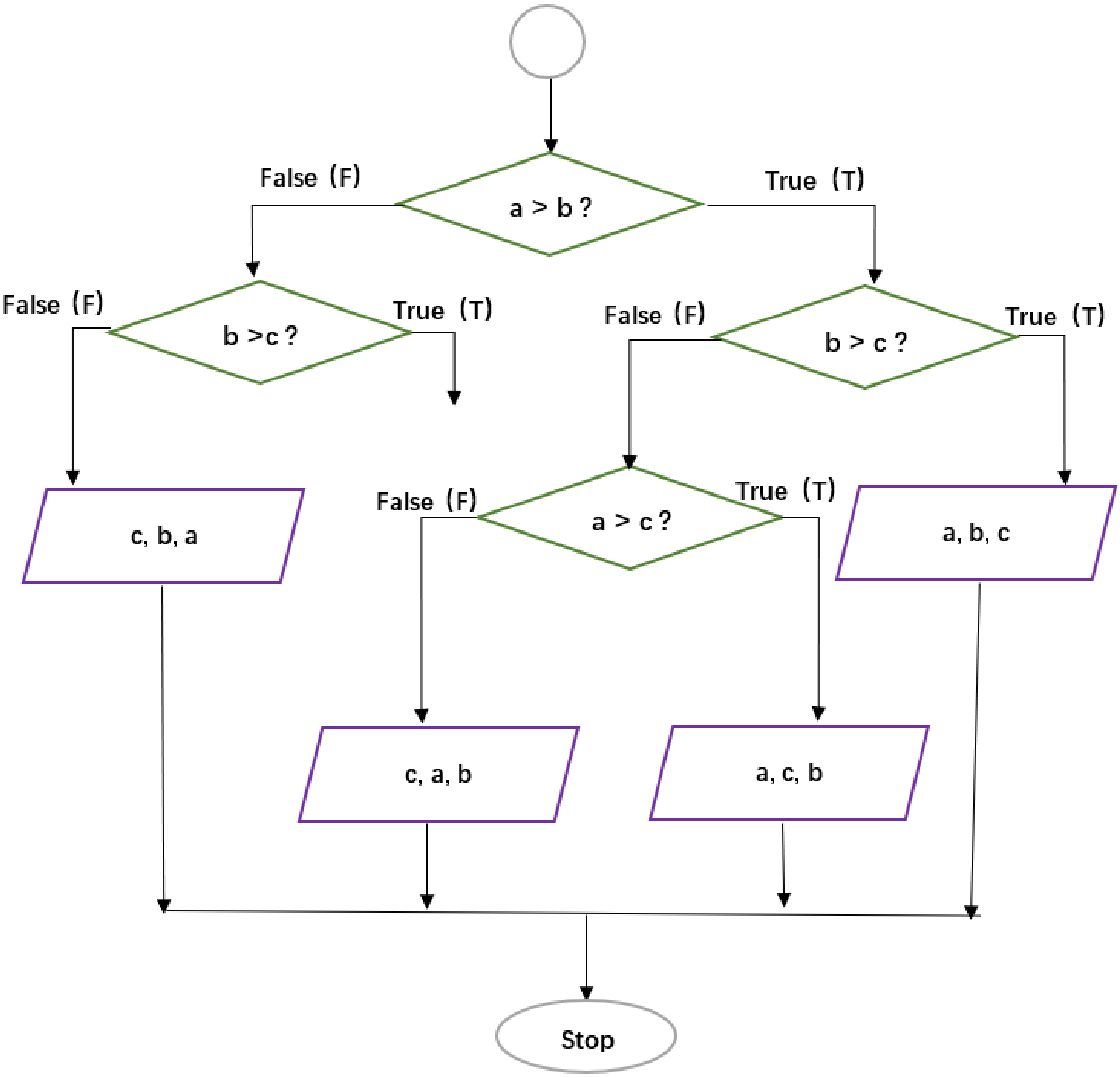
冯汇然, 12231095, Assignment 01, Due: 10/25 19:00

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.



In [5]:
```python
def Print_values(a,b,c):
    if a>b:
        if b>c:
            print(a,b,c)
        else:
            if a>c:
                print(a,c,b)
            else:
                print(c,a,b)
    else:
        if b>c:
            print(c,a,b)
        else:
            print(c,b,a)
```

In [7]:
```python
a = 3
b = 2
```

```
c = 1
Print_values(a,b,c)
```

```
3 2 1
```

In [8]:

```
a = 3
b = 1
c = 2
Print_values(a,b,c)
```

```
3 2 1
```

In [9]:

```
a = 2
b = 1
c = 3
Print_values(a,b,c)
```

```
3 2 1
```

In [10]:

```
a = 1
b = 2
c = 3
Print_values(a,b,c)
```

```
3 2 1
```

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.

