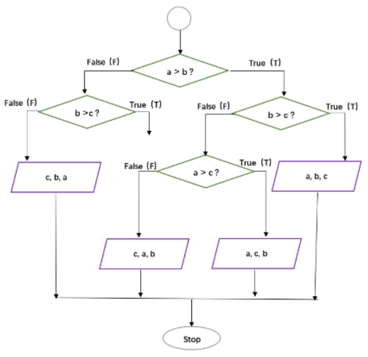
1. Flowchart

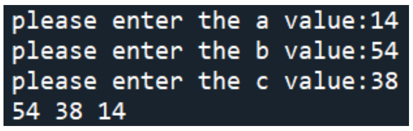
[10 points] Write a function Print_values with arguments a, b, and c to reflect the following flowchart. Here the purple parallelogram operator is to print values in the given order. Report your output with some random a, b, and c values.



Program:

```
#define the Print_values function
def Print_values(a, b, c):
    if a >= b and a >= c:
        if b >= c:
            print(a, b, c)
        else:
            print(a, c, b)
    elif b >= a and b >= c:
        if a >= c:
            print(b, a, c)
        else:
            print(b, c, a)
    else:
        if a >= b:
            print(c, a, b)
        else:
            print(c, b, a)
#enter the values
a = int(input("please enter the a value:"))
b = int(input("please enter the b value:"))
c = int(input("please enter the c value:"))
Print_values(a, b, c)
```

The result of the function:



```
please enter the a value:14
please enter the b value:54
please enter the c value:38
54 38 14
```

Problem-solving ideas:

According to the flowchart given by the title, six sorting results can be obtained, based on the above flowchart, through the if statement, compare the size of A, B, C, and output the value according to the comparison result

2. Matrix multiplication

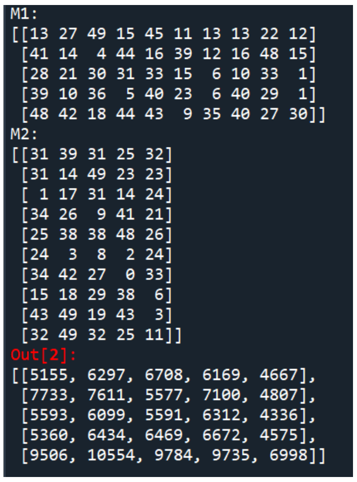
2.1 [5 points] Make two matrices M1 (5 rows and 10 columns) and M2 (10 rows and 5 columns); both are filled with random integers from 0 and 50.

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.

Program：

```
#Q2-1
import numpy as np
# Generate 5×10 matrix M1
M1 = np.random.randint(0, 50, size=(5, 10))
# Generate 10×5 matrix M2
M2 = np.random.randint(0, 50, size=(10, 5))
print("M1:")
print(M1)
print("M2:")
print(M2)
#Q2-2
def Matrix_multip(M1, M2):
    result = [[0 for j in range(len(M2[0]))] for i in range(len(M1))]

    for i in range(len(M1)):
        for j in range(len(M2[0])):
            for k in range(len(M2)):
                result[i][j] += M1[i][k] * M2[k][j]

    return result
Matrix_multip(M1, M2)
```

The result of the function：



```
M1:
[[13 27 49 15 45 11 13 13 22 12]
 [41 14  4 44 16 39 12 16 48 15]
 [28 21 30 31 33 15  6 10 33  1]
 [39 10 36  5 40 23  6 40 29  1]
 [48 42 18 44 43  9 35 40 27 30]]
M2:
[[31 39 31 25 32]
 [31 14 49 23 23]
 [ 1 17 31 14 24]
 [34 26  9 41 21]
 [25 38 38 48 26]
 [24  3  8  2 24]
 [34 42 27  0 33]
 [15 18 29 38  6]
 [43 49 19 43  3]
 [32 49 32 25 11]]
Out[2]:
[[5155, 6297, 6708, 6169, 4667],
 [7733, 7611, 5577, 7100, 4807],
 [5593, 6099, 5591, 6312, 4336],
 [5360, 6434, 6469, 6672, 4575],
 [9506, 10554, 9784, 9735, 6998]]
```

Problem-solving ideas：

In the first question, according to the random function, 50 integers in the range of 0 and 50 are randomly generated, where M1 is placed according to 5 rows and 10 columns, M2 is placed according to 10 rows and 5 columns, and in the second question, the matrix product is calculated by the for loop, and the number of loops can be identified by the range function, the size of the number of rows and columns of the matrix can be identified

