Python 计算实验报告

组　　号	第 10 组
实验序号	实验二
学　　号	19122206
姓　　名	陈宝润
日　　期	2020 年 5 月 4 日

一、 实验目的与要求

1. 熟悉 Python 的流程控制;

2. 熟悉 Python 的数据结构;

3. 掌握 Python 语言基本语法;

二、 实验环境

1. 操作系统：windows10

2. Python IDLE、VS Code

三、 实验内容

1 .Python 流程控制: 编写循环控制代码用下面公式逼近圆周率(精确到小数点后 15 位)，并和 math.py 的值做比较

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{k!^4 (396^{4k})}.$$

2．Python 流程控制：阅读 https://en.wikipedia.org/wiki/Koch_snowflake，通过修改 koch.py 绘制其中一种泛化的 Koch 曲线。

3．生日相同情形的概率分析:

（1）生成 M（M>=1000）个班级，每个班级有 N 名同学，用 input 接收 M 和 N;

（2）用 random 模块中的 randint 生成随机数作为 N 名同学的生日;

（3）计算 M 个班级中存在相同生日情况的班级数 Q，用 P=Q/M 作为对相同生日概率的估计;

（4）分析 M，N 和 P 之间的关系。

4．文本分析算法设计:

参照验证实验 1 中反序词实现的例示代码，设计 Python 程序找出 words.txt 中最长的“可缩减单词”（所谓“可缩减单词”是指：每次删除单词的一个字母，剩下的字母依序排列仍然是一个单词，直至单字母单词'a'或者'i'）。

提示：

（1）可缩减单词例示：sprite —〉spite —〉spit—〉pit—〉it—〉i

（2）如果递归求解，可以引入单词空字符串''作为基准。

（3）一个单词的子单词不是可缩减的单词，则该单词也不是可缩减单词。因此，记录已经查找到的可缩减单词可以提速整个问题的求解。

四、 实验内容的设计与实现

1. 实验内容 1

（1） 公式分析即程序架构

公式中，需要计算阶乘，还涉及幂运算，所以，导入 math 模块，利用其中的库函数，可以极大降低难度

同时，有累加计算，利用循环结构完成精确度判断，可以利用前后两次循环的差值来完成，若两者插值小于精度，则退出循环。

（2） 程序初次架构

```
while True:
    累加变量+=累加变量
    计算 pi
    if pi-上一次循环中的 pi < 1e-15:
        break
```

（3） 代码优化

在循环体类，我们可以通过记录，来减少计算量，从而节省计算时间。比如，我们可以记录 2*math.sqrt(8)/9801 来减少循环中不必要的计算量。具体代码实现见 pi.py 文件。

2. 实验内容 2

（1） 解读代码

在 koch.py 文件中，我们不难看出，一条科赫曲线，就是从一个正三角形出发，将一条边分成 3 等分，然后然后以各边的中间长度为底边，向外做正三角形，反复进行这一过程，就可得到“科赫雪花”

（2） 编写程序

我们可以对 koch.py 做出一些调整，改变 koch 函数，海龟左转变为右转，右转变为左转，其余不变，就可以得到科赫反雪花。

3. 实验内容 3

（1） 设计流程，确定程序结构

在这个小实验中，（1）（2）要求比较简单，在要求（3）中，需要查找存在相同生日情况的班级，在这里，我们可以利用集合来做生日的容器，我们随机生成 N 名同学的生日，送入集合，由于集合中不能出现重复元素，所以，集合中存储的就是不同生日的个数。所以，我们就可以通过比较集合长度和 N 的大小关系，来判断该班级中是否存在相同生日情况。如果集合长度小于 N，则存在。

（2） 编写程序

我们可以利用字典于集合来作为数据容器，字典的键表示班级编号，字典的值是集合，存储着生日信息。然后，我们根据字典推导式和集合推导式来生成数据。

```
birthday = {i:{random.randint(0, 365) for j in range(n)} for i in range(m)}
```