In [34]:

```
import random
import numpy as np

# Always create the same random matrices by using seeds.
np.random.seed(12231095)

# Get 50 integers from 0 to 50, and using `reshape` to get the rows and columns
M1 = np.mat(np.random.randint(0,50,50).reshape(5,10))
M2 = np.mat(np.random.randint(0,50,50).reshape(10,5))

print(M1)
print(M2)
```

```
[[28 33 31 32  5 38 47 22 13 24]
 [39 41 35  5 23 17 49 46 21 11]
 [ 9 16  8 11 15 22 42 43 12 38]
 [32 19 17 32 45 20  3  5 12  6]
 [ 3 27  5 17 44 29 45 39 11 19]]
[[40 20 31 12 12]
 [29 24 11 11 41]
 [42 19 12  8 41]
 [30 32  6 34 48]
 [40 19 29  2 32]
 [14 46  4 17 47]
 [46  5  5  7 32]
 [22 34 31 40  9]
 [34 10 41  2 34]
 [37 13 32 28 27]]
```

Reference 2.1

(1) I search `python创建矩阵` in bing and refer to this page: CSDN: python numpy--矩阵的创建

(2) I search `python用随机种子创建随机矩阵` in bing and refer to this page: 知乎：numpy.random.seed()的作用

In [55]:

```
def Matrix_multip(m1,m2):
    if m1.shape[1] == m2.shape[0]:
        commonCR = m1.shape[1]
        newR = m1.shape[0]
        newC = m2.shape[1]
        m1_mul_m2 = np.zeros((newR,newC))
        for i in range(newR):
            for j in range(newC):
                for k in range(commonCR):
                    m1_mul_m2[i,j] += m1[i,k]*m2[k,j]
        print('the mutiplication of m1 and m2 is:\n',m1_mul_m2)
    else:
        print('m1 cannot mulitiple m2, because the rows of m2 and the columns o
```

In [56]:

```
# My function
Matrix_multip(M1,M2)
```

```
the mutiplication of m1 and m2 is:
 [[9047. 6233. 4310. 4598. 9234.]
 [9914. 5970. 5729. 4237. 8352.]
 [7090. 4651. 4391. 4228. 6299.]
 [6463. 4601. 3836. 2660. 6487.]
 [7794. 5425. 4437. 3961. 7613.]]
```

In [58]:

```
# Collaboration and validation
print('the result from numpy calculation is:\n',np.dot(M1,M2))
```

```
the result from numpy calculation is:
 [[9047 6233 4310 4598 9234]
 [9914 5970 5729 4237 8352]
 [7090 4651 4391 4228 6299]
 [6463 4601 3836 2660 6487]
 [7794 5425 4437 3961 7613]]
```

Reference 2.2

I search `python创建0矩阵` in bing and refer to this page: CSDN: 【Python】生成全0矩阵的方法

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)`.

In [195...

```
# n>2
def Pascal_triangle(n):
    old_tri = [1,1]
#     print('1')
#     print('1,1')
    for t in range(3,n+1):
        new_tri = np.zeros(t)#.astype(int)
        new_tri[0] = 1
        new_tri[t-1] = 1
#        print(1, end=',')
        for i in range(1,t-1):
            new_tri[i] = old_tri[i-1]+old_tri[i]
#            print(int(new_tri[i]), end=',')
#        print('1')
        old_tri = new_tri
    print(new_tri)
```

