PS1_1

Train of thought:

To print the values in the given order, we can do comparisons by using “if” structure.

The script is as follows:

```
a = float(input("Please write down the first number: "))
b = float(input("Please write down the second number: "))
c = float(input("Please write down the third number: "))

def Print_values():

    if a > b:
        if b > c:
            return a, b, c
        elif a > c:
            return a, c, b
        else:
            return c, a, b
    elif b > c:
        if a > c:
            return b, a, c
        else:
            return b, c, a
    else:
        return c, b, a

print(Print_values())
```

Result:

Set 1, 2, 3 in different order:

```
In [1]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 1
Please write down the second number: 2
Please write down the third number: 3
(3.0, 2.0, 1.0)

In [2]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 1
Please write down the second number: 3
Please write down the third number: 2
(3.0, 2.0, 1.0)

In [3]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 2
Please write down the second number: 1
Please write down the third number: 3
(3.0, 2.0, 1.0)
```

```
In [4]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 2
Please write down the second number: 3
Please write down the third number: 1
(3.0, 2.0, 1.0)

In [5]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 3
Please write down the second number: 1
Please write down the third number: 2
(3.0, 2.0, 1.0)

In [6]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_1.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
Please write down the first number: 3
Please write down the second number: 2
Please write down the third number: 1
(3.0, 2.0, 1.0)
```

It turns out that the script is right.

PS1_2

Train of thought:

2.1 Use numpy library to generate two arrays, each of which including 50 random integers. Then use reshape() to transform them into 5×10 and 10×5 matrices respectively.

2.2 The result of a $m\times n$ matrix ($\boldsymbol{A}$) multiplied by a $n\times l$ matrix ($\boldsymbol{B}$) is a $m\times l$ matrix ($\boldsymbol{C}=\boldsymbol{AB}$). Each element ($c_{ij}$) in $\boldsymbol{C}$ can be calculated as the following formula:

$$c_{ij}=\sum_k a_{ik}b_{kj}$$

where a_{ik} is the element of ith row and kth column in $\boldsymbol{A}$, b_{kj} is the element of kth row and jth column in $\boldsymbol{B}$, c_{ij} is the element of ith row and jth column in $\boldsymbol{C}$.

Create a 5×5 matrix called $\boldsymbol{M}$ in Python, and $\boldsymbol{M}=\boldsymbol{M_1M_2}$. For each m and l, $\mathbf{M}[\mathrm{m,l}]=\sum_n \mathbf{M1}[\mathrm{m,n}]\times\mathbf{M2}[\mathrm{n,l}]$. Here m∈[1,5], n∈[1,10], l∈[1,5].

The script is as follows:

```
import numpy as np
arr1 = np.random.randint(0, 51, 50)
arr2 = np.random.randint(0, 51, 50)
M1 = arr1.reshape(5, 10)
M2 = arr2.reshape(10, 5)
print(M1)
print(M2)

def Matrix_multip():
    M = np.zeros((5, 5))
    for m in range(5):
        for l in range(5):
            for n in range(10):
                M[m,l] += M1[m,n]*M2[n,l]
    return M

print(Matrix_multip())
```

Result:

(In the following screenshot, the first matrix is M1, the second matrix is M2, and the third matrix is M.)

```
In [5]: runfile('C:/ESE5023/ESE5023_Assignments_12332283/PS1_2.py', wdir='C:/ESE5023/
ESE5023_Assignments_12332283')
[[ 9 26 28  1 39 23  9 46  0 42]
 [33 12  7 32 48 13 14 41 23 26]
 [ 2 19 46 16 19  2 16  0 20  6]
 [30  4 32 21 49 46  0 36 11 34]
 [22 40 27 19 43 29 12 48  9 38]]
[[ 3 43  1 48 14]
 [50 20  9 27 23]
 [13  5 46 15 10]
 [49 25 26 17  6]
 [47  3 12 10 14]
 [24 14 19 38 42]
 [23 50  1 41 43]
 [11 45 10 36 15]
 [ 5 41 18  4  4]
 [ 4 35 41 50 49]]
[[5006. 5501. 4653. 6960. 5657.]
 [5918. 7218. 4022. 6973. 4801.]
 [3771. 3011. 3593. 2873. 2433.]
 [5729. 6107. 5498. 7663. 5826.]
 [7066. 7350. 5397. 8552. 6566.]]
```

PS1_3

Train of thought:

For the k-th element in the n-th row of the Pascal Triangle, it can be calculated as:

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

In Python, I first define a function factorial(n) to calculate $n!$, then define another function Pascal_triangle(k) and use the formula above to calculate each element of k-th row of the Pascal triangle.

