上海大学

SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号　第十组

实 验 序 号　**2**

学　　号　**22121630**

姓　　名　汪江豪

日　　期　**2025 年 3 月 20 日**

Python 计算实验二

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 windows11

2. vscode 编辑器

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)，并且和 math.pi 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、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. 实验一要求使用公式计算 pi 精确到小数点后 15 位。首先需要引入 decimal 库。该库用于解决 python 内置浮点数在高精度计算中出现的舍入误差问题。

   具体可以通过设置 getcontext().prec 来控制有效数字的位数，减小计算 pi 值的误差。

   主要思想是：设置一个主循环，每次迭代 k，计算 pi 的近似值，并于 math 库中的 pi 值比较，如果误差小于 1e-15，则达到题目要求，跳出循环，输出结果。在这途中，用到了 math 库中的阶乘函数 factorial。

2. Koch.py 文件，通过调整不同参数，实现了绘制不同 koch 曲线的功能。本质利用递归，细分和适当角度旋转，每次将线段分解为更小的曲线片段，最终生成具有分形特性的 koch 曲线。其主要绘制逻辑如下：

   如果线段长度小于 5，不再细分，直接绘制一条短直线。

   第一部分：直接绘制长度为 m 的直线；

   第二部分：旋转 60 度后绘制长度为 m 的直线；

   第三部分：旋转-120 度绘制长度为 3m 的直线；

   第四部分：旋转 60 度绘制长度为 m 的直线；

   最后一部分：继续绘制长度为 m 的直线。

3. 实验三探讨的是著名的生日悖论问题。

   主要问题描述是：在一个班级中，当班级人数达到一定数量时，至少两个人生日相同的概率会急剧增加，远低于直观上认为“需要很多人才能碰撞“的数量。

   同时，我也计算了理论概率统计，也就是：在 N 个学生中，至少两人生日相同的概率。

```
# 理论概率统计
if N <= 365:
    theoretical_p = 1.0
    for i in range(1, N):
        theoretical_p *= (365 - i) / 365
    theoretical_p = 1 - theoretical_p
    print(f"\n理论概率:{theoretical_p:.4f} ({theoretical_p*100:.2f}%)")
```

如果人数大于 365，那么必定有两人生日同一天，概率为 100%；如果人数小于等于 365，计算两两生日都不同的概率，利用乘法原理，然后 1-p 得出生日相同概率。

4. 实验四要求找出 words.txt 中最长的可缩减单词。是一个典型的递归问题。核心思想是通过递归和记忆化搜索来判断一个单词是否可缩减，然后从所有单词中找到最长可缩减单词。

   题目要求了最终缩减到单字母 a 或 i，如果某个单词缩减到 a 或者 i，即认为该单词是可缩减的。对于如何判断一个词是否为英语单词。这里采取的方式是：判断该词是否在 words.txt 文件中。因为 words.txt 文件中包含了约 11 万个单词，基本上涵盖了日常使用中的所有单词。

   可以在查找前先将所有单词按照长度降序排列，这样能最快找到最长的可缩减单词。

```
sorted_words = sorted(word_set, key=lambda x: len(x), reverse=True)
```

实测未通过排序操作，寻找耗时 1 秒左右，降序排列后寻找耗时 90ms 左右，提升了十倍以上的速度。

主要递归过程是：对于当前单词，删除其中的任意一个字母后，生成所有子单词，通过列表推导式和切片操作实现：

```
# 生成所有可能的子单词(删除一个字母)
    children = [word[:i] + word[i+1:] for i in range(len(word))]
```

然后循环判单每个子单词是否为可缩减单词，如果是，记录到缓存字典 memo 中，该字典的值是一个元组，记录了布尔型判断结果和缩减路径。在求取最长可缩减单词时，需要将’a’和’i’单词添加到 word_set 单词集合中，否则会出现找不到可缩减单词的情况。最后通过主循环，边判断每个单词是否可缩减，边更新最长可缩减单词即可。

```
# 检查每个子单词是否可缩减
    for child in children:
        if child in word_set:
            reducible, path = is_reducible(child, word_set, memo) # 递归入口
            if reducible:
                # 缓存结果
                memo[word] = (True, [word] + path)
                print(f'找到可缩减单词：{word} (长度：{len(word)})')
                return memo[word]
```

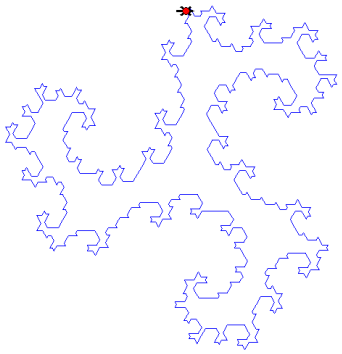
该算法时间复杂度为$O(N*2^k)$,N 表示单词总数量，k 表示每个单词的平均长度，但通过缓存 memo 方法，进行了大量剪枝操作，实现了效率的大幅提高。同时通过对 words.txt 文本中的单词进行降序排序，优先找长度最大的单词，效率提升了近十倍。

五、测试用例

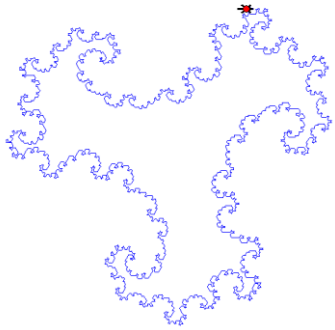
1.整个函数执行时间 0.47ms 左右，使计算机结果达到了小数点后 15 位。可以看到，math 库对 pi 的精确度高达小数点后几十位。

```
函数calculate_pi的运行时间为：0.473500000ms
计算得到的pi值：3.1415926535897938781
math.pi的值：3.14159265358979311599796346854418516159057617187 5
两者的差值：7.6210203653145581484E-16
```

2.如图是 n = 20 时绘制的 koch 曲线：



如图是 n = 5 时绘制的 koch 曲线：



3. 当每个班级人数较少时：取班级数 5000，班级人数 23，概率结果如下，约为 50%：

```
===生日相同概率分析===
请输入班级数量M(M>=1000)：5000
请输入每个班级的学生数量N(N>=2)：23

实验结果：
班级数量M：5000
每个班级的学生数量N：23
存在相同生日的班级数：2499
生日相同的概率：0.4998 (49.98%)
```

当每个班级人数略微增加到 40 时，生日相同的概率显著增加：

```
===生日相同概率分析===
请输入班级数量M(M>=1000)：5000
请输入每个班级的学生数量N(N>=2)：40

实验结果：
班级数量M：5000
每个班级的学生数量N：40
存在相同生日的班级数：4482
生日相同的概率：0.8964 (89.64%)

理论概率:0.8912 (89.12%)
```

当每个班级人数增加到 57 时，概率已经接近为 1：

```
===生日相同概率分析===
请输入班级数量M(M>=1000)：5000
请输入每个班级的学生数量N(N>=2)：57

实验结果：
班级数量M：5000
每个班级的学生数量N：57
存在相同生日的班级数：4945
生日相同的概率：0.9890 (98.90%)
```

4. 结果：

未排序运行耗时：

```
函数find_longest_reducible的运行时间为：1086.744400000ms
```

降序排序后运行耗时：

```
函数find_longest_reducible的运行时间为：93.190100000ms
```

再次运行，算法耗时 80ms 左右，找到了最长可缩减单词 complecting，并给出了缩减路径：

```
已加载113809个单词
找到可缩减单词：pi（长度：2）
找到可缩减单词：pig（长度：3）
找到可缩减单词：ping（长度：4）
找到可缩减单词：oping（长度：5）
找到可缩减单词：coping（长度：6）
找到可缩减单词：comping（长度：7）
找到可缩减单词：compting（长度：8）
找到可缩减单词：competing（长度：9）
找到可缩减单词：completing（长度：10）
找到可缩减单词：complecting（长度：11）
找到新的最长可缩减单词：complecting（长度：11）
函数find_longest_reducible的运行时间为：82.805500000ms
最长可缩减单词为：complecting（长度： 11）
缩减路径：complecting->completing->competing->compting->comping->coping->oping->ping->pig->pi->i
```

六、收获与体会

本次实验中，我更加体会到了 pyton 的跨学科应用的强大之处。可以用于数学分析，文本处理，精确度估算，统计学理论分析等，无处不体现着 pyton 应用场景的广泛。

我了解到了通过简单的递归规则，能生成无限复杂图案，修改参数会产生截然不同的分形形态，体会到了简单规则产生复杂行为的计算之美。蒙特卡洛模拟验证了反直觉的概率现象（23 人中即有 50%生日重复），认识到大数定律在实际中的表现形式。了解了记忆化技术可将指数级复杂度优化为多项式级别，体会到子问题重用对算法效率的关键影响，学习了如何通过剪枝策略提前终止无效搜索。

同时，我了解到，空字符串处理，和边界条件常常是 bug 的源头。优化程序设计需要数学思维、工程实践、调试耐心的三维结合。每个实验都像是一个微型的科研项目，从问题理解、方案设计、实现调试到结果分析的全流程训练，对培养计算思维和系统化解决问题能力有着不可替代的价值。