In [197…

```
Pascal_triangle(100)
```

```
[1.00000000e+00 9.90000000e+01 4.85100000e+03 1.56849000e+05
 3.76437600e+06 7.15231440e+07 1.12052926e+09 1.48870315e+10
 1.71200863e+11 1.73103095e+12 1.55792785e+13 1.26050526e+14
 9.24370525e+14 6.18617197e+15 3.80007707e+16 2.15337701e+17
 1.13052293e+18 5.51961194e+18 2.51448989e+19 1.07196674e+20
 4.28786696e+20 1.61305471e+21 5.71901217e+21 1.91462581e+22
 6.06298174e+22 1.81889452e+23 5.17685364e+23 1.39966784e+24
 3.59914587e+24 8.81170195e+24 2.05606379e+25 4.57640004e+25
 9.72485009e+25 1.97443926e+26 3.83273504e+26 7.11793650e+26
 1.26541093e+27 2.15461861e+27 3.51543037e+27 5.49849366e+27
 8.24774049e+27 1.18686997e+28 1.63901091e+28 2.17264238e+28
 2.76518120e+28 3.37966592e+28 3.96743390e+28 4.47391483e+28
 4.84674106e+28 5.04456723e+28 5.04456723e+28 4.84674106e+28
 4.47391483e+28 3.96743390e+28 3.37966592e+28 2.76518120e+28
 2.17264238e+28 1.63901091e+28 1.18686997e+28 8.24774049e+27
 5.49849366e+27 3.51543037e+27 2.15461861e+27 1.26541093e+27
 7.11793650e+26 3.83273504e+26 1.97443926e+26 9.72485009e+25
 4.57640004e+25 2.05606379e+25 8.81170195e+24 3.59914587e+24
 1.39966784e+24 5.17685364e+23 1.81889452e+23 6.06298174e+22
 1.91462581e+22 5.71901217e+21 1.61305471e+21 4.28786696e+20
 1.07196674e+20 2.51448989e+19 5.51961194e+18 1.13052293e+18
 2.15337701e+17 3.80007707e+16 6.18617197e+15 9.24370525e+14
 1.26050526e+14 1.55792785e+13 1.73103095e+12 1.71200863e+11
 1.48870315e+10 1.12052926e+09 7.15231440e+07 3.76437600e+06
 1.56849000e+05 4.85100000e+03 9.90000000e+01 1.00000000e+00]
```

