上海大学

SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号　第 8 组

实 验 序 号　2

学　　号　20121034

姓　　名　胡才郁

日　　期　2022 年 4 月 5 日

实验二 PYTHON 数据结构与流程控制

1 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

2 实验环境

Python 3.9.7 [MSC v.1916 64 bit (AMD64)]
PyCharm 2021.3.2

3 实验内容

3.1 逼近圆周率

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

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

3.2 Koch 曲线

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。

3.3 生日相同

（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。

3.4 最长可缩减单词

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’）。

4 设计与实现

4.1 逼近圆周率

此题比较简单，如上图所示，以为该程序的核心代码。

```
while True:
    molecular = math.factorial(4 * k) * (1103 + 26390 * k)  # 分子
    denominator = math.pow(396, 4 * k) * math.pow(math.factorial(k), 4)  # 分母
    step = factor * molecular / denominator  # 每一次迭代大小
    base += step
    k += 1
    if step < epsilon:
        pi = 1 / base
        return k, pi
```

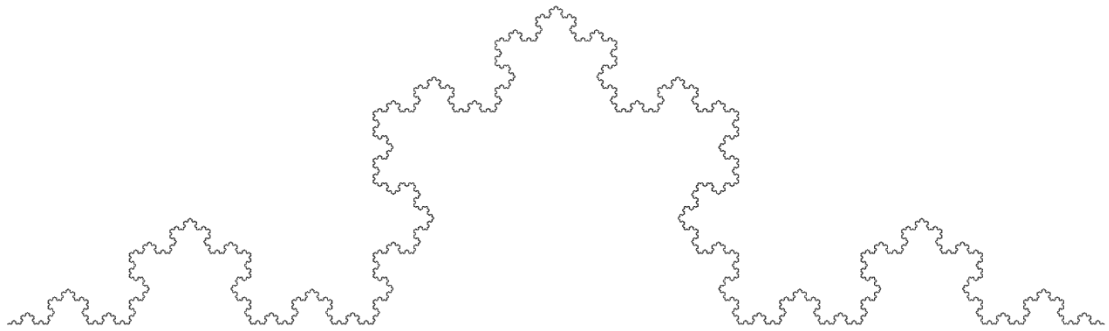
首先调用 math 库，然后再通过调用内置 factorial 函数计算阶乘。

其次余部分直接对应原始公式转换即可，代码中将公式分为分子、分母两个部分。这里测试了小数点后 15 位的情况，当满足小于规定的极小值时，结束循环。

4.2 Koch 曲线

Koch 雪花可以在一系列阶段中迭代构建。第一阶段是一个等边三角形，每个连续的阶段都是通过在前一阶段的每一侧增加向外弯曲形成的，从而形成更小的等边三角形。雪花构造的连续阶段的周长无限增加，因此，雪花包围了一个有限的区域，但有一个无限的周长。

而利用迭代算法与作图工具，通过修改角度与迭代边界条件，可以绘制出不同种类的 Koch 曲线。



此题核心在于递归函数的处理，对于此递归函数而言，递归的终止条件为 order 为 0，在满足递归终止条件时，同时为画笔重新上色。当在递归调用栈中执行完某一函数后，控制画笔转向，即可完成水平的部分曲线。(下图左)

而当完成整个一次调用后，控制画笔转向，此处角度为 360 / 5 = 72 度。将 5 段曲线拼接，即可获得完整的曲线。

```
def koch(t: Turtle, order, size):
    if order == 0:
        set_color(t)
        t.fd(size)
    else:
        koch(t, order - 1, size / 3)
        t.rt(85)
        koch(t, order - 1, size / 3)
        t.lt(170)
        koch(t, order - 1, size / 3)
        t.rt(85)
        koch(t, order - 1, size / 3)
```

```
for i in range(5):
    koch(bob, 5, 1000)
    bob.rt(72)
```

4.3 生日相同

从题目的实现层面来看:

这里假设了一年只有固定的 365 天，简化了运算速度。这里测试使用的程序运行用时 5 秒（计算了当班级数 m 取 1000 时，N 取 1-365 的所有情况，并包含绘图的时间）

使用随机数，生成 1-365 中的一个数字，代表生日的一天，并且在生成时同时将这一天添加到集合之中，利用集合的 Key 值唯一的特点，集合之中无重复元素自动去重，再将集合的长度与班级人数对比即可，如果数量相同，则证明班级内无生日相同的情况，反之，有生日相同的情况。