The script is as follows:

```
def factorial(n):
    y = 1
    for i in range(1, n+1):
        y = y * i
    return y

def Pascal_triangle(k):
    for i in range(1, k+1):
        x_ki = factorial(k-1) / (factorial(i-1) * factorial(k-i))
        print(x_ki)

print(Pascal_triangle(100))
print(Pascal_triangle(200))
```

The result of k=100:

1.0
99.0
4851.0
156849.0
3764376.0
71523144.0
1120529256.0
14887031544.0
171200862756.0
1731030945644.0
15579278510796.0
126050526132804.0
924370524973896.0
6186171974825304.0
3.8000770702498296e+16
2.1533770064749034e+17
1.1305229283993243e+18
5.519611944537878e+18
2.514489885845033e+19
1.0719667408076194e+20
4.2878669632304775e+20
1.6130547147390845e+21
5.719012170438572e+21
1.914625813581609e+22
6.062981743008428e+22
1.8188945229025285e+23
5.1768536421071965e+23
1.3996678365697236e+24
3.599145865465003e+24
8.811701946483283e+24
2.0560637875127662e+25
4.576400043173577e+25
9.72485009174385e+25
1.974439261051024e+26
3.8327350361578704e+26
7.117936495721758e+26
1.265410932572757e+27
2.154618614921181e+27
3.515430371713506e+27
5.498493658321124e+27
8.247740487481687e+27
1.1868699725888282e+28
1.6390109145274292e+28
2.1726423750712436e+28
2.765181204636128e+28
3.37966591677749e+28
3.96743390230401e+28
4.473914826002394e+28
4.846741061502594e+28
5.04456722727821e+28
5.04456722727821e+28
4.846741061502594e+28
4.473914826002394e+28
3.96743390230401e+28
3.37966591677749e+28
2.765181204636128e+28
2.1726423750712436e+28
1.6390109145274292e+28
1.1868699725888282e+28
8.247740487481687e+27
5.498493658321124e+27
3.515430371713506e+27
2.154618614921181e+27
1.265410932572757e+27
7.117936495721758e+26
3.8327350361578704e+26
1.974439261051024e+26
9.72485009174385e+25
4.576400043173577e+25
2.0560637875127662e+25
8.811701946483283e+24
3.599145865465003e+24
1.3996678365697236e+24
5.1768536421071965e+23
1.8188945229025285e+23
6.062981743008428e+22
1.914625813581609e+22
5.719012170438572e+21
1.6130547147390845e+21
4.2878669632304775e+20
1.0719667408076194e+20
2.514489885845033e+19
5.519611944537878e+18
1.1305229283993243e+18
2.1533770064749034e+17
3.8000770702498296e+16
6186171974825304.0
924370524973896.0
126050526132804.0
15579278510796.0
1731030945644.0
171200862756.0
14887031544.0
1120529256.0
71523144.0
3764376.0
156849.0
4851.0
99.0
1.0

The result of k=200:

1.0
199.0
19701.0
1293699.0
63391251.0
2472258789.0
79936367511.0
2203959847089.0
52895036330136.0
1122550215450664.0
2.1328454093562616e+16
3.664616203348486e+17
5.741232051912628e+18
8.258541490058933e+19
1.0972062265364012e+21
1.3532210127282282e+22
1.5562041646374625e+23
1.6752080125215036e+24
1.6938214348828537e+25
1.61358778796735e+26
1.452229009170615e+27
1.2378523459120956e+28
1.0015350798743319e+29
7.707465614685076e+29
5.652141450769056e+30
3.956499015538339e+31
2.6478108796295038e+32
1.6965603043552007e+33
1.0421727583896233e+34
6.145225575331917e+34
3.482294492688086e+35
1.8984121589170535e+36
9.96666383431453e+36
5.043735940395535e+37
2.462529900310761e+38
1.1609069530036446e+39
5.2885761192388254e+39
2.3298321822592664e+40
9.932442461210556e+40
4.100315990397178e+41
1.6401263961588712e+42
6.360490170469769e+42
2.3927558260338655e+43
8.736341039239928e+43
3.097430004821429e+44
1.06689255721627e+45
3.571770735028382e+45
1.1627253669347713e+46
3.6819636619601086e+46
1.134645944808115e+47
3.4039378344243454e+47
9.944837986847597e+47
2.830453888564316e+48
7.850504181489708e+48
2.1225437231435135e+49
5.595797088287444e+49
1.4389192512739143e+50
3.609920226880171e+50
8.838080555465246e+50
2.1121514547806774e+51
4.9283533944882477e+51
1.123018232514535e+52
2.4996212272097715e+52
5.435684255995853e+52
1.1550829043991188e+53
2.399018339905862e+53
4.870734205263416e+53
9.66877088507514e+53
1.8768790541616448e+54
3.563350088335876e+54
6.617650164052342e+54
1.2023617903700734e+55
2.1375320717690194e+55
3.7187201796529513e+55
6.33187490049016e+55
1.0553124834150268e+56
1.7218256308350438e+56
2.7504487349702646e+56
4.301983918799644e+56
6.589114609807051e+56
9.883671914710577e+56
1.4520456269759982e+57
2.089529072965461e+57
2.9454807414091436e+57
4.067568642898341e+57
5.503181105097756e+57
7.294914488152839e+57
9.475003875416906e+57
1.2059095841439698e+58
1.503999593707648e+58
1.8382217256426806e+58
2.2018260230225515e+58
2.584752287896039e+58
2.9738547828481305e+58
3.3534958189564025e+58
3.70649537884655e+58
4.015369993750429e+58
4.2637433954257135e+58
4.437773738096151e+58
4.527425732805164e+58
4.527425732805164e+58
4.437773738096151e+58

