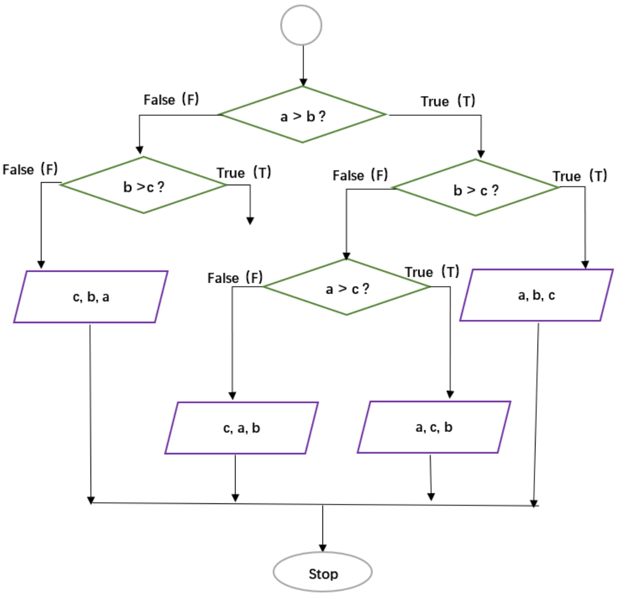
Homework #1

Name: 温承彦　　　SID：12332279

Problem 1：Flowchart

[10 points] Write a function Print_values with arguments a, b, and c to reflect the following flowchart. Here the purple parallelogram operator is to print values in the given order. Report your output with some random a, b, and c values.



Answer:

Through PS1_1.py, the output is calculated as follows:

Table 1

	a	***b***	***c***	***Print value***
#1	11	2	13	13,11,2
#2	2	4	7	7,4,2

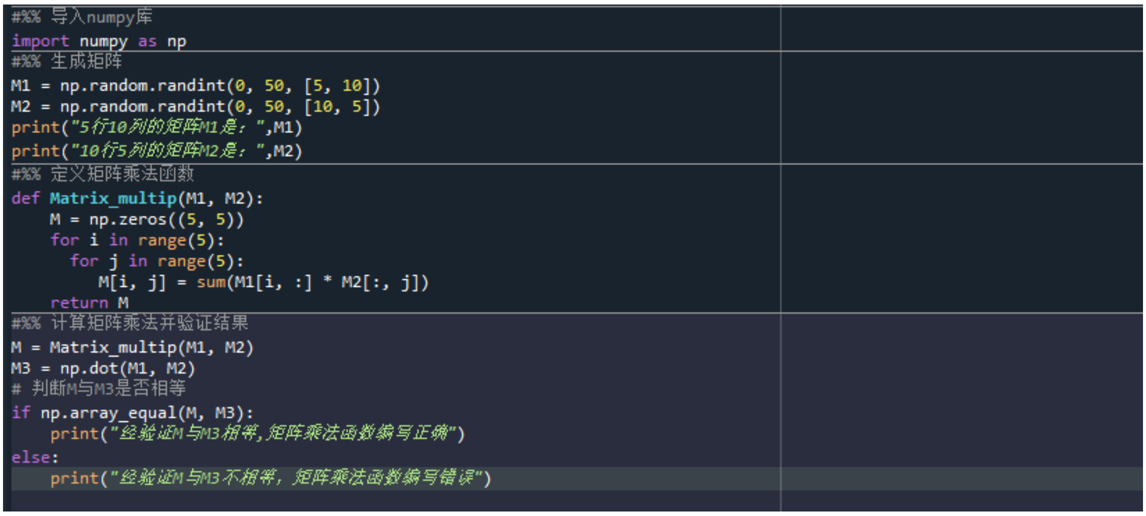
#3	8	1	5	8,5,1
#4	15	13	7	15,13,7
#5	3	12	6	6,3,12
#6	2	8	1	2,1,8

Problem 2：Matrix multiplication

2.1 [5 points] Make two matrices M1 (5 rows and 10 columns) and M2 (10 rows and 5 columns); both are filled with random integers from 0 and 50.