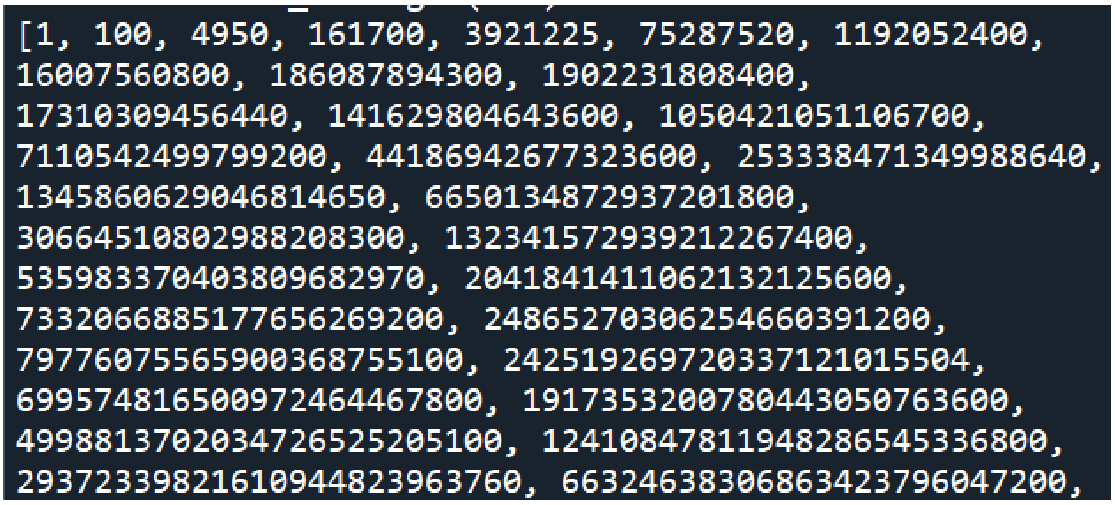
3. Pascal triangle

[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 kth line of the Pascal triangle. Report Pascal_triangle(100) and Pascal_triangle(200).

Program:

```
#define the Pascal_triangle function
def Pascal_triangle(k):
    row = [1]
    for i in range(k):
        row.append(row[i]*(k-i)//(i+1))
    print(row)

# Report Pascal_triangle(100) and Pascal_triangle(200)
Pascal_triangle(100)
Pascal_triangle(200)
```

Part of the result of the function:



```
[1, 100, 4950, 161700, 3921225, 75287520, 1192052400,
16007560800, 186087894300, 1902231808400,
17310309456440, 141629804643600, 1050421051106700,
7110542499799200, 44186942677323600, 253338471349988640,
1345860629046814650, 6650134872937201800,
30664510802988208300, 132341572939212267400,
535983370403809682970, 2041841411062132125600,
7332066885177656269200, 24865270306254660391200,
79776075565900368755100, 242519269720337121015504,
699574816500972464467800, 1917353200780443050763600,
4998813702034726525205100, 12410847811948286545336800,
29372339821610944823963760, 66324638306863423796047200,
```

Problem-solving ideas:

The value of each number in Pascal's triangle is the value of the number in the upper left corner plus the value in the upper right corner (if not, it is recorded as 0), where the first line is 1, so it can be directly calculated with the plus operator, and the number of loops can be identified by the range function as the number of loops

4. Add or double

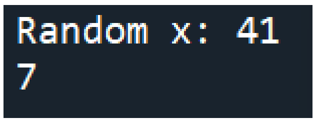
[20 points] If you start with 1 RMB and, with each move, you can either double your money or add another 1 RMB, what is the smallest number of moves you have to make to get to exactly x RMB? Here x is an integer randomly selected from 1 to 100. Write a function Least_moves to print your results. For example, Least_moves(2) should print 1, and Least_moves(5) should print 3.

import random

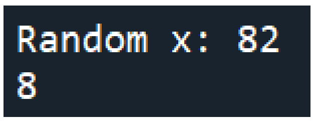
Program:

```
#define the Least_moves function
import random

def Least_moves(x):
    moves = 0

    while x > 1:
        if x%2 == 0:
            x = x / 2
            moves += 1
        else:
            x = x - 1
            moves += 1

    print(moves)
#test
x = random.randint(1, 100)
print("Random x:", x)
Least_moves(x)
```

The result of the function:



```
Random x: 41
7
```



```
Random x: 82
8
```

Problem-solving ideas:

This problem can be programmed in reverse, if the target value is even, divide by 2, if not, subtract one, and each time a loop is realized, add one step until the value is 1

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

and

1+2+34−56+78−9=50

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?

