# Report of assignment 1

1. Flowchart

According to the flowchart, I used “if-elif-else” structure to define a method “Print_values(a, b ,c)” with three arguments a, b, c. Either applied random.random() to create random a, b, c or set certain a, b, c and then call for method Print-value(a, b, c), the console windows showed like this:

c = 1
a = 2
b = 3
Print_values(a, b, c) : when b is the biggest one, the output is to print empty.

c = 1
a = 3
b = 2
Print_values(a, b, c)
3, 2, 1

c = 1
a = 3
b = 1
Print_values(a, b, c)
3, 1, 1

c = 4
a = 3
b = 1
Print_values(a, b, c)
4, 3, 1

c = 4
a = 2
b = 3
Print_values(a, b, c)
4, 3, 2

a = random.random();
b = random.random();
c = random.random();

Print_values(a, b, c)
0.2703991133069813, 0.10372005239302096, 0.08213798389241322

**2. Matrix multiplication**

**2.1**

For making matrix, I referred this website:

https://blog.csdn.net/qq_43287650/article/details/82860938

In this website, I learned how to use "numpy" to generate all types of matrices or array. Selecting from it, I used "np.random.randint(lower_lim, upper_lim, [nrow, ncol])".

This radint() is a method to fill integer number randomly chosen from number "lower_lim" to number "upper_lim" into a matrix with row and column number equating to nrow and ncol respectively.

So M1 (5 rows and 10 columns ) is :

```
M1 = np.random.randint(0, 51, [5, 10])
M1
Out[52]:
array([[45,  6, 36, 26, 21,  8, 17, 39, 33, 24],
[12, 17,  4,  6, 33, 34, 24,  0, 16, 27],
[ 6, 45, 22, 39, 18,  6, 20, 44, 36, 12],
[47, 13, 32, 41, 12, 32, 14, 25, 29, 13],
[21, 40, 13, 21, 25, 22, 23,  5, 26, 44]])
```

M2 (10 rows and 5 columns ) is:

```
M2 = np.random.randint(0, 51, [10, 5])
M2
Out[54]:
array([[16, 25, 37, 30, 44],
       [16, 31, 13,  2, 31],
       [47,  4, 47, 39, 10],
       [16, 20, 41, 43, 15],
       [48,  4, 28, 34, 45],
       [39, 48, 35, 45, 29],
       [46, 10, 10, 27,  6],
       [19,  9, 44,  1,  6],
       [50, 35,  7,  4, 43],
       [22, 44, 10, 24, 11]])
```

**2.2**

I defined a method Matrix_multip(M1, M2). In this method, get one row and one column for M1, M2 by x, y in for loop respectively, then, multiple each single elements and make a sum to form M3[x,y]. Finally, I did a test by python 'dot' function. For specific results like this:

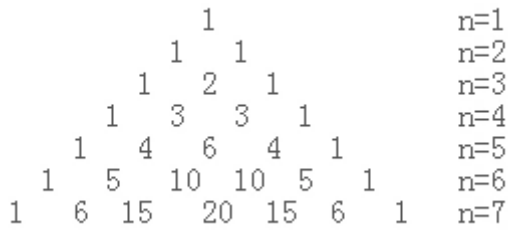
```
M3
Out[57]:
array([[7945., 5175., 7726., 6164., 6112.],
       [6156., 4715., 3835., 4820., 4785.],
       [7392., 5157., 6662., 4703., 5512.],
       [7799., 6062., 8122., 7126., 6498.],
```

```
        [7402., 6514., 5311., 5746., 6142.]])

M4 = np.dot(M1,M2)
M3 == M4
Out[56]:
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

**3. Pascal_triangle**

Pascal triangle is showed in figure below (from Baidu):




So, the first line is defined as first line not 0 line here.

Firstly, a “jc(k)” function is defined to calculate factorial of number k. Then combination number of k is “b = jc(k)/jc(k-a)/jc(a)”. So we can do Pascal_triangle(k) to show the kth line of pascal triangle. Results like this:

Pascal_triangle(1)
Out[111]: [1]

Pascal_triangle(2)
Out[112]: [1, 1]

Pascal_triangle(3)
Out[113]: [1, 2, 1]

Pascal_triangle(4)
Out[114]: [1, 3, 3, 1]

Pascal_triangle(5)
Out[115]: [1, 4, 6, 4, 1]

Pascal_triangle(6)
Out[116]: [1, 5, 10, 10, 5, 1]

