PS1_1

Code:

```
# -*- coding: utf-8 -*-
import random
a=random.random()*100
b=random.random()*100
c=random.random()*100
if (a>b):
    if (b>c):
        print(a,b,c)
    elif (a>c):
        print(a,c,b)
    else:
        print(c,a,b)
elif (b>c):
    if (a>c):
        print(b,a,c)
    else:
        print(b,c,a)
else:
    print(c,b,a)
```

Output:

```
75.05415888359256 60.1397862631461 0.48579948382303506

Process finished with exit code 0
```

Statement:

A, *b* and *c* are number from 0 to 100, and the result is the order of *a*, *b* and *c*.

PS1_2

Code:

```
# -*- coding: utf-8 -*-
import numpy as np
def Matrix_multip (M1,M2):
    S=np.zeros((5,5))
    for i in range(0,5):
        for j in range(0,5):
            for m in range(0,10):
                S[i,j]+=M1[i,m]*M2[m,j]
    print(M1,M2,S)

M1=np.random.randint(0,50,(5,10))
M2=np.random.randint(0,50,(10,5))
Matrix_multip(M1,M2)
```

Output:

```
[[48 39 46 42 47 17 30 22 39 26]
 [ 9 25 39 41 27 18  4 12 19  8]
 [27 48 20 17 11 34 47 43 46 23]
 [20 36 42 14 31 17  4  8  6 44]
 [17  2 30 13 23 14  4 45 40 45]] [[42 15 11  4 22]
 [30 35  4 26  6]
 [29 47 40 23 39]
 [ 2 44 37  5 36]
 [18 34  1 39 38]
 [41 49  4 30 13]
 [33 24 29 38 44]
 [18  8 32 46 48]
 [ 1 27 49 16 38]
 [42 29 26 36 16]] [[ 8664. 11229.  8354.  8529. 10877.]
 [ 4268.  7384.  5014.  4677.  6207.]
 [ 8117.  9194.  7656.  8678.  9382.]
 [ 6551.  7635.  4471.  5971.  5689.]
 [ 5530.  6616.  6641.  6674.  7656.]]

Process finished with exit code 0
```

Statement:

M1 (5 rows and 10 columns) and M2 (10 rows and 5 columns) are matrices, and the "*Matrix_multip*" is the function to calculate M1*M2.

PS1_3

Code:

```
# -*- coding: utf-8 -*-
def Pascal_triangle(a):
    global b
    K=[1]
    if a >2:
        for i in range(0,a-1):
            if (i+1<a-1):
                K.append(Pascal_triangle(a-1)[i]+Pascal_triangle(a-1)[i+1])
            else:
                break
        K.append(1)
    elif a==1:
        K = [1]
    else:
        K=[1,1]
    if (len(K)==b):
```

```
            print(K)
    return (K)

a=10 # Change a to get Pascal_triangle(100) and Pascal_triangle(200)
b=a
Pascal_triangle(a)
```

Output:

PS: The output below is the result of Pascal_triangle(10) because the results of Pascal_triangle(100) and Pascal_triangle(200) are too much and the calculation takes too much time.

```
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]

Process finished with exit code 0
```

Statement:

A represents the layer of Pascal_triangle that need to be calculated, and the results are obtained in the order from back to front using an iterative method.

PS1_4

Code:

```
# -*- coding: utf-8 -*-
import math
import random
def Least_moves(a,step):
    if (a!=1):
        if (math.ceil(a / 2) == (a / 2)):
            step += 1
            Least_moves(a / 2,step)
        else:
            step += 1
            Least_moves(a - 1,step)
    else:
        print("least step:"+str(step))
a=int(random.random()*100)
print("RMB:"+str(a))
step=0
Least_moves(a,step)
```

Output:

```
RMB:50
least step:7

Process finished with exit code 0
```

Statement:

A represents RMB, the calculation process of Least_moves function is: calculate from back to front, if *a* is odd, then subtract 1, if *a* is even, then divide by 2, until you get 1.

