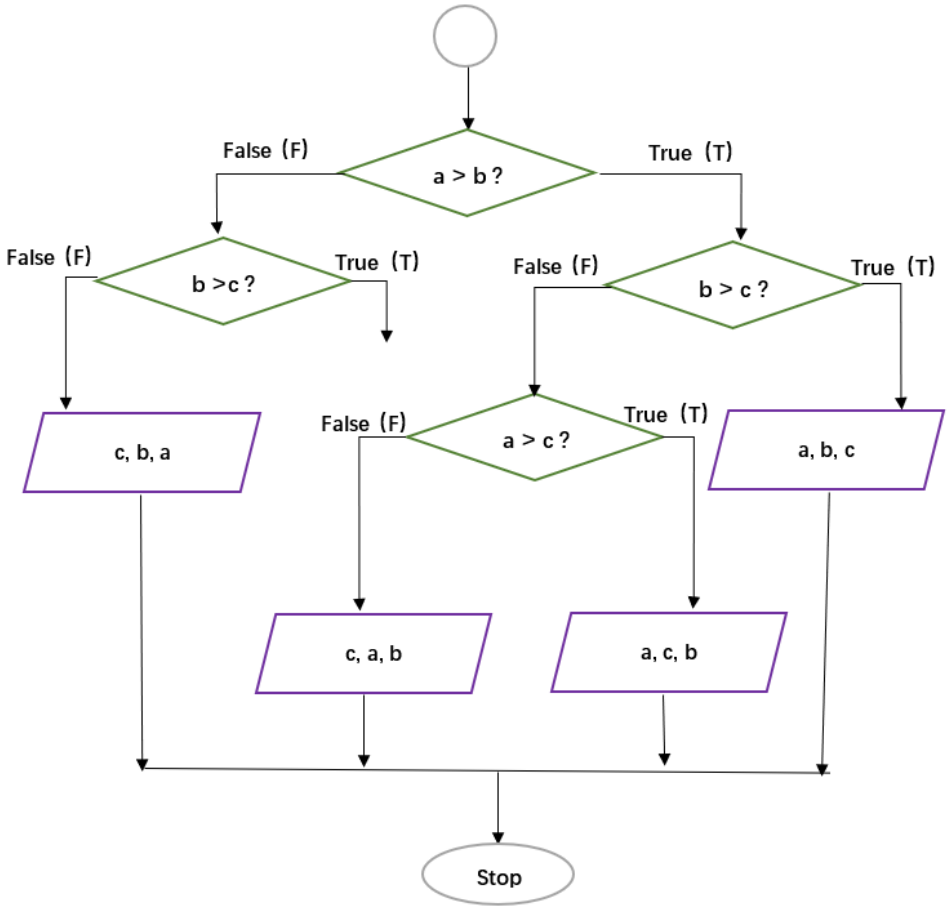
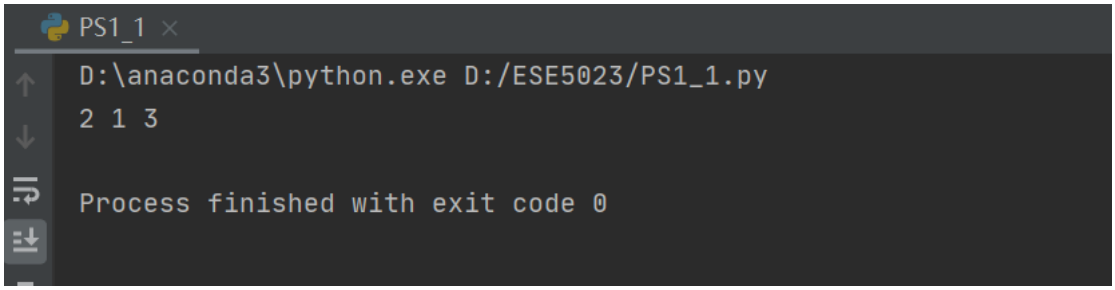
1. Flowchart

[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.



