其中是否有 swampy 库

第二种方法就是利用 try/except 函数，编写脚本查看，用法较为简单，此处省略。

d. Pycharm 安装

个人倾向于在 pycharm 的虚拟环境下安装 swampy，因为这个库的调用次

数较少，不必要安装在全局环境下。

（2） 理解库中重要函数

在利用 swampy 库画图时，我们需要用到其中的 TurtleWorld 模块，在此模块中，有 3 组（6 个）比较重要的函数。

第一组函数为 fd(self, dist)和 bk(self, dist)，两个函数的参数时相同的，第一个参数时画图的“实例”，也就是我们能看到的小海龟，第二个参数是步长，就是海龟移动的距离。第一个函数的功能就是让海龟向前移动 dist 长度，第二个与前者相反，是让海龟向后移动。

第二组函数为 rt(self, angle)和 lt(self, angle)，两个函数的第一个参数与第一组函数中的相同，第一个参数表示角度。两个函数，前者的功能是让海龟右转，后者为让海龟左转，转过的角度为 angle，默认为 90 度。

第三组函数为 pd(self)和 pu(self)，前者的功能为画出小海龟的移动路径，后者为让小海龟的移动路径不再图形中显示。

当然，还有其它函数，比如更改移动路径的颜色，更改海龟的颜色，在此不做说明。

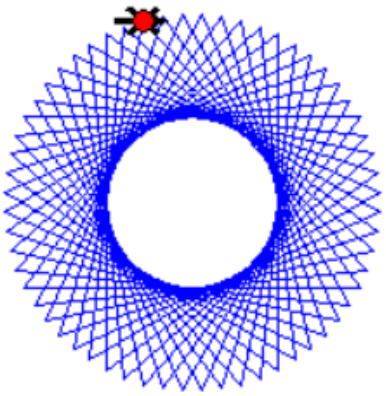
在老师所给的文件中，有 polygon.py 文件，在该 python 脚本中，对前面所讲的这几组函数做出了应用，可以画出 n 边形与圆形，设计较为简单，不做过多说明。

（3） 调用函数、编写简单程序

我自己也利用 TurtleWorld 模块中的函数，画出了自己的图形。

实现较为简单，利用一些数学关系式，画出一个基础图形，然后利用循环，重复画图，得到最终图形，如右图。

```
for i in range(60):
    rt(bob,102)
    fd(bob,100)
    lt(bob,60)
    fd(bob,100)
    rt(bob,132)
```



2. 实验内容 2

（1） 分析程序功能，确定程序结构

在这个小实验中，输入是关键，我们需要设计友好提示，让用户输入正确的格式的时间。假如用户输入的时间是 25 时 X 分 X 秒，由于 24 小时制的时间，伟大为 23 时，所以，需要提醒用户重新输入。用户输入正确的时间后，进行下一步——计算下一秒的时间，计算过程中，只需合理利用分支结构，判断时分秒这算个数是否需要进位即可。

（2） 编写程序

在用户输入时，我采用的是 input()函数，值得注意的是，在 python3 中，input 函数默认接收到的是 str 类型的数据，所以，我们需要进行强制类型转换，比如：h = int(input("Please input the hous:"))。同时，用一个 bool 变量记录输入的数据是否符合要求，judge_h = h < 24 and h >= 0。在输入分和秒时，进行同样的操作，输入结束后，判断是否符合要求，判断用户是否需要重新输入。

只要输入的数据符合规范，计算一下秒时间的算法就较为简单了，此处忽略。

3. 实验内容 3

（1） 设计流程，确定程序结构

在这个小实验中，首先，我们需要读取并存储 word.txt 中的词汇，所以，这里就涉及到用哪种数据结构去存储读取的内容。选择了合适的数据结构之后，利用选择的数据结构类型，设计算法。实验过程中，我们采用了两种数据结构，列表与集合。

（2） 编写程序

实验过程中，我们对程序的算法不断改进，程序性能也不断提高。

在第一版程序中，算法乏善可陈，用列表存储读取的数据，然后，对每一个元素，逐个查找其反序对。查找速度极慢，耗时 1000s 以上，在代码压缩包中，reverse1.1—1000s.py 对应着第一版程序。