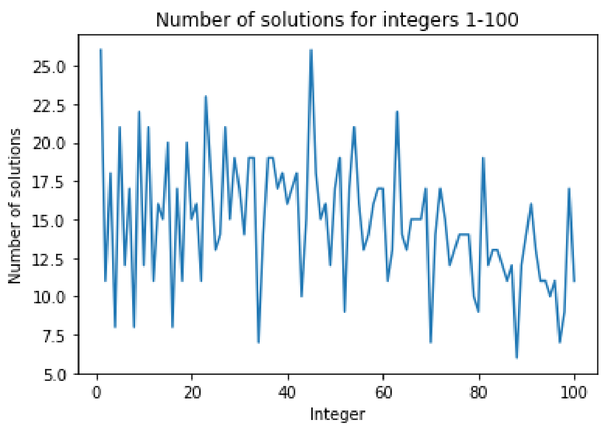
import random

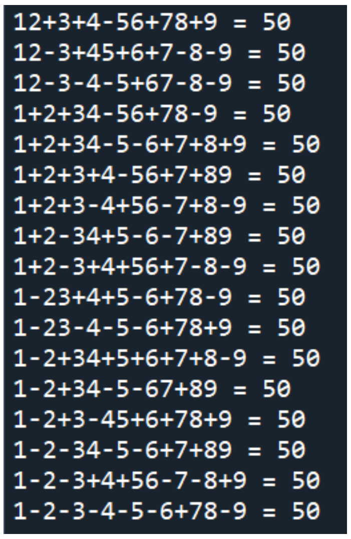
Program：

```
#Q5-1
from functools import reduce

operator = {
    1: '+',
    2: '-',
    0: ''
}

base = ['1', '2', '3', '4', '5', '6', '7', '8', '9']

def operator_evaluation(num):

    arr = []
    for index in range(8):
        index = 7 - index
        arr.append(num // (3 ** index))
        num -= (num // (3 ** index)) * (3 ** index)
    arr = map(lambda x: operator[x], arr)

    formula = reduce(lambda x, y: x + y, zip(base, arr))

    formula = list(formula)
    formula.append('9')

    formula = ''.join(formula)
    res = eval(formula)
    return res, formula
```

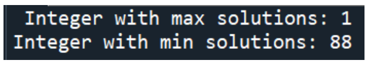
```
def Find_expression(target):
    total = 3 ** 8
    for i in range(total):
        res, formula = operator_evaluation(i)
        if res == target:
            print(formula + ' = %d' % target)
#test
Find_expression(50)
#Q5-2
import matplotlib.pyplot as plt

# Existing code

Total_solutions = []
for i in range(1, 101):
    count = 0
    for j in range(3**8):
        res, formula = operator_evaluation(j)
        if res == i:
            count += 1
    Total_solutions.append(count)

plt.plot(range(1, 101), Total_solutions)
plt.xlabel('Integer')
plt.ylabel('Number of solutions')
plt.title('Number of solutions for integers 1-100')
plt.show()
# Get index of max and min counts
max_idx = Total_solutions.index(max(Total_solutions))
min_idx = Total_solutions.index(min(Total_solutions))

print("Integer with max solutions:", max_idx + 1)
print("Integer with min solutions:", min_idx + 1)
```

The result of the function:





```
12+3+4-56+78+9 = 50
12-3+45+6+7-8-9 = 50
12-3-4-5+67-8-9 = 50
1+2+34-56+78-9 = 50
1+2+34-5-6+7+8+9 = 50
1+2+3+4-56+7+89 = 50
1+2+3-4+56-7+8-9 = 50
1+2-34+5-6-7+89 = 50
1+2-3+4+56+7-8-9 = 50
1-23+4+5-6+78-9 = 50
1-23-4-5-6+78+9 = 50
1-2+34+5+6+7+8-9 = 50
1-2+34-5-67+89 = 50
1-2+3-45+6+78+9 = 50
1-2-34-5-6+7+89 = 50
1-2-3+4+56-7-8+9 = 50
1-2-3-4-5-6+78-9 = 50
```



```
 Integer with max solutions: 1
Integer with min solutions: 88
```

Problem-solving ideas:

The main idea is to place three types of operators (+/-/no operator) at eight intervals, while considering the aesthetics of the program. Instead of nesting the for loop eight times, the for loop reads a ternary value and corresponds each bit of value to an operator. Then, the value is calculated. If it is the target value, solution is added by 1