此外，最后调用 matplot 下的内置函数 plt.plot，可以绘制出简单的图形，这里我们假定了班级数固定为 1000，M 的取值与 N，P 没有明显的关系，M 越大，其结果越趋近于真实的值。核心代码如下:

```
for i in range(1, 101):
    for j in range(m):
        # 集合推导式生成日期集合，并且与人数比较
        # 利用了集合去重的特点
        birthday_set = {random.randint(1, 365) for j in range(i)}
        if len(birthday_set) < i:
            q += 1
        rate = q / m
    q = 0
    rate_list.append(rate)
```

从问题实际意义层面来看：

这是著名的生日悖论问题，此类问题对于密码学的研究十分重要。这里会使用到一种签名的技术，进行校验是否被篡改了。一种常见的签名技术是使用 RSA 算法。RSA 算法是建立在整数因式分解问题上的：两个大素数相乘在计算上是非常简单的；但是反过来，知道其乘积结果，要想对乘积结果进行因式分解确是非常困难的。需要知道在实际上签名的时候，不是直接对原始信息进行 RSA 签名，而是先对原始信息计算出一串很短的固定的哈希值，然后对该哈希值进行签名。对哈希值进行签名有很多好处，比如由于哈希值非常短，所以比起对原始信息签名来说快很多，签名的结果也相对变短了，并且更加安全。生日悖论最重要的结论就是:找到一个冲突所需要哈希的消息的数目大概等于可能输出值个数的平方根。

4.4 最长可缩减单词

通过设置成功词集与失败词集，分别存储可缩减单词与不可缩减单词，由于可缩减单词的子单词一定是可缩减单词，而非可缩减单词的子单词一定不是可缩减单词。因此，可以想到此题的算法思想使用递归处理。而对于此类 dfs 问题而言，重要的是性质函数 find，此处 find 内的判断条件较为复杂，满足条件的要求可以具体分为下面几种情况：

此单词已经为可缩减单词。

此单词不是已知的可缩减单词，且此单词的子单词之中有可缩减单词。

其中，第二种情况的“此单词的子单词之中有可缩减单词”的条件，可以用递归进行判断。

将所有的可缩减单词存入集合之后，依次比较，取得最长单词即可。

find 函数核心代码如下：

```
if len(w) > 1:
    for i in range(len(w)):
        nw = w[:i] + w[i + 1:]  # 第i个字母左边与右边拼起来成为一个子单词nw
        # 条件：nw已经是可缩减单词了 或 (nw是个单词 且 nw不是已知的非可缩减单词 且 nw的子单词里有可缩减单词)
        if nw in succeed_words_set or \
                (nw in words_set and nw not in failed_words_set and find(nw) is not None):
            break  # 满足条件就是可缩减单词
    else:  # 循环完毕break没有执行，进入for-else的else
        failed_words_set.add(w)
        return None
succeed_words_set.add(w)
max_len = max(max_len, len(w))  # 更新已找到最长的可缩减单词长度
return w
```

5 测试用例

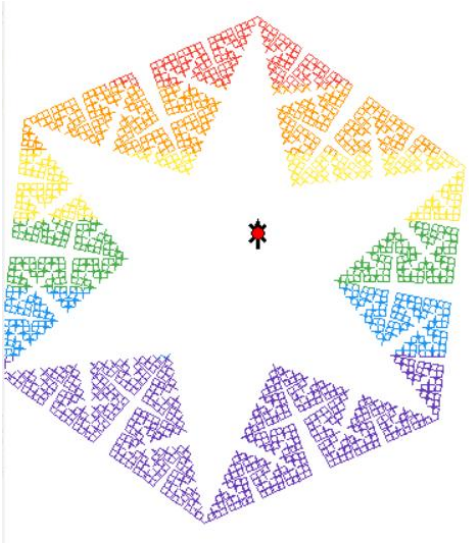
5.1 逼近圆周率

根据实验输出，在内部进行循环时，发现当循环第 3 次时，每一次迭代的大小就已经满足了小于 1e-15 的界限，并且此处使用公式逼近的 Pi 值的精度十分接近 Pi 的真实值。具体实验结果如下图所示：

