**上海电力大学**

**实践课程报告**



学 院： 数理学院

专 业： 信息与计算科学

课程编号： 2812126.02 课程名称： Python程序设计

学生姓名： 刘伟涛 学号： 20222421班级：2022122

指导老师： 邓化宇

2023 年 11 月 19 日

成绩：

|  |
| --- |
| 教师评语： |

## 选题结果

[14, 20, 33, 35, 43, 48]

[1, 2, 12]

[1]

## 基础题

14. 编辑代码实现以下功能，随机生成 50 个介于[1,20]之间的整数，

然后统计每个整数出现频率。

import random

numbers = [random.randint(1, 20) for \_ in range(50)]

count = {}

for i in numbers:

if i in count:

count[i] += 1

else:

count[i] =

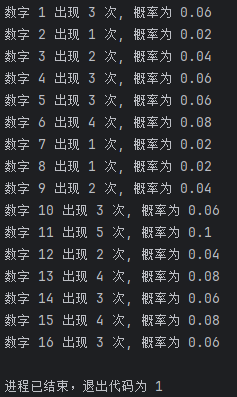
for num in range(1, 21):

c = count[num]

print(f"数字 {num} 出现 {c} 次, 概率为 {c / 50}")

解析：使用random模块生成一个包含50个1到20之间的随机整数的列表。然后使用一个字典count统计每个数字在列表中出现的次数。遍历1到20的数字，输出每个数字出现的次数和概率。

运行结果：



20大衍数列：中国古代文献中，曾记载过“大衍数列”，主要用于

解释中国传统文化中的太极衍生原理 它的前几项是：0、2、4、8、

12、18、24、32、40、50... 其规律是：偶数项，是序号平方再除 2，

奇数项，是序号平方减 1 再除 2。打印大衍数列的前 100 项。

count = {}

for i in range(1, 101):

if i % 2 == 0:

count[i] = int(i \* i / 2)

else:

count[i] = int((i \* i - 1) / 2)

for i in range(1, 101):

print(count[i]end=" ")

解析：建立一个字典来存储数字1到100之间的偶数的平方除以2的结果，以及奇数的平方减1再除以2的结果，并打印出所有的值。

输出结果：

0 2 4 8 12 18 24 32 40 50 60 72 84 98 112 128 144 162 180 200 220 242 264 288 312 338 364 392 420 450 480 512 544 578 612 648 684 722 760 800 840 882 924 968 1012 1058 1104 1152 1200 1250 1300 1352 1404 1458 1512 1568 1624 1682 1740 1800 1860 1922 1984 2048 2112 2178 2244 2312 2380 2450 2520 2592 2664 2738 2812 2888 2964 3042 3120 3200 3280 3362 3444 3528 3612 3698 3784 3872 3960 4050 4140 4232 4324 4418 4512 4608 4704 4802 4900 5000

33斐波那契数列又称兔子数列，因数学家列昂纳多·斐波那契以兔

子繁殖为例子引入。这个数列中的数据满足以下公式：

F(1)=1, F(2)=1, F(n)= F(n-1)+F(n-2), n>3,n 属于自然数

编写程序，利用递归实现根据用户输入的数字输出斐波那契数列

的功能。

num = int(input("查第几个？"))

if num == 1:

number=[0]

elif num == 2:

number=[0, 1]

else:

number = [0, 1]

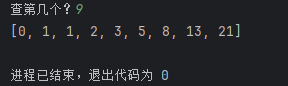
for i in range(2, num):

number.append(number[i-1] + number[i-2])

print(number)

解析：根据输入的值，生成一个列表来存储斐波那契数列，使用循环来计算并添加斐波那契数到列表中。然后打印前输入值个斐波那契列表。

输出结果：



35天天向上得力量计算。一年 365 天,以第 1 天得能力值为基数,

记为 1、0,当好好学习时能力值相比前一天提高 1‰,当没有学习时由

于遗忘等原因能力值相比前一天下降 1‰。每天努力与每天放任,一

年下来得能力值相差多少呢？

如果好好学习时能力值相比前一天提高 1%,当放任时相比前一天

下降 1%。效果相差多少呢？

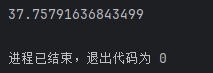
a = 1.01

b = 0.99

c = a\*\*365 - b\*\*365

print(c)

