这个单元格给助教，请忽略!

Score:

Comment:

请实现每个 function 内容，确保最终提交的notebook是可以运行的。

每一题除了必须要报告的 输出/图表，可以添加解释（中文即可）。此外可以自定义其他 function / 变量，自由添加单元格，但请确保题目中给出的 function （如第一题的 Print_values）可以正常调用。

Collaboration:

Collaboration on solving the assignment is allowed, after you have thought about the problem sets on your own. It is also OK to get clarification (but not solutions) from online resources, again after you have thought about the problem sets on your own.

There are two requirements for collaboration:

- Cite your collaborators **fully and completely** (*e.g.*, "XXX explained to me what is asked in problem set 3"). Or cite online resources (*e.g.*, "I got inspired by reading XXX") that helped you.
- Write your scripts and report **independently** - the scripts and report must come from you only.

1. Flowchart

Write a function `Print_values` with arguments `a`, `b`, and `c` to reflect the following flowchart. Here the purple parallelogram operator on a list `[x, y, z]` is to compute and print `x+y-10z`. Try your output with some random `a`, `b`, and `c` values. Report your output when `a = 10, b = 5, c = 1`.

```
# def Print_values(a, b, c):
# # Your code here
def Print_values(a, b, c):
#判断a,b的大小
  if a>b:
#判断b,c的大小，若b>c直接输出，否则判断a,c的大小
      if b>c:
          value=[a,b,c]
```

```
        else:
            if a>c:
                value=[a,c,b]
            else:
                value=[c,a,b]
#否则若a<b
  else:
#判断b,c的大小，若b>c,则继续判断a,c，否则直接输出
        if b>c:
            if a>c:
                value=[b,a,c]
            else:
                value=[b,c,a]
        else:
            value[c,b,a]
#计算并打印x+y-10*z
  result=value[0] + value[1] -10*value[2]
  print(result)
Print_values(10, 5, 1)
```

```
5
```

Report your output when `a = 10, b = 5, c = 1`: [5]

先定义一个函数 Print_values，它接受三个参数 a、b 和 c，然后根据这些参数的大小关系，将它们排序并存储在列表 value 中。最后，函数计算 value 列表中第一个元素和第二个元素的和减去第三个元素的十倍，并将结果打印出来。

2. Continuous ceiling function

Given a list with `N` positive integers. For every element `x` of the list, find the value of continuous ceiling function defined as `F(x) = F(ceil(x/3)) + 2x`, where `F(1) = 1`.

```
import math
def continuous_ceil_function(x):
#当x=1时
   if x==1:
      return 1
#递增情况
   else:
      return continuous_ceil_function(math.ceil(x / 3)) + 2 * x
#构造示例
numbers=[1,4,6,9,12]
```

#计算示例中每个x的值
function_values = [continuous_ceil_function(x) for x in numbers]
#打印结果
print(function_values)

```
import math
def continuous_ceil_function(x):
#当x=1时
    if x==1:
        return 1
#递增情况
    else:
        return continuous_ceil_function(math.ceil(x / 3)) + 2 * x
#构造示例
numbers=[1,4,6,9,12]
#计算示例中每个x的值
function_values = [continuous_ceil_function(x) for x in numbers]
#打印结果
print(function_values)
```

```
[1, 13, 17, 25, 37]
```

先定义了一个递归函数 continuous_ceil_function，如果输入 x 等于1，函数直接返回1。如果 x 不等于1，函数首先计算 math.ceil(x / 3)，这是对 x 除以3后的结果向上取整。然后，函数递归调用自身，传入这个向上取整后的值，即 continuous_ceil_function(math.ceil(x / 3))。这个递归调用的结果是与 2 * x 相加得到最终结果。然后定义了一个列表 numbers，包含了一系列的测试值。然后使用列表推导式，对 numbers 列表中的每个元素 x 应用 continuous_ceil_function 函数，并将结果存储在 function_values 列表中。最后，使用 print 函数打印出 function_values 列表，这个列表包含了输入列表 numbers 中每个元素对应的函数计算结果。

3. Dice rolling

3.1 Given `10` dice each with `6` faces, numbered from `1` to `6`. Write a function `Find_number_of_ways` to find the number of ways to get sum `x`, defined as the sum of values on each face when all the dice are thrown.

```
def Find_number_of_ways(n,s):
#n为骰子的数量，s为sumx,创建一个二维数组，初始化动态规划数组
    dp = [[0 for _ in range(s+1)] for _ in range(n+1)]
```

```
    dp[0][0] = 1
#填充动态规划数组
    for i in range (1,n+1):
      for j in range (1,s+1):
#每个骰子都可以显示1-6的任意数字
          for k in range (1,7):
              if j >=k:
                  dp[i][j] += dp[i-1][j-k]
#返回n个骰子数字之和为s的方法数量
    return dp[n][s]
#计算10个骰子的到总数为30的方法数量
print (Find_number_of_ways(10,30))
```

```
2930455
```