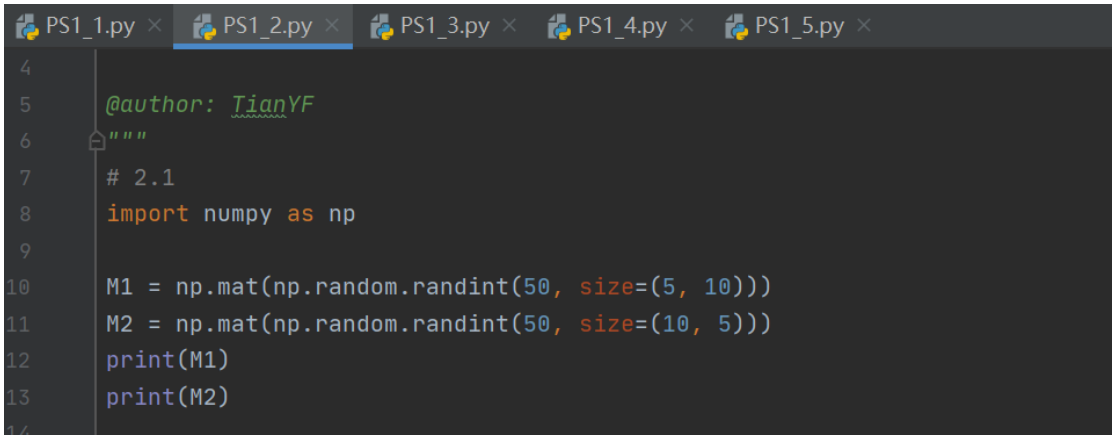
PS1_1.py PS1_2.py PS1_3.py PS1_4.py PS1_5.py

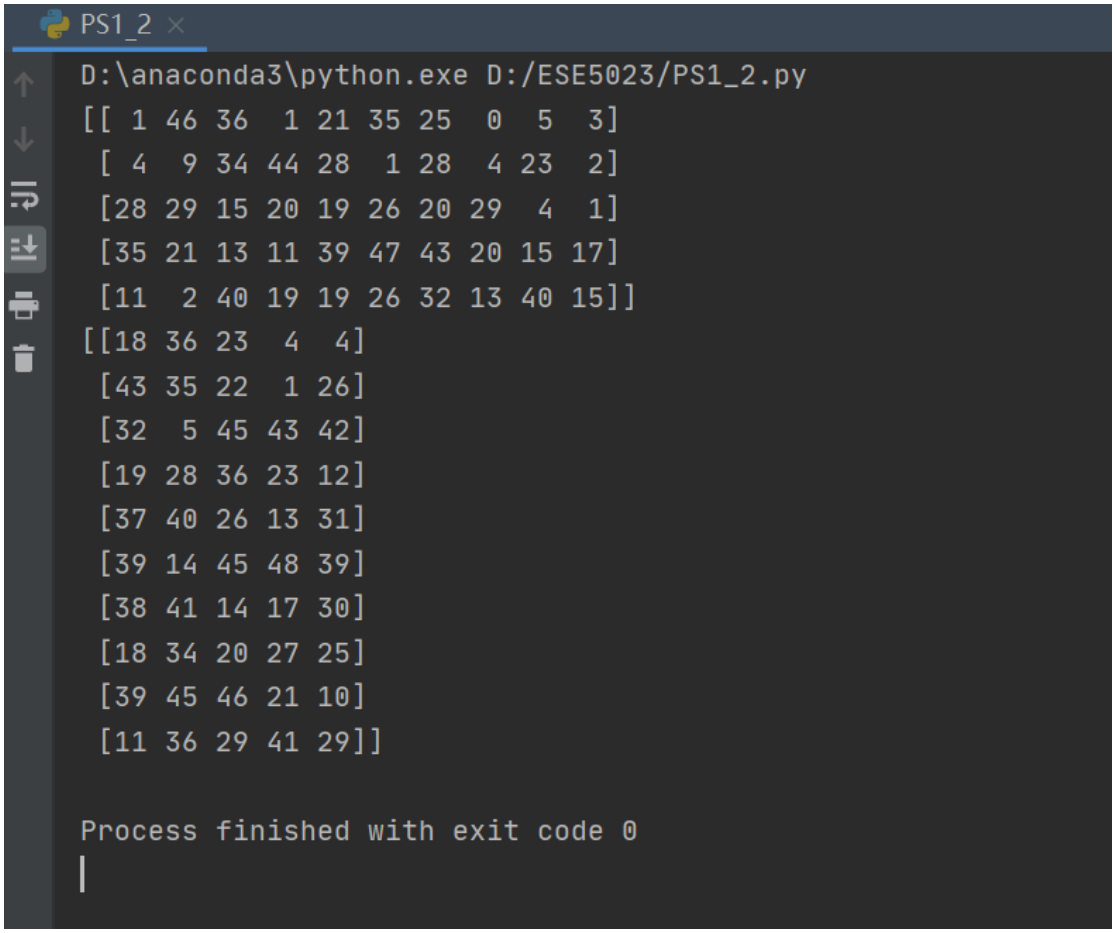
```
            print(a,c,b)
        else:
            print(c,a,b)
    elif b > c:
        print(c,a,b)
    else:
        print(c,b,a)

Print_values(1,3,2)
```



