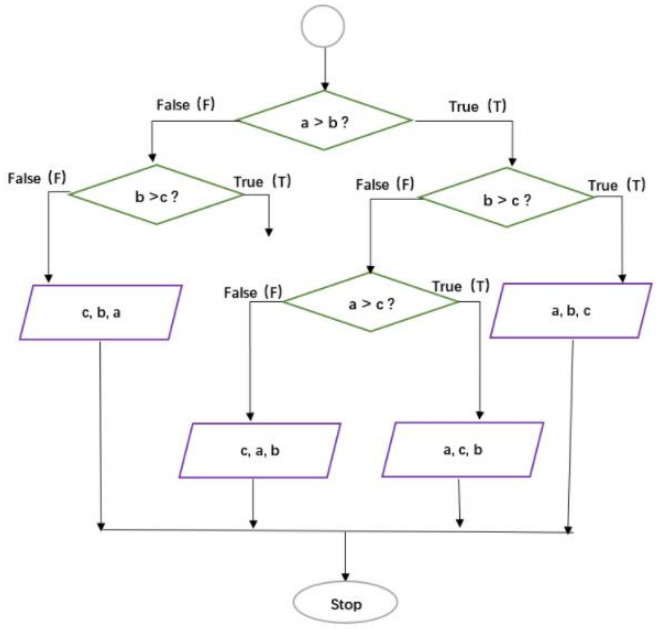
1. Flowchart

[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.



Here are my codes for problem 1:

```
@author: zyq17713
"""
import random

def Print_values():
    a=random.random()
    b=random.random()
    c=random.random()
    if (a>b):
        if (b>c):
            return a,b,c
        elif (a>c):
            return a,c,b
        else:
            return c,a,b
    elif (b>c):
        if (a>c):
            return None
        else:
            return None
    else:
        return c,b,a

print(Print_values())
```

Firstly, I use “random” to produce 3 random numbers. Then translate the flow chart to “if” statements. Here are my outputs:

```
In [23]: runfile('D:/ESE5023/PS1/PS1_1.py', wdir='D:/ESE5023/PS1')
(0.9618996374989408, 0.549375072167861, 0.2325820510403629)
```

In addition, there is no description about the range of the numbers, so I assume that they are random numbers in [0,1].

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`.

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.

Problem 2 requires us to write codes of matrix multiplication. After checking online, I know I can use numpy. Here are my codes for problem 2:

```
import numpy as np
M1=np.mat(np.random.randint(0,51,[5,10]))
M2=np.mat(np.random.randint(0,51,[10,5]))
print(M1)
print(M2)

def Matrix_multip(M1,M2):
    M3=[]
    for i in range(len(M1)):
        temp=[]
        for j in range(len(M2[0].T)):
            s=0
            for k in range(len(M2)):
                s+=(M1[i,k])*(M2[k,j])
            temp.append(s)
        M3.append(temp)
    return np.mat(M3)

print(Matrix_multip(M1,M2))

print(M1*M2)#To prove the accuracy of the function
```

Firstly, I need to create two random matrixes and print them. It is the result of 2.1.

```
In [24]: runfile('D:/ESE5023/PS1/PS1_2.py', wdir='D:/ESE5023/PS1')
[[27 15  0 20 45 46  4 48 39 46]
 [39 16 28  7 34 27 46 37 47  0]
 [18 26 46 30 24 33 26  7  6 11]
 [ 9  2 20 18  0  2 46 43 29 22]
 [16 23 29 30 30 32 35 28 47  9]]
[[45 21 33 49 11]
 [ 1 47 41 10  6]
 [41 25 21 24 26]
 [17  9 43 39 18]
 [ 7 44 11  0 26]
 [36 44 40 29 45]
 [26 35  8 35 47]
 [ 9 21 38  0  6]
 [48 44 44 30 48]
 [ 5 32  5 42 13]]
```

Secondly, I write the for loop according to the rule of matrix multiplication. Finally, I write “print(M1*M2)” to prove the accuracy of the codes I wrote before. Here are my outputs for 2.2:

```
[[6179 9792 8503 6829 6933]
 [8033 9473 8128 6819 8118]
 [5670 7201 6293 5925 5894]
 [4690 5526 5041 5105 5143]
 [7267 9309 8437 6821 8014]]
[[6179 9792 8503 6829 6933]
 [8033 9473 8128 6819 8118]
 [5670 7201 6293 5925 5894]
 [4690 5526 5041 5105 5143]
 [7267 9309 8437 6821 8014]]
```

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)`.