第二版程序就有了很大的提升，也是用列表存储数据。核心思想是对列表进行“切片”，即根据单词的长度，对该列表排序，然后，在同长度的单词中，查找每个单词的反序对。

首先，读取文件时，我们记录下各个长度单词的个数：

```
def read_file():#读取文件
    global words   #全局变量，列表，存储数据
    global frequency #列表，存储各个长度单词的个数
    ans = 0
    file = open("words.txt")
    for line in file:
        words.append(line.strip())
        frequency[len(words[ans])] += 1
        ans += 1
```

然后，我们利用 sort()函数，对列表排序，调整 sort()中的 key 参数，让其等于每个单词的长度，同时根据列表 frequency，计算长度为 n 的单词范围中，第一个单词出现的索引值。

```
def calculate_range():#计算相同长度单词所在范围
    #frequency[i]表示长度为 i 的最后一个单词的索引
    global frequency
    for i in range(21):
        frequency[i + 1] += frequency[i]
```

最后，根据这个索引值，来进行“切片”查找。相比于第一版，取得极大进步，所用时间从 1000s 缩短为 30s。

第三版代码，我们同样采用列表，时间极大得缩短，但也付出了更大空间消耗的代价。

第三版代码中，我们用一个列表，存储读取到的单词（列表 x），同时，构造一个新的列表对象（列表 y），存储每个单词的“逆序”，然后对 y 依据首字母顺序排序：

```
word = open('words.txt', 'r')
x = word.read().split()
word.close()
y=list(map(lambda s: s[::-1],x))
y.sort()
```

由此，x 与 y 列表中，单词是按照字母顺排列，然后再构造两个列表，分别存储列表 x 与列表 y 中单词首字母发生变化的位置索引，比如说，以字母 b 开头的第一个单词的位置被记为 i，以字母 c 为首字母的第一个单词的位置被积为 j，根据这个位置索引，我们可以对列表进行一次切片，依据首字母切片。

所以，第三版代码，就是在第二版代码的基础之上，再多一层切片，就是在首字母相同的情况下，比较原单词列表与逆序单词列表，减少不必要的比较。最后，计算时间大约在 0.9s，源代码文件被命名为 reverse1.4.1—0.9s.py

第四版代码的思维量就少了很多，性能与第三版代码相差无几，它在读入数据时，同时存储原单词与其逆序单词，然后再将得到的列表依照字母排序，然后再每两个比较一次，若相同，则为逆序对，输出。

```
for line in file:
        t = line.strip()
        z.append(t)
        z.append(t[::-1])
z.sort()
for i in range(int(len(z)-1)):
        if z[i] ==z[i+1]:
                print(z[i], "&", z[i][::-1])
                i+=1
```

第五版代码，选择了集合作为数据存储的方式，速度非常快，同时，空间消耗也降低了不少。是这五版代码中，性能最强大的一版。代码清晰易懂。

```
file = open('words.txt', 'r')
words = set(file.read().split())
reverse = set()
for word in words:
        if word[::-1] in words:
                reverse.add(word)
```

这版代码的速度很快，时间大约为 0.2s，集合的速度比列表快很多，其中原理涉及到一些知识盲区，有待学习。

所以，似乎无论怎样改善算法，前面选择列表的的四版算法的速度都无法与第五版选择集合的代码相比。

4. 实验内容 4

（1） 实验要求分析分析

在完成第三个小实验之后，第四个实验就显得尤为简单了，只需选择合适的数据结构存储，然后根据数据结构，选择合适的函数完成实验。在此题中，无论单词出现的次数还是频率，与其单词都是一对一的关系，为方便查找，同时，考虑到速度问题，我们选择字典来存储单词与其频数或频率的关系。

（2） 程序编写

代码比较简单，相对重要的部分就是频数的计算

```
def calculate_frequency():
    global frequency
    words_set = set(words)
    for word in words_set:
        frequency[word] = words.count(word)
```

