上海大学

SHANGHAI UNIVERSITY

Python 计算实验报告

组　　号　第 7 组

实 验 序 号　2

学　　号　21122782

姓　　名　陈诺

日　　期　2023 年 4 月 10 日

Python 计算实验报告

1 实验目的与要求

1.1 熟悉 Python 的流程控制

1.2 熟悉 Python 的数据结构

1.3 掌握 Python语言基本语法

2 实验环境

系统：MacOS 13.2.1 (22D68)
硬件：Apple M1，8G
Python版本：anaconda Python 3.9.6

3 实验内容

3.1 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。

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

3.2 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。

3.3 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。

3.4 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')。

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。

4 实验内容的设计与实现

4.1 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。

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

4.1.1 程序设计特点

将计算π的公式转化为两个函数，使得计算过程更加清晰直观。

4.1.2 函数测试:

```
def func1(k: int) -> float:
    return math.factorial(4 * k) * (1103 + 26390 * k) /
(pow(math.factorial(k), 4) * pow(396, 4 * k))

def CountPi(k: int) -> float:
    return 9801 / (2 * math.sqrt(2) * sum([func1(i) for i in range(k + 1)]))
```

4.1.3 函数/代码分析:

`func1(k: int) -> float`函数解释:

实现函数:

$$func1(k)=\frac{(4k)!\,(1103+26390k)}{k!^4\left(396^{4k}\right)}$$

使代码更具可读性。

`CountPi(k: int) -> float`函数解释:

通过以下公式实现π的计算:

$$CountPi(k_n)=\frac{9801}{2\sqrt{2}\sum_{k=0}^{k_n}func1(k)}$$

当$k\to\infty$时，根据题目给出的公式，有:

$$\pi=\frac{9801}{2\sqrt{2}\sum_{k=0}^{\infty}func1(k)}$$

运行如下测试函数:

```
countpi = CountPi(k)
print(f'pi counted: {countpi}')
print(f'math.pi: {math.pi}')
print(f"误差: {abs((countpi - math.pi)) / math.pi}%")
```

有输出结果：

```
pi counted: 3.141592653589793
math.pi: 3.141592653589793
误差： 0.0%
```

4.2 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。

4.2.1 程序设计特点：

修改了样例代码中的`koch(t, n)`函数与`snowflack(t, n)`函数，将参数分别扩充为`koch(t, n, degree=60, reversed=False)`与`snowflake(t, n, sides=3, degree=60, reversed=False)`，加强分型图的可定义性。

4.2.2 源代码片段展示：

```
def koch(t, n, degree=60, reversed=False):
    if n < 4:
        fd(t, n)
        return
    rev = 1
    if reversed == True:
        rev = -1
    m = n/3.0
    koch(t, m, degree, reversed)
    lt(t, rev * degree)
    koch(t, m, degree, reversed)
    rt(t, 2 * rev * degree)
    koch(t, m, degree, reversed)
    lt(t, rev * degree)
    koch(t, m, degree, reversed)


def snowflake(t, n, sides=3, degree=60, reversed=False):
    for i in range(sides):
        koch(t, n, degree, reversed)
        rt(t, 360 / sides)
```

4.2.3 函数/代码分析：

以如下参数调用`snowflake()`函数：

```
snowflake(bob, 1200, 3, 85, True)
```

有以下结果：

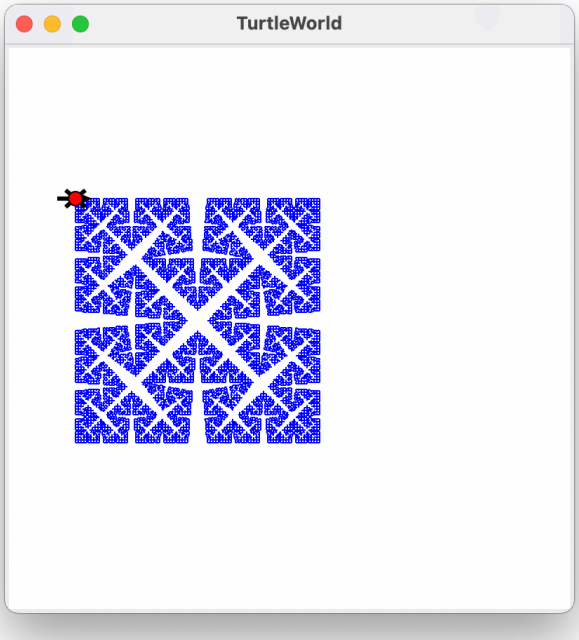

图1 Cesàro fractal (85°)

4.3 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。

4.3.1 程序设计特点：

随机生成M组学生的生日($M \geq 1000$)，每组有N个学生，生日以一年(365天)中的第n天表示($1 \leq n \leq 365$)；使用`tuple`类型作为存储结构，加快计算速度；使用`set`判断同一班级是否有同一天出生的学生。

