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 |

1. Matrix multiplication
   1. 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>)

* 1. Define the function Matrix\_multip to do matrix multiplication and use the np.dot() method to do the same calculation to test the result

1. 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.

1. 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.