Assignment 01

胡蓓慧 12332288

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.

The results of my code are as follows:

```
In [24]: runfile('D:/ese2023/ESE5023_Assignments_12332288/PS1_1.py',
wdir='D:/ese2023/ESE5023_Assignments_12332288')
a:0.7404246544670767
b:0.11757678040719
c0.33504539973616576
output:0.7404246544670767 0.11757678040719 0.33504539973616576
```

Fig 1 result of problem 1

```
In [25]: runfile('D:/ese2023/ESE5023_Assignments_12332288/PS1_1.py',
wdir='D:/ese2023/ESE5023_Assignments_12332288')
a:0.8058922588434229
b:0.5633520387941982
c0.1629556304470584
output:0.8058922588434229 0.1629556304470584 0.5633520387941982
```

Fig 2 result of problem 1

```
In [31]: runfile('D:/ese2023/ESE5023_Assignments_12332288/PS1_1.py'
wdir='D:/ese2023/ESE5023_Assignments_12332288')
a:0.05583848519203094
b:0.2482954295865295
c0.43786334320459475
output:0.43786334320459475 0.2482954295865295 0.05583848519203094
```

Fig 3 result of problem 1

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.

I use function “np.random.randint()” to create matrices filled with random integers. The results are as follows:

```
In [39]: runfile('D:/ese2023/ESE5023_Assignments_12332288/
PS1_2.py', wdir='D:/ese2023/ESE5023_Assignments_12332288')
M1:[[21 46  8 27 37 42 17  6 25 22]
 [ 5 24  5 50 39 25  6 45 30 28]
 [14  1 10  6 35 49 27 26 42 32]
 [24 44 48 49 33  5  1 40 46 27]
 [ 7 16 20  7  9 38  3 28 11  7]]
M2:[[19 40 48 39 29]
 [ 9 45  5 36 16]
 [ 4 49 32 27 17]
 [47 20 32 22 34]
 [14 15  5 21 33]
 [24 16 37 16  9]
 [10 15  4 43 34]
 [32 16 41 41 13]
 [47 38  4  8  0]
 [49 17 24 10 36]]
```

Fig 4 result of problem 2.1

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.

I use nesting loop , * operator and + operator to do matrix multiplication, the main part of my code are as follows:

```
for i in range(0,row):
    for j in range(0,col):
        for k in range(0,middle):
        M3[i,j]+=M1[i,k]*M2[k,j]
```

I use M1 and M2 which created in 2.1 to test my code, the result are as follows:

```
M1*M2:[[6255. 6744. 5039. 6131. 5446.]
 [8109. 5936. 5901. 6136. 5623.]
 [6907. 5485. 5287. 5386. 4800.]
 [8704. 9709. 7302. 7988. 6542.]
 [3510. 3893. 4103. 3775. 2394.]]
```

Fig 5result of problem 2.2

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).

I got inspired by reading 【精选】【Python 实现杨辉三角】_python 杨辉三角-CSDN 博客. The main part of my code are as follows:

```
def Pascal_triangle(k):
```

