PS1_1 Flowchart

1. 第一步随机生成 a，b，c 三个数

```
import random
a=round(random.random()*100)
b=round(random.random()*100)
c=round(random.random()*100)
print(a,b,c)
```

2. 定义一个函数叫做 Print_values(a,b,c)，按照 flowchart 来比较 a，b，c 的大小。用 if 函数写代码。先写一个空的 list1，将每次输出的 xyz 打印到 list1 中，再计算 x+y-10z。

按照流程图的顺序，用 if 判断，当 a>b 错误，b>c 正确时，此时没有比较 ac 的大小，清空 list1，并输出请重新输入值。最后当 a = 10, b = 5, c =1 时，输出值为 5.

完整代码如下：

```
# 1.Flowchart
import random
a=round(random.random()*100)
b=round(random.random()*100)
c=round(random.random()*100)
print(a,b,c)

def Print_values(a, b, c):
    list1 = []
    if a > b:
        if b > c:
            list1 = [a, b, c]
            print(list1)
            print(a+b-10*c)
        elif a > c:
            list1 = [a, c, b]
            print(list1)
            print(a+c-10*b)
        else:
            list1 = [c, a, b]
            print(list1)
            print(c+a-10*b)
    elif b > c:
        print(list1.clear())
        print("Please re-enter the values!")
    else:
        list1 = [c, b, a]
        print(c,b,a)
        print(c+b-10*a)

Print_values(a, b, c)
Print_values(10, 5, 1)
```

```
24 49 86
86 49 24
-105
[10, 5, 1]
5
```

PS1_2 Continuous ceiling function

1.首先输入 list1=[1]，因为 F(1)=1，所以 1 先单独列出来。

2.listN 里面包含 N 个正整数

```
# listN have N positive numbers
listN = input("N = ")
m = int(listN)
```

3.用 for 循环和 range 函数，从 2 到 m+1 进行循环，m+1 是代表循环的最后一个到 m。然后计算 F(ceil(x/3))+2x，令 n=list1[ceil(i/3)-1]+2*i，-1 是因为 python 里面的[0]对应的是 list 里面的第一个数。

```
# for i in range from 2 to m+1
for i in range(2, m+1):
    n = list1[ceil(i/3)-1]+2*i
```

4.最后再将循环后得到的数字 append 到 list1 的末尾，print(list1)就得到最后输出的 list。

完整代码如下：

```
# 2
from math import ceil
list1=[1]
listN = input("N = ")
m = int(listN)

for i in range(2, m+1):
    n = list1[ceil(i/3)-1]+2*i
    list1.append(n)
print(list1)
```

```
N = 8
[1, 5, 7, 13, 15, 17, 21, 23]
```

PS1_3 Dice rolling

3.1 定义一个函数 Find_number_of_ways(n)，n 代表骰子的个数。

思路：利用递归，dp[n][x]代表扔 n 个骰子，和为 x 的排列组合数。那么 x 最小值是 n，x 最大值是 6n。

例如：

dp[1][1]=1, dp[1][2]=1, dp[1][3]=1,…, dp[1][6]=1，代表只有一个骰子时，x 值取 1-6 都只有一种可能性；

dp[2][2]=1, dp[2][3]=2,即扔第 n 个骰子时，与前一个骰子有关，即和可能是 x-1，x-2，x-3，x-4，x-5，x-6。即可得到下列关系式：

dp[n][x]=dp[n-1][x-1]+dp[n-1][x-2]+dp[n-1][x-3]+dp[n-1][x-4]+dp[n-1][x-5]+dp[n-1][x-6].

首先创建一个数组 dp，初始都是 0，长度是 6n，行数是 n。初始化只有一个骰子(即 n=1)时，x 取值从 1 到 6，此时 x 的可能性只有一种情况，即 dp[1][1]=1,dp[1][2]=1,…,dp[1][6]=1。

```
def Find_number_of_ways(n):
    dp=[[0]*(6*n+1) for m in range(n+1)]

    for x in range(1,7):
        dp[1][x]=1
```

当骰子的个数大于等于 2 时，从 2 开始取到 n，i 代表有几个骰子。x 代表点数之和，点数最小是 i，点数最大是 6*i。利用上述公式求每个 x 出现的可能性，此时已经填充完 dp。

```
    for i in range(2,n+1):
        for x in range(i,6*i+1):
            dp[i][x]=dp[i-1][x-1]+dp[i-1][x-2]+dp[i-1][x-3]+dp[i-1][x-4]+dp[i-1][x-5]+dp[i-1][x-6]
```

写一个 result[]空集，将所有 x 出现的可能性转化为一个 list，append 到 result 中，最后得到所有 x 出现的次数。

```
    result=[]
    for x in range(n,6*n+1):
        result.append(dp[n][x])
    return result
Find_number_of_ways(10)
```

