```python
#Assignments 01 by Yi Yang

#1.Flowchart
def print_values(a,b,c):
    if a>b and b>c:
        print(a,b,c)
    elif a>b and b<c and a>c:
        print(a,c,b)
    elif a>b and b<c and a<c:
        print(c,a,b)
    else:
        a<b and b<c
        print(c,b,a)
    #compute and print x+y-10z
    print(a+b-10*c)
print_values(10,5,1)
```

```
10 5 1
5
```

```python
#2.Continuous ceiling functon
N=int(input("输入一个正整数"))
#创建一个包含从1到N的正整数的列表
New_List=[i for i in range(1,N+1)]
print(New_List)
import math
def F(x):
    return math.ceil(x/3)
F(1)==1
print(F(N)+2*N)
```

```
输入一个正整数 3
[1, 2, 3]
7
```

```python
#3.Dice rolling
#3.1
import random
def rolling():
    return random.randint(1, 6)
# 定义函数计算得到特定总和 x 的投掷方式数量
def find_number_of_ways(x):
    count = 0
    # 遍历所有可能的骰子点数组合
    for dice1 in range(1, 7):
        for dice2 in range(1, 7):
            for dice3 in range(1, 7):
                for dice4 in range(1, 7):
                    for dice5 in range(1, 7):
                        for dice6 in range(1, 7):
                            for dice7 in range(1, 7):
                                for dice8 in range(1, 7):
                                    for dice9 in range(1, 7):
                                        for dice10 in range(1, 7):
                                            if (dice1 + dice2 + dice3 + dice4 + dice5 + dice6 + dice7 + dice8 + dice9 + dice10) == x:
                                                count += 1
    return count
# 测试函数
x = 60
print(f"得到总和 {x} 的投掷方式有：{find_number_of_ways(x)} 种")
```

```python
#3.2
def count_number_of_ways():
    # 从10到60，共51种可能的总和值
    ways = [0] * (60 - 10 + 1)
    for _ in range(100000):
        roll_results = [rolling() for _ in range(10)]
        total_sum = sum(roll_results)  # 计算总和
        if 10 <= total_sum <= 60:  # 确保总和值在10到60之间
            # 更新对应总和值的方式数量
            ways[total_sum - 10] += 1
    return ways
def find_max_ways(ways):
    max_ways = max(ways)
    max_sum = ways.index(max_ways) + 10  # 将索引转换回总和值
    return max_sum, max_ways
# 计算每种总和值的方式数量
WAYs = count_number_of_ways()
# 找出方式数量最多的总和值
max_sum, max_ways = find_max_ways(WAYs)
print(f"方式数量：{WAYs}")
print(f"达到方式数量最多的总和值是：{max_sum} ，有{max_ways} 种方式.")
```

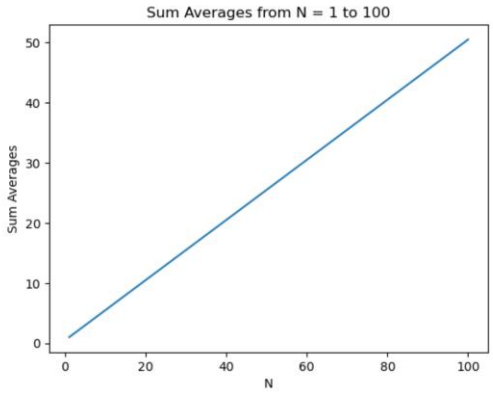
```
得到总和 60 的投掷方式有：1 种
方式数量：[0, 0, 0, 0, 0, 4, 7, 18, 29, 84, 147, 278, 385, 647, 987, 1383, 1908, 2553, 3234, 4062, 4919, 5474, 6386, 6809, 7163, 7414, 7196, 6907, 6330, 5612, 4779, 3875, 3156, 2504, 1898, 1338, 973, 606, 394, 258, 136, 83, 35, 18, 8, 2, 0, 1, 0, 0, 0]
达到方式数量最多的总和值是： 35 ， 有7414 种方式.