```
Pascal_triangle(100)
Out[119]:
[1,
 99,
 4851,
 156849,
 3764376,
 71523144,
 1120529256,
 14887031544,
 171200862756,
 1731030945644,
 15579278510796,
 126050526132804,
 924370524973895,
 6186171974825304,
 38000770702498296,
 215337700647490336,
 1130522928399324288,
 5519611944537877504,
 25144898858450329600,
 107196674080761937920,
 428786696323047751680,
 1613054714739084230656,
 5719012170438571720704,
 19146258135816089894912,
 60629817430084289560576,
 181889452290252818350080,
 517685364210719585730560,
 1399667836569723550564352,
 3599145865465002725474304,
 8811701946483283423395840,
 20560637875127662394998784,
 45764000431735766071115776,
 97248500917438495384928256,
 197443926105102399720914944,
 383273503615786966757408768,
 711793649572175834674888704,
 1265410932572756856170086400,
 2154618614921180756379172864,
 3515430371713505560356716544,
 5498493658321124166161399808,
 8247740487481687348753727488,
 11868699725888281821365403648,
```

163901091452742920589132431 36,
217264237507124360630732062 72,
276518120463612790635138908 16,
337966591677749024972456591 36,
396743390230400903110679592 96,
447391482600239415463730216 96,
484674106150259381412529438 72,
504456722727821010358574776 32,
504456722727821010358574776 32,
484674106150259469373459660 80,
447391482600239415463730216 96,
396743390230400991071609815 04,
337966591677748980991991480 32,
276518120463612790635138908 16,
217264237507124360630732062 72,
163901091452742942579364986 88,
118686997258882818213654036 48,
824774048748168734875372748 8,
549849365832112526567302758 4,
351543037171350556035671654 4,
215461861492118075637917286 4,
126541093257275713104799334 4,
711793649572175972113842176,
383273503615786966757408768,
197443926105102399720914944,
972485009174384953849282 56,
457640004317357574811811 84,
205606378751276623949987 84,
881170194648328342339584 0,
359914586546500326234521 6,
139966783656972328212889 6,
517685364210719652839424,
181889452290252851904512,
60629817430084281171968,
191462581358160857006 08,
571901217043857172070 4,
161305471473908449280 0,
428786696323047686144,
107196674080761921536,
25144898858450333696,
5519611944537877504,
1130522928399324288,
215337700647490368,
38000770702498296,

6186171974825304,
924370524973895,
126050526132804,
15579278510796,
1731030945643,
171200862756,
14887031543,
1120529256,
71523144,
3764376,
156849,
4851,
99,
1]

Pascal_triangle(200)
Traceback (most recent call last):

File "<ipython-input-120-6e8cb372cac1>", line 1, in <module>
Pascal_triangle(200)

File "<ipython-input-117-21e9c58fe877>", line 8, in Pascal_triangle