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.

```python
# 2.2
def Matrix_multip(M1,M2):
    r1, c1 = M1.shape
    r2, c2 = M2.shape
    result = np.zeros((r1, c2))
    for i in range(r1):
        for j in range(c2):
            for k in range(c1):
                result[i][j] += M1[i][k] * M2[k][j]
    print(result)
    return(result)

Matrix_multip(M1,M2)
```

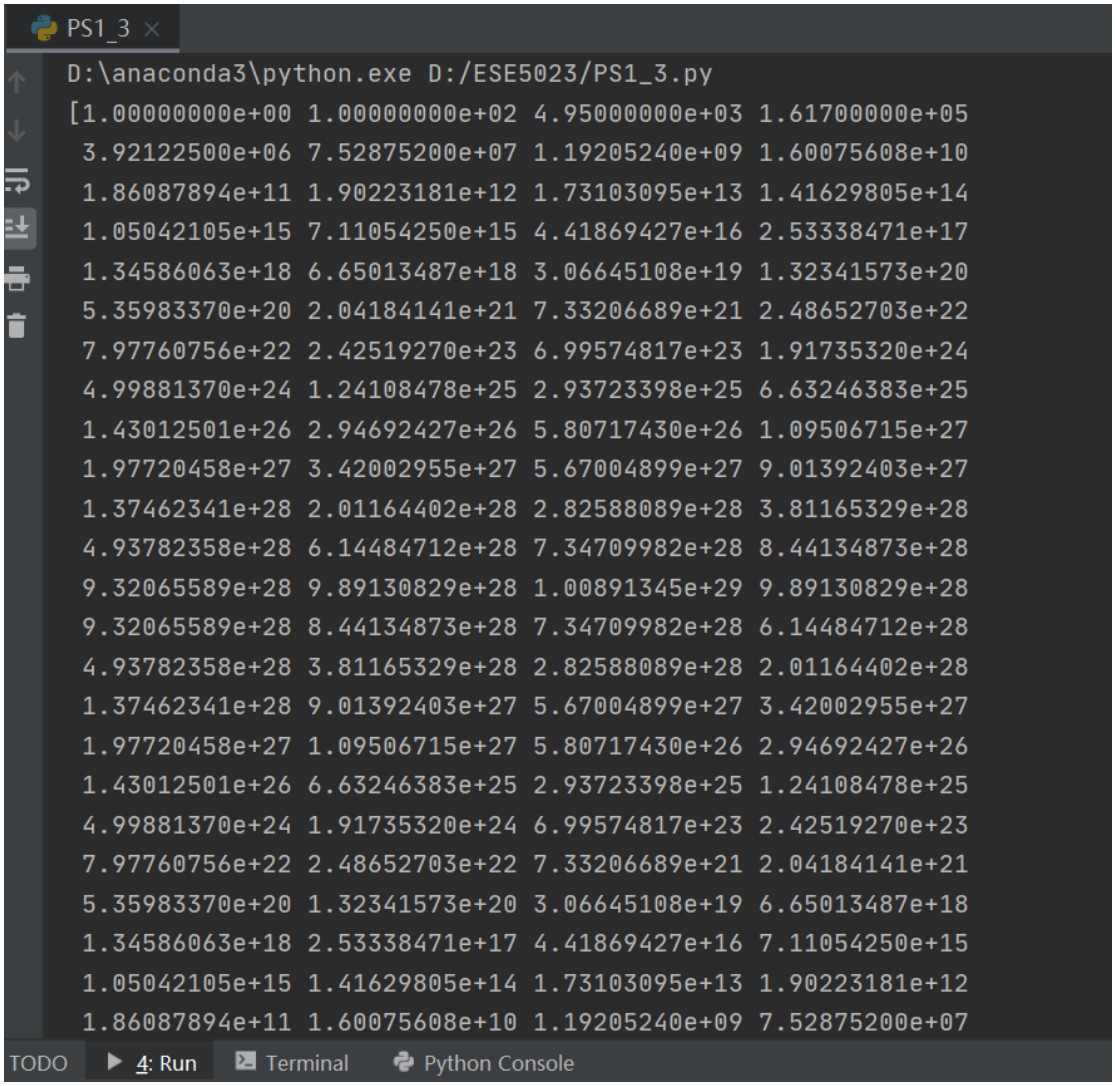
```
[[5560. 4958. 4754. 5144. 5203.]
 [9067. 6148. 5914. 7416. 6440.]
 [7503. 5783. 5600. 6709. 6907.]
 [5898. 6880. 6207. 7345. 7391.]
 [4928. 3837. 2681. 4807. 3629.]]

Process finished with exit code 0
```