4.2637433954257135e+58
4.015369993750429e+58
3.70649537884655e+58
3.3534958189564025e+58
2.9738547828481305e+58
2.584752287896039e+58
2.2018260230225515e+58
1.8382217256426806e+58
1.503999593707648e+58
1.2059095841439698e+58
9.475003875416906e+57
7.294914488152839e+57
5.503181105097756e+57
4.067568642898341e+57
2.9454807414091436e+57
2.089529072965461e+57
1.4520456269759982e+57
9.883671914710577e+56
6.589114609807051e+56
4.301983918799644e+56
2.7504487349702646e+56
1.7218256308350438e+56
1.0553124834150268e+56
6.33187490049016e+55
3.7187201796529513e+55
2.1375320717690194e+55
1.2023617903700734e+55
6.617650164052342e+54
3.563350088335876e+54
1.8768790541616448e+54
9.66877088507514e+53
4.870734205263416e+53
2.399018339905862e+53
1.1550829043991188e+53
5.435684255995853e+52
2.4996212272097715e+52
1.123018232514535e+52
4.9283533944882477e+51
2.1121514547806774e+51
8.838080555465246e+50
3.609920226880171e+50
1.4389192512739143e+50
5.595797088287444e+49
2.1225437231435135e+49
7.850504181489708e+48
2.830453888564316e+48
9.944837986847597e+47
3.4039378344243454e+47
1.134645944808115e+47
3.6819636619601086e+46
1.1627253669347713e+46
3.571770735028382e+45
1.06689255721627e+45
3.097430004821429e+44
8.736341039239928e+43
2.3927558260338655e+43
6.360490170469769e+42
1.6401263961588712e+42
4.100315990397178e+41
9.932442461210556e+40
2.3298321822592664e+40
5.2885761192388254e+39
1.1609069530036446e+39
2.462529900310761e+38
5.043735940395535e+37
9.96666383431453e+36
1.8984121589170535e+36
3.482294492688086e+35
6.145225575331917e+34
1.0421727583896233e+34
1.6965603043552007e+33
2.6478108796295038e+32
3.956499015538339e+31
5.652141450769056e+30
7.707465614685076e+29
1.0015350798743319e+29
1.2378523459120956e+28
1.452229009170615e+27
1.61358778796735e+26
1.6938214348828537e+25
1.6752080125215036e+24
1.5562041646374625e+23
1.3532210127282282e+22
1.0972062265364012e+21
8.258541490058933e+19
5.741232051912628e+18
3.664616203348486e+17
2.1328454093562616e+16
1122550215450664.0
52895036330136.0
2203959847089.0
79936367511.0
2472258789.0
63391251.0
1293699.0
19701.0
199.0
1.0

PS1_4

Train of thought:

Let's think reversely — if starting with x, this problem can be transferred to: "If you start with number x and, with each move you can either halve or subtract 1 , what is the smallest number of moves you have to make to get to 1?"

So we have to find the fastest reduction of each move. Obviously, reducing to half is faster than minus 1. But when the number is odd, we can't divide it into two equal integers, and it's the time when we have to subtract 1. If the number is even, we can simply halve it.

The script is as follows:

```
def Least_moves(x):
    n = 0
    while x > 1:
        if x %2 == 0:
            x = x / 2
        else:
            x = x - 1
        n = n + 1
    return n
```

Result:

The result of Least_moves(5) is 3.

The result of Least_moves(60) is 8. (60←30←15←14←7←6←3←2←1)

PS1_5

5.1

[Note: I sought ChatGLM for help and referred to the functions on it (its script is wrong and I write the script by myself). Also, I asked Zhe Yu for the train of thought of this problem and felt grateful to him.]

Train of thought:

Among 9 digits, there are 8 vacancies in which operators can be put. Each vacancy can be filled with “+”, “-” or “Null” (“Null” is to simply join the two neighboring digits together to form a bigger number). Arrange them and we can get 3^8 expressions. (Here I use “product” function in itertools library to achieve full permutation.) Now let the computer try one by one. Print the expression if it equals to a given number.