PS1_5.1

Code:

```
# -*- coding: utf-8 -*-
def Find_expression(diction,value):
    keylist=[]
    for k,v in diction.items():
        if v==value:
            keylist.append(k)
    return keylist

def string_repeat(n_string,repeat_list):
    for s_index in range(0,len(n_string)):
        if (n_string[s_index]=="+") or (n_string[s_index]=="-"):
            repeat_list.append(s_index)
    return repeat_list

def make_dic(string_list,answer_list):
    formula_list=[]
    for i in string_list:
        new_string2=i.replace("a","")
        repeat_list2=string_repeat(new_string2,[])
        formula_list.append(new_string2)
        if len(repeat_list2) >0:
            answer=int(new_string2[:repeat_list2[0]])
            for j in range(0,len(repeat_list2)):
                if new_string2[repeat_list2[j]]=="+" and j< len(repeat_list2)-1:
                    answer=answer+int(new_string2[repeat_list2[j]+1:repeat_list2[j+1]])
                if new_string2[repeat_list2[j]]=="+" and j== len(repeat_list2)-1:
                    answer = answer + int(new_string2[repeat_list2[j] + 1:])
                if new_string2[repeat_list2[j]] == "-" and j< len(repeat_list2)-1:
                    answer = answer - int(new_string2[repeat_list2[j] + 1:repeat_list2[j + 1]])
                if new_string2[repeat_list2[j]] == "-" and j == len(repeat_list2) - 1:
                    answer = answer - int(new_string2[repeat_list2[j] + 1:])
            answer_list.append(answer)
        else:
            answer=123456789
            answer_list.append(answer)
    diction = dict(zip(formula_list, answer_list))
    return diction

def make_list():
```

```
    str_list = []
    for i in ['a', '-', '+']:
        for ii in ['a', '-', '+']:
            for iii in ['a', '-', '+']:
                for iiii in ['a', '-', '+']:
                    for iiiii in ['a', '-', '+']:
                        for iiiiii in ['a', '-', '+']:
                            for iiiiiii in ['a', '-', '+']:
                                for iiiiiiii in ['a', '-', '+']:
                                    str_list.append("1"+i                                +"2"+
ii+"3"+iii+"4"+iiii+"5"+iiiii+"6"+iiiiii+"7"+iiiiiii+"8"+iiiiiiii+"9")
    return str_list

string_list=make_list()
diction2=make_dic(string_list,[])
value=input("Please enter a number: ")
formula = Find_expression(diction2,int(value))
for s in range(0,len(formula)):
    print(formula[s]+"="+value)
```

Output:

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

Process finished with exit code 0
```

Statement:

I checked the information about dynamic programming which should be used to consume space in exchange for saving time. The "*make_list*" function is to build a list of all expression possibilities,

a total of 8 positions, you can place 3 symbols: “+”, ”-” and “NULL” (here NULL is represented by *a*), which means that you will get 3^8 list contents. The “*make_dic”* function is to build a dictionary, the keys in the dictionary are the expressions obtained by the “*make_list*” function, and the values are the results of the corresponding calculation. The “*Find_expression*” function is used to pick out the expressions that get the same result from the dictionary to form a new list. If the dictionary is stored on the computer and referenced by calling, it will greatly improve the calculation time efficiency.

PS1_5.2

Code:

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
def Find_expression(diction,value):
    keylist=[]
    for k,v in diction.items():
        if v==value:
            keylist.append(k)
    return keylist

def string_repeat(n_string,repeat_list):
    for s_index in range(0,len(n_string)):
        if (n_string[s_index]=="+") or (n_string[s_index]=="-"):
            repeat_list.append(s_index)
    return repeat_list

def make_dic(string_list,answer_list):
    formula_list=[]
    for i in string_list:
        new_string2=i.replace("a","")
        repeat_list2=string_repeat(new_string2,[])
        formula_list.append(new_string2)
        if len(repeat_list2) >0:
            answer=int(new_string2[:repeat_list2[0]])
            for j in range(0,len(repeat_list2)):
                if new_string2[repeat_list2[j]]=="+" and j< len(repeat_list2)-1:
                    answer=answer+int(new_string2[repeat_list2[j]+1:repeat_list2[j+1]])
                if new_string2[repeat_list2[j]]=="+" and j== len(repeat_list2)-1:
                    answer = answer + int(new_string2[repeat_list2[j] + 1:])
                if new_string2[repeat_list2[j]] == "-" and j< len(repeat_list2)-1:
                    answer = answer - int(new_string2[repeat_list2[j] + 1:repeat_list2[j + 1]])
                if new_string2[repeat_list2[j]] == "-" and j == len(repeat_list2) - 1:
                    answer = answer - int(new_string2[repeat_list2[j] + 1:])
            answer_list.append(answer)
        else:
```

```
                    answer=123456789
                    answer_list.append(answer)
    diction = dict(zip(formula_list, answer_list))
    return diction

def make_list():
    str_list = []
    for i in ['a', '-', '+']:
        for ii in ['a', '-', '+']:
            for iii in ['a', '-', '+']:
                for iiii in ['a', '-', '+']:
                    for iiiii in ['a', '-', '+']:
                        for iiiiii in ['a', '-', '+']:
                            for iiiiiii in ['a', '-', '+']:
                                for iiiiiiii in ['a', '-', '+']:
                                    str_list.append("1"+i                    +"2"+
ii+"3"+iii+"4"+iiii+"5"+iiiii+"6"+iiiiii+"7"+iiiiiii+"8"+iiiiiiii+"9")
    return str_list

string_list=make_list()
diction2=make_dic(string_list,[])
value_list=[]
solutions=[]
for value in range(1,101):
    solution = len(Find_expression(diction2,int(value)))
    value_list.append(value)
    solutions.append(solution)

total_solutions = dict(zip(value_list, solutions))
print(total_solutions)
max_list=[]
max_value=max(total_solutions.values())
for m,n in total_solutions.items():
    if n==max_value:
        max_list.append(m)
print(max_list)
min_list=[]
min_value=min(total_solutions.values())
for m,n in total_solutions.items():
    if n==min_value:
        min_list.append(m)
print(min_list)
x=value_list
y=solutions
```