4.3.2 源代码片段展示：

```
def GetClassAndNum() ->tuple:
    string = input("请输入班级数M与同学数N，用空格间隔")
    ans = re.findall(r"\b\d+\b", string)
    if len(ans) == 2:
        return (int(ans[0]), int(ans[1]))
    else:
        print("输入格式有误...")
        return GetClassAndNum()

def PofSameBrithdaysExistClass(student_class_num: int, student_num: int) ->
float:
    student_birthdays = tuple(tuple(random.randint(1, 366) for i in range(
```

```
        student_num)) for j in range(student_class_num))
    return CountSameBirthdaysExistClass(student_birthdays) /
student_class_num

def CountSameBirthdaysExistClass(student_birthdays: tuple) -> int:
    ans = 0
    for classes in student_birthdays:
        if len(classes) > len(set(classes)):
            ans += 1
    return ans
```

鉴于题目要求$M \geq 1000$，而`PofSameBrithdaysExistClass(1000, 100)`的运行时间已达到50s，因此实际使用`numpy`库的`np.random.randint()`函数生成数据，可大约加速十倍。

```
import numpy as np

def PofSameBrithdaysExistClassWithNp(student_class_num: int, student_num:
int) -> float:
    student_birthdays = np.random.randint(
        1, 366, (student_class_num, student_num))
    return CountSameBirthdaysExistClassWithNp(student_birthdays) /
student_class_num

def CountSameBirthdaysExistClassWithNp(student_birthdays: tuple) -> int:
    ans = 0
    for classes in student_birthdays:
        if len(classes) > len(np.unique(classes)):
            ans += 1
    return ans
```

为了比较生成数据所得的概率的准确性，即分析M，N和P之间的关系，本题分别生成了$M = 1000, 5000, 10000$时$N \in [1, 365]$的概率估计P，同时设计函数`SameBirthday(student_num: int) -> float`，生成实际概率进行对比。

```
def SameBirthday(student_num: int) -> float:
    all_symple = 365 ** student_num
    no_same = 1
    for i in range(1, student_num + 1):
        no_same = no_same * (365 - i + 1)
    return 1 - no_same / all_symple

student_class_num = 5000
y_labels_5000 = [PofSameBrithdaysExistClassWithNp(
    student_class_num, student_num) for student_num in range(1, 366)]
```

```
11
12  student_class_num = 10000
13  y_labels_10000 = [PofSameBrithdaysExistClassWithNp(
14      student_class_num, student_num) for student_num in range(1, 366)]
15
16  y_labels_count = [SameBirthday(student_num) for student_num in range(1,
    366)]
```

为了比较不同M对于P的影响，使用`pandas`的`Series`计算三组数据相对理论值的协方差。

```
1   from pandas import Series
2
3   def calc_corr(a, b):
4       s1 = Series(a)
5       s2 = Series(b)
6       return s1.corr(s2)
7
8   Corr_1000 = calc_corr(y_labels_1000, y_labels_count)
9   Corr_5000 = calc_corr(y_labels_5000, y_labels_count)
10  Corr_10000 = calc_corr(y_labels_10000, y_labels_count)
```