输出结果：



43编写程序，实现抓狐狸游戏。假设墙上有 5 个洞（编号分别为 0、

1、2、3、4），其中一个洞里有狐狸，人类玩家输入洞口编号，如果

洞里有狐狸就抓到了；如果洞里没有狐狸就第二天再来抓。但在第二天人类玩家来抓之前，狐狸会跳到隔壁的洞里。

import random

holes = [1, 2, 3, 4, 5]

fox = random.choice(holes)

while True:

print("墙上的洞口:", holes)

human\_choice = int(input("请输入洞口编号："))

if human\_choice == fox:

print("抓到了狐狸！")

break

else:

print("没有抓到狐狸，狐狸跳到了隔壁的洞里")

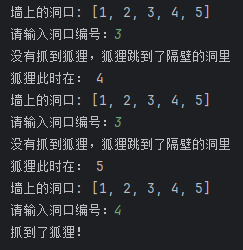
if 1 < fox < 5: fox = fox + random.choice([1, -1])

elif fox == 1: fox = fox+1

elif fox == 5: fox = fox-1

解析：随机选择一个位置，玩家输入猜测的数字，若猜中则游戏结束，否则根据狐狸的实际位置进行相应操作，并提示玩家"没有抓到狐狸，狐狸跳到了隔壁的洞里"，直到玩家猜中为止。

输出结果：

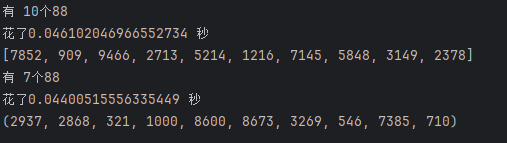


48分别用列表推导式和元组生成器生成1000个1-9999之间的随机

整数，输出它们前 10 个元素，再各自统计它们中含整数 88 的个数，

注意比较两者快慢。

import random  
import time  
# 列表推导式  
start = time.time()  
random\_list = [random.randint(1, 10000) for x in range(100000)]  
count = sum(num == 88 for num in random\_list)  
print(f"有 {count}个88")  
print(f"花了{time.time() - start} 秒")  
print(random\_list[:10])  
# 元组生成器  
start = time.time()  
random\_tup = tuple(random.randint(1, 10000) for t in range(100000))  
count = sum(num == 88 for num in tuple(random\_tup))  
print(f"有 {count}个88")  
print(f"花了{time.time() - start} 秒")  
print(random\_tup[:10])



## 提升题

1冒泡排序：这个算法的名字由来是因为越小的元素会经由交换慢

慢“浮”到数列的顶端(升序或降序排列)，就如同碳酸饮料中二氧化

碳的气泡最终会上浮到顶端一样，故名“冒泡排序”。

从前到后(即从下标较小的元素开始) 依次比较相邻元素的值，若发

现比后一个值大则交换位置，使值较大的元素逐渐从前移向后部。

假设有一个列表 [29, 12, 19, 37, 14] 想按升序排列，

第一轮：初始列表为 [29, 12, 19, 37, 14]

比较 29 > 12 交换位置：[12, 29, 19, 37, 14]

比较 29 > 19 交换位置：[12, 19, 29, 37, 14]

比较 29 > 37 不大于，不交换：列表同上不变

比较 37 > 14 交换位置：[12, 19, 29, 14, 37]

第二轮：列表继承上一轮为 [12, 19, 29, 14, 37]

比较 12 > 19 不大于，不交换：列表同上不变

比较 19 > 29 不大于，不交换：列表同上不变

比较 29 > 14 交换位置：[12, 19, 14, 29, 37]

