In [1]:
```
import numpy as np
import random
```

1. Flowchart

1.1 Write a function

In [2]:
```
def Print_values(a,b,c):
    if a>b:
        if b>c:
            print(str(a) + ", " + str(b) + ", " + str(c))
        else:
            if a>c:
                print(str(a) + ", " + str(c) + ", " + str(b))
            else:
                print(str(c) + ", " + str(a) + ", " + str(b))
    else:
        if b>c:
            if a>c:
                print(str(b) + ", " + str(a) + ", " + str(c))
            else:
                print(str(b) + ", " + str(c) + ", " + str(a))
        else:
            print(str(c) + ", " + str(b) + ", " + str(a))
```

1.2 Check output with some random a, b, and c

In [3]:
```
for _ in range(3):
    a = random.randint(1,1000)
    b = random.randint(1,1000)
    c = random.randint(1,1000)
print("Random numbers a, b, c:\n" + str(a) + ", " + str(b) + ", " + str(c) +"\n")


print("Print a, b and c in the given order:")
Print_values(a,b,c)
```

```
Random numbers a, b, c:
938, 781, 928

Print a, b and c in the given order:
938, 928, 781
```

2. Matrix multiplication

2.1 Make two matrices (M1, M2) filled with random integers

In [4]:
```
r1 =5
c1 = 10
M1 = [[random.randint(0,50) for _ in range(c1)] for _ in range(r1)]

r2 =10
c2 = 5
M2 = [[random.randint(0,50) for _ in range(c2)] for _ in range(r2)]
```

In [5]:
```
M1
```

Out[5]:
```
[[8, 43, 3, 24, 5, 20, 38, 40, 24, 3],
 [14, 16, 4, 46, 30, 12, 19, 49, 17, 31],
 [22, 42, 16, 25, 1, 5, 48, 29, 35, 28],
 [32, 15, 33, 10, 19, 11, 16, 21, 25, 32],
 [50, 15, 28, 7, 31, 2, 42, 2, 36, 9]]
```

In [6]:
```
M2
```

Out[6]:
```
[[3, 34, 17, 49, 27],
 [6, 20, 11, 25, 45],
 [23, 21, 1, 9, 47],
```

```
[17, 5, 34, 10, 49],
[48, 6, 31, 45, 34],
[15, 12, 14, 33, 5],
[44, 39, 9, 40, 2],
[49, 33, 0, 47, 6],
[28, 6, 44, 50, 48],
[20, 16, 30, 30, 20]]
```

2.2 Write a function to do multiplication

In [7]:
```
def Matrix_nultip(M1,M2):

    mul = [[0 for _ in range(c2)] for _ in range(r1)]

    for i in range(r1):
        for j in range(c2):
            for k in range(c1):
                mul[i][j] += M1[i][k] * M2[k][j]

    return mul
```

In [8]:
```
result1 = Matrix_nultip(M1, M2)
result1
```

Out[8]:
```
[[5663, 4579, 3351, 7309, 5266],
 [6965, 4390, 4929, 8171, 6388],
 [6307, 5602, 4615, 8605, 7030],
 [5265, 4356, 4029, 7395, 6279],
 [5655, 4897, 4502, 8452, 6752]]
```

Use the function "np.dot( )" to check the result

In [9]:
```
result2 = np.dot(M1, M2)
result1 == result2
```

Out[9]:
```
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

In [10]:
```
n=300
k=100
```

In [11]:
```
def coeff(n,k):
    if k==0 or k==n:
        return 1
    else:
        return coeff(n-1,k-1)+coeff(n-1,k)
```

In [12]:
```
def Pascal_triangle(k) :

    line = []

    line.append(1)  #The first number of a line

# For the input do not match the row numbers:
    if k<=0:
        return "Please input a positive integer to get the k-th line of Pascal's triangle"

# For the 1-st line:
    if k==1:
        return line

    # Generate the previous row
    prev = Pascal_triangle(k - 1)

    for i in range(1, len(prev)) :
        num = prev[i - 1] + prev[i]
        line.append(num)
```