最后使用`matplotlib.pyplot`绘图进行可视化比较。

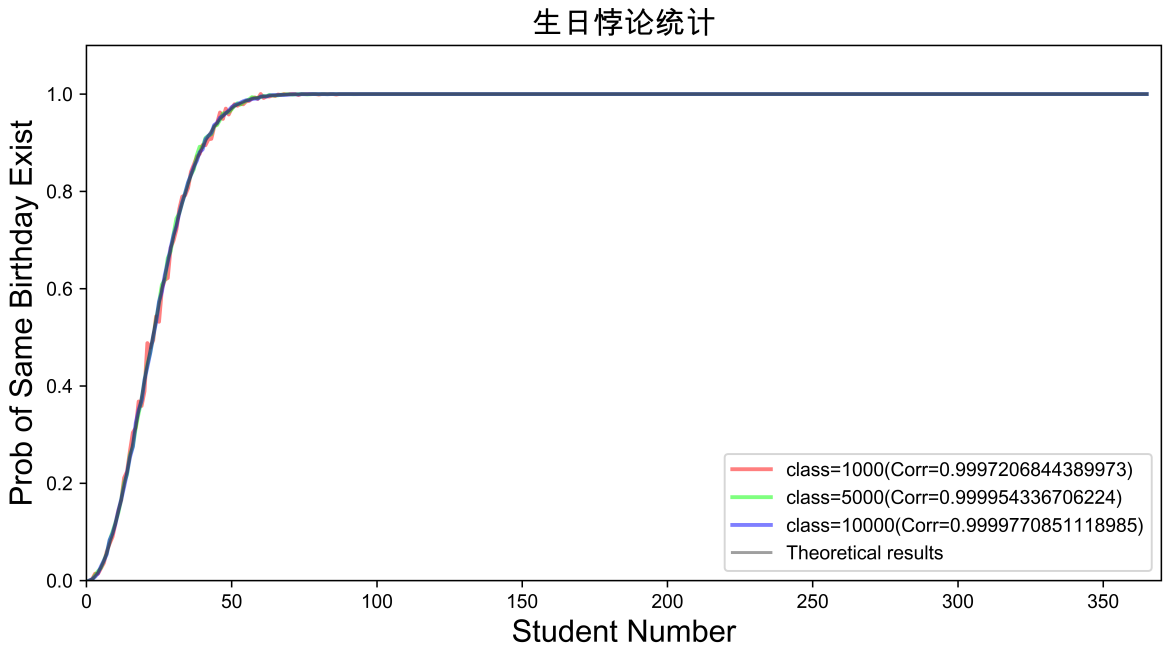

图2 生日悖论统计

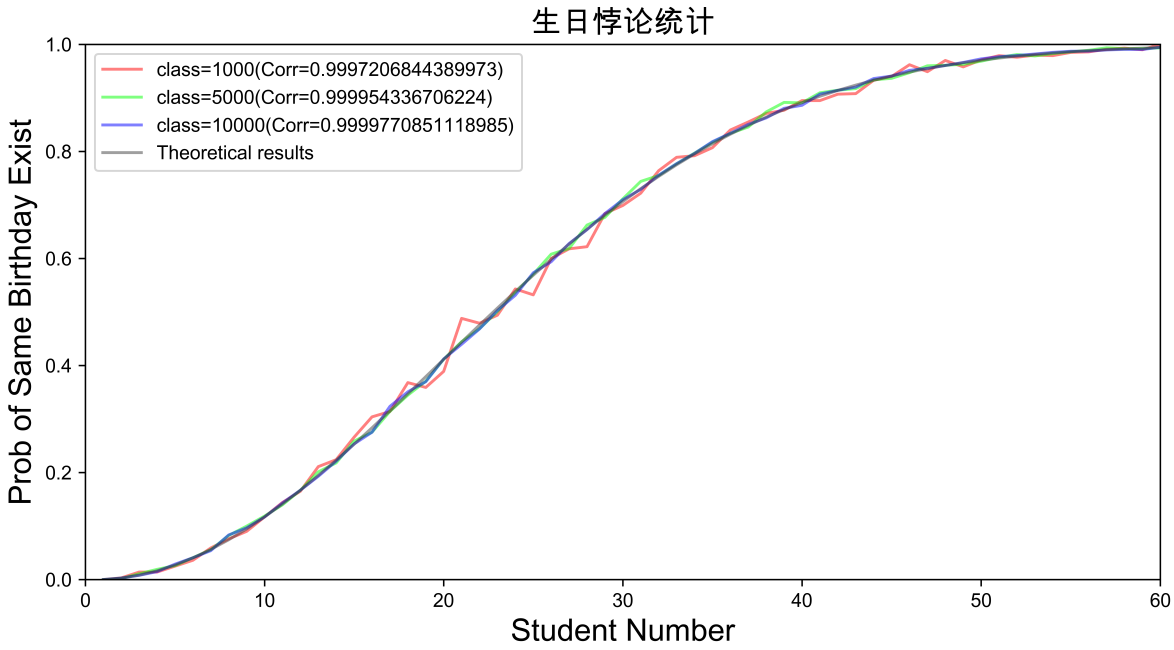

图3 生日悖论统计(放大)

4.4 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')。

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。

4.4.1 程序设计特点：

通过递归求解，判断单词删除某一单词后的剩余部分是否在单词表中，如果存在于单词表则将其添加至`reducable_words`以剪枝，简化程序运行。运行结束后，`reducable_words`即为单词表中所有的可缩减单词。

4.4.2 源代码片段展示：

```
reducable_words = set()

def ReducableWord(word: str, alphabet: set[str]):
    if word == 'i' or word == 'a' or word in reducable_words:
        return True
    for i in range(len(word)):
        reduced_word = word[:i] + word[i + 1:]
        if reduced_word == 'i' or reduced_word == 'a' or reduced_word in
alphabet:
            if ReducableWord(reduced_word, alphabet):
                return True
    return False
```

4.4.3 函数/代码分析：

`word[:i] + word[i + 1:]`：将单词`word`的第`i`个字母删除。

`ReducableWord(word: str, alphabet: set[str])` 函数分析：

对于长度为n的单词`word`，逐个删除第`i`个字符，生成长度为n - 1的`reduced_word`，若其等于`'a'`或`'i'`或存在于`alphabet`中，则继续递归并返回`True`；否则返回`False`退出递归。直至`word`等于`'a'`或`'i'`或存在于`reducable_words`中(该单词为可缩减单词，不必继续运行)，退出递归并返回`True`。

经计算得，word.txt中共有9767个可缩减单词。

5 测试用例：

无

6 收获与体会

通过本次实验，我对Python的流程控制，科学计算、库函数使用、递归函数设计有了一定的理解，通过练习与实验，我深刻的体会到即使是相同功能，程序也需要调用合适的运行库才能更好更快的完成计算。