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)`.

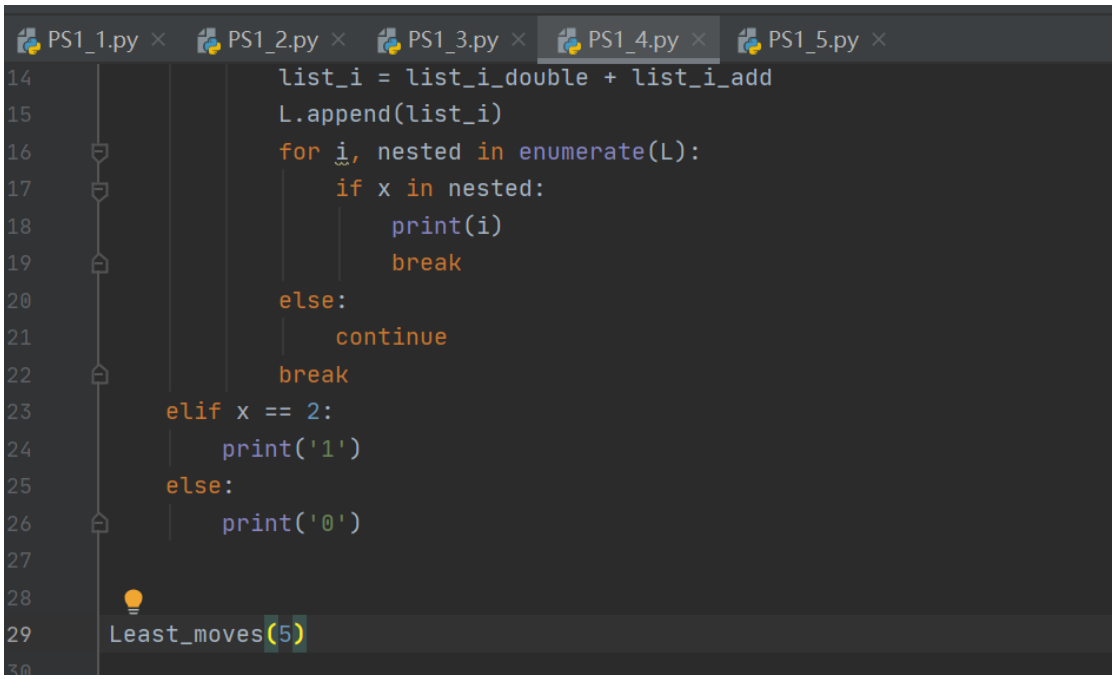
PS1_1.py PS1_2.py PS1_3.py PS1_4.py PS1_5.py

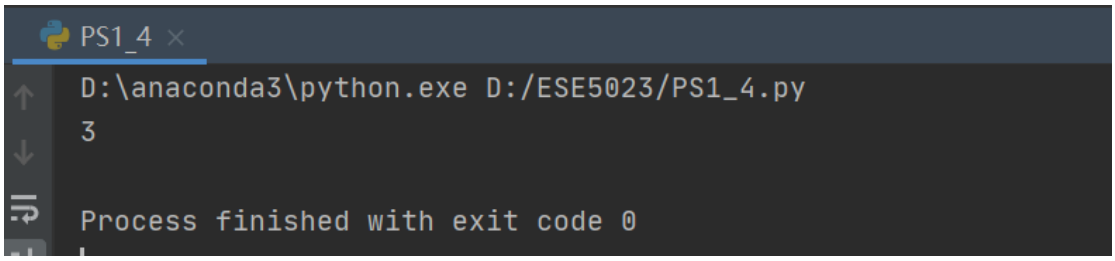
```
7
8   import math
9   import numpy as np
10
11  def Pascal_triangle(n):
12      array1 = np.zeros(n+1)
13      for k in range(0,n+1):
14          array1[k] = math.factorial(n) / (math.factorial(k)*math.factorial(n-k))
15      return array1
16
17  print(Pascal_triangle(100))
18  print(Pascal_triangle(200))
19
```



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`.