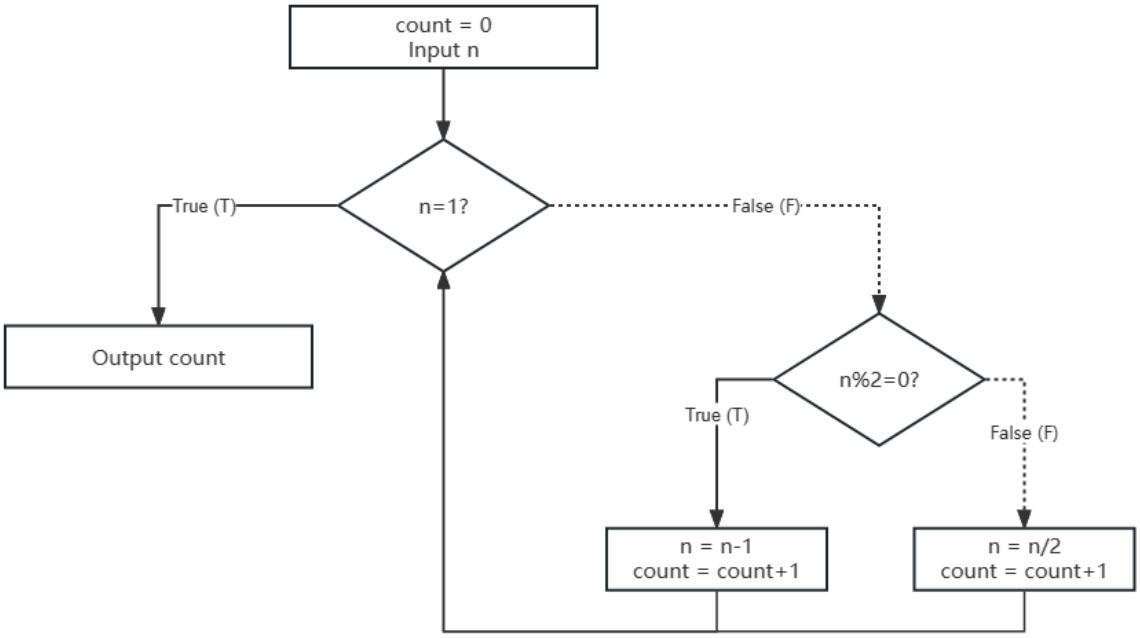
In [196…

```
Pascal_triangle(200)
```

```
[1.00000000e+00 1.99000000e+02 1.97010000e+04 1.29369900e+06
 6.33912510e+07 2.47225879e+09 7.99363675e+10 2.20395985e+12
 5.28950363e+13 1.12255022e+15 2.13284541e+16 3.66461620e+17
 5.74123205e+18 8.25854149e+19 1.09720623e+21 1.35322101e+22
 1.55620416e+23 1.67520801e+24 1.69382143e+25 1.61358779e+26
 1.45222901e+27 1.23785235e+28 1.00153508e+29 7.70746561e+29
 5.65214145e+30 3.95649902e+31 2.64781088e+32 1.69656030e+33
 1.04217276e+34 6.14522558e+34 3.48229449e+35 1.89841216e+36
 9.96666383e+36 5.04373594e+37 2.46252990e+38 1.16090695e+39
 5.28857612e+39 2.32983218e+40 9.93244246e+40 4.10031599e+41
 1.64012640e+42 6.36049017e+42 2.39275583e+43 8.73634104e+43
 3.09743000e+44 1.06689256e+45 3.57177074e+45 1.16272537e+46
 3.68196366e+46 1.13464594e+47 3.40393783e+47 9.94483799e+47
 2.83045389e+48 7.85050418e+48 2.12254372e+49 5.59579709e+49
 1.43891925e+50 3.60992023e+50 8.83808056e+50 2.11215145e+51
 4.92835339e+51 1.12301823e+52 2.49962123e+52 5.43568426e+52
 1.15508290e+53 2.39901834e+53 4.87073421e+53 9.66877089e+53
 1.87687905e+54 3.56335009e+54 6.61765016e+54 1.20236179e+55
 2.13753207e+55 3.71872018e+55 6.33187490e+55 1.05531248e+56
 1.72182563e+56 2.75044873e+56 4.30198392e+56 6.58911461e+56
 9.88367191e+56 1.45204563e+57 2.08952907e+57 2.94548074e+57
 4.06756864e+57 5.50318111e+57 7.29491449e+57 9.47500388e+57
 1.20590958e+58 1.50399959e+58 1.83822173e+58 2.20182602e+58
 2.58475229e+58 2.97385478e+58 3.35349582e+58 3.70649538e+58
 4.01536999e+58 4.26374340e+58 4.43777374e+58 4.52742573e+58
 4.52742573e+58 4.43777374e+58 4.26374340e+58 4.01536999e+58
 3.70649538e+58 3.35349582e+58 2.97385478e+58 2.58475229e+58
 2.20182602e+58 1.83822173e+58 1.50399959e+58 1.20590958e+58
 9.47500388e+57 7.29491449e+57 5.50318111e+57 4.06756864e+57
 2.94548074e+57 2.08952907e+57 1.45204563e+57 9.88367191e+56
 6.58911461e+56 4.30198392e+56 2.75044873e+56 1.72182563e+56
 1.05531248e+56 6.33187490e+55 3.71872018e+55 2.13753207e+55
 1.20236179e+55 6.61765016e+54 3.56335009e+54 1.87687905e+54
 9.66877089e+53 4.87073421e+53 2.39901834e+53 1.15508290e+53
 5.43568426e+52 2.49962123e+52 1.12301823e+52 4.92835339e+51
 2.11215145e+51 8.83808056e+50 3.60992023e+50 1.43891925e+50
 5.59579709e+49 2.12254372e+49 7.85050418e+48 2.83045389e+48
 9.94483799e+47 3.40393783e+47 1.13464594e+47 3.68196366e+46
 1.16272537e+46 3.57177074e+45 1.06689256e+45 3.09743000e+44
 8.73634104e+43 2.39275583e+43 6.36049017e+42 1.64012640e+42
 4.10031599e+41 9.93244246e+40 2.32983218e+40 5.28857612e+39
 1.16090695e+39 2.46252990e+38 5.04373594e+37 9.96666383e+36
 1.89841216e+36 3.48229449e+35 6.14522558e+34 1.04217276e+34
 1.69656030e+33 2.64781088e+32 3.95649902e+31 5.65214145e+30
 7.70746561e+29 1.00153508e+29 1.23785235e+28 1.45222901e+27
 1.61358779e+26 1.69382143e+25 1.67520801e+24 1.55620416e+23
 1.35322101e+22 1.09720623e+21 8.25854149e+19 5.74123205e+18
 3.66461620e+17 2.13284541e+16 1.12255022e+15 5.28950363e+13
 2.20395985e+12 7.99363675e+10 2.47225879e+09 6.33912510e+07
 1.29369900e+06 1.97010000e+04 1.99000000e+02 1.00000000e+00]
```

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`.



This picture was plotted on Procese On by myself.

