1. Flowchart

   I was provided with a flowchart for this question, so I simply followed it to define a function called 'Print_values.' Afterward, I utilized the random function to generate three random numbers for testing the function. The results of all ten tests demonstrated that the function works perfectly, printing the numbers in descending order.

No.1	78	51	14
No.2	94	87	71
No.3	75	53	30
No.4	94	54	42
No.5	61	50	22
No.6	94	54	23
No.7	89	82	1
No.8	96	51	29
No.9	68	60	21
No.10	75	28	2

2. Matrix multiplication
   a) Use np.random.randint() method to generate matrix filled with random integers form 0 to 50 (I learned this from a csdn bolg https://blog.csdn.net/weixin_39274808/article/details/105181122)
   b) Define the function Matrix_multip to do matrix multiplication and use the np.dot() method to do the same calculation to test the result

3. Pascal triangle

   All the numbers in row 'j' at location 'i,' except the first and last ones, are the sum of the numbers in row 'j-1' at location 'i' and 'i-1.' To implement this rule of Pascal's triangle, we need an array 'prev_row' to calculate the ‘new row.' A multidimensional array is created to store the 'triangle,' and we calculate it row by row until reaching the desired row. The results are quite long, so I won't display them here. I have already added 'Pascal_triangle(100)' and 'Pascal_triangle(200)' in the 'PS1_3.py' script. Running the script will provide the answers immediately.

4. Add or double

   Define the function Least_moves() to calculate how many steps are needed to reach 'x' by performing calculations starting from 'x'. If 'x' is odd, subtract 1 to obtain a new 'x'; if 'x' is even, divide it by 2 to get a new 'x', and repeat the process until 'x' equals 1."

   2 and 5 are used as input to test the function and the output are correct.

5.

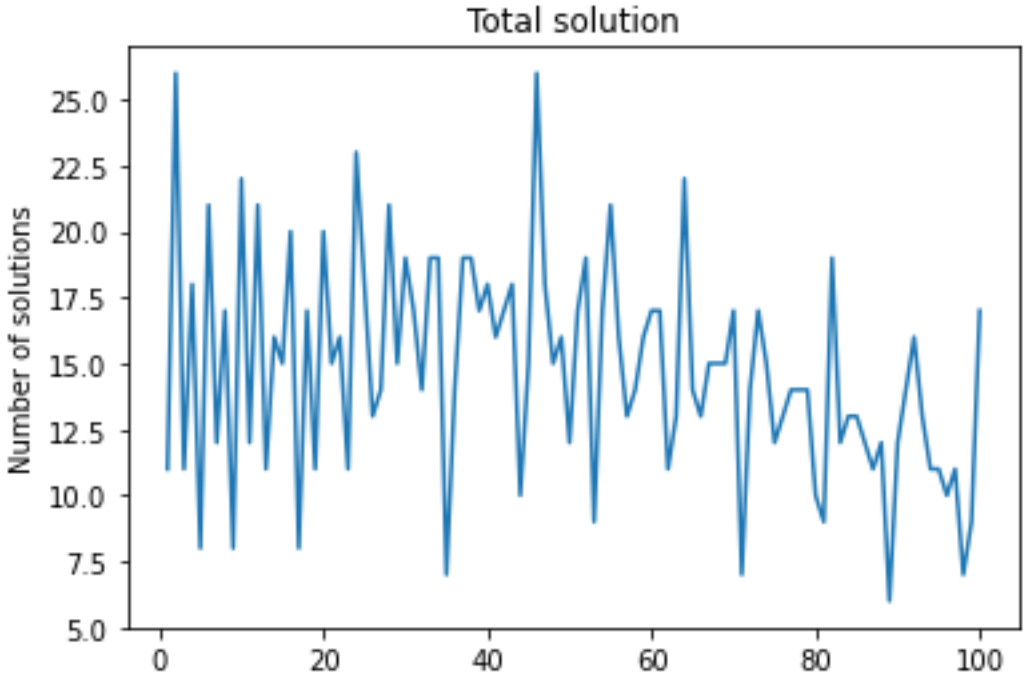
   5.1 In this question, there are three possible states between every two

numbers: addition, subtraction, and no operator. These three states are represented using the numbers 0, 1, and 2, respectively. There are a total of 9 numbers, which means there are 8 spaces that can be filled with one 8-digit ternary number to represent the states of these 8 spaces. The smallest number is 00000000 (corresponding to decimal 0), and the largest number is 22222222 (corresponding to decimal 6560). This means there are 6561 possible states for these 8 spaces in total.

I first wrote a function to convert decimal numbers from 0 to 6560 into ternary numbers and stored each digit of the ternary number in a list called "three_list." If the ternary number has less than 8 digits, I padded it with zeros to make it 8 digits long. (such as 200112 is ternary number of 500, it is store as [0,0,2,0,0,1,1,2] in the three_list). Then, I added all the "three_list" instances to the "ternary_list," thus obtaining all possible states between the 9 numbers 123456789. (The idea about use ternary is inspired by Xingyu Nan)

Then I iterate through the entire list to calculate all the solutions and store the results and equations in a dictionary as values and keys. When the function Find_expression() is called, all the keys with value equal to argument will be printed.

5.2



The result show that 1 and 45 has the most solution:26 xand 88 has the least solution:6