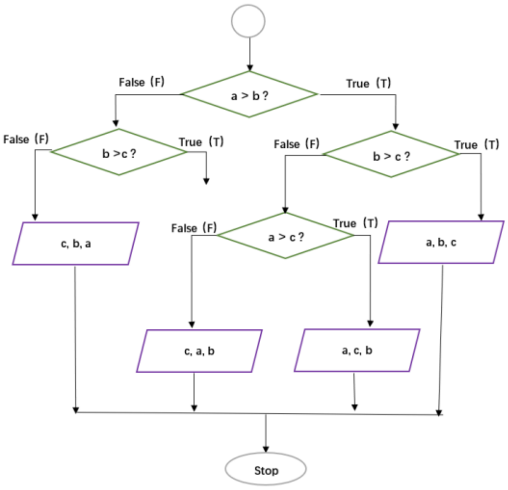
1. Flowchart

[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.



```
#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)
```

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.

```
#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)
```

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).

```
#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)
```

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

#define the Least_moves function
def Least_moves(x):
    moves = 0
    money = 1

    while money < x:
        if money*2 <= x:
            money = money * 2
            moves += 1
        else:
            money += 1
            moves += 1

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

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

#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)
```