

# 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 `k`th 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,
126050526132804, 924370524973896, 6186171974825304, 38000770702498296, 215337700647490344, 1130522928399324306,
5519611944537877494, 25144898858450330806, 107196674080761936594, 428786696323047746376, 1613054714739084379224,
5719012170438571889976, 19146258135816088501224, 60629817430084280253876, 181889452290252840761628, 517685364210719623706172,
1399667836569723427057428, 3599145865465003098147672, 8811701946483283447189128, 20560637875127661376774632,
45764000431735762419272568, 97248500917438495140954207, 197443926105102399225573693, 383273503615787010261407757,
711793649572175876199757263, 1265410932572757113244012912, 2154618614921181030658724688, 3515430371713505892127392912,
5498493658321124600506947888, 8247740487481686900760421832, 11868699725888281149874753368, 16390109145274293016493707032,
21726423750712434928840495368, 27651812046361280818524266832, 33796659167774898778196326128, 39674339023040098565708730672,
44739148260023940935799206928, 48467410615025936013782474172, 50445672272782096667406248628, 50445672272782096667406248628,
48467410615025936013782474172, 44739148260023940935799206928, 39674339023040098565708730672, 33796659167774898778196326128,
27651812046361280818524266832, 21726423750712434928840495368, 16390109145274293016493707032, 11868699725888281149874753368,
8247740487481686900760421832, 5498493658321124600506947888, 3515430371713505892127392912, 2154618614921181030658724688,
1265410932572757113244012912, 711793649572175876199757263, 383273503615787010261407757, 197443926105102399225573693,
97248500917438495140954207, 45764000431735762419272568, 20560637875127661376774632, 8811701946483283447189128,
3599145865465003098147672, 1399667836569723427057428, 517685364210719623706172, 181889452290252840761628,
60629817430084280253876, 19146258135816088501224, 5719012170438571889976, 1613054714739084379224, 428786696323047746376,
107196674080761936594, 25144898858450330806, 5519611944537877494, 1130522928399324306, 215337700647490344, 38000770702498296,
6186171974825304, 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, 366461620334848584, 5741232051912627816, 82585414900589338584, 1097206226536401212616,
13532210127282281622264, 155620416463746238656036, 1675208012521503627885564, 16938214348828536681954036,
161358778796735007338614764, 1452229009170615066047532876, 12378523459120956991548018324, 100153507987433197477070330076,
770746561468507650149628192324, 5652141450769056101097273410376, 3956499015538392707680913872632,
264781087962950397351403038993768, 1696560304355200694140471323923032, 1042172758389623283543423846955768,
61452255753319166029629978545842632, 348229449268808607501236545093108248, 1898412158917053376377708901120493352,
996663834314530225982971762382590098, 50437359403955349931489584373269471102, 246252990031076120253743264881256829498,
1160906953003644566910503963011639339062, 5288576119238825249258962498164134766838, 23298321822592662584573267212641999107962,
99324424612105561544759718155421154091838, 410031599039717830992469605718533482276562,
1640126396158871323969878422874133929106248, 6360490170469769280761235835048470603119352,
23927558260338655865720839569944246554591848, 87363410392399278393445856104215039745835352,
3097430004821242896122217126187671504553416248, 106689255721626997553208121242401849017322632,
357177073502832091998706667681023581492775768, 11627253669347711916506428088408438467412653032,
36819636619601087735603688946626721813473401268, 113464594480811515266860347570217040690499665132,
3403937834424345458005810427106511220714988959396, 994483798684759751456599516938961121346144123804,
2830453888564316215684167855903197037677487121596, 7850504181489707239727786317316414425256426544804,
11225437231435134388893644487559194557174782880396, 55957970882874445207083244558110603832551700321044,
1438919251723914305324997717208558384265615111256, 360992022688017097651709953615480436754356081770344,
88380805546524618388669196782727965846871786403256, 211215145478067747844107741463807511600151218353544,
4928353394488247448302918063415550860400352842824936, 11230182325145350742854190341225599501568017133650264,
24996212272097716169578681727244076309941715555544136, 54356842559958525638607609470356165943841508430310264,
115508290439911866982041170124506852630663205414409311, 239901833990586185270393199489360386232915888168388569,
487073420526341648882313465629913511442586803259070731, 966877088057514019423099864608634283908418579587747869,
1876879054161644861233076207769701845233989007435039981, 356350088335876475674391061127984662690616811217249819,
6617650160534202625244054209482865928257407794892521, 12023617903700734104036124365214547845738761352940297679,
2137511717690193627308877608381418392424464627449418096, 37187201796529515524203051309156714189560369968302412304,
633187490049016075141835738072629713357565081163566896, 105531248341502679190308956417877161889292841801939278160,
1718256308304371310499192050220632556214799782111453840, 275044873497026463262225982106196594862524939911684530160,
43019839187996468179379100217384417605487726528532213840, 658911460980705071515251533244348285193215378606992378160,

```

Fig 7 a) result of Pascal\_triangle(200)