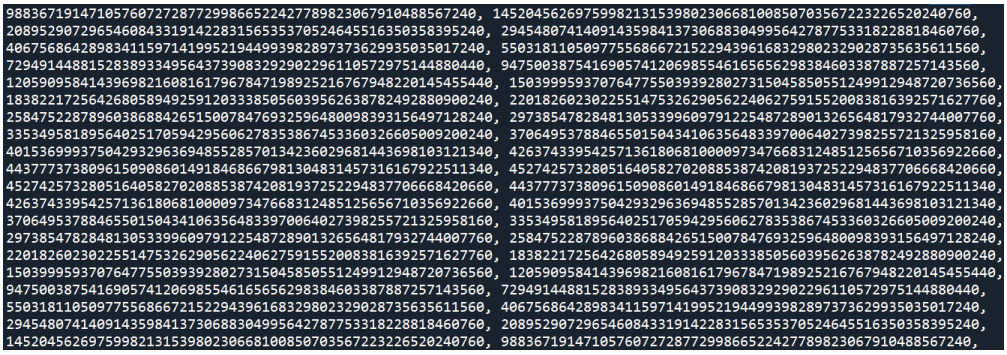
```
        List=[]
        for i in range(0,k):
            tempList=[]
            for j in range(0,i+1):
                if j==0 or j==i:
                    tempList.append(1)
                else:
                    tempList.append(List[i-1][j-1]+List[i-1][j])
            List.append(tempList)
    return List[-1]
```

```
ipdb> !runfile('D:/ese2023/ESE5023_Assignments_12332288/untitled4.py', wdir='D:/ese2023/ESE5023_Assignments_12332288')
line 100:[1, 99, 4851, 156849, 3764376, 71523144, 1120529256, 14887031544, 171200862756, 1731030945644, 15579278510796,
1260505261328O4, 9243705249738896, 6186171974825304, 38000770702498296, 215337700647490344, 1130522928399324306,
5519611944537877494, 25144898858450330806, 107196674080761936594, 428786696323047746376, 161305471473908437922 4,
57190121704385718899 76, 191462581358160885012 24, 606298174300842802538 76, 181889452290252840761628, 517685364210719623706172,
139966783656972342705 7428, 359914586546500309814767 2, 881170194648328344718912 8, 205606378751276613767746 32,
457640004317357624192725 68, 972485009174384951409542 07, 197443926105102399225573693, 383273503615787010261407757,
711793649572175876199757263, 126541093257275711324401291 2, 215461861492118103065872468 8, 351543037171350589212739291 2,
549849365832112460050694788 8, 824774048748168690076042183 2, 118686997258882811498747533 68, 163901091452742930164937070 32,
217264237507124349288404953 68, 276518120463612808185242668 32, 337966591677748987781963261 28, 396743390230400985657087306 72,
447391482600239409357992069 28, 484674106150259360137824741 72, 504456722727820966674062486 28, 504456722727820966674062486 28,
484674106150259360137824741 72, 447391482600239409357992069 28, 396743390230400985657087306 72, 337966591677748987781963261 28,
276518120463612808185242668 32, 217264237507124349288404953 68, 163901091452742930164937070 32, 118686997258882811498747533 68,
824774048748168690076042183 2, 549849365832112460050694788 8, 351543037171350589212739291 2, 215461861492118103065872468 8,
126541093257275711324401291 2, 711793649572175876199757263, 383273503615787010261407757, 197443926105102399225573693,
972485009174384951409542 07, 457640004317357624192725 68, 205606378751276613767746 32, 881170194648328344718912 8,
359914586546500309814767 2, 139966783656972342705 7428, 517685364210719623706172, 181889452290252840761628,
606298174300842802538 76, 191462581358160885012 24, 571901217043857188997 6, 161305471473908437922 4, 428786696323047746376,
107196674080761936594, 251448988584503308 06, 551961194453787749 4, 113052292839932 4306, 215337700647490344, 38000770702498296,
618617197482530 4, 924370524973896, 126050526132804, 15579278510796, 1731030945644, 171200862756, 14887031544, 1120529256,
71523144, 3764376, 156849, 4851, 99, 1]
```

Fig 6 result of Pascal_triangle(100)

```
line 200:[1, 199, 19701, 1293699, 63391251, 2472258789, 79936367511, 2203959847089, 52895036330136, 1122550215450664,
21328454093562616, 3664616203348485 84, 5741232051912627816, 8258541490058933858 4, 1097206226536401212616,
135322101272822816222 64, 155620416463746238656036, 167520801252150362788556 4, 169382143488285366819545036,
161358778796735007338614764, 145222900917061506604753 2876, 123785234591209569915480183 24, 100153507987433197477070330076,
770746561468507650149628192324, 565214145076905610109727 3410376, 395649901553833927076809138726 32,
264781087962950397351403038993768, 169656030435520069414047132392303 2, 104217275838962328354343238469557 68,
614522557533191660296299785 45842632, 348229449268808607501236545093108248, 189841215891705337637777089071204 93352,
996666383431453022598297176238259 0098, 504373594039553499314895843732694711 02, 246252990031076120253743264881256829498,
116090695300364456691050396301163933906 2, 528857611923882524925896249816413476683 8, 232983218225926625845732672216419991079 62,
993244246121055615447597181554211540918 38, 410031599039717830992469605718533482276562,
164012639615887132396987842287413392910624 8, 636049017046976928076123583504847060311 9352,
239275582603386558657208395699442465549184 8, 873634103923992783934458561042150397458353 52,
309743000482142896122217126187671504553416248, 106689255721626997553208121242420184901732 2632,
357177073502838209199870666768102358149277 5768, 116272536693477119165064280884084384674126530 32,
368196366196010877356036889466267218134734012 68, 113464594480811515266860347570217040690499665 132,
340393783442434545800581042710651122071498995396, 994483798684759751456599516938961121346144123804,
283045388856431621568416785590319703767748712159 6, 785050418148970723972778631731641442525642654484 4,
212254372314351343888936444875591945571747828803 96, 559579708828744452070832445581106038325517003210 44,
143891925127391430532499771720855838426561515111256, 360992022688017097651709953615480436754356081770344,
883808055546524618388669196782727965846871786403256, 211215145478067747784410774146380751160015121835354 4,
492835339448824744830291806341555086040035284282482 9436, 112301823251453507428541903412255995015680171336502 64,
249962122720977161695786817272440763099417155555441 36, 543568425599585256386076094703561659438415084303102 64,
115508290439911866982041170124506852630663205414409311, 239901833990586185270393199489360386232915888168388569,
487073420526341648882313465629913511442586803250970731, 966877088507514019423099864608634283908418579587747869,
187687905416164486123307620776970184523398900743503998 1, 356335008833587647567439106112798466269061681121724981 9,
661765016405234202625244054209482865928257407797489252 1, 120236179037007341040361243652145478457387613529402976 79,
213753207176901939627308877603814183924244646274494180 96, 371872017965295155242030513091567141895603699683024123 04,
633187490049016075141835738507262971335757650816356689 6, 105531248341502679190305956417877161889292941801939278160,
172182563083504371310499192050220632556214799782111453840, 275044873497026463262225982106196594862524939911684530160,
430198391879964468179379100217384417605487726528532213840, 658911460980705071515251533244348285193215378606992378160,
```