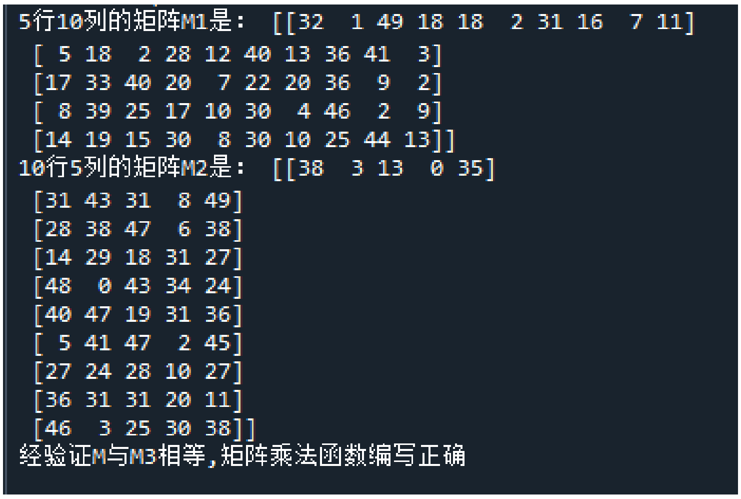
2.2 [10 points] Write a function Matrix_multip to do matrix multiplication, *i.e.*, M1 * M2. Here you are **ONLY** allowed to use for loop, * operator, and + operator.

Answer:



```
#%% 导入numpy库
import numpy as np
#%% 生成矩阵
M1 = np.random.randint(0, 50, [5, 10])
M2 = np.random.randint(0, 50, [10, 5])
print("5行10列的矩阵M1是：",M1)
print("10行5列的矩阵M2是：",M2)
#%% 定义矩阵乘法函数
def Matrix_multip(M1, M2):
    M = np.zeros((5, 5))
    for i in range(5):
      for j in range(5):
         M[i, j] = sum(M1[i, :] * M2[:, j])
    return M
#%% 计算矩阵乘法并验证结果
M = Matrix_multip(M1, M2)
M3 = np.dot(M1, M2)
# 判断M与M3是否相等
if np.array_equal(M, M3):
    print("经验证M与M3相等,矩阵乘法函数编写正确")
else:
    print("经验证M与M3不相等，矩阵乘法函数编写错误")
```

Through PS1_2.py, the output is as followed:

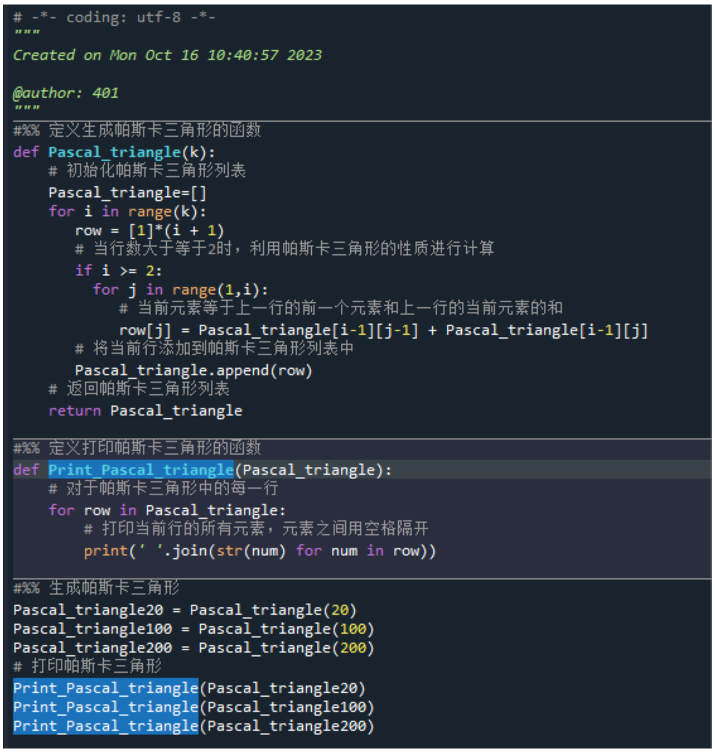


```
5行10列的矩阵M1是：  [[32  1 49 18 18  2 31 16  7 11]
 [ 5 18  2 28 12 40 13 36 41  3]
 [17 33 40 20  7 22 20 36  9  2]
 [ 8 39 25 17 10 30  4 46  2  9]
 [14 19 15 30  8 30 10 25 44 13]]
10行5列的矩阵M2是：  [[38  3 13  0 35]
 [31 43 31  8 49]
 [28 38 47  6 38]
 [14 29 18 31 27]
 [48  0 43 34 24]
 [40 47 19 31 36]
 [ 5 41 47  2 45]
 [27 24 28 10 27]
 [36 31 31 20 11]
 [46  3 25 30 38]]
经验证M与M3相等,矩阵乘法函数编写正确
```