3.2 Count the number of ways for any `x` from `10` to `60`, assign the number of ways to a list called `Number_of_ways`, so which `x` yields the maximum of `Number_of_ways`?

```
#10个骰子总数和为10到60的每个可能的总和
Number_of_ways =[Find_number_of_ways(10,s) for s in range(10,61)]
#找到最大的s
max_value =max(Number_of_ways)
#找出Number_of_ways中的s_max，从10开始，所以加上10
s_max = Number_of_ways .index(max_value)+10
print(s_max)
```

```
35
```

So which `x` yields the maximum of `Number_of_ways`? [35]

首先创建一个二维数组 dp，其中 dp[i][j] 表示使用 i 个骰子得到总和 j 的方法数量。数组的大小为 (n+1) x (s+1)，其中 n 是骰子的数量，s 是可能的最大总和。初始化时，dp[0][0] 设置为1，表示没有骰子得到总和0有一种方法。使用三层循环来填充 dp 数组。外层循环遍历骰子的数量 i，中间循环遍历总和 j，内层循环遍历骰子的每个面 k（从1到6）。如果当前总和 j 大于或等于骰子的面值 k，则可以通过添加一个显示数字为 k 的骰子来增加总和，因此 dp[i][j] 会增加 dp[i-1][j-k] 的值。函数返回 dp[n][s]，即使用 n 个骰子得到总和 s 的方法数量。计算时，首先计算使用10个骰子得到总和30的方法数量，并打印结果。使用列表推导式计算使用10个骰子得到从10到60的每个可能总和的方法数量，并将结果存储在 Number_of_ways 列表中。然后使用 max 函数找到 Number_of_ways 列表中的最大值，即方法数量最多的总和。使用 index 方法找到最大值Number_of_ways 列表中的索引，并加上10（因为列表的索引从0开始，而总和从10开始），得到 s_max，即方法数量最多的总和。

4. Dynamic programming

4.1 [5 points] Write a function `Random_integer` to fill an array of `N` elements by randomly selecting integers from `0` to `10`.

```
import random
import matplotlib.pyplot as plt
#定义函数，参数为N列表中的每个元素都是0-10之间的整数
def Random_integer(N):
    return [random.randint(0,10)for _ in range(N)]
N = 5
random_array = Random_integer(N)
random_array
```

```
[7, 3, 1, 9, 2]
```

4.2 [15 points] Write a function `Sum_averages` to compute the sum of the average of all subsets of the array. For example, given an array of `[1, 2, 3]`, you `Sum_averages` function should compute the sum of: average of `[1]`, average of `[2]`, average of `[3]`, average of `[1, 2]`, average of `[1, 3]`, average of `[2, 3]`, and average of `[1, 2, 3]`.

```
def Sum_average(arr):
    n = len(arr)    #获取长度
    total_sum =0    #初始化一个变量total_sum，用于累加所有子集的平均值
    for i in range (1<<n):
#初始化两个变量，分别用于计算当前子集的元素和以及子集中元素的数量
        subset_sum = 0
        subset_count = 0
        for j in range (n):
#检查当前整数i的第j位是否为1，如果是，则表示第j个元素属于当前子集
            if (i & (1 << j)):
#如果第j个元素属于子集，则将其值加到subset_sum中，并将subset_count加1
                subset_sum += arr[j]
                subset_count += 1
#如果子集中有元素，则计算子集的平均值并加到total_sum中
        if subset_count > 0:
            total_sum += subset_sum/subset_count
    return total_sum
arr=[1,2,3]
```

```
sum_of_averages = Sum_averages(arr)

print(sum_of_averages)
```

4.3 [5 points] Call `Sum_averages` with `N` increasing from `1` to `100`, assign the output to a list called `Total_sum_averages`. Plot `Total_sum_averages`, describe what you see.

```
#初始化一个空列表，用于存储每个数组的平均值总和
Total_sum_average =[]
for N in range(1,101):
#使用Random_integer函数生成一个长度为N的随机整数数组
    arr =Random_integer(N)
#计算这个数组所有非空子集的平均值的总和，并将其添加到Total_sum_average列表中
    Total_sum_average . append(Sum_average(arr))
#使用matplotlib.pyplot的plot函数绘制一个图表，横轴是数组的长度N，纵轴是对应的平均值总和
plt.plot (range(1,101), Total_sum_average)
#设置图表的横轴标签、纵轴标签和标题
plt.xlabel('N')
plt.ylabel('Sum of Average')
plt.title('Sum of Average for Subsets Arreys with Random Integers')
plt.show()
```

Describe what you see. [Your answer]

先定义了 Random_integer(N) 函数，它生成一个包含 N 个元素的列表，每个元素是一个0到10之间的随机整数。然后定义了 Sum_average(arr) 函数，它计算一个数组 arr 所有非空子集的平均值的总和。这个函数通过所有可能的子集（使用位运算，从 1 到 2^n - 1）来实现。对于每个子集，它计算子集中元素的和以及子集中元素的数量，然后计算平均值并累加到 total_sum 中。使用一个循环，从1到100，对于每个长度 N，生成一个随机整数数组。对于每个生成的数组，使用 Sum_average 函数计算所有非空子集的平均值的总和，并将结果存储在 Total_sum_average 列表中。最后使用 matplotlib.pyplot 的 plot 函数绘制一个图表，横轴是数组的长度 N，纵轴是对应的平均值总和。然后设置图表的横轴标签、纵轴标签和标题，并显示图表。

