PS1_1.py

Use if and else conditional statements to complete the requirements of the flowchart
Here are some output:

```
x=rand.randint(0,90)
y=rand.randint(0,90)
z=rand.randint(0,90)

print_values(x,y,z)
```

float is also ok:

```
In [4]: print_values(5.5,10.9,8.8)
8.8 5.5 10.9
```

PS1_2.py

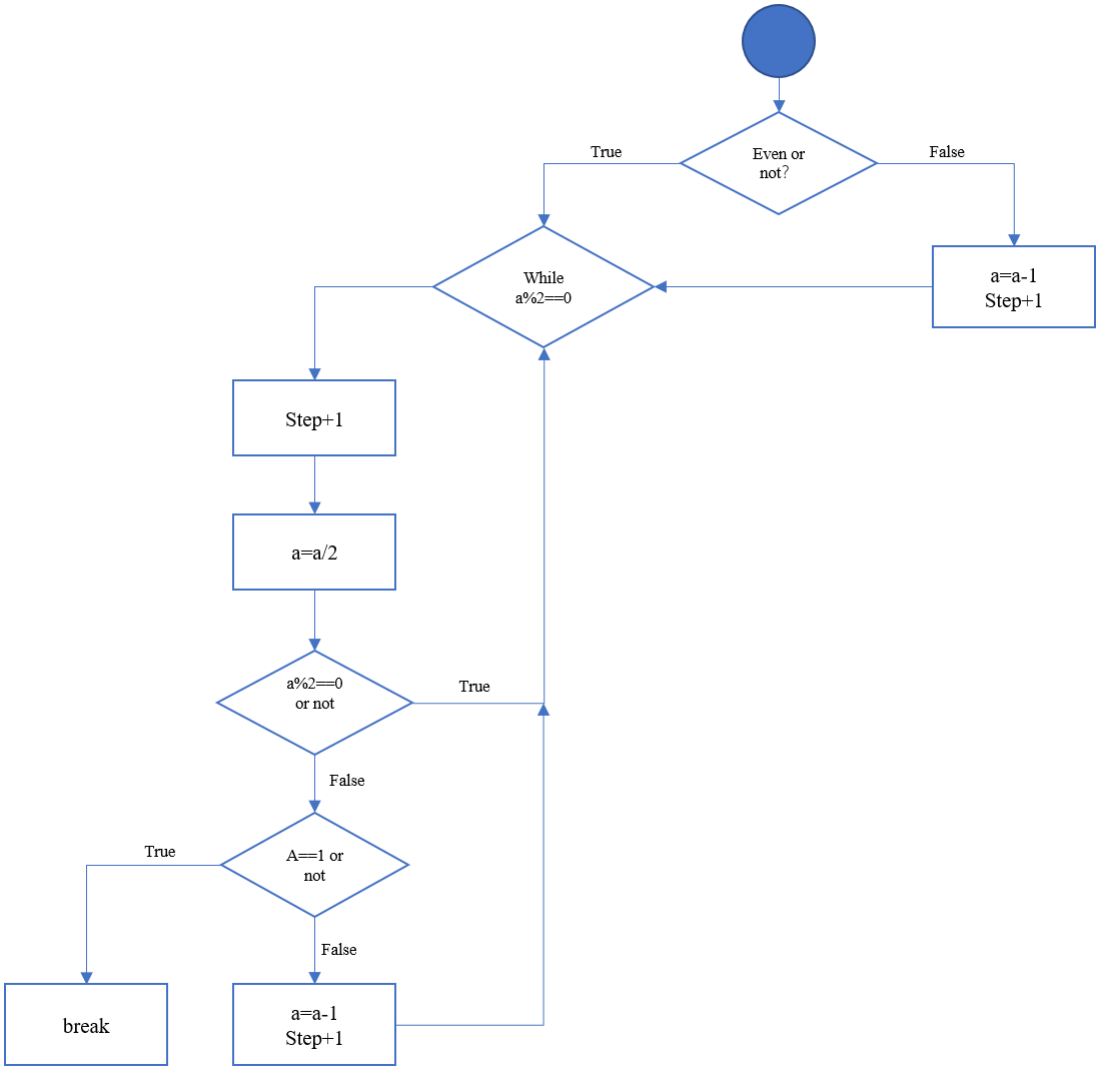
The first step is to use numpy to generate a matrix that meets the requirements, but whose elements are all zeros.
The matrix is then assigned a value (random integer) through 1 for loop.(Here I defined a function *random_matrix* to reach it.)
Finally, 3 for loops are used to multiply the matrix(*Matrix_multip* function)

PS1_3.py

The elements of each row of PASCAL triangle are actually the result of the expansion of binomials to different powers, which can be calculated by permutation and combination formula. The program considers the number of rows in the PASCAL triangle to be counted from row 1, not row 0.
Here are some reports(line 100),line 200 is too long, please see in the script:
Line 100

```
1 99 4851 156849 3764376 71523144 1120529256 14887031544 171200862756 1731030945644 15579278510796
126050526132804 924370524973896 6186171974825304 38000770702498296 215337700647490344 1130522928399324306
5519611944537877494 25144898858450330806 107196674080761936594 428786696323047746376 1613054714739084379224
5719012170438571889976 19146258135816088501224 60629817430084280253876 181889452290252840761628
517685364210719623706172 1399667836569723427057428 3599145865465003098147672 8811701946483283447189128
20560637875127661376774632 45764000431735762419272568 97248500917438495140954207 197443926105102399225573693
383273503615787010261407757 711793649572175876199757263 1265410932572757113244012912
2154618614921181030658724688 3515430371713505892127392912 5498493658321124600506947888
8247740487481686900760421832 11868699725888281149874753368 16390109145274293016493707032
21726423750712434928840495368 27651812046361280818524266832 33796659167774898778196326128
39674339023040098565708730672 44739148260023940935799206928 48467410615025936013782474172
50445672272782096667406248628 50445672272782096667406248628 48467410615025936013782474172
44739148260023940935799206928 39674339023040098565708730672 33796659167774898778196326128
27651812046361280818524266832 21726423750712434928840495368 16390109145274293016493707032
11868699725888281149874753368 8247740487481686900760421832 5498493658321124600506947888
3515430371713505892127392912 2154618614921181030658724688 1265410932572757113244012912
711793649572175876199757263 383273503615787010261407757 197443926105102399225573693 97248500917438495140954207
45764000431735762419272568 20560637875127661376774632 8811701946483283447189128 3599145865465003098147672
1399667836569723427057428 517685364210719623706172 181889452290252840761628 60629817430084280253876
19146258135816088501224 5719012170438571889976 1613054714739084379224 428786696323047746376
107196674080761936594 25144898858450330806 5519611944537877494 1130522928399324306 215337700647490344
38000770702498296 6186171974825304 924370524973896 126050526132804 15579278510796 1731030945644 171200862756
14887031544 1120529256 71523144 3764376 156849 4851 99 1
```

PS1_4.py



PS1_5.py

Use 0,1 and 2 to represent "","+","-" respectively. So the total number of combinations is 3^8. Here I've combined all the cases with 8 for loops. Then use eval() to determine if the string operation is equal to the desired value. If it's equal, put that combination in the list, then print it.

The length of the list "expression" for each number is stored in the new list "Total_solutions" by the for loop and calling the function “find_expression”.

Finally, the number(s) with the most and least number of combinations is obtained by comparison