一行代码即可，当然，我们没有准确的生成 N 名同学的生日，我们在写这行代码，只是为要求（3）和（4），即数据分析服务的。

4. 实验内容 4

（1） 实验分析与程序逻辑架构

首先，每个可缩减单词都带有字母'i'和'a'，所以，我们在存取数据时，可以直接剔除不含这两个字母的单词。由于，我们的操作中会频繁判断，一个单词的子单词是否还是一个单词，即是否还在 words.txt 中，所以，我们可以采用字典或者集合的方式存储数据。

现在，似乎有两种方式，一种方式是自下而上，另一种为自上而下。

第一种方式，就是从长度短的单词判断，因为，一个单词的子单词不是可缩减单词，则该单词也不是可缩减单词。所以，从长度短的单词，开始判断，然后记录其中的可缩减单词。但是，此中方法就需要完全遍历所有单词，才可找到最长的"可缩减单词"

另一种方式，就是从长度长的单词开始判断，因为，我么需要找出的是最长的"可缩减单词"，所以，如果我们从最长的开始找，就不需要遍历所有的单词，程序中，出现的第一个可缩减单词，便是最长的可缩减单词。

（2） 程序逻辑

第一种方法，我们可以采用列表来存储所有单词，然后按照单词长度为列表排序，遍历所有单词，判读，如果为可缩减单词，则将其加入到一个"可缩减单词"组成的集合中。

```
将单词存入 Wordslist 中
Cutable_words = {'I', 'a'}#将可缩减单词存入集合中
Wordslist.sort(key=len(word))#按照单词长度排序
for word in wordslist:
#如果该单词的子单词为可缩减单词，那么它也是可缩减单词
#由于是从短到长遍历，所以，最后出现的可缩减单词便为最长
    if word 的子单词在 Cutable_words 中：
        Cutable_words.appen(word)
        Maxword = word
```

第二种方法，我们可以采用字典方式存储单词，字典的值是相同长度的单词所组成的集合，字典的键位单词的长度，此种，存储的所有单词都应该含有字母'i'或者'a'。然后，我们从最长的单词开始判断，是否为可缩减单词。

```
    将单词存入 word_dic 字典中
    #从最长的单词开始遍历
    for i in range(maxlen, minlen-1, -1):
        for word in word_dic[i]:
            调用函数判断是否可缩减
            if cutable(word):
                #输出单词，结束运行
                print(word)
                return
```

```
编写 cut(函数):
#利用全局变量 flag，判断是否为可缩减单词
#如果 flag=True，则让 cutable 函数返回 True
def cut(word):
    if word 的子单词 subword 是在 word_dic[len(subword)]:
        cutable(subword)#递归
    if subword == 'I' or subword =='a':
        flag = True
```

五、 测试用例

实验任务 1:

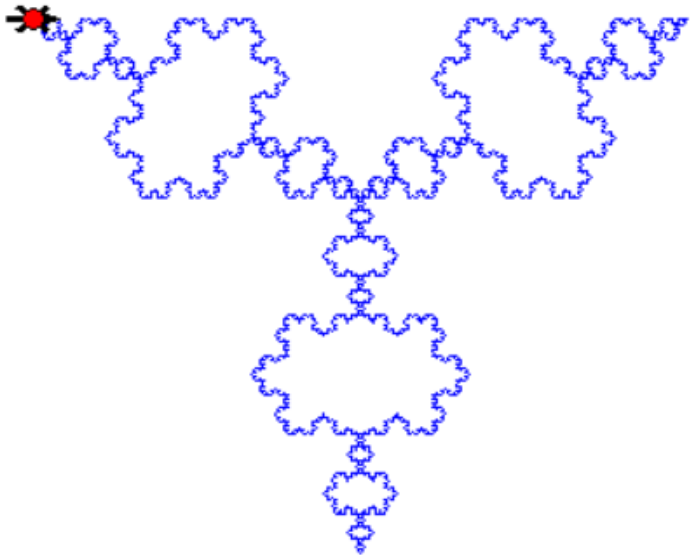
运行程序，得到以下输出

```
D:\codefile\python_code\E2>python pi.py
math.pi =  3.141592653589793
my__.pi =  3.141592653589794
2
0.0
```