b = jc(k)/jc(k-a)/jc(a)

OverflowError: integer division result too large for a float

Pascal_triangle(200) is too large to print, sorry…

**4. Least moves**

Least moves is :

Divide RMB K by 2, if the reminder is 1, make (K – 1) replace K, otherwise, make K/2 replace K. Use i to count the times of doing each operation.

Result is:

1, 2, 4, 5
Least_moves(5)
Out[128]: 3

1, 2, 4, 5, 10
Least_moves(10)
Out[129]: 4

1, 2, 3, 6, 12, 24, 48, 49, 98, 99
Least_moves(99)
Out[130]: 9

**5. Dynamic programming**

In this question, though I made a lot of reference, question was truly settled down.

My idea is:

1. I need to insert "+", "-", and nothing (this is to make N-digital number) into 123456789, therefore, I should firstly make probabilities of (+, - , "") into a list. The list should have a number to the eighth power of 3.

Here, I referred: https://code-examples.net/zh-CN/q/197e4. And used "itertools" to get the list.

itertools.product([a,b,c], repeat = N) function can get all the probabilities of arbitrarily place these number, a, b, c, in these N postions.

2. Once I get the list, I should combine each one in the list with 123456789, which means inserting each probability of "+-'" between each single number of 12456789, to make the final expression that we want, of course "= X" should be added in the end. however, before adding "=X", we should judge whether the result of expression match X or not.

Here, I referred: http://c.biancheng.net/view/4277.html to use "join()" function to combine and get expression. "join()" can join two string into one.

I referred: https://blog.csdn.net/weixin_43097301/article/details/82933099 to use "eval()" calculate string, which mean making the string expression into real calculation formula and return the result.

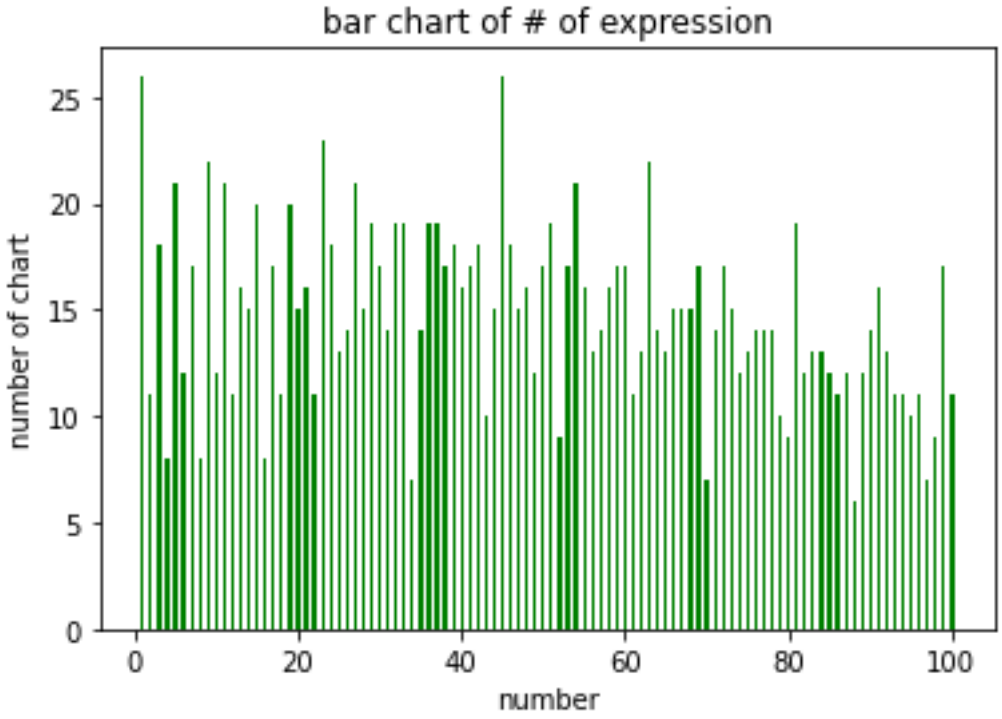
I referred: https://m.jingyanlib.com/resultpage?id=CsFJGPp1X_PdaFvnJN-oLg to use "for index, value in **enumerate**(res)" to ergodic and match result.

From this process, I learned a lot and realized the power and wisdom of python and programmer behind it. More importantly, I started thinking a question by a process of from building concept model to creating math model: figure out problem logically by logically, module by module.

Last is the figure and results:

```
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
Out[137]: 17
```

Figure:




```
Max = max(Total_solutions)
for index, value in enumerate(Total_solutions):
    if (value == Max):
        print(index+1)
```

1

45

Max

Out[156]: 26

The 1 and 45 have maximal number of expressions.