以此类推。

def bubble\_sort(example):

n = len(example)

for ii in range(n):

for j in range(0, n - ii - 1):

if example[j] > example[j + 1]:

example[j], example[j + 1] = example[j + 1], example[j]

return example

arr = [29, 12, 19, 37, 14]

sorted\_arr = bubble\_sort(arr)

print("Sorted array is:")

for i in range(len(sorted\_arr)):

print("%d " % sorted\_arr[i],end="")

输出结果：



2选择排序：从未排序的序列中找到一个最小的元素，放到第一位，

再从剩余未排序的序列中找到最小的元素，放到第二位，依此类推，直到所有元素都排序完毕。

def selection\_sort(arr):

"""

选择排序

:param arr: list, 需要排序的列表

:return: list, 排序后的列表

"""

for i in range(len(arr)):

min\_index = i

for j in range(i+1, len(arr)):

if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

12编写程序，用一个变量存储组员成绩，要求有低于 60 的成绩，

利用列表推导式，筛选出低于 60 分的成绩。

scores = [89, 76, 50, 94, 45, 67, 59, 78, 82]

low60 = [score for score in scores if score < 60]

print(low60)



## 实验题（hog）

PROBLEM 1  
sumdice: int = 0  
flag: int = 0  
for i in range(num\_rolls):  
 k = dice()  
 sumdice += k  
 if k == 1:  
 flag = 1  
if flag == 1:  
 return 1  
else:  
 return sumdice

//首先定义了两个变量sumdice和flag，用于记录骰子的点数总和和是否出现1点。使用for循环进行多次模拟掷骰子的过程。在每次掷骰子后，将点数累加到sumdice，并且如果点数为1，则将flag设置为1。根据flag的值返回不同的结果，如果出现过1点则返回1，否则返回骰子点数的总和。

PROBLEM 2  
a: int = 0  
b: int = 0  
a = player\_score % 10  
if opponent\_score >= 10:  
 b = int(opponent\_score % 100 / 10)  
elif opponent\_score < 10:  
 b = 0  
if a == b:  
 return 1  
else:  
 return 3 \* abs(a - b)

//

通过计算玩家得分的个位数和对手得分的十位数，比较它们是否相等，如果相等返回1，否则返回3倍的它们之差的绝对值。

PROBLEM 3  
if num\_rolls == 0:  
 return boar\_brawl(player\_score, opponent\_score)  
else:  
 return roll\_dice(num\_rolls, dice)

//模拟掷骰子游戏。如果已经完成了所有掷骰子的次数，就决定两个玩家的得分；否则，模拟掷骰子并将结果作为参数传递给下一个掷骰子次数的递归调用。

PROBLEM 4  
factors = []  
for i in range(1, int(n \*\* 0.5) + 1):  
 if n % i == 0:  
 factors.append(i)  
 if n // i != i:  
 factors.append(n // i)  
return len(factors)

//

遍历1到n的平方根的整数，判断是否能整除n，如果能则将该整数和n除以该整数的结果添加到列表中，最后返回列表的长度即为因子的个数。

PROBLEM 4  
nums = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103,  
 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197]  
if num\_factors(score) == 3 or num\_factors(score) == 4:  
 for i in range(len(nums)):  
 if score <= nums[i]:  
 return nums[i]

//函数返回小于等于给定数score的质数，如果质数的因子数量等于3或4，则返回该质数；否则返回score本身。

PROBLEM 4  
score = player\_score + take\_turn(num\_rolls, player\_score, opponent\_score, dice)  
return sus\_points(score)

//计算游戏中得分的函数，先计算玩家得分，然后根据玩家得分和对手得分计算附加分数，最后返回总分数。

PROBLEM 5  
while (score0 < goal) and (score1 < goal):  
 # score0 = update(strategy0(score0,score1), score0, score1, dice)  
 # score1 = update(strategy1(score1,score0),score1,score0,dice)  
 if who == 0:  
 score0 = update(strategy0(score0, score1), score0, score1, dice)  
 who = 1  
 elif who == 1:  
 score1 = update(strategy1(score1, score0), score1, score0, dice)  
 who = 0  
 else:  
 print("出现错误！")  
return score0, score1