第一、二行分别代表 math 中的 pi 值与我通过计算得到的 pi 值，可以看见，两者差别很小。第 3 行代表的是累加次数，这里，累加次数为 2 + 1 = 3 次。第 4 行代表运行时间，很快！相比较于没有优化的代码（pi1.py）（运行时间为 0.001s），速度提升了，但是由于运行时间快，所以再次不做过多比较。

实验任务 2:

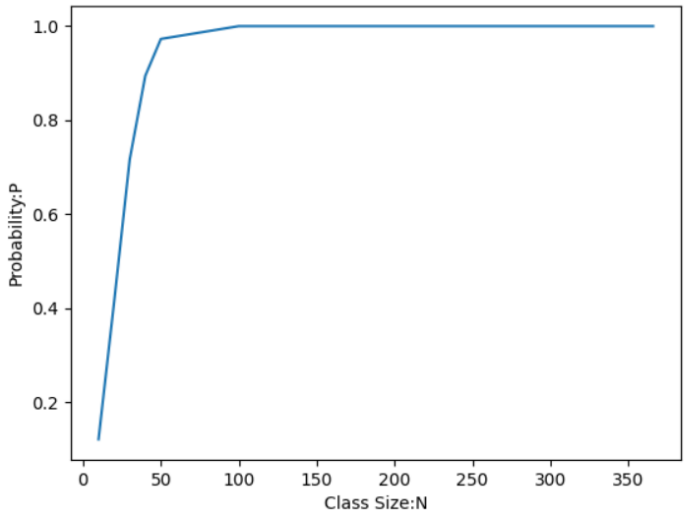
运行科赫反雪花.py 文件，可得到以下图案:



试验任务 3:

这题是模拟经典的生日问题。我们可以通过分析，发现班级数并没有实际的意义，只需保证足够大即可，可以理解为样本数足够大，当 M 足够大，N 相同的情况下，M 如何变化，对 P 的影响较小，我们可以通过数据验证，在此就省略。

接下来分析 N 于 P 的关系，我们可以接用 python 中的 Matplotlib 库画图来直观展示。



由图，我们不难发现，当班级人数 N>50 时，P 的概率就很接近 1 了，在 N>100 时，P 几乎等于 1

试验任务 4:

运行第一版代码:

```
D:\codefile\python_code\E2>python cutable_word1.py
complecting
completing
competing
compting
comping
coping
oping
ping
pig
pi
i
pin
in
i
pi
i
0.2602715492248535
```

前面输出的是最长可缩减单词的缩减过程，最后一行输出的时间

运行第二版程序

```
D:\codefile\python_code\E2>python cutable_word2.py
complecting
completing
competing
compting
comping
coping
oping
ping
pig
pi
i
pin
in
i
pi
i
0.11964273452758789
```

最长可缩减单词的缩减过程相同，运行时间缩短了一半有余。

分析两版代码，可以发现，第一版代码（cutable_word1.py）遍历了所有单词，最终找到最长可缩减单词，而第二版只遍历了一部分，所以，时间会快上许多。

六、 收获与体会

第二次 python 上机试验，相比于第一次，就显得轻松许多了。当然，在代码编写过程中，也会有出现各种各样的问题。在写递归函数的时候，终止条件的选择，以及返回类型的选择，还有循环嵌套中变量“清 0”的位置，都需要反复琢磨。

在此次实验中，也深刻的体会到 python 的灵活性。函数的返回值是任意的，同一个函数，可以返回多种不同类型的返回值，返回 bool 型，int，string 都可，当然，也可返回 None，这给我编写函数带来了极大的方便。当然，这灵活性也有带来麻烦的时候，比如深浅拷贝！到底是深拷贝还是浅拷贝，我的数据偶尔会被意外得改变。

收获也是颇多，我在实验中，逐渐熟悉各种数据结构的操作，了解各种他们的特性、优缺点。现在，能过通过分析问题，选择合适的数据结构，与其对应的方法来解决问题。