Pascal triangle is a famous rule in mathematics. One is its properties is that

every number equals the sum of the two numbers above it when the row number is more than 3 and the number is not on the side. Here are my codes for problem 3:

```
@author: zyq17713
"""

def Pascal_triangle(k):
    tri=[]
    for i in range(k):
        if i==0:
            tri.append([1])
        else:
            temp=[]
            for j in range(i+1):
                if j==0 or j==i:
                    temp.append(1)
                else:
                    temp.append(tri[i-1][j]+tri[i-1][j-1])
            tri.append(temp)

    s=str(tri[len(tri)-1])
    print(s.center(10*k))

Pascal_triangle(100)
Pascal_triangle(200)
```

Here I create a list named tri to set every row of the triangle. For each row, I use the list named temp to store. Then I can get the 100th row and the 200th row:

```
    ...: Pascal_triangle(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,
606298174300842802538876, 1818894522902528407616628, 5176853642107196237061726, 13996678365697234270574428, 35991458654650030981476 72,
881170194648328344718912 8, 2056063787512766137677463 2, 4576400043173576241927256 8, 9724850091743849514095420 7, 1974439261051023992255736 93,
383273503615787010261407757, 711793649572175876199757263, 1265410932572757113244012912, 2154618614921181030658724688,
351543037171350589212739291 2, 5498493658321124600506947888, 8247740487481686900760421832, 1186869972588828114987473533 68,
163901091452742930164937070 32, 217264237507124349288404953 68, 2765181204636128081852426683 2, 3379665916777489877819632612 8,
396743390230400985657087306 72, 447391482600239403957992069 28, 484674106150259360137824741 72, 504456722727820966674062486 28,
504456722727820966674062486 28, 484674106150259360137824741 72, 447391482600239403957992069 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, 972485009174384951409542077, 457640004317357624192725 68, 205606378751276613767746 32, 881170194648328344718912 8,
359914586546500309814767 2, 139966783656972342705742 8, 517685364210719623706172, 181889452290252840761628, 606298174300842802538 76,
191462581358160885012 24, 571901217043857188997 6, 161305471473908437922 4, 428786696323047746376, 107196674080761936594, 25144898858450330806,
551961194453787749 4, 113052292839932430 6, 215337700647490344, 38000770702498296, 6186171974825304, 924370524973896, 126050526132804,
15579278510796, 1731030945644, 171200862756, 14887031544, 1120529256, 71523144, 3764376, 156849, 4851, 99, 1]
```

```
In [26]: Pascal_triangle(200)
[1, 199, 19701, 1293699, 63391251, 2472258789, 79936367511, 2203959847089, 52895036330136, 1122550215450664, 21328454093562616, 366461620334848584, 5741232051912627816,
825854149005893385 84, 109720622653640121261 6, 1353221012728228161222264, 15562041646374623865603 6, 167520801252150362788556 4, 169382143488285366819540 36, 161358778796735007338614764,
145222990917061506604753287 6, 1237852345912095699154801832 4, 10015350798743319747707033007 6, 7707456146850765014962819232 4, 565214145076905610109727341037 6,
395649015538339270760091387263 2, 264781087962950397351403038993768, 16965603043552006941404713239230 32, 1042172758389623283543233846955768, 61452255753319160029629978545826 32,
348229449268808607501236545089108248, 18984121580170533763777089071204933 52, 996666638343145302259829717623825909 98, 50437359403955349931489558437326947102,
246252990031076120253743264881256829498, 11609069530036445669105039630111639339062, 528857611923882524925896249816413476683 8, 232983218225926625845732672216419991076 2,
993244246121055615447597181554211540918 38, 4100315990397178309924696057185334822766 2, 16401263961588713239698784228741339291062 48, 6360490170469769280761235835048470603119352,
239275582603386558657208395699442465549518 48, 873634103923992783944585610421503974585352, 2392755826033865586572083956994424655459184 8,
636049017046976928076123583504847060311935 2, 16401263961588713239698784228741339291062 48, 4100315990397178309924696057185334822765 62, 993244246121055615447597181554211540918 38,
232983218225926625845732672216419991076 2, 528857611923882524925896249816413476683 8, 116090695300364456691050396301163933906 2, 246252990031076120253743264881256829498,
504373594083955349931489584373269471102, 9966663834314530225982971762382590098, 1898412158917053376377089071204933 52, 348229449268808607501236545093108248,
614522557533191660296299785458426 32, 10421727583896232835434323846955768, 16965603043552006941404713239230 32, 2647810879629503973514030389937 68, 395649901553833927076809138726 32,
565214145076905610109727341037 6, 7707456146850765014962819232 4, 10015350798743319747707033007 6, 1237852345912095699154801832 4, 145222990917061506604753287 6, 161358778796735007338614764,
169382143488285366819540 36, 167520801252150362788556 4, 155620416463746238656036 6, 135322101272822816222264, 109720622653640121261 6, 825854149005893385 84, 574123205191262781 6,
366461620334848584, 21328454093562616, 1122550215450664, 52895036330136, 2203959847089, 79936367511, 2472258789, 63391251, 1293699, 19701, 199, 1]
```

```
5503181105097755686672152294396168329802329028735635611560, 7294914488152838933495643739083292902296110572975144880440, 9475003875416905741206985546165656298384603387887257143560,