```

988367191471057607272877299866522427789823067910488567240, 1452045626975998213153980230668100850703567223226520240760,
2089529072965460843319142283156535370524645516350358395240, 2945480741409143598413730688304995642787753318228818460760,
4067568642898341159714199521944993982897373629935035017240, 5503181105097755686672152294396168329802329028735635611560,
7294914488152838933495643739083292902296110572975144880440, 9475003875416905741206985546165656298384603387887257143560,
12859095841439698216081617967847198925216767948220145455440, 15039995937076477550393928027315045850551249912948720736560,
18382217256426805894925912033385056039562638782492880900240, 22018260230225514753262905622406275915520083816392571627760,
25847522878960386884265150078476932596480098393156497128240, 29738547828481305339960979122548728901326564817932744007760,
33534958189564025170594295606278353867453360326605009200240, 37064953788465501504341063564833970064027398255721325958160,
40153609937584293296369485528570134236029681443698103121340, 42637433954257136180681000097347668312485125656710356922660,
44377737380961509086014918468667981304831457316167922511340, 45274257328051640582702088538742081937252294837706668420660,
45274257328051640582702088538742081937252294837706668420660, 44377737380961509086014918468667981304831457316167922511340,
42637433954257136180681000097347668312485125656710356922660, 40153609937504293296369485528570134236029681443698103121340,
37064953788465501504341063564833970064027398255721325958160, 33534958189564025170594295606278353867453360326605009200240,
29738547828481305339960979122548728901326564817932744007760, 25847522878960386884265150078476932596480098393156497128240,
22018260230225514753262905622406275915520083816392571627760, 18382217256426805894925912033385056039562638782492880900240,
15039995937076477550393928027315045850551249912948720736560, 12059095841439698216081617967847198925216767948220145455440,
9475003875416905741206985546165656298384603387887257143560, 7294914488152838933495643739083292902296110572975144880440,
5503181105097755686672152294396168329802329028735635611560, 4067568642898341159714199521944993982897373629935035017240,
2945480741409143598413730688304995642787753318228818460760, 2089529072965460843319142283156535370524645516350358395240,
1452045626975998213153980230668100850703567223226520240760, 988367191471057607272877299866522427789823067910488567240,

```

#### b) result of Pascal\_triangle(200)

```

658911460980705071515251533244348285193215378606992378160, 430198391879964468179379100217384417605487726528532213840,
275044873497026463262225982106196594862524939911684530160, 172182563083504371310499192050220632556214799782111453840,
105531248341502679190305956417877161889292941801939278160, 6331874900490160751418357385072629713357565081163566896,
37187201796529515524203051309156714189560369968302412304, 2137532071690193962730887760381418392424464627449418096,
12023617903700734104036124365214547845738761352940297679, 6617650164052342026252440542094828659282574077974892521,
356355008835876475674391061127984662690616811217249819, 1876879054161644861233076207769701845233989007435039981,
966877088507514019423099864608634283908418579587747869, 487073420526341648882313465629913511442586803250970731,
239901833990586185270391199489360386232915888168388569, 1155082904399118669882041170124506852630663205414409311,
5435684255995852563860760947035616594384150843010264, 24996212272097716169578681727244076309941715555544136,
11230182325145350742854190341225599501568017133650264, 4928353394488247448302918063415550860400352842824936,
2112151454780677477844107741463807511600151218353544, 8838080554652461838866919678272965846871786403256,
360992022688017097651709953615480436754356081770344, 14389192512739143053249977172085583842656151111256,
55957970882874445207083244558110603832551700321044, 21225437231435134388893644487559194557174782880396,
7850504181489702739727786317316414425256426544804, 2830453888564316215684167855903197037677487121596,
994483798684759751456599516938961121346144123804, 34039378344243454800581042710651122071498995396,
11346459448081151526686034750217040690499665132, 36819636619601087735603688946626721813473401268,
11627253669347711916506428088408438467412653032, 3571770735028382091998706667681023581492775768,
106689255721626997532081212424201849017322632, 309743000482142896122217126187671504553416248,
8736341039239927839434585104215039745835352, 23927558260338655865720839569944246554591848,
6360490170469769280761235835048470603119352, 1640126396158871323969878422874133929106248,
410031599039717830992469605718533482276562, 99324424612105561544759718155421154091838,
23298321822592662584573267212614999107962, 5288576119238825249258962498164134766838, 1160906953003644566910503963011639339062,
246252990031076120253743264881256829498, 50437359403955349931489584373269471102, 9966663834314530225982971762382590098,
1898412158917053376377708907120493352, 348229449268808607501236545093108248, 6145225573319166029629978455842632,
10421727583896232835434323846955768, 1696560304355200694140471323923032, 264781087962950397351403038993768,
39564990155383392707680913872632, 5652141450769056101097273410376, 770746561468507650149628192324,
100153507987433197477070330076, 12378523459120956991548018324, 1452229009170615066047532876, 161358778796735007338614764,
16938214348828536681954036, 1675208012521503627885564, 155620416463746238656036, 13532210127282281622264,
1097206226536401212616, 82585414900589338584, 5741232051912627816, 366461620334848584, 21328454093562616, 1122550215450664,
52895036330136, 2203959847089, 79936367511, 2472258789, 63391251, 1293699, 19701, 199, 1]

```

#### 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