Problem 3：Pascal triangle

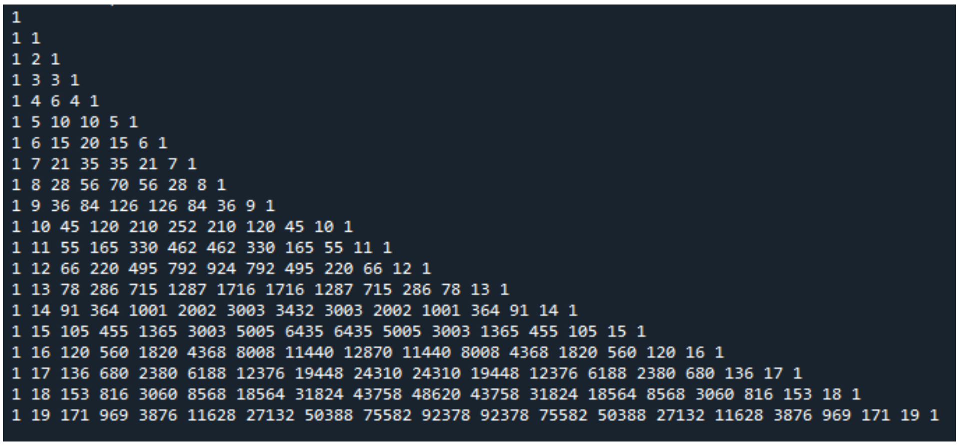
[20 points] One of the most interesting number patterns is Pascal’s triangle (named after Blaise Pascal). Write a function Pascal_triangle with an argument k to print the *k*th line of the Pascal triangle. Report Pascal_triangle(100) and Pascal_triangle(200).

Answer:



```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 16 10:40:57 2023

@author: 401
"""
#%% 定义生成帕斯卡三角形的函数
def Pascal_triangle(k):
    # 初始化帕斯卡三角形列表
    Pascal_triangle=[]
    for i in range(k):
       row = [1]*(i + 1)
       # 当行数大于等于2时，利用帕斯卡三角形的性质进行计算
       if i >= 2:
         for j in range(1,i):
            # 当前元素等于上一行的前一个元素和上一行的当前元素的和
            row[j] = Pascal_triangle[i-1][j-1] + Pascal_triangle[i-1][j]
       # 将当前行添加到帕斯卡三角形列表中
       Pascal_triangle.append(row)
    # 返回帕斯卡三角形列表
    return Pascal_triangle

#%% 定义打印帕斯卡三角形的函数
def Print_Pascal_triangle(Pascal_triangle):
    # 对于帕斯卡三角形中的每一行
    for row in Pascal_triangle:
        # 打印当前行的所有元素，元素之间用空格隔开
        print(' '.join(str(num) for num in row))

#%% 生成帕斯卡三角形
Pascal_triangle20 = Pascal_triangle(20)
Pascal_triangle100 = Pascal_triangle(100)
Pascal_triangle200 = Pascal_triangle(200)
# 打印帕斯卡三角形
Print_Pascal_triangle(Pascal_triangle20)
Print_Pascal_triangle(Pascal_triangle100)
Print_Pascal_triangle(Pascal_triangle200)
```

Through PS1_3.py, the Pascal_triangle(20) output is as followed, due to the long results of Pascal_triangle(100) and Pascal_triangle(200), they cannot be included in the report. Please refer to the detailed output in the code execution results.