1205909584143969821608161679678471989252167679482201454554 40, 1503999593707647755039392802731504585055124991294872073656 0, 183822172564268058949259120333850560395626387824928809002 40,
2201826023022551475326290562240627591552008381639257162776 0, 2584752287896038688426515007847693259648009839315649712824 0, 2973854782848130533996097912254872890132656481793274400776 0,
3353495818956402517059429560627835386745336032660500920024 0, 3706495378465501504341063564833970064027398255721325958160, 4015369993750429329636948552857013423602968144369810312134 0,
4263743395425713618068100009734766831248512565701035692266 0, 4437773738096150908601491846866798130483145731616792251134 0, 4527425732805164058270288853874208193725229483770666842066 0,
4527425732805164058270208853874208193725229483770666842066 0, 4437773738096150908601491846866798130483145731616792251134 0, 4263743395425713618068100009734766831248512565701035692266 0,
4015369993750429329636948552857013423602968144369810312134 0, 3706495378465501504341063564833970064027398255721325958160, 3353495818956402517059429560627835386745336032660500920024 0,
2973854782848130533996097912254872890132656481793274400776 0, 2584752287896038688426515007847693259648009839315649712824 0, 2201826023022551475326290562240627591552008381639257162776 0,
1838221725642680589492591203338505603956263878249288090024 0, 1503999593707647755039392802731504585055124991294872073656 0, 1205909584143969821608161679678471989252167679482201454554 40,
947500387541690574120698554616565629838460338788725714356 0, 729491448815283893349564373908329290229611057297514488044 0, 550318110509775568667215229439616832980232902873563561160,
406756864289834115971419952194499398289737362999503505172 40, 294548074140914359841373068834099564278775331822881846607 60, 208952907296546068433191422831565353705246455163505385295 40,
145204562697599821315390023066810085078356722326520240760, 98836719147105760727287729866522422709823067910488556724 0, 65891146098307050715152515332443482851932157386069923781 60,
430198391879964468179379100217384417605487726528532213840, 275044873497026463262225982106196598625249399116845301 60, 172182563083504371310409192085220623556214799782111453840,
105531248341502679190305956417877161889292941801939278160, 63318749004901607514183573850726297133575765081163566896, 371872017965295155242030351309156714189560369968302412304,
213753207176901939627308877603814183924244646274494318096, 120236179037007341040361243652145478457387613529402979 76, 661765016405234202625244054204948286592825747077974892521,
356335008833587647567439106112798466269061681121724981 9, 187687905416164486123307620776970184523398900743503981, 966877088507514019423099864608634283908418579587747869,
487073420526341648888231346562991351144258680325097073 1, 239901833990586185270393199489360386232915888168388569, 115508290439911866982041170124506852630663205414409311,
543568425599585256386076094703561659438415084303102 64, 249962122720977161695786817272440763099417155555441 36, 112301823251453507428541903412255995015688017133650264,
492835339448824744830291806341555086040035284282493 6, 211215154578067747784410774146380751160015121853544 4, 883808055546524618388669196782727965846871786403256,
360992022688017097651709953615480436754356081770344, 143891925127391430532499771720855838426561515111256, 559579708828744452070832445581106038325517003210 44,
212254372314351343888936444875591945571747828803 96, 785050418148970723972778631731641442525642654480 4, 283045388856431621568416785590319703767748712159 6,
```

```
994483798684759751456599516938961121346144123804, 340393783442434545800581042710651122071498995396, 113464594480811515266860347570217040690499665132,
368196366196010877356036889466262718134734012 68, 116272536693477119165064280884084384674126533032, 357177071502838209199870666768102358149277576 8,
106689255721626997553208121242420184901732263 2, 30974300048214289612221712618767158455341624 8, 873634103923992783944585610421503974585352, 2392755826033865586572083956994424655459184 8,
636049017046976928076123583504847060311935 2, 16401263961588713239698784228741339291062 48, 4100315990397178309924696057185334822765 62, 993244246121055615447597181554211540918 38,
232983218225926625845732672216419991076 2, 528857611923882524925896249816413476683 8, 116090695300364456691050396301163933906 2, 246252990031076120253743264881256829498,
504373594083955349931489584373269471102, 9966663834314530225982971762382590098, 1898412158917053376377089071204933 52, 348229449268808607501236545093108248,
614522557533191660296299785458426 32, 10421727583896232835434323846955768, 16965603043552006941404713239230 32, 2647810879629503973514030389937 68, 395649901553833927076809138726 32,
565214145076905610109727341037 6, 7707456146850765014962819232 4, 10015350798743319747707033007 6, 1237852345912095699154801832 4, 145222990917061506604753287 6, 161358778796735007338614764,
169382143488285366819540 36, 167520801252150362788556 4, 155620416463746238656036 6, 135322101272822816222264, 109720622653640121261 6, 825854149005893385 84, 574123205191262781 6,
366461620334848584, 21328454093562616, 1122550215450664, 52895036330136, 2203959847089, 79936367511, 2472258789, 63391251, 1293699, 19701, 199, 1]
```