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
k的值为3
pi的值为3.141592653589793
```

5.2 Koch曲线

此处选取了 Koch 曲线的变种之一切萨罗曲线进行绘制，并且当迭代一定次数之或进行转向操作，通过 for 流程控制，画出了六芒星内部与外部的轮廓。

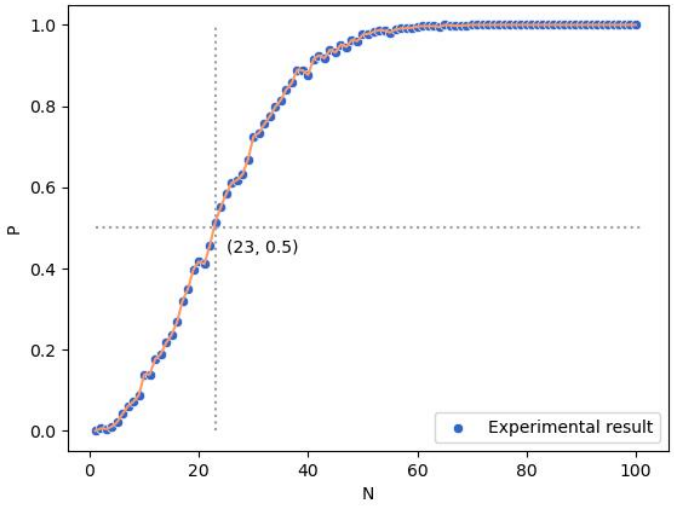


在 Koch 曲线的基础之上，编写了 set_color 函数对于画笔坐标的纵坐标(Y)进行判断，对于此六边形而言，当画笔位置位于 6 个不同的区间时，更换不同的画笔颜色。

5.3 生日相同

对于本题,要求的数据量较大,这里使用了 numpy 库来进行验证,并通过调用 matplotlib 库以及 seaborn 库进行数据可视化。NumPy(Numerical Python) 是 Python 语言的一个扩展程序库，支持大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。Matplotlib 是 Python 的绘图库。 它可与 NumPy 一起使用，提供了一种有效的 MatLab 开源替代方案，而 Seaborn 是基于 Matplotlib 的拓展，更好的集成了功能。

观察图像可知，当 n=23 时，计算结果与图像拟合结果相似，发生的概率大约是 0.5。



从上述图中，可以非常直观地看到，起初 P 随着 M 的增加缓慢上升，约在 24 人左右，概率接近 50%，之后随着 N 的继续上升，概率在越来越趋近于 1 的时候增长速度趋于 0，这里可以看出，在大于 50 个人以上的班级，概率已经非常接近于 100%。

此外，对于传统的列表类型，经过测试后可以发现，其速度相较于 numpy 库的运行速度慢 10 倍左右，这里从侧面也可以体现出该方法的效率之高。

5.4 最长可缩减单词

找到了最长可缩减单词 complecting,它的长度为 11，并且将它每一步的缩减情况完整的打印出来，且由于数据量只有 11 万，递归算法也可以较快的完成问题，算法效率较好。

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
i → pi → pig → ping → oping → coping → comping → compting → competing → completing → complecting (11)
1.8071472644805908 s
```

6 收获与体会

本次实验中，我学习到了递归算法与流程控制相结合，可以解决很多问题。我也明白了 Pyhton 可以用作科学计算的工具，帮助解决绘图、计算、拟合等众多需求。

附录：

实验 1：

```
import math


def estimate_pi(epsilon=1e-15):
    factor = 2 * math.sqrt(2) / 9801
    base = 0.0
    k = 0
    while True:
        molecular = math.factorial(4 * k) * (1103 + 26390 * k)  # 分子
        denominator = math.pow(396, 4 * k) * math.pow(math.factorial(k), 4)  # 分母
        step = factor * molecular / denominator  # 每一次迭代大小
        base += step
        k += 1
        if step < epsilon:
            pi = 1 / base
            return k, pi


def main():
    epsilon = 1e-15
    k, pi = estimate_pi(epsilon)
    print("k 的值为{}".format(k))
    print("pi 的值为{}".format(pi))


if __name__ == '__main__':
    main()
```