In [209…

```python
# 1<= n <=100
def Least_moves(n):
    count = 0
    while n != 1:
        if n%2 == 0:
            n = n/2
            count += 1
        else:
            n = n-1
            count += 1
    print(count)
```

In [212…

```
for i in range(1,101):
    print(i,end=',')
    Least_moves(i)
```

```
1,0
2,1
3,2
4,2
5,3
6,3
7,4
8,3
9,4
10,4
11,5
12,4
13,5
14,5
15,6
16,4
17,5
18,5
19,6
20,5
21,6
22,6
23,7
24,5
25,6
26,6
27,7
28,6
29,7
30,7
31,8
32,5
33,6
34,6
35,7
36,6
37,7
38,7
39,8
40,6
41,7
42,7
43,8
44,7
45,8
46,8
47,9
48,6
49,7
50,7
51,8
52,7
53,8
54,8
55,9
56,7
57,8
```

```
58,8
59,9
60,8
61,9
62,9
63,10
64,6
65,7
66,7
67,8
68,7
69,8
70,8
71,9
72,7
73,8
74,8
75,9
76,8
77,9
78,9
79,10
80,7
81,8
82,8
83,9
84,8
85,9
86,9
87,10
88,8
89,9
90,9
91,10
92,9
93,10
94,10
95,11
96,7
97,8
98,8
99,9
100,8
```

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:

$$1-2+34+5+6+7+8-9=50 \tag{1}$$

and

$$1+2+34-56+78-9=50 \tag{2}$$

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`?

In [321…

```python
from itertools import product

def find_expression(target):
    valid_expressions = []

    # Generate all possible combinations of + and - operators
    for operators in product('+- ' , repeat=7):

        # Loop through all combinations and evaluate expressions
        for begin in ['1','1+','1-']:
            expression = begin
            num_str = '2'
            for op, num in zip(operators, range(3, 11)):
                if op == ' ':
                    num_str += str(num)
                else:
                    expression += num_str
                    expression += op
                    num_str = str(num)
            expression += num_str

            # Evaluate the expression
            result = eval(expression)

            # Check if the result matches the target
            if result == target:
                valid_expressions.append(expression)

    # Print the valid expressions
    for valid_expression in valid_expressions:
        print(valid_expression + '=' + str(target))

# Example usage:
find_expression(50)
```

```
1+2+3+4-56+7+89=50
12+3+4-56+78+9=50
1+2+3-4+56-7+8-9=50
1-2+3-45+6+78+9=50
1-2+34+5+6+7+8-9=50
1+2+34-5-6+7+8+9=50
1-2+34-5-67+89=50
1+2+34-56+78-9=50
1+2-3+4+56+7-8-9=50
1-2-3+4+56-7-8+9=50
12-3+45+6+7-8-9=50
12-3-4-5+67-8-9=50
1-2-3-4-5-6+78-9=50
1+2-34+5-6-7+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
```

In [313…

```python
def count_expression(target):
    count = 0
    valid_expressions = []

    # Generate all possible combinations of + and - operators
    for operators in product('+- ', repeat=7):

        # Loop through all combinations and evaluate expressions
        for begin in ['1','1+','1-']:
            expression = begin
            num_str = '2'
            for op, num in zip(operators, range(3, 11)):
                if op == ' ':
                    num_str += str(num)
                else:
                    expression += num_str
                    expression += op
                    num_str = str(num)
            expression += num_str

            # Evaluate the expression
            result = eval(expression)

            # Check if the result matches the target
            if result == target:
                valid_expressions.append(expression)
                count += 1

    return count

# Example usage:
count_expression(50)
```