The script is as follows:

```
digits = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
operators = ['+', '-', '']
def Find_expression(x):
    import itertools
    for op1, op2, op3, op4, op5, op6, op7, op8 in itertools.product(operators, repeat=8):
        expression = digits[0] + op1 + digits[1] + op2 + digits[2] + op3 + digits[3] +
        op4 + digits[4] + op5 + digits[5] + op6 + digits[6] + op7 + digits[7] + op8 +
        digits[8]
        if eval(expression) == x:
            print(expression+'='+str(x))
    return expression
Find_expression(50)
```

Result:

The screenshot on the right side shows the result of Find_expression(50).

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

5.2

[Note: I asked ChatGLM how to plot the list.]

Train of thought:

Based on 5.1, we only have to add a list called Total_solutions and a variable to count the total solutions in the structure of function “Find_expression”. Then use matplot library to Plot the list “Total_solutions”. Lastly we can do a for-loop to find which number(s) yields the maximum and minimum of Total_solutions.

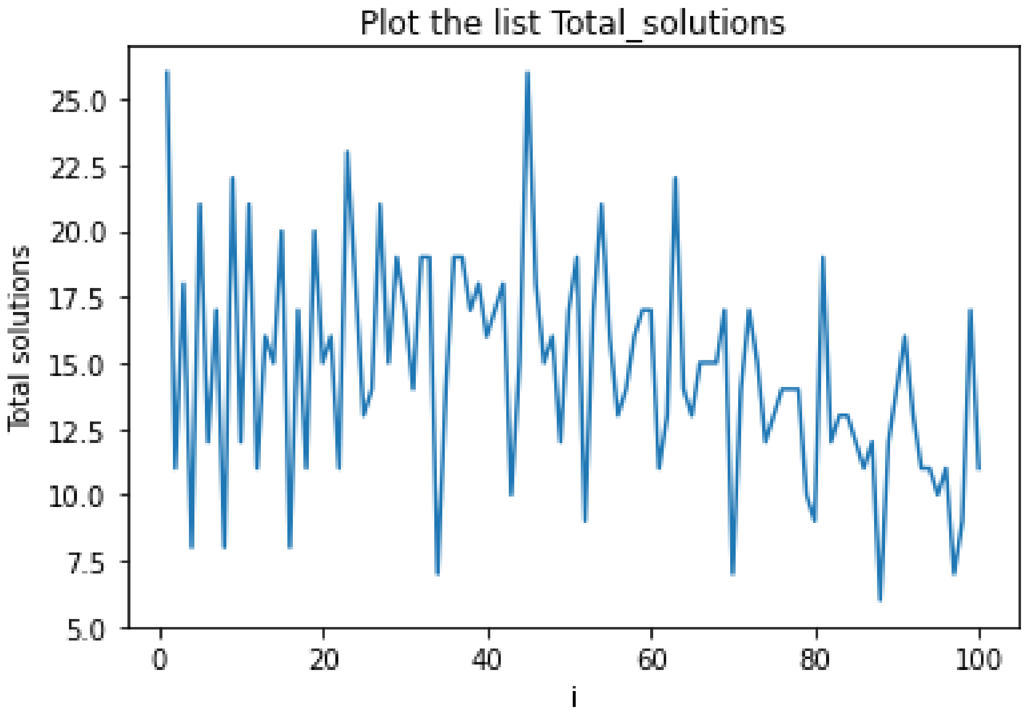
The script is as follows:

```
import itertools
Total_solutions = []
for i in range(1, 101):
    n = 0
    for op1, op2, op3, op4, op5, op6, op7, op8 in itertools.product(operators, repeat=8):
        expression = digits[0] + op1 + digits[1] + op2 + digits[2] + op3 + digits[3] +
        op4 + digits[4] + op5 + digits[5] + op6 + digits[6] + op7 + digits[7] + op8 +
        digits[8]
        if eval(expression) == i:
            n += 1
    Total_solutions.append(n)
print(Total_solutions)

import matplotlib.pyplot as plt
i_list = list(range(1, 101))
plt.figure()
plt.plot(i_list, Total_solutions)
plt.title('Plot the list Total_solutions')
plt.xlabel('i')
plt.ylabel('Total solutions')
plt.show()

for i in range(100):
    if Total_solutions[i] == max(Total_solutions):
        maximum = i + 1
        print(str(maximum) + ' yields the maximum of Total_solutions.')
    elif Total_solutions[i] == min(Total_solutions):
        minimum = i + 1
        print(str(minimum) + ' yields the minimum of Total_solutions.')
    else:
        pass
```

The result is as the following chart:



1 and 45 yield the maximum of Total_solutions, and 88 yields the minimum of Total_solutions.

```
1 yields the maximum of Total_solutions.
45 yields the maximum of Total_solutions.
88 yields the minimum of Total_solutions.
```