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`.

In this problem, my classmate Weihao Deng told me a fact. And it is the key to solve this problem.

The thing that should be thought carefully is that x/2 is always no more than x-1 when x is larger than 1. So, for a certain number x, if x can be divided by 2, use 2 to divide first. Otherwise, minus 1 first, then divide by 2.

Here are my codes for problem 4:

```
@author: zyq17713
"""

import random
x=random.randint(1,100)
print("x="+str(x))

def Least_moves(x):
    n=0
    if x==1:
        n=0
    else:
        #For x>=2,x/2<=x-1
        while x!=1:
            if x%2==0:
                x=x/2
                n=n+1
            else:
                x=x-1
                n=n+1
    return n

print("Least_moves("+str(x)+")="+str(Least_moves(x)))
```

Here are my outputs:

```
    ...: def Least_moves(x):
    ...:     n=0
    ...:     if x==1:
    ...:         n=0
    ...:     else:
    ...:         #For x>=2,x/2<=x-1
    ...:         while x!=1:
    ...:             if x%2==0:
    ...:                 x=x/2
    ...:                 n=n+1
    ...:             else:
    ...:                 x=x-1
    ...:                 n=n+1
    ...:     return n
    ...:
    ...: print("Least_moves("+str(x)+")="+str(Least_moves(x)))
x=12
Least_moves(12)=4
```

In addition, for any x, it can print the Least_moves.

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

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

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`?

Actually, this problem has puzzled me for a long time. I found no way to deal with it. And I asked my classmate Shuai Wang for advice, and read his codes for 5.1. I noticed that he used the thought of recursion, which means use the function in the function itself. In this way, I get the codes for 5.1:

```
#5.1 The space between each number can be filled with "+","-" or "",递归思想
import random
x=random.randint(1,100)
print("x="+str(x))

def fun(x, digit, aa: str):
    if len(digit) == 1:
        for i in range(len(digit)):
            a = digit[i]
            b = digit[:]
            b.pop(i)
            temps = str(a)
            aa = aa + temps
            if eval(aa) == x:
                print(aa + "=" + str(x))
    else:
        for i in range(len(digit)):
            a = digit[i]
            b = digit[:]
            b.pop(i)
            temps = str(a)
            aa = aa + temps
            fun(x, b, aa + '+')
            fun(x, b, aa + '-')
            fun(x, b, aa + '')
            return True

def Find_expression(x):
    digit = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    return fun(x,digit,'')

result=Find_expression(x)
```

Here are my outputs for 5.1:

```
x=22
1+2-3-4-56-7+89=22
1-2+34+5-6+7-8-9=22
1-23+4-56+7+89=22
1-23-4+56-7+8-9=22
12+3+4+5+6-7+8-9=22
12+3-4-5+6-7+8+9=22
12-3+4+5-6-7+8+9=22
12-3+4-5+6+7-8+9=22
12-3-4+5+6+7+8-9=22
12-3-4-5-67+89=22
123-4+5-6-7-89=22
```

However, in this way, it is hard to count the number of solutions for a certain x by adding “count” to codes. That is because the codes of recursion can set the count 0 each time. Therefore, I changed another view to solve 5.2.

The new way is the method of exhaustion. I use 8 for loops to calculate all possibilities. Here are my codes for 5.2:

```
def Find_count(x):
    sym = ['+','-','']
    s=''
    count=0
    for i in range(3):
        for j in range(3):
            for k in range(3):
                for l in range(3):
                    for m in range(3):
                        for n in range(3):
                            for o in range(3):
                                for p in range(3):
                                    digit = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
                                    digit.insert(1,sym[i])
                                    digit.insert(3,sym[j])
                                    digit.insert(5,sym[k])
                                    digit.insert(7,sym[l])
                                    digit.insert(9,sym[m])
                                    digit.insert(11,sym[n])
                                    digit.insert(13,sym[o])
                                    digit.insert(15,sym[p])
                                    ex=s.join(digit)
                                    if eval(ex) == x:
                                        count=count+1
                                        continue
    return count

Total_solutions=[]

for i in range(100):
    temp=Find_count(i+1)
    Total_solutions.append(temp)
print(Total_solutions)

a=max(Total_solutions)
b=min(Total_solutions)
for j in range(100):
    if Total_solutions[j] == a:
        print("i="+str(j+1)+" can yield the maximum number "+str(a)+" of Total_solutions.")
    if Total_solutions[j] == b:
        print("i="+str(j+1)+" can yield the minimum number "+str(b)+" of Total_solutions.")
```

We can easily get the list Total_solutions and find the index of maximum and minimum number. Here are my outputs for 5.2:

```
[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]
i=1 can yield the maximum number 26 of Total_solutions.
i=45 can yield the maximum number 26 of Total_solutions.
i=88 can yield the minimum number 6 of Total_solutions.
```