```
    line.append(1)   #The last number of a line

    # Return the row
    return line
```

In [13]:
```
Pascal_triangle(100)
```

Out[13]:
```
[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,
 6062981743008428025387 6,
 1914625813581608850122 4,
 5719012170438571889976,
 1613054714739084379224,
 428786696323047746376,
 107196674080761936594,
 25144898858450330806,
 5519611944537877494,
 1130522928399324306,
 215337700647490344,
 38000770702498296,
 6186171974825304,
 924370524973896,
 126050526132804,
 15579278510796,
 1731030945644,
 171200862756,
 14887031544,
 1120529256,
 71523144,
 3764376,
 156849,
 4851,
 99,
 1]
```

In [15]:
```
Pascal_triangle(200)
```

Out[15]:
```
[1,
 199,
 19701,
 1293699,
 63391251,
 2472258789,
 79936367511,
 2203959847089,
 52895036330136,
 1122550215450664,
 21328454093562616,
 366461620334848584,
 5741232051912627816,
 82585414900589338584,
 1097206226536401212616,
 13532210127282281622264,
 155620416463746238656036,
 1675208012521503627885564,
 16938214348828536681954036,
 161358778796735007338614764,
 14522290091706150660475328 6,
 12378523459120956991548018324,
 10015350798743319747707033007 6,
 77074656146850765014962819232 4,
 56521414507690561010972734103 6,
 39564990155383392707680913872632,
 26478108796295039735140303899376 8,
 16965603043552006941404713239230 2,
 10421727583896232835434323846955768,
 61452255753319166029629978545842632,
 34822944926880860750123654509310824 8,
 18984121589170533763777089071204933 52,
 99666638343145302259829717623825900 98,
 50437359403955349931489584373269471102,
 24625299003107612025374326488125682949 8,
 11609069530036445669105039630116393390 62,
 52885761192388252492589624981641347668 38,
 23298321822592662584573267221641999107962,
 99324424612105561544759718155421154091838,
 41003159903971783099246960571853348227656 2,
 16401263961588713239698784228741339291062 48,
 63604901704697692807612358350484706031193 52,
 23927558260338655865720839569944246554591848,
 87363410392399278393445856104215039745835352,
 30974300048214289612221712618767150455341624 8,
 10668925572162699755320812124242018490173226 32,
 35717707350283820919987066676810235814927757 68,
```

116272536693477119165064280884084384674126530 32,
368196366196010877356036889466267218134734012 68,
113464594480811515266860347570217040690499665132,
340393783442434545800581042710651122071498995396,
994483798684759751456599516938961121346144123804,
283045388856431621568416785590319703767748712159 6,
785050418148970723972778631731641442525642654480 4,
212254372314351343888936444875591945571747828803 96,
559579708828744452070832445581106038325517003210 44,
143891925127391430532499771720855838426561515111256,
360992022688017097651709953615480436754356081770344,
883808055546524618388669196782727965846871786403256,
211215145478067747784410774146380751160015121835354 4,
492835339448824744830291806341555086040035284282493 6,
112301823251453507428541903412255995015680171336502 64,
249962122720977161695786817272440763099417155555441 36,
543568425599585256386076094703561659438415084303102 64,
115508290439911866982041170124506852630663205414409311,
239901833990586185270393199489360386232915888168388569,
487073420526341648882313465629913511442586803250970731,
966877088507514019423099864608634283908418579587747869,
187687905416164486123307620776970184523398900743503998 1,
356335008833587647567439106112798466269061681121724981 9,
661765016405234202625244054209482865928257407797489252 1,
120236179037007341040361243652145478457387613529402976 79,
213753207176901939627308877603814183924244646274494180 96,
371872017965295155242030513091567141895603699683024123 04,
633187490049016075141835738507262971335757650811635668 96,
105531248341502679190305956417877161889292941801939278160,
172182563083504371310499192050220632556214799782111453840,
275044873497026463262225982106196594862524939911684530160,
430198391879964468179379100217384417605487726528532213840,
658911460980705071515251533244348285193215378606992378160,
988367191471057607272877299866522427789823067910488567240,
145204562697599821315398023066810085070356722322652024076 0,
208952907296546084331914228315653537052464551635035839524 0,
294548074140914359841373068830499564278775331822881846076 0,
406756864289834115971419952194499398289737362993503501724 0,
550318110509775568667215229439616832980232902873563561156 0,
729491448815283893349564373908329290229611057297514488044 0,
947500387541690574120698554616565629838460338788725714356 0,
120590958414396982160816179678471989252167679482201454554 40,
150399959370764775503939280273150458505512499129487207365 60,
183822172564268058949259120333850560395626387824928809002 40,
220182602302255147532629056224062759155200838163925716277 60,
258475228789603868842651500784769325964800983931564971282 40,
297385478284813053399609791225487289013265648179327440077 60,
335349581895640251705942956062783538674533603266050092002 40,
370649537884655015043410635648339700640273982557213259581 60,
401536999375042932963694855285701342360296814436981031213 40,
426374339542571361806810000973476683124851256567103569226 60,
443777373809615090860149184686679813048314573161679225113 40,
452742573280516405827020885387420819372522948377066684206 60,
452742573280516405827020885387420819372522948377066684206 60,
443777373809615090860149184686679813048314573161679225113 40,
426374339542571361806810000973476683124851256567103569226 60,
401536999375042932963694855285701342360296814436981031213 40,
370649537884655015043410635648339700640273982557213259581 60,
335349581895640251705942956062783538674533603266050092002 40,
297385478284813053399609791225487289013265648179327440077 60,
258475228789603868842651500784769325964800983931564971282 40,
220182602302255147532629056224062759155200838163925716277 60,
183822172564268058949259120333850560395626387824928809002 40,
150399959370764775503939280273150458505512499129487207365 60,
120590958414396982160816179678471989252167679482201454554 40,
947500387541690574120698554616565629838460338788725714356 0,
729491448815283893349564373908329290229611057297514488044 0,
550318110509775568667215229439616832980232902873563561156 0,
406756864289834115971419952194499398289737362993503501724 0,
294548074140914359841373068830499564278775331822881846076 0,
208952907296546084331914228315653537052464551635035839524 0,
145204562697599821315398023066810085070356722322652024076 0,
988367191471057607272877299866522427789823067910488567240,
658911460980705071515251533244348285193215378606992378160,
430198391879964468179379100217384417605487726528532213840,
275044873497026463262225982106196594862524939911684530160,
172182563083504371310499192050220632556214799782111453840,
105531248341502679190305956417877161889292941801939278160,
633187490049016075141835738507262971335757650811635668 96,
371872017965295155242030513091567141895603699683024123 04,