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.

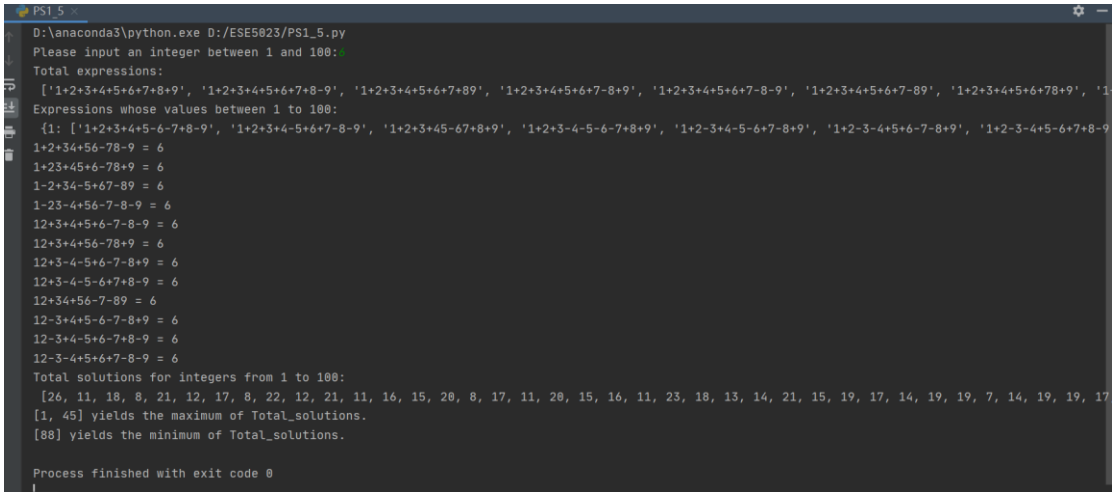
5.1 [30 points] Write a function `Find_expression`, which should be able to print every possible solution that makes the expression evaluate to a random integer from `1` to `100`. For example, `Find_expression(50)` should print lines include:

$$1-2+34+5+6+7+8-9=501-2+34+5+6+7+8-9=50$$

and

$$1+2+34-56+78-9=501+2+34-56+78-9=50$$

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`?


PS1_5
D:\anaconda3\python.exe D:/ESE5023/PS1_5.py
Please input an integer between 1 and 100:6
Total expressions:
['1+2+3+4+5+6+7+8+9', '1+2+3+4+5+6+7+8-9', '1+2+3+4+5+6+7+89', '1+2+3+4+5+6+7-8+9', '1+2+3+4+5+6+7-8-9', '1+2+3+4+5+6+7-89', '1+2+3+4+5+6+78+9', '1
Expressions whose values between 1 to 100:
{1: ['1+2+3+4+5-6-7+8-9', '1+2+3+4-5+6+7-8-9', '1+2+3+45-67+8+9', '1+2+3-4-5-6-7+8+9', '1+2-3+4-5-6+7-8+9', '1+2-3-4+5+6-7-8+9', '1+2-3-4+5-6+7+8-9
1+2+34+56-78-9 = 6
1+23+45+6-78+9 = 6
1-2+34-5+67-89 = 6
1-23-4+56-7-8-9 = 6
12+3+4+5+6-7-8-9 = 6
12+3+4+56-78+9 = 6
12+3-4-5+6-7-8+9 = 6
12+3-4-5-6+7+8-9 = 6
12+34+56-7-89 = 6
12-3+4+5-6-7-8+9 = 6
12-3+4-5+6-7+8-9 = 6
12-3-4+5+6+7-8-9 = 6
Total solutions for integers from 1 to 100:
[26, 11, 18, 8, 21, 12, 17, 8, 22, 12, 21, 11, 16, 15, 20, 8, 17, 11, 20, 15, 16, 11, 23, 18, 13, 14, 21, 15, 19, 17, 14, 19, 19, 7, 14, 19, 19, 17
[1, 45] yields the maximum of Total_solutions.
[88] yields the minimum of Total_solutions.
Process finished with exit code 0