实验 2：

```
from swampy.TurtleWorld import *


def set_color(t: Turtle):
    if t.get_y() > 106:
        t.set_pen_color('#F44336')
    elif t.get_y() > 53:
        t.set_pen_color('#FF9800')
    elif t.get_y() > 0:
        t.set_pen_color('#FFEB3B')
    elif t.get_y() > -53:
        t.set_pen_color('#4CAF50')
```

```
    elif t.get_y() > -106:
        t.set_pen_color('#2196F3')
    else:
        t.set_pen_color('#673AB7')


def koch(t: Turtle, order, size):
    if order == 0:
        set_color(t)
        t.fd(size)
    else:
        koch(t, order - 1, size / 3)
        t.rt(85)
        koch(t, order - 1, size / 3)
        t.lt(170)
        koch(t, order - 1, size / 3)
        t.rt(85)
        koch(t, order - 1, size / 3)


if __name__ == '__main__':
    world = TurtleWorld()
    bob = Turtle()
    bob.delay = 0

    bob.x = 0
    bob.y = 160
    bob.redraw()
    bob.rt(36)
    for i in range(5):
        koch(bob, 5, 1000)
        bob.rt(72)

    bob.y = -10
    bob.heading = 90
    bob.redraw()

    world.mainloop()
```

实验 3：

```
import random
import time
import seaborn as sns
import matplotlib.pyplot as plt
```

```
def get_rate():
    m = 100
    q = 0
    rate_list = []
    for i in range(1, 366):
        for j in range(m):
            birthday_set = {random.randint(1, 365) for j in range(i)}
            if len(birthday_set) < i:
                q += 1
            rate = q / m
        q = 0
        rate_list.append(rate)
    fig = sns.scatterplot(x=range(1, 366), y=rate_list, marker='x', color='#3F51B5', label='Experimental result')
    plt.xlabel('N')
    plt.ylabel('P')
    scatter_fig = fig.get_figure()
    scatter_fig.savefig('figures/m={}.png'.format(m))
    plt.show()
    print("图片已保存")


def main():
    start = time.time()
    get_rate()
    end = time.time()
    print("所用时间为{}".format(end - start))


if __name__ == '__main__':
    main()
```

实验 4：

```
import time


# 递归查找，判断子单词里有没有可缩减单词，返回单词本身为可缩减单词，返回 None 为非可缩减单词
def find(w: str):
    global max_len, words_set, succeed_words_set, failed_words_set
    if len(w) > 1:
        for i in range(len(w)):
```

```
                nw = w[:i] + w[i + 1:]  # 第 i 个字母左边与右边拼起来成为一个子单词 nw
                # 条件：nw 已经是可缩减单词了 或 (nw 是个单词 且 nw 不是已知的非可缩减单词 且 nw 的子单词里有可缩减单词)
                if nw in succeed_words_set or (nw in words_set and nw not in failed_words_set and find(nw) is not None):
                    break  # 满足条件就是可缩减单词
            else:  # 循环完毕 break 没有执行，进入 for-else 的 else
                failed_words_set.add(w)
                return None
        succeed_words_set.add(w)
        max_len = max(max_len, len(w))  # 更新已找到最长的可缩减单词长度
        return w


# 打印输出
def print_seq(w: str, *, root: bool = True):
    global words_set
    arrow = ' → '
    if len(w) > 1:
        for i in range(len(w)):
            nw = w[:i] + w[i + 1:]
            if nw in {'a', 'i'}:
                print(nw, end=arrow)
                break
            elif nw in words_set and print_seq(nw, root=False) is not None:
                break
        else:
            return None
    print(w, end=' ({})\n'.format(len(w)) if root else arrow)
    return w


if __name__ == '__main__':
    start = time.time()  # 记录开始时间

    # 制作词表
    with open('words.txt') as fp:
        words_list = [x.strip() for x in fp if 'a' in x or 'i' in x]  # 读取文件并初筛，不含'a'和'i'的一定不是可缩减单词

    words_set = set(words_list)  # 全部词集
    succeed_words_set = {'a', 'i'}  # 成功词集，可缩减单词的子单词一定是可缩减单词
    failed_words_set = set()  # 失败词集，非可缩减单词的子单词一定不是可缩减单词
    max_len = 1  # 记录已找到最长的可缩减单词长度
```

```
result = 'a'
for word in words_list:
    if len(word) > max_len:
        sub_result = find(word)
        if sub_result is not None:
            result = sub_result

end = time.time()    # 记录结束时间

print_seq(result)    # 打印输出序列
# print(result)
print(end - start, 's')
```