Fig 7 a) result of Pascal_triangle(200)



b) result of Pascal_triangle(200)

c) result of 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.

The main part of my code are as follows:

```
def Least_moves(RMB):
    step=0
    while RMB!=1:
        if RMB%2==1:
            step+=1
            RMB-=1
        else:
```

step+=1

RMB/=2

return step

```
ipdb> !runfile('D:/ese2023/ESE5023_Assignments_12332288/
PS1_4.py', wdir='D:/ese2023/ESE5023_Assignments_12332288')
27
step numbers:7
```

Fig 8 result of problem 4

```
ipdb> !runfile('D:/ese2023/ESE5023_Assignments_12332288/
PS1_4.py', wdir='D:/ese2023/ESE5023_Assignments_12332288')
87
step numbers:10
```

Fig 9 result of problem 4

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

I got inspired by reading C 语言算法解决 123456789=100-CSDN 博客. So I use recursive functions to solve this problem. Firstly, I create a function named “generate_num” to divide 9 numbers into different combination, and save the numbers into a list named “num”. Then , I create a function named ” Find_expression” to calculate the sum using different operator. 9 numbers can be divided into 1 to 9 numbers, I use a loop to traverse all the possibilities, if sum is equal to 50, the formula will be printed to the screen.

```
ipdb> !runfile('D:/ese2023/ESE5023_Assignments_12332288/
PS1_5.py', wdir='D:/ese2023/ESE5023_Assignments_12332288')
1-2-3-4-5-6+78-9=50
1+2+3-4+56-7+8-9=50
1+2-3+4+56+7-8-9=50
1-2-3+4+56-7-8+9=50
1+2+34-5-6+7+8+9=50
1-2+34+5+6+7+8-9=50
1+2+3+4-56+7+89=50
1-2+3-45+6+78+9=50
1+2-34+5-6-7+89=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
12-3-4-5+67-8-9=50
12-3+45+6+7-8-9=50
1-2+34-5-67+89=50
1+2+34-56+78-9=50
12+3+4-56+78+9=50
```

Fig 10 result of 5.1

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?

Based on the code of 5.1, I use loop to count the total number of suitable solutions for any integer from 1 to 100, the result are recorded into a List named “Total_solutions”. Then, I use functions(ie. list.index(), max()) to search which numbers yields the maximum and minmum of Total_solutions.

```
Total_solutions:
[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, 18, 16, 17, 18, 10, 15, 26,
18, 15, 16, 12, 17, 19, 9, 17, 21, 16, 13, 14, 16, 17, 17,
11, 13, 22, 14, 13, 15, 15, 15, 17, 7, 14, 17, 15, 12, 13,
14, 14, 14, 10, 9, 19, 12, 13, 13, 12, 11, 12, 6, 12, 14,
16, 13, 11, 11, 10, 11, 7, 9, 17, 11]
[1, 45] yields the maximum of Total_solutions
[88] yields the minimum of Total_solutions
```

Fig 11 result of 5.2