3.2 求哪个 x 产生最大的路径数。

上述 3.1 的结果已经输出了所有 x 出现的次数，将 3.1 输出的结果赋到名为 Number_of_ways 这个 list 中，再求 list 的最大值就是产生此 x 的最多次数。

```
Number_of_ways = Find_number_of_ways(10)
print(max(Number_of_ways))
```

扔 10 个骰子即 n=10，最后输出结果是 4395456

完整代码如下：

```python
# 3. Dice rolling
# 3.1
def Find_number_of_ways(n):
    dp=[[0]*(6*n+1) for m in range(n+1)]

    for x in range(1,7):
        dp[1][x]=1

    for i in range(2,n+1):
        for x in range(i,6*i+1):
            dp[i][x]=dp[i-1][x-1]+dp[i-1][x-2]+dp[i-1][x-3]+dp[i-1][x-4]+dp[i-1][x-5]+dp[i-1][x-6]

    result=[]
    for x in range(n,6*n+1):
        result.append(dp[n][x])
    return result
Find_number_of_ways(10)

# 3.2
Number_of_ways = Find_number_of_ways(10)
print(max(Number_of_ways))
```

```
4395456
```

傅玮韬向我解释了如何将 3.1 定义好的函数运行后放到一个新的 list 里面，用 return 代替 print，感谢他！

做完题之后和石绍讨论了两个人的不同思路和解法。

参考资料：

https://blog.csdn.net/yue_luo_/article/details/95517498?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-95517498-blog-88960673.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-3-95517498-blog-88960673.pc_relevant_default&utm_relevant_index=4

PS1_4 Dynamnc programmnng

4.1 array 里面有 N 个元素，定义一个函数 Random_nnteger(N)，利用 np.random.randnnt(0,11,N)，表示从 0 到 10 里面随机选取 N 个数，代表 array 里面有 N 个元素。

完整代码如下：

```
# 4.1
import numpy as np
import math
def Random_integer(N):
    return np.random.randint(0, 11, N)
```

4.2 求所有子集的平均数的和

举例[1,2,3,4]，此时只有一个元素时，和 S=1/1+2/1+3/1+4/1；有两个元素时 [1,2][1,3][1,4][2,3][2,4][3,4]，和 S=(1+2)/2+(1+3)/2+(1+4)/2+(2+3)/2+(2+4)/2+(3+4)/2=3S/2;

有三个元素时[1,2,3][1,2,4][1,3,4]，和 S=(1+2+3)/3+(1+2+4)/3+(1+3+4)/3=3S/3;

有四个元素时[1,2,3,4]，和 S=1S/4.

得到通项公式：

$$S = \sum_{m=1}^{n} C_n^m \frac{S}{m}$$

代码：

先定义函数 Cnm1(n,m)，求出 Cnm。

再定义函数 Sum_averages1(arr)，最后根据得出的通项公式，得到所有子集平均值的和。

代码如下：

```
# 4.2
import math
def Cnm1(n,m):
    return math.factorial(n)//(math.factorial(n-m)*math.factorial(m))

def Sum_averages1(arr):
    result = 0
    s = 0
    for i in range(len(arr)):
        s += arr[i]
    for n in range(1, len(arr)+1):
        result += (s*(Cnm1(len(arr) - 1, n - 1)))/n
    return result

arr = Random_integer(3)
print(Sum_averages1(arr))
```

4.3

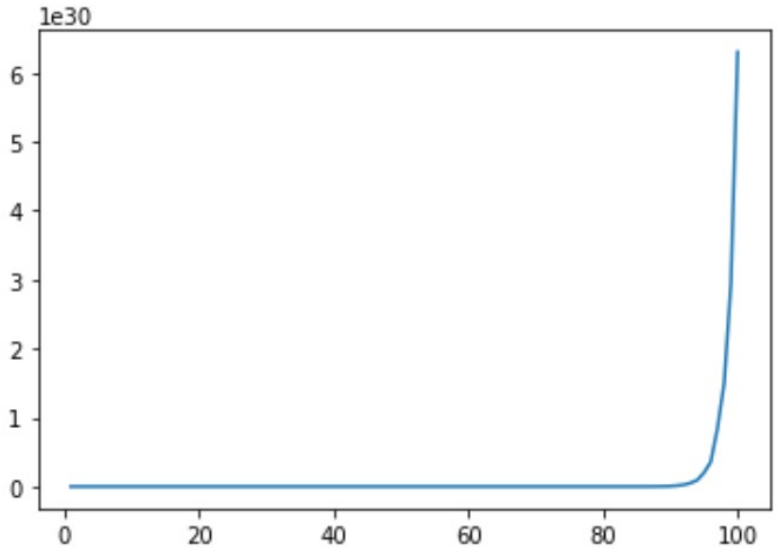
先写空列表 Total_sum_averages = []，N 的个数从 1 增加到 100，利用 4.1Random_integer(N)