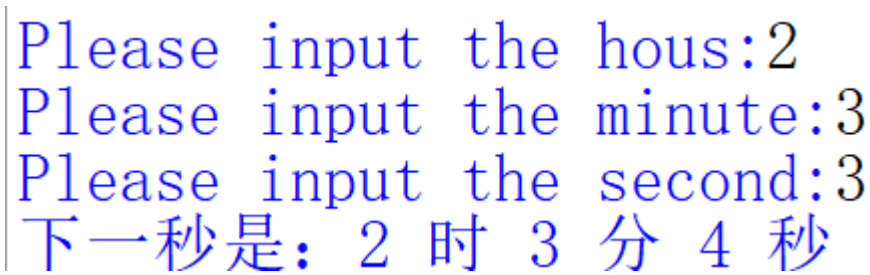
五、 测试用例

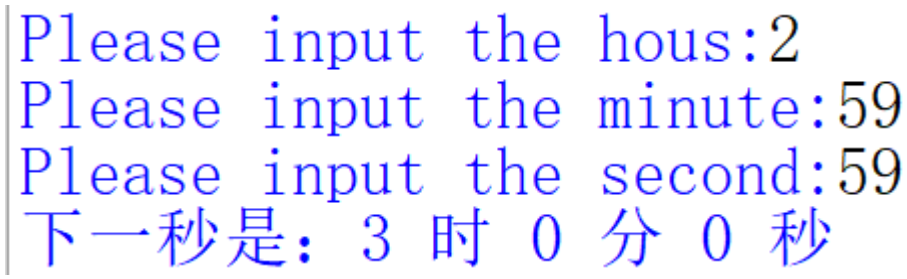
对于实验 2，我们可以尝试输入错误数据与正确数据，来测试程序功能。

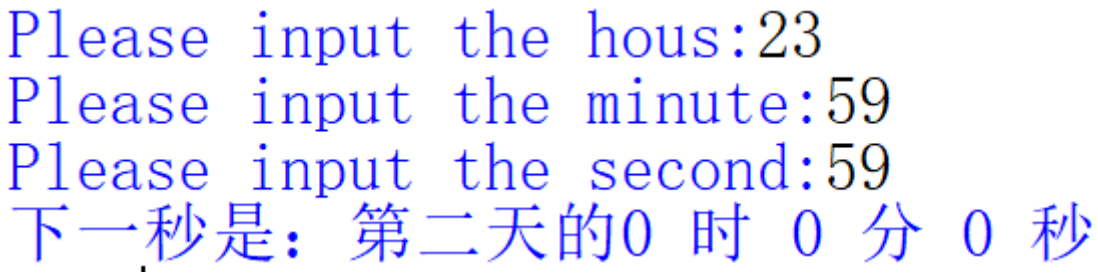
比如输入一个错误时间，程序会提醒你，让你再次输入时间

```
Please input the hous:25
Please input the minute:30
Please input the second:30
Please input the correct time!
```

当输入正确时：

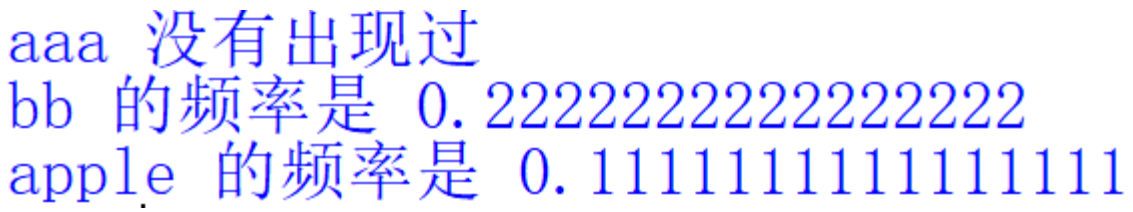






实验三的测试用例，在此省略。

对于实验四，我们需编辑两个 txt 文件，第一个命名为 word.txt，里面存储着单词总体，第二个命名为 newword.txt，里面存储着查找对象，比如在里面存储 apple，运行程序，将输出 apple 的频率或者频数：



六、 收获与体会

初学 python，收获很多。

首先，我熟悉了各种开发环境配置与使用，比如 IDLE，VS Code，Pycharm，也了解到了各种开发环境的各种优势，IDLE 轻便快捷， Pycharm 功能强大，VS Code 介于两者之间。个人倾向于用 VS Code，只需注意文件路径与文件名不能出现中文即可。但同样的，Pycharm

的虚拟环境也是相当有吸引力。

其次，我认识到在编程过程中，选择合适的数据结构有多么重要，它能给你带来极大的方便，就拿实验 3 来说，起初我们用列表来存储数据，我们花费很多精力，改善算法，但是，也敌不过集合的简单几行代码。

python Wiki 上说:“使用集和字典进行成员资格测试比搜索序列 O(n)更快, O(1)。测试“a in b”时，b 应该是集合或字典，而不是列表或元组。”用这句话来作为实验 3 的体会、总结，也最好不过了。