//根据策略更新分数并交换玩家，直到分数达到目标分数。

PROBLEM 6  
def strategy\_always\_roll(score, opponent\_score):  
 return n  
return strategy\_always\_roll

//函数strategy\_always\_roll总是返回一个值n，并且不接受任何参数。它返回的值可能因代码上下文中的赋值语句而异。函数最后返回strategy\_always\_roll函数本身

PROBLEM 7  
m = 0  
rem = strategy(0, 0)  
flag = 1  
while m < goal:  
 n = 0  
 while n < goal:  
 if rem != strategy(m, n):  
 flag = 0  
 break  
 n = n + 1  
 if flag == 0:  
 break  
 m = m + 1  
if flag == 1:  
 return True  
else:  
 return False

检查策略是否满足条件

PROBLEM 8  
def put\_average(\*args):  
 total = 0  
 for i in range(samples\_count):  
 total = total + original\_function(\*args)  
 return total / samples\_count  
return put\_average

//接受任意数量的参数，调用一个函数并返回其平均值。

PROBLEM 9  
a = 0  
b = 0  
for n in range(1, 11):  
 avg = make\_averaged(roll\_dice, samples\_count)(n, dice)  
 if avg > a:  
 b = n  
 a = avg  
return b

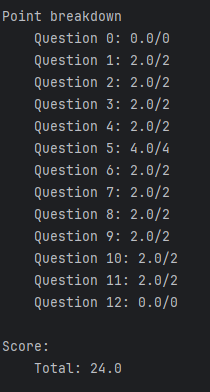
//这个函数通过调用make\_averaged函数计算平均值，并找到具有最高平均值的n。

PROBLEM 10  
score\_0 = 3 \* abs(score % 10 - (opponent\_score//10) % 10)  
if score\_0 >= threshold:  
 return 0  
else:  
 return num\_rolls

//这个函数根据得分和对手得分的个位数差异计算得分分值，如果分值大于等于给定的阈值，返回0，否则返回骰子的滚动次数。

PROBLEM 11  
score\_in = sus\_update(0, score, opponent\_score) - score  
if score\_in >= threshold:  
 return 0  
else:  
 return num\_rolls

//根据得分和对手得分计算得分变化量，根据阈值判断是否停止，停止则返回0，不停止返回还需要进行的投掷次数。



## 五、体会与收获

作为一个编程爱好者，我深知完成一个完整的Python大作业的重要。在课程中，我们需要掌握Python的基本语法、数据类型、函数、模块等知识，通过编写程序来解决实际问题。完成大作业的过程中，我获得了丰富的体会和收获。

完成大作业让我更加熟练掌握了Python的基本语法和常用数据类型。在编写程序的过程中，我需要考虑到变量、数据类型、运算符等基本概念，这有助于我更好地理解Python编程的本质。我也学会了如何使用Python内置的数据类型来处理文本、数字、布尔等数据。我还学会了如何使用列表、元组等结构体来组织代码和数据。这些知识对于我日后的编程工作有着很大的帮助。完成大作业让我更加熟练地掌握了Python中的函数和模块的使用。在编写程序的过程中，需要调用不同的函数来实现不同的功能，或者使用不同的模块来组织代码和数据。通过不断地调用函数和导入模块，我逐渐了解了它们的作用和使用方法。这有助于我在以后的工作中更加高效地编写代码和组织项目结构，让我更加深入地了解了Python中的面向对象编程思想的应用。在编写一些复杂的程序时，我需要使用面向对象编程的思想来进行代码的组织和管理。通过对类、对象的理解和使用，我更好地理解了Python中的面向对象编程思想的应用和方法论的意义。这对于我在以后的学习和工作有着很大的帮助作用！

“路漫漫其修远兮”，虽然这次大作业经历给我带来了很多收获,但我知道,这只是我在学习过程中的一小步,前进的路还很长很长……我将不断地学习,不断地实践,为自己的未来打下坚实的基础!