函数将每次输出的值放到 arr1 中，最后 append 到 Total_sum_averages 中，再打印。最后 plot 出趋势图。
代码如下：

```
# 4.3
Total_sum_averages = []
for N in range(1,101):
    arr1 = Random_integer(N)
    Total_sum_averages.append(Sum_averages1(arr1))
    |
print(Total_sum_averages)

import matplotlib.pyplot as plt
x = np.arange(1,101)
y = np.array(Total_sum_averages)
print(plt.plot(x,y))
```

图形如下：



问题：虽然能正确输出值，但是运行数据很大时会有下面的提示。

```
C:\Users\李彦辰\AppData\Local\Temp/ipykernel_20664/72248357.py:19: RuntimeWarning: overflow encountered in lon
g_scalars
  result += (s*(Cnm1(len(arr) - 1, n - 1)))/n
```

4.2，我的思路是将每个子集都打印出来，再算每个子集的平均值，最后求和。但是这种方法用在 4.3 时，用时很长。
石邵给我讲解了 4.2 新方法的思路，感谢！
参考资料：Python 算法——求集合的所有子集_Aamax 的博客-CSDN 博客_python 子集

PS1_5 Path countnng

5.1

先输入想要的 N 和 M 值，利用 np.random.randnnt(0,2,(n,m))，随机生成 n,m 的矩阵，全部随机填充 0 或 1。再将左上角和右下角的数字改成 1，对应的位置是 arr[[0,-1],[0,-1]]=1。

代码如下：

```
# 5.1
import numpy as np
n = int(input("N = "))
m = int(input("M = "))
arr2 = np.random.randint(0,2,(n,m))
arr2[[0,-1],[0,-1]]=1
print(arr2)
```

5.2

思路：

障碍物是 0，通路是 1，计算到达某一个位置的所有路径是等于这个位置左边位置的所有路径+这个位置上面位置的所有路径和。障碍物时标记为 0，如果前一个是 1，后一个位置也依次加 1，最后的位置就是所有的加和。

代码：

给定 arr2，初始化 r 代表行数，c 代表列数；

给定一个二维数组 cache，cache 是一个 m*n 的二维数组，里面的元素全初始化为 0，更新第一个元素为 1。

接着对二维数组的每一行和每一列都进行遍历，如果当前数值=0，代表有障碍物，此时，更新中间的 cache=0。

如果没有障碍物，就要判断 i 是否大于 0，如果是在第一行，i=0 就不执行，同理第一列也不执行。

最后返回最后一个元素即可。

代码如下：

```
# 5.2
def Count_path(arr2):
    r = len(arr2)
    c = len(arr2[0])
    cache = [[0]*c for _ in range(r)]
    cache[0][0] = 1
    for i in range(r):
        for j in range(c):
            if (arr2[i][j] == 0):
                cache[i][j] = 0
            else:
                if i > 0:
                    cache[i][j]+=cache[i-1][j]
                if j > 0:
                    cache[i][j]+=cache[i][j-1]
    return cache[-1][-1]
Count_path(arr2)
```

5.3

当 N=10,M=8 时，Count_path 运行 1000 次，求均值。

写一个 for 循环 1000 次，求每次的加和，最后打印出平均值即可。每次的结果大概是 0.35 左右。

代码如下：

```
# 5.3
s = 0
for i in range(1000):
    arr3 = np.random.randint(0,2,(10,8))
    arr3[[0,-1],[0,-1]]=1
    i = Count_path(arr3)
    s += i
#    print(i)
print(s/1000)
```

整题完整代码如下：

```python
# 5
# 5.1
import numpy as np
n = int(input("N = "))
m = int(input("M = "))
arr2 = np.random.randint(0,2,(n,m))
arr2[[0,-1],[0,-1]]=1
print(arr2)

# 5.2
def Count_path(arr2):
    r = len(arr2)
    c = len(arr2[0])
    cache = [[0]*c for _ in range(r)]
    cache[0][0] = 1
    for i in range(r):
        for j in range(c):
            if (arr2[i][j] == 0):
                cache[i][j] = 0
            else:
                if i > 0:
                    cache[i][j]+=cache[i-1][j]
                if j > 0:
                    cache[i][j]+=cache[i][j-1]
    return cache[-1][-1]
Count_path(arr2)

# 5.3
s = 0
for i in range(1000):
    arr3 = np.random.randint(0,2,(10,8))
    arr3[[0,-1],[0,-1]]=1
    i = Count_path(arr3)
    s += i
#    print(i)
print(s/1000)
```

```
N = 10
M = 8
[[1 1 1 0 1 1 0 0]
 [1 0 1 1 0 0 0 1]
 [1 1 0 1 1 0 0 1]
 [0 0 1 1 0 1 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 1 0 1 0 0 1]
 [0 1 0 0 1 0 1 1]
 [1 1 0 0 1 0 0 0]
 [0 1 1 1 1 1 0 1]
 [1 0 1 1 1 0 1 1]]
0.318
```

石绍向我解释了 5.2 的思路，在写完代码后，发现代码正确，但是每次运行结果错误，最后发现是我在 5.2 定义函数下引用了 5.1 的变量，石绍和姜少杰师兄向我解释了这个困惑，感谢他们！