```

```python
#4.Dynamic programming
#4.1(通过查阅网页得知random.sample函数含义)
import random
N=int(input("输入一个0-11之间的整数"))
Random_integer=random.sample(range(11),N)
print(Random_integer)

#4.2(通过查阅网页得知itertools.combinations函数含义)
import itertools
#定义一个集合
set={1,2,3}
#设置一个列表储存所有子集
All=[]
n=len(set)
#生成所有长度为i的子集
for i in range(1,n+1):
    New=list(itertools.combinations(set,i))
    All.extend(New)
#分别计算所有非空子集的平均值
New2=[]
for subset in All:
    Average=sum(subset)/len(subset)
    New2.append(Average)
#计算所有子集平均值的总和
Sum_averages=sum(New2)
print(Sum_averages)
```

```python
#4.3
import matplotlib.pyplot as plt
# 计算总和平均值并存储到列表中
Total_sum_averages = []
for N in range(1, 101):
    sum_numbers = sum(range(1, N + 1))
    average = sum_numbers / N
    Total_sum_averages.append(average)
# 绘制 Total_sum_averages
plt.plot(range(1, 101), Total_sum_averages)
plt.xlabel('N')
plt.ylabel('Sum Averages')
plt.title('Sum Averages from N = 1 to 100')
plt.show()
```

```
输入一个0-11之间的整数 3
[8, 9, 4]
14.0
```



绘制的图像可以看到随着 N 从 1 增加到 100，总和平均值呈现出一种逐渐上升的趋势，但上升的速度逐渐减缓。一开始，当 N 较小时，总和平均值增长较快，因为少量数字的总和相对较小，随着 N 的增加，新加入的数字对总和的影响逐渐减小，导致总和平均值的增长速度变慢。图像大致呈现出一条弯曲程度逐渐变小的曲线，趋近于一个稳定的值。这表明随着数字数量的不断增加，总和平均值会逐渐稳定在一个特定的数值附近。

```python
#5.Path counting
#5.1
import random
N=int(input("输入行数N"))
M=int(input("输入列数M"))
#创建一个空矩阵
matrix=[]
#创建N行、M列，每个单元格都随机填充0或1的矩阵
for i in range(N):
    ROW=[random.randint(0,1) for j in range(M)]
    matrix.append(ROW)
matrix[0][0]=1
matrix[N-1][M-1]=1
for row in matrix:
    print(row)
```

```python
#5.2
def Count_paths(matrix):
    N = len(matrix)
    M = len(matrix[0])
    # 创建一个与原矩阵相同大小的矩阵，初始值全为 0
    dp = [[0] * M for _ in range(N)]
    dp[0][0] = 1 if matrix[0][0]==1 else 0
    # 初始化第一行和第一列
    for j in range(1, M):
        if matrix[0][j] == 1:
            dp[0][j] = dp[0][j - 1]
    for i in range(1, N):
        if matrix[i][0] == 1:
            dp[i][0] = dp[i - 1][0]
    # 计算路径总数
    for i in range(1, N):
        for j in range(1, M):
           if matrix[i][j] == 1:
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
    return dp[N - 1][M - 1]
result = Count_paths(matrix)
print(f"从左上角到右下角的路径总数为：{result}，矩阵大小为{N}行{M}列。")
```

```python
#5.3
def create_matrix(N, M):
    matrix = []
    for i in range(N):
        ROW = [random.randint(0, 1) for j in range(M)]
        matrix.append(ROW)
    matrix[0][0] = 1
    matrix[N-1][M-1] = 1
    return matrix
def average(N,M,times):
    total_paths=0
    for _ in range(times):
        matrix = create_matrix(N,M)
        total_paths += Count_paths(matrix)
    return total_paths / times
# 设置N和M的值
N = 10
M = 8
# 设置试验次数
times = 1000
# 计算平均路径数
average_result = average(N, M, times)
print(f"在{times}次运行中，从左上角到右下角的路径总数的平均值为：{average_result}")
```

```
输入行数N 3
输入列数M 3
[1, 0, 1]
[0, 0, 0]
[1, 0, 1]
从左上角到右下角的路径总数为：0，矩阵大小为3行3列。
在1000次运行中，从左上角到右下角的路径总数的平均值为：0.315
```