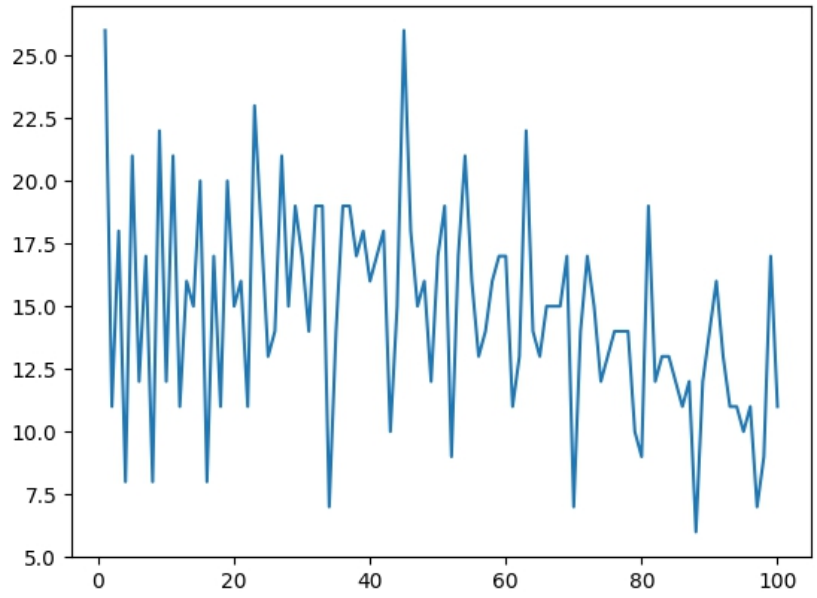
```
plt.plot(x,y)
plt.show()
```

Output:

PS: The first line shows the total number of solutions from 1 to 100. Because of limited space, only part of the results are shown below. The second line shows the maximum of total-solutions, which means 1 and 45 have the most solutions. The third line shows the minimum of total-solutions, which means 88 has the least solutions. The plot below shows the list *Total_solutions*.

```
{1: 26, 2: 11, 3: 18, 4: 8, 5: 21, 6: 12, 7: 17, 8: 8,
[1, 45]
[88]

Process finished with exit code 0
```



Statement:

Based on the results of the previous question, calculate the total-solutions from 1 to 100 to form a dictionary. The keys in the dictionary represent 1 to 100, and the values represent the corresponding total-solutions, and the maximum and minimum values are obtained.