Out[313]:

```
17
```

In [350…

```
Total_solutions = []
Total_numubers = []

for i in range(1,101):
    count = count_expression(i)
    Total_solutions.append(count)
    Total_numubers.append(str(i)+'-'+str(count))

print(Total_solutions,'\n')
print(Total_numubers)
```

```
[26, 11, 18, 8, 21, 12, 17, 8, 22, 12, 21, 11, 16, 15, 20, 8, 17, 11, 20, 15, 1
6, 11, 23, 18, 13, 14, 21, 15, 19, 17, 14, 19, 19, 7, 14, 19, 19, 17, 18, 16, 1
7, 18, 10, 15, 26, 18, 15, 16, 12, 17, 19, 9, 17, 21, 16, 13, 14, 16, 17, 17, 1
1, 13, 22, 14, 13, 15, 15, 15, 17, 7, 14, 17, 15, 12, 13, 14, 14, 14, 10, 9, 1
9, 12, 13, 13, 12, 11, 12, 6, 12, 14, 16, 13, 11, 11, 10, 11, 7, 9, 17, 11]

['1-26', '2-11', '3-18', '4-8', '5-21', '6-12', '7-17', '8-8', '9-22', '10-12',
'11-21', '12-11', '13-16', '14-15', '15-20', '16-8', '17-17', '18-11', '19-20',
'20-15', '21-16', '22-11', '23-23', '24-18', '25-13', '26-14', '27-21', '28-1
5', '29-19', '30-17', '31-14', '32-19', '33-19', '34-7', '35-14', '36-19', '37-
19', '38-17', '39-18', '40-16', '41-17', '42-18', '43-10', '44-15', '45-26', '4
6-18', '47-15', '48-16', '49-12', '50-17', '51-19', '52-9', '53-17', '54-21',
'55-16', '56-13', '57-14', '58-16', '59-17', '60-17', '61-11', '62-13', '63-2
2', '64-14', '65-13', '66-15', '67-15', '68-15', '69-17', '70-7', '71-14', '72-
17', '73-15', '74-12', '75-13', '76-14', '77-14', '78-14', '79-10', '80-9', '81
-19', '82-12', '83-13', '84-13', '85-12', '86-11', '87-12', '88-6', '89-12', '9
0-14', '91-16', '92-13', '93-11', '94-11', '95-10', '96-11', '97-7', '98-9', '9
9-17', '100-11']
```

In [328…

```
sol_max = max(Total_solutions)
num_max = Total_solutions.index(sol_max)+1
print('Number',num_max,'yields the maximum of Total_solutions: ',sol_max)

sol_min = min(Total_solutions)
num_min = Total_solutions.index(sol_min)+1
print('Number',num_min,'yields the minimum of Total_solutions: ',sol_min)
```

```
Number 1 yields the maximum of Total_solutions:  26
Number 88 yields the minimum of Total_solutions:  6
```

In [349…

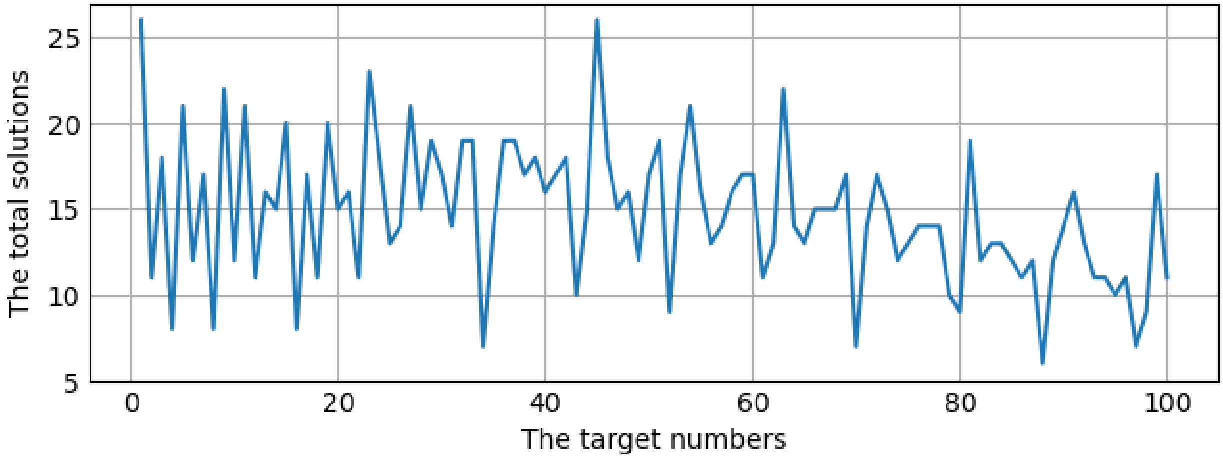
```
import matplotlib.pyplot as plt

# choose data
x = range(1,101)
y = Total_solutions

# plot
fig, ax = plt.subplots(figsize=(6,2))

ax.plot(x, y, linewidth=1.5)
ax.set_xlabel('The target numbers')
ax.set_ylabel('The total solutions')

plt.show()
```



Reference 5.2:

I draw the x-y plot after refering to page: matplotlib > Plot types > Pairwise data > plot(x, y) and CSDN：Spyder使用弹出绘图窗口的设置方法

In [ ]:

```

```