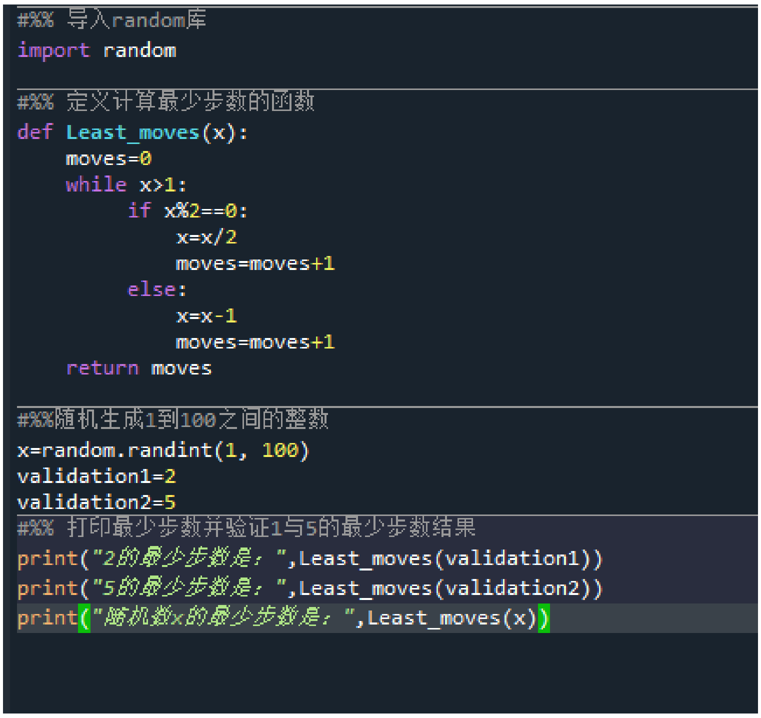


```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1
```

Problem 4：Add or double

[20 points] If you start with 1 RMB and, with each move, you can either double your money or add another 1 RMB, what is the smallest number of moves you have to make to get to exactly x RMB? Here x is an integer randomly selected from 1 to 100. Write a function Least_moves to print your results. For example, Least_moves(2) should print 1, and Least_moves(5) should print 3.

Answer:



```python
#%% 导入random库
import random

#%% 定义计算最少步数的函数
def Least_moves(x):
    moves=0
    while x>1:
         if x%2==0:
             x=x/2
             moves=moves+1
         else:
             x=x-1
             moves=moves+1
    return moves

#%%随机生成1到100之间的整数
x=random.randint(1, 100)
validation1=2
validation2=5
#%% 打印最少步数并验证1与5的最少步数结果
print("2的最少步数是： ",Least_moves(validation1))
print("5的最少步数是： ",Least_moves(validation2))
print("随机数x的最少步数是： ",Least_moves(x))
```

Through PS1_4.py, the output is as followed:

Table 2

	x	$move$
#1	12	4
#2	15	6
#3	14	5
#4	23	7
#5	62	9
#6	78	9

Problem 5：Dynamic programming

Insert + or - operation anywhere between the digits 123456789 in a way that the expression evaluates to an integer number. You may join digits together to form a bigger number. However, the digits must stay in the original order.

5.1 [30 points] Write a function Find_expression, which should be able to print every possible solution that makes the expression evaluate to a random integer from 1 to 100. For example, Find_expression(50) should print lines include:

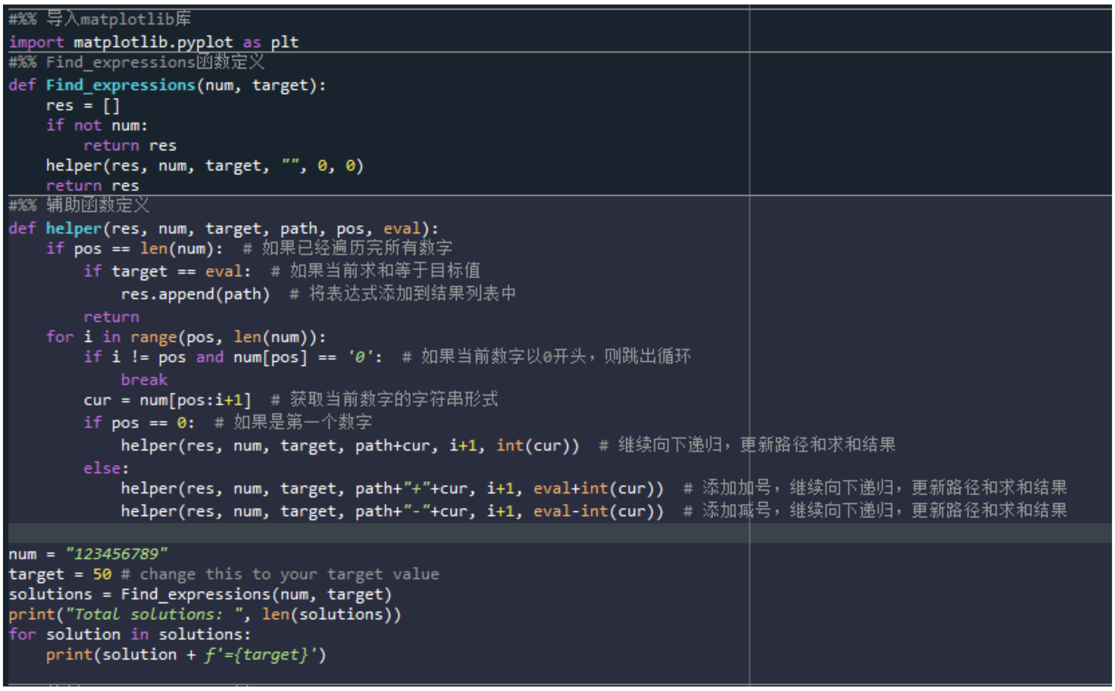
$$1 - 2 + 34 + 5 + 6 + 7 + 8 - 9 = 50$$

and

$$1 + 2 + 34 - 56 + 78 - 9 = 50$$

5.2 [5 points] Count the total number of suitable solutions for any integer *i* from 1 to 100, assign the count to a list called Total_solutions. Plot the list Total_solutions, so which number(s) yields the maximum and minimum of Total_solutions?

Answer:

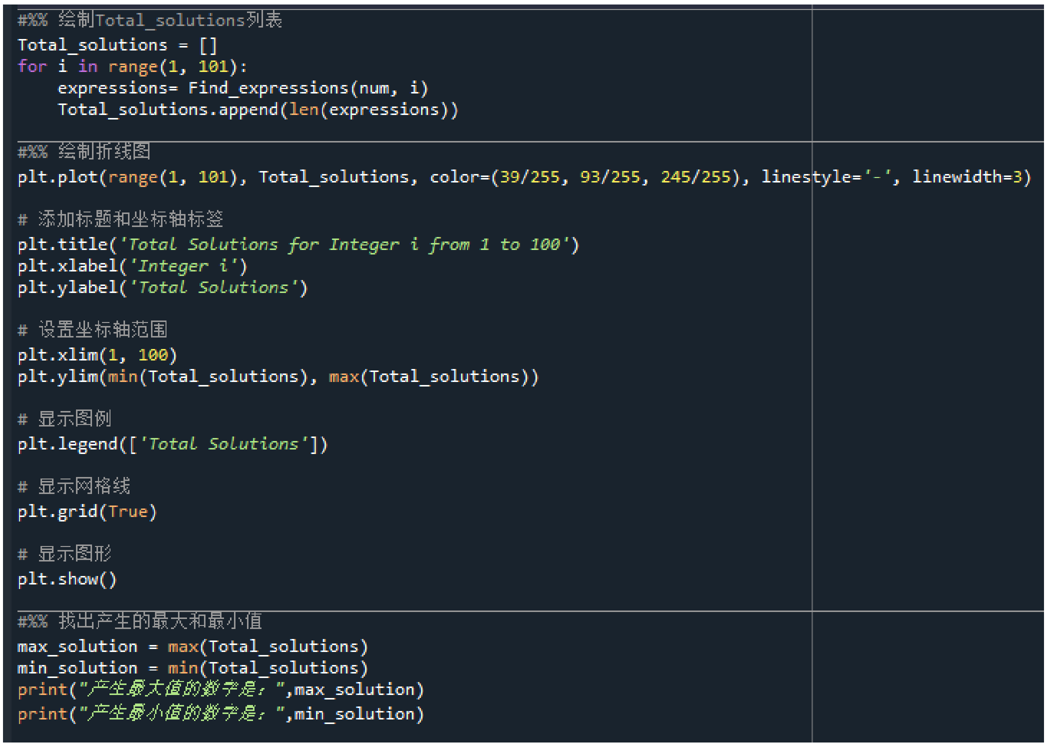


```python
#%% 导入matplotlib库
import matplotlib.pyplot as plt
#%% Find_expressions函数定义
def Find_expressions(num, target):
    res = []
    if not num:
        return res
    helper(res, num, target, "", 0, 0)
    return res
#%% 辅助函数定义
def helper(res, num, target, path, pos, eval):
    if pos == len(num):  # 如果已经遍历完所有数字
        if target == eval:  # 如果当前求和等于目标值
            res.append(path)  # 将表达式添加到结果列表中
        return
    for i in range(pos, len(num)):
        if i != pos and num[pos] == '0':  # 如果当前数字以0开头，则跳出循环
            break
        cur = num[pos:i+1]  # 获取当前数字的字符串形式
        if pos == 0:  # 如果是第一个数字
            helper(res, num, target, path+cur, i+1, int(cur))  # 继续向下递归，更新路径和求和结果
        else:
            helper(res, num, target, path+"+"+cur, i+1, eval+int(cur))  # 添加加号，继续向下递归，更新路径和求和结果
            helper(res, num, target, path+"-"+cur, i+1, eval-int(cur))  # 添加减号，继续向下递归，更新路径和求和结果

num = "123456789"
target = 50 # change this to your target value
solutions = Find_expressions(num, target)
print("Total solutions: ", len(solutions))
for solution in solutions:
    print(solution + f'={target}')
```

- **res: Result list used to store the expressions that satisfy the condition.**
- **num: The numbers to be used.**
- **target: The desired value.**
- **path: The current path, which represents the generated expression so far.**
- **pos: The index of the current number.**
- **eval: The current sum.**

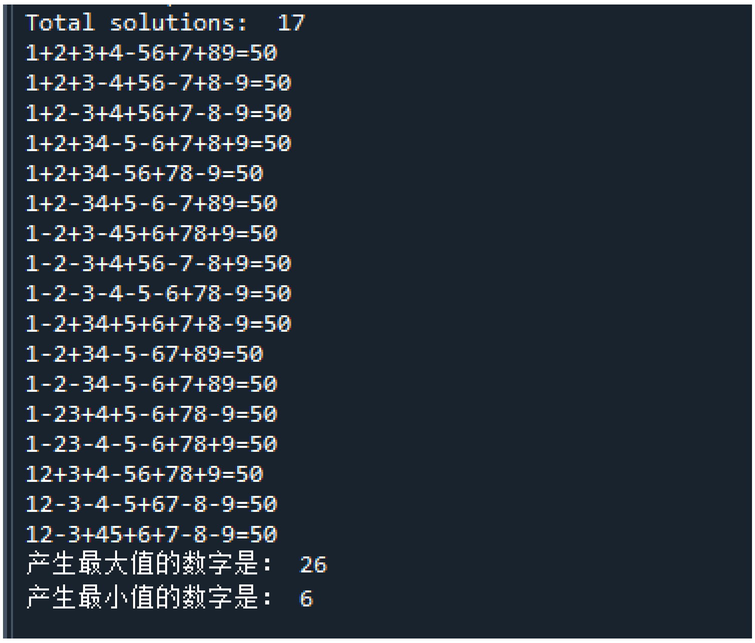
Inside the function, firstly, it checks whether all numbers have been traversed. If so, it checks if the current sum equals the target value. If yes, it adds the current path to the result list. Then, it uses a loop to iterate through all numbers starting from the current position. Inside the loop, it firstly checks if the current number starts with 0. If yes, it exits the loop because numbers starting with 0 cannot form valid expressions. Then, it obtains the string form of the current number and makes recursive calls based on whether the current position is the first number. If it is the first number, it directly adds the current number to the path and updates the current sum. If it is not the first number, it makes recursive calls for both addition and subtraction, updating the path and the current sum accordingly. In this way, through recursive calls, the function generates all expressions that meet the conditions and adds them to the result list.