5. Path counting

5.1 [5 points] Create a matrix with `N` rows and `M` columns, fill the right-bottom corner and top-left corner cells with `1`, and randomly fill the rest of matrix with integer `0` or `1`.

```
import numpy as np
def create_matrix (N,M):
    matrix = np.ones ((N,M), dtype = int)  #先全部填充为1
    matrix [0,0] = 1  #设置左上角和右下角
    matrix [-1,-1] = 1
    for i in range (1,N-1):
        for j in range (1,M-1):
            matrix [i,j] = np.random . randint(0,2)
    return matrix
N,M = 10, 8
matrix = create_matrix(N,M)
print (matrix)
```

```
[[1 1 1 1 1 1 1 1]
 [1 1 0 1 0 0 1 1]
 [1 1 0 0 0 1 0 1]
 [1 0 0 1 1 1 1 1]
 [1 0 0 0 1 0 1 1]
 [1 1 1 1 0 1 1 1]
 [1 0 1 1 0 0 0 1]
 [1 0 0 0 0 1 0 1]
 [1 0 1 1 0 1 0 1]
 [1 1 1 1 1 1 1 1]]
```

5.2 [25 points] Consider a cell marked with `0` as a blockage or dead-end, and a cell marked with `1` is good to go. Write a function `Count_path` to count the total number of paths to reach the right-bottom corner cell from the top-left corner cell.

Notice: for a given cell, you are **only allowed** to move either rightward or downward.

```
#定义一个函数Count_path，用于计算从矩阵的左上角到右下角的路径数量，路径只能向右或向下移动
def Count_path (matrix):
    N,M = matrix .shape  #获取行数和列数
 #如果矩阵的左上角或右下角是0，则没有路径，直接返回0
    if matrix [0,0] == 0 or matrix [N-1,M-1] == 0:
        return 0
#创建动态规划数组，初始值设为0
    dp = np.zeros((N,M),dtype = int)
#起始点的路径数量为1
    dp[0,0] = 1
```

```
#两个循环初始化第一行和第一列的dp值，如果单元格的值为1，则该单元格的路径数量等于它上方或左方单元格的路径数量
    for i in range (1,N):
        dp [i,0] = dp [i-1,0] if matrix [i,0] ==1 else 0
    for j in range (1,M):
        dp [0,j] = dp [0,j-1] if matrix [0,j] ==1 else 0
#两个嵌套循环遍历矩阵的其余部分，使用动态规划的方法计算每个单元格的路径数量，如果当前单元格的值为1，则该单元格的路径数量是它上方和左方单元格路径数量的总和
    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]
#调用Count_path函数计算当前矩阵的路径数量
total_paths = Count_path(matrix)
#打印矩阵和路径数量
print(matrix)
print(total_paths)
```

```
[[1 1 1 1 1 1 1 1]
 [1 1 0 1 0 0 1 1]
 [1 1 0 0 0 1 0 1]
 [1 0 0 1 1 1 1 1]
 [1 0 0 0 1 0 1 1]
 [1 1 1 1 0 1 1 1]
 [1 0 1 1 0 0 0 1]
 [1 0 0 0 0 1 0 1]
 [1 0 1 1 0 1 0 1]
 [1 1 1 1 1 1 1 1]]
3
```

5.3 [5 points] Let `N = 10, M = 8`, run `Count_path` for `1000` times, each time the matrix (except the right-bottom corner and top-left corner cells, which remain being `1`) is re-filled with integer `0` or `1` randomly, report the mean of total number of paths from the `1000` runs.

```
#计算1000次运行，路径的平均数量
for _ in range (1000):
    matrix = create_matrix (N,M)
    total_paths += Count_path(matrix)
average_paths = total_paths /1000
print (average_paths)
```

```
16.526
```

Report the mean of total number of paths from the `1000` runs. [16.526]

定义 create_matrix(N, M) 函数，它创建一个 N 行 M 列的矩阵。矩阵的元素初始化为1，除了左上角和右下角也设置为1。对于矩阵中间的部分（除去第一行、第一列、最后一行和最后一列），每个元素随机设置为0或1。定义 Count_path(matrix) 函数，它使用动态规划的方法计算从矩阵的左上角到右下角的路径数量。如果矩阵的左上角或右下角是0，则直接返回0，因为没有路径。创建一个与输入矩阵同样大小的动态规划数组 dp，初始值设为0。起始点（左上角）的路径数量设为1。如果单元格的值为1，则该单元格的路径数量等于它上方或左方单元格的路径数量。使用两个嵌套循环遍历矩阵的其余部分，如果当前单元格的值为1，则该单元格的路径数量是它上方和左方单元格路径数量的总和，调用 Count_path 函数计算当前矩阵的路径数量，并打印矩阵和路径数量。循环1000次，每次创建一个新的矩阵并计算路径数量，然后将这些路径数量相加。最后，将总路径数量除以1000，得到平均路径数量，并打印结果。