```
213753207176901939627308877603814183924244646274494180 96,
120236179037007341040361243652145478457387613529402976 79,
661765016405234202625244054209482865928257407797489252 1,
356335008833587647567439106112798466269061681121724981 9,
187687905416164486123307620776970184523398900743503998 1,
966877088507514019423099864608634283908418579587747869,
487073420526341648882313465629913511442586803250970731,
239901833990586185270393199489360386232915888168388569,
115508290439911866982041170124506852630663205414409311,
543568425599585256386076094703561659438415084303102 64,
249962122720977161695786817272440763099417155555441 36,
112301823251453507428541903412255995015680171336502 64,
492835339448824744830291806341555086040035284282493 6,
211215145478067747784410774146380751160015121835354 4,
883808055546524618388669196782727965846871786403256,
360992022688017097651709953615480436754356081770344,
143891925127391430532499771720855838426561515111256,
559579708828744520708324455811060383255170032104 4,
212254372314351343888936444875591945571747828803 96,
785050418148970723972778631731641442525642654480 4,
283045388856431621568416785590319703767748712159 6,
994483798684759751456599516938961121346144123804,
340393783442434545800581042710651122071498995396,
113464594480811515266860347570217040690499665132,
368196366196010877356036889466267218134734012 68,
116272536693477119165064280884084384674126530 32,
357177073502838209199870666768102358149277576 8,
106689255721626997553208121242420184901732263 2,
309743000482142896122217126187671504553416248,
873634103923992783934458561042150397458353 52,
239275582603386558657208395699442465545918 48,
636049017046976928076123583504847060311935 2,
164012639615887132396987842287413392910624 8,
410031599039717830992469605718533482276562,
993244246121055615447597181554211540918 38,
232983218225926625845732672216419991079 62,
528857611923882524925896249816413476683 8,
116090695300364456691050396301163933906 2,
246252990031076120253743264881256829498,
504373594039553499314895843732694711 02,
996666383431453022598297176238259009 8,
189841215891705337637770890712049335 2,
348229449268808607501236545093108248,
614522557533191660296299785458426 32,
104217275838962328354343238469557 68,
169656030435520069414047132392303 2,
264781087962950397351403038993768,
395649901553833927076809138726 32,
565214145076905610109727341037 6,
770746561468507650149628192324,
100153507987433197477070330076,
123785234591209569915480183 24,
145222900917061506604753287 6,
161358778796735007338614764,
169382143488285366819540 36,
167520801252150362788556 4,
155620416463746238656036,
13532210127282281622264,
109720622653640121261 6,
82585414900589338584,
57412320519126278 16,
366461620334848584,
213284540935626 16,
112255021545066 4,
52895036330136,
2203959847089,
79936367511,
2472258789,
63391251,
1293699,
19701,
199,
1]
```

Reference: https://www.geeksforgeeks.org/find-the-nth-row-in-pascals-triangle/

I got inspired by reading the reference code, while my code do several improvements based on it, including: 1) The determination of the kth row starts from 1; 2) Check the input parameter "k" is negtive or not;

4. Add or Double

In [16]:
```
# Define a recursive function to get the least times to reach the target number
def moveTimes(current, target):
        if current == target:
            return 0
        if current > target:
            return float('inf')