Then we plotted a list of solutions corresponding to each number and generated a corresponding line graph.

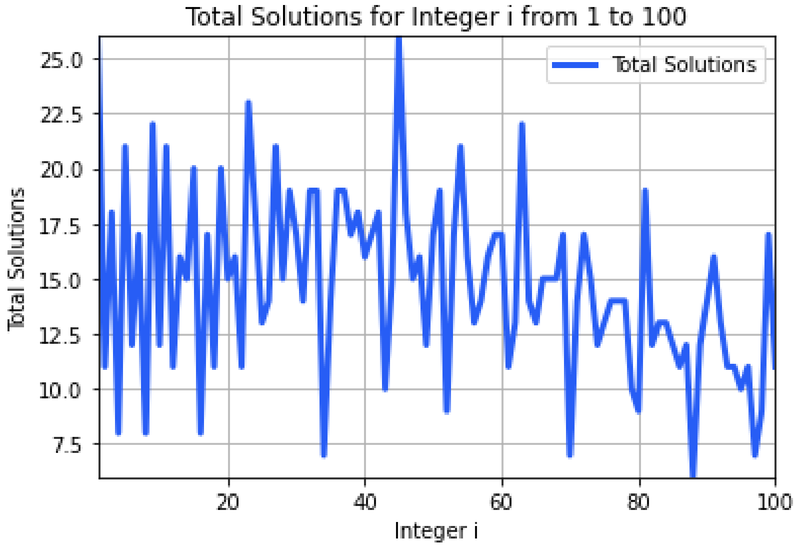


```
#%% 绘制Total_solutions列表
Total_solutions = []
for i in range(1, 101):
    expressions= Find_expressions(num, i)
    Total_solutions.append(len(expressions))

#%% 绘制折线图
plt.plot(range(1, 101), Total_solutions, color=(39/255, 93/255, 245/255), linestyle='-', linewidth=3)

# 添加标题和坐标轴标签
plt.title('Total Solutions for Integer i from 1 to 100')
plt.xlabel('Integer i')
plt.ylabel('Total Solutions')

# 设置坐标轴范围
plt.xlim(1, 100)
plt.ylim(min(Total_solutions), max(Total_solutions))

# 显示图例
plt.legend(['Total Solutions'])

# 显示网格线
plt.grid(True)

# 显示图形
plt.show()

#%% 找出产生的最大和最小值
max_solution = max(Total_solutions)
min_solution = min(Total_solutions)
print("产生最大值的数字是：",max_solution)
print("产生最小值的数字是：",min_solution)
```

Through PS1_5.py, the output and the graph is as followed:



```
Total solutions:  17
1+2+3+4-56+7+89=50
1+2+3-4+56-7+8-9=50
1+2-3+4+56+7-8-9=50
1+2+34-5-6+7+8+9=50
1+2+34-56+78-9=50
1+2-34+5-6-7+89=50
1-2+3-45+6+78+9=50
1-2-3+4+56-7-8+9=50
1-2-3-4-5-6+78-9=50
1-2+34+5+6+7+8-9=50
1-2+34-5-67+89=50
1-2-34-5-6+7+89=50
1-23+4+5-6+78-9=50
1-23-4-5-6+78+9=50
12+3+4-56+78+9=50
12-3-4-5+67-8-9=50
12-3+45+6+7-8-9=50
产生最大值的数字是:  26
产生最小值的数字是:  6
```


Total Solutions for Integer i from 1 to 100
Total Solutions
Total Solutions
Integer i