        double_moves = 1 + moveTimes(current * 2, target)
        add_moves = 1 + moveTimes(current + 1, target)

        return min(double_moves, add_moves)
```

In [17]:
```
# Define whether the input number is an integer between 1 and 100
def inputCheck(num):
    if num.isnumeric():  # if the input is number or not
        num = int(num)

        if 1 <= num <= 100:  # if the input is in the interval
            return True
    return False
```

In [18]:
```
def least_moves():
    x = input("Please input an integer from 1 to 100:")

    if inputCheck(x):
        min_moves = moveTimes(1, int(x))

        print("\nThe smallest times of moves from 1 RMB to", x, "RMB is", min_moves, ".")

    else:
        print("\nPlease check your input is an integer ranging from 1 to 100")
```

Check the output

In [19]:
```
least_moves()
```

```
Please input an integer from 1 to 100:1

The smallest times of moves from 1 RMB to 1 RMB is 0 .
```

In [20]:
```
least_moves()
```

```
Please input an integer from 1 to 100:2

The smallest times of moves from 1 RMB to 2 RMB is 1 .
```

In [21]:
```
least_moves()
```

```
Please input an integer from 1 to 100:5

The smallest times of moves from 1 RMB to 5 RMB is 3 .
```

In [22]:
```
least_moves()
```

```
Please input an integer from 1 to 100:100

The smallest times of moves from 1 RMB to 100 RMB is 8 .
```

In [23]:
```
least_moves()
```

```
Please input an integer from 1 to 100:13.4

Please check your input is an integer ranging from 1 to 100
```

5. Dynamic programming

In [24]:

```
def Find_expression():

    target = input("Please input an integer from 1 to 100:")

    if inputCheck(target):

        # Initialize a list to store expressions
        expressions = ['1']

        for i in range(2, 10):
            new_expressions = []
            for expression in expressions:
                new_expressions.append(expression + '+' + str(i))
                new_expressions.append(expression + '-' + str(i))
                new_expressions.append(expression + str(i))
            expressions = new_expressions

        # Evaluate and print expressions that match the target
        for expression in expressions:
            if eval(expression) == int(target):
                print(expression + '=' + str(target))


    else:
        print("\nPlease check your input is an integer ranging from 1 to 100")
```

Check the output

In [25]:

```
Find_expression()
```

```
Please input an integer from 1 to 100:50
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
```

In [ ]: