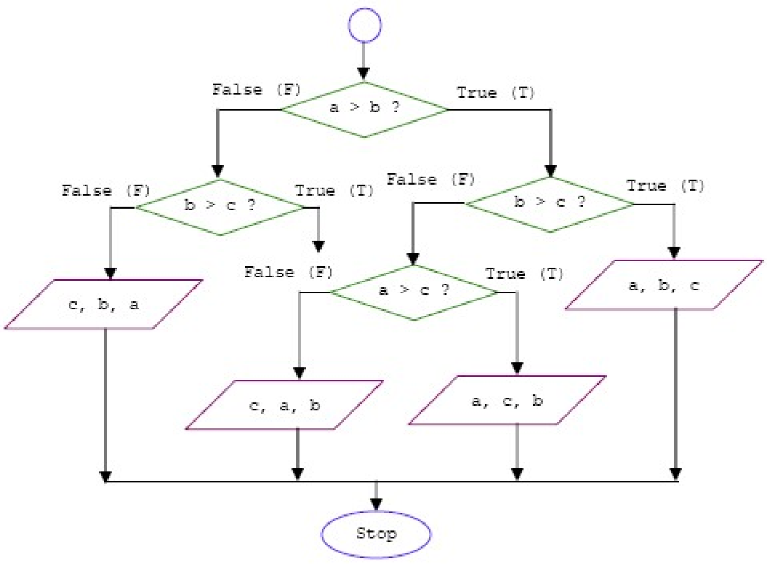
Table of Contents

Assignment 01 Shijing Liang

In [5]:
```
import numpy as np
```

1. Flowchart



In [44]:
```
def Print_values(a,b,c):
    if (a>b):     # if: if condition satisfies, stop running
        if(b>c): # if & elif: elif allows judging from previous states
            print(a,b,c)
        if(b<c):
            if (a>c):
                print(a,c,b)
            else:
                print(c,a,b)
    if (a<b):
        if(b>c):
                print(c,a,b)
        else:
                print(c,b,a)
```

In [51]:
```
a,b,c = 2,3,1
```

In [52]:
```
test = Print_values(a,b,c)
```

```
1 2 3
```

Matrix multiplication

Create random matrices

In [118]:
```
M1 = np.matrix(np.random.rand(5,10)*50)
M2 = np.matrix(np.random.rand(10,5)*50)
```

In [119]:
```
M1.shape, M2.shape
```

Out[119]:
```
((5, 10), (10, 5))
```

Matrix multiplication

In [164]:
```
def Matrix_multip(M1,M2):
    row1,col1 = M1.shape
    row2,col2 = M2.shape

    if (col1 != row2):
        M2_t = M2.reshape((col2,row2))
        row2_t,col2_t = M2_t.shape
        M2 = M2_t
        print('Transpose M2')

        if (col1 != row2_t):
            return print('Matrix Multiplication False: Shape error')

    elif (col1 == row2):
        global out_matrix
        out_matrix = np.matrix(np.zeros((M1.shape[0],M2.shape[1])))
        for i in range(out_matrix.shape[0]):
            for j in range(out_matrix.shape[1]):
                out_matrix[i,j] = M1[i,:]*M2[:,j]

    return out_matrix
```

In [165]:
```
Mul_M = Matrix_multip(M1,M2)
Mul_M
```

Out[165]:
```
matrix([[6978.72739073, 6056.09336446, 7250.3046309 , 4853.52425858,
         5214.79769824],
        [5928.53839183, 7071.11287022, 6962.99832526, 5547.45671514,
         3545.66849203],
        [8567.92537979, 5908.97300459, 8031.38173379, 4924.41726514,
         5718.57890333],
        [7921.98302375, 8875.03474619, 7810.31515819, 7256.9861673 ,
         6130.09163933],
        [5070.0761337 , 6075.58796698, 5668.16034239, 4332.87332894,
         4965.09278146]])
```

Check our results

In [166]:
```
M1*M2
```

Out[166]:
```
matrix([[6978.72739073, 6056.09336446, 7250.3046309 , 4853.52425858,
         5214.79769824],
        [5928.53839183, 7071.11287022, 6962.99832526, 5547.45671514,
         3545.66849203],
        [8567.92537979, 5908.97300459, 8031.38173379, 4924.41726514,
         5718.57890333],
        [7921.98302375, 8875.03474619, 7810.31515819, 7256.9861673 ,
         6130.09163933],
        [5070.0761337 , 6075.58796698, 5668.16034239, 4332.87332894,
         4965.09278146]])
```

In [167]:
```
Mul_M2 = Matrix_multip(M1,M2.transpose())
```

```
Transpose M2
```

In [168]:
```
Mul_M2
```
Out[168]:
```
matrix([[6978.72739073, 6056.09336446, 7250.3046309 , 4853.52425858,
         5214.79769824],
        [5928.53839183, 7071.11287022, 6962.99832526, 5547.45671514,
         3545.66849203],
        [8567.92537979, 5908.97300459, 8031.38173379, 4924.41726514,
         5718.57890333],
        [7921.98302375, 8875.03474619, 7810.31515819, 7256.9861673 ,
         6130.09163933],
        [5070.0761337 , 6075.58796698, 5668.16034239, 4332.87332894,
         4965.09278146]])
```
In [172]:
```
Mul_M3 = Matrix_multip(M1,np.identity(4))
```
```
Transpose M2
Matrix Multiplication False: Shape error
```

Notice the difference between np.multiply and *

In [114]:
```
M3 = np.matrix(np.ones((5,5)))
M4 = np.identity(5)
```
In [115]:
```
M3,M4
```
Out[115]:
```
(matrix([[1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.]]),
 array([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]]))
```

Except for 1-dimension vectors, '*' is matrix multiplication

In [116]:
```
M3*M4
```
Out[116]:
```
matrix([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])
```

np.multiply is the multuplication among elements

In [117]:
```
np.multiply(M3,M4)
```
Out[117]:
```
matrix([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]])
```

Pascal triangle

In [204]:
```
# Import libraries
from math import factorial
import matplotlib.pyplot as plt
```

$$\frac{n!}{k!(n-k)!} = \binom{n}{k}$$

https://www.mathsisfun.com/pascals-triangle.html (https://www.mathsisfun.com/pascals-triangle.html)

In [187]:
```
def Pascal_triangle(n):
    out_array = np.zeros(n+1)
    for k in range(0,n+1):
        out_array[k] = factorial(n) / (factorial(k)*factorial(n-k))
    return out_array
```

In [198]:
```
Pascal_triangle(100)
```

Out[198]:
```
array([1.00000000e+00, 1.00000000e+02, 4.95000000e+03, 1.61700000e+05,
       3.92122500e+06, 7.52875200e+07, 1.19205240e+09, 1.60075608e+10,
       1.86087894e+11, 1.90223181e+12, 1.73103095e+13, 1.41629805e+14,
       1.05042105e+15, 7.11054250e+15, 4.41869427e+16, 2.53338471e+17,
       1.34586063e+18, 6.65013487e+18, 3.06645108e+19, 1.32341573e+20,
       5.35983370e+20, 2.04184141e+21, 7.33206689e+21, 2.48652703e+22,
       7.97760756e+22, 2.42519270e+23, 6.99574817e+23, 1.91735320e+24,
       4.99881370e+24, 1.24108478e+25, 2.93723398e+25, 6.63246383e+25,
       1.43012501e+26, 2.94692427e+26, 5.80717430e+26, 1.09506715e+27,
       1.97720458e+27, 3.42002955e+27, 5.67004899e+27, 9.01392403e+27,
       1.37462341e+28, 2.01164402e+28, 2.82588089e+28, 3.81165329e+28,
       4.93782358e+28, 6.14484712e+28, 7.34709982e+28, 8.44134873e+28,
       9.32065589e+28, 9.89130829e+28, 1.00891345e+29, 9.89130829e+28,
       9.32065589e+28, 8.44134873e+28, 7.34709982e+28, 6.14484712e+28,
       4.93782358e+28, 3.81165329e+28, 2.82588089e+28, 2.01164402e+28,
       1.37462341e+28, 9.01392403e+27, 5.67004899e+27, 3.42002955e+27,
       1.97720458e+27, 1.09506715e+27, 5.80717430e+26, 2.94692427e+26,
       1.43012501e+26, 6.63246383e+25, 2.93723398e+25, 1.24108478e+25,
       4.99881370e+24, 1.91735320e+24, 6.99574817e+23, 2.42519270e+23,
       7.97760756e+22, 2.48652703e+22, 7.33206689e+21, 2.04184141e+21,
       5.35983370e+20, 1.32341573e+20, 3.06645108e+19, 6.65013487e+18,
       1.34586063e+18, 2.53338471e+17, 4.41869427e+16, 7.11054250e+15,
       1.05042105e+15, 1.41629805e+14, 1.73103095e+13, 1.90223181e+12,
       1.86087894e+11, 1.60075608e+10, 1.19205240e+09, 7.52875200e+07,
       3.92122500e+06, 1.61700000e+05, 4.95000000e+03, 1.00000000e+02,
       1.00000000e+00])
```

In [193]:
```
Pascal_triangle(200)
```

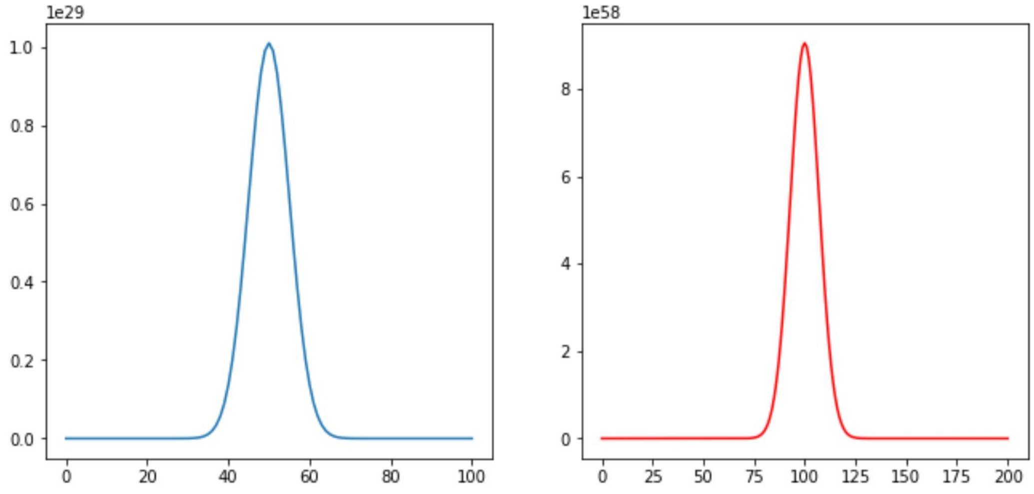
Out[193]:
```
array([1.00000000e+00, 2.00000000e+02, 1.99000000e+04, 1.31340000e+06,
       6.46849500e+07, 2.53565004e+09, 8.24086263e+10, 2.28389621e+12,
       5.50989962e+13, 1.17544525e+15, 2.24510043e+16, 3.87790074e+17,
       6.10769367e+18, 8.83266470e+19, 1.17979164e+21, 1.46294164e+22,
       1.69152627e+23, 1.83082843e+24, 1.86134224e+25, 1.78296993e+26,
       1.61358779e+27, 1.38307525e+28, 1.12532031e+29, 8.70900069e+29,
       6.42288801e+30, 4.52171316e+31, 3.04346078e+32, 1.96134139e+33,
       1.21182879e+34, 7.18739833e+34, 4.09681705e+35, 2.24664161e+36,
       1.18650760e+37, 6.04040232e+37, 2.96690349e+38, 1.40715994e+39,
       6.44948307e+39, 2.85868979e+40, 1.22622746e+41, 5.09356024e+41,
       2.05015800e+42, 8.00061657e+42, 3.02880484e+43, 1.11290969e+44,
       3.97106411e+44, 1.37663556e+45, 4.63866329e+45, 1.51990244e+46,
       4.84468903e+46, 1.50284231e+47, 4.53858378e+47, 1.33487758e+48,
       3.82493769e+48, 1.06809581e+49, 2.90759414e+49, 7.71834081e+49,
       1.99849896e+50, 5.04883948e+50, 1.24480008e+51, 2.99595951e+51,
       7.04050485e+51, 1.61585357e+52, 3.62263946e+52, 7.93530548e+52,
       1.69865133e+53, 3.55410124e+53, 7.26975255e+53, 1.45395051e+54,
       2.84375614e+54, 5.44022914e+54, 1.01810003e+55, 1.86412681e+55,
       3.33989386e+55, 5.85625225e+55, 1.00505951e+56, 1.68849997e+56,
       2.77713811e+56, 4.47227437e+56, 7.05243265e+56, 1.08910985e+57,
       1.64727865e+57, 2.44041282e+57, 3.54157470e+57, 5.03500981e+57,
       7.01304938e+57, 9.57074975e+57, 1.27980956e+58, 1.67699184e+58,
       2.15340997e+58, 2.70990918e+58, 3.34222132e+58, 4.04004775e+58,
       4.78657831e+58, 5.55860707e+58, 6.32735060e+58, 7.05999120e+58,
       7.72186537e+58, 8.27911339e+58, 8.70151713e+58, 8.96519947e+58,
       9.05485147e+58, 8.96519947e+58, 8.70151713e+58, 8.27911339e+58,
       7.72186537e+58, 7.05999120e+58, 6.32735060e+58, 5.55860707e+58,
       4.78657831e+58, 4.04004775e+58, 3.34222132e+58, 2.70990918e+58,
       2.15340997e+58, 1.67699184e+58, 1.27980956e+58, 9.57074975e+57,
       7.01304938e+57, 5.03500981e+57, 3.54157470e+57, 2.44041282e+57,
       1.64727865e+57, 1.08910985e+57, 7.05243265e+56, 4.47227437e+56,
       2.77713811e+56, 1.68849997e+56, 1.00505951e+56, 5.85625225e+55,
       3.33989386e+55, 1.86412681e+55, 1.01810003e+55, 5.44022914e+54,
       2.84375614e+54, 1.45395051e+54, 7.26975255e+53, 3.55410124e+53,
       1.69865133e+53, 7.93530548e+52, 3.62263946e+52, 1.61585357e+52,
       7.04050485e+51, 2.99595951e+51, 1.24480008e+51, 5.04883948e+50,
       1.99849896e+50, 7.71834081e+49, 2.90759414e+49, 1.06809581e+49,
       3.82493769e+48, 1.33487758e+48, 4.53858378e+47, 1.50284231e+47,
       4.84468903e+46, 1.51990244e+46, 4.63866329e+45, 1.37663556e+45,
       3.97106411e+44, 1.11290969e+44, 3.02880484e+43, 8.00061657e+42,
       2.05015800e+42, 5.09356024e+41, 1.22622746e+41, 2.85868979e+40,
       6.44948307e+39, 1.40715994e+39, 2.96690349e+38, 6.04040232e+37,
       1.18650760e+37, 2.24664161e+36, 4.09681705e+35, 7.18739833e+34,
       1.21182879e+34, 1.96134139e+33, 3.04346078e+32, 4.52171316e+31,
       6.42288801e+30, 8.70900069e+29, 1.12532031e+29, 1.38307525e+28,
       1.61358779e+27, 1.78296993e+26, 1.86134224e+25, 1.83082843e+24,
       1.69152627e+23, 1.46294164e+22, 1.17979164e+21, 8.83266470e+19,
       6.10769367e+18, 3.87790074e+17, 2.24510043e+16, 1.17544525e+15,
       5.50989962e+13, 2.28389621e+12, 8.24086263e+10, 2.53565004e+09,
       6.46849500e+07, 1.31340000e+06, 1.99000000e+04, 2.00000000e+02,
       1.00000000e+00])
```

Plot distribution

Here we plot the distribution of these values

In [213]:
```
fig,ax = plt.subplots(1,2,figsize=(11,5))
ax[0].plot(Pascal_triangle(100))
ax[1].plot(Pascal_triangle(200),'r')
```

Out[213]:
```
[<matplotlib.lines.Line2D at 0x7f0304a9eb10>]
```



Add or double

Method1: idea from loss function --> failed

In [1]:
```
def Loss(x_step,x):
    delta = x - sum(x_step)
    return delta
```

In [271]:
```
def Gradient(x,x_step,iters):
    for i in range(iters):
        if (Loss(x_step,x)>0):
            test1 = list(np.asarray(x_step)*2)
            loss1 = Loss(test1,x)

            x_step.append(1)
            test2 = x_step
            loss2 = Loss(test2,x)
            x_step.pop(-1)

            if (loss1<loss2):
                if (loss1>=0):
                    x_step = test1
                    loss[i] = loss1
                else:
                    x_step.append(1)
                    loss[i] = loss2

            else:
                x_step.append(1)
                loss[i] = loss2
        else:
            loss[i] = 'NAN'
        test1 = []
        test2 = []
    return x_step,loss

def Least_moves(rand):
    moves = Gradient(rand,x0_step,iters)[1]
    return len([x for x in moves if np.isnan(x)==False])+1
```

In [279]:
```python
rand=11
iters = 100
loss = np.zeros(iters)
x0_step = [1,]

Gradient(rand,x0_step,iters)
```

Out[279]:
```
([4, 4, 1, 1, 1],
 array([ 9.,   7.,   3.,   2.,   1.,   0., nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
        nan, nan, nan, nan, nan, nan, nan, nan, nan]))
```

In [280]:
```python
Least_moves(11)
```

Out[280]: 6

Problem of method 1 in finding the "least" move

Method 1 confirms the convergence. But the idea of minimizing the routes cannot be accomplished by defining the Loss function.Our Gradient function is able to find "the most rapid descending way", instead of the minimum steps. Here the "most rapif descending way" is always doubling the money, then add 1 to fill in the rest of the money.

Method 2

In [605]:
```python
def Least_moves():
    iters = 100
    x0_step = [1,]
    rand = int(input("Please type a random interger: "))
    xx_step = np.zeros(iters)
    xx_step[0]=rand
    for i in range(1,iters):
        rand_i = rand/2
        if (rand_i != 1):
            if (rand%2 != 0):
                rand_i = rand-1
                xx_step[i] = rand_i
            if (rand%2 ==0):
                xx_step[i] = rand_i
        if (rand_i == 1):
            xx_step[i] = 1
            break
        rand = rand_i
    return xx_step,len([x for x in xx_step if x>0])-1
```

In [610]:
```python
Least_moves()[1]
```

```
Please type a random interger: 15
```

Out[610]: 6

In [609]:
```python
Least_moves()[0]
```

```
Please type a random interger: 15
```

Out[609]:
```
array([15., 14.,  7.,  6.,  3.,  2.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

7: 7-1 --> 6 --> 3×2: 1,2,3,6,7
10: 5×2 --> 5 --> 5-1 --> 4 --> 2×2: 1,2,4,5,10
15: 15-1 --> 7x2 --> 7-1 -> 6 --> 3×2 : 1,2,3,6,7,14,15

Now it works well.

Dynamic programming

Find expression

In [6]:
```python
def calcp(num_x):
    return '+'+ num_x

def calcm(num_x):
    return '-'+ num_x

def calcc(num_xi):
    return num_xi+''
```

In [7]:
```python
def Find_expression(num):
    # Three operations for digits 123456789
    M = []
    for i in range(2,10):
        M.append([calcp('%s'%i),calcm('%s'%i),calcc('%s'%i)])
    test = []
    # List all possible expressions
    for a in range(3):
        for b in range(3):
            for c in range(3):
                for d in range(3):
                    for e in range(3):
                        for f in range(3):
                            for g in range(3):
                                for h in range(3):
                                    test.append(('1'+M[0][a]+M[1][b]+M[2][c]+M[3][d]+M[4][e]+M[5][f]+M
[6][g]+M[7][h]))
    # The result of each espression
    num_sum = np.zeros((len(test)))
    for i in range(len(test)):
        indx = []
        for j in range(len(test[i])):
            if(test[i][j].isdigit()==False):
                indx.append(j)
        indx.append(len(test[i]))
        num_sum[i] = int(test[i][:indx[0]])
        for k in range(1,len(indx)):
            num_sum[i] += int(test[i][indx[k-1]:indx[k]])

    index = np.argwhere(num_sum == num)
    return [test[int(index[i])] for i in range(len(index))]
```

In [8]:
```python
Find_expression(100)
```

Out[8]:
```
['1+2+3-4+5+6+78+9',
 '1+2+34-5+67-8+9',
 '1+23-4+5+6+78-9',
 '1+23-4+56+7+8+9',
 '12+3+4+5-6-7+89',
 '12+3-4+5+67+8+9',
 '12-3-4+5-6+7+89',
 '123+4-5+67-89',
 '123+45-67+8-9',
 '123-4-5-6-7+8-9',
 '123-45-67+89']
```

The iterations here is quite exhausting, but I cannot come up with other simple thoughts to write this.

Total solutions

In [105]:
```python
import matplotlib.pyplot as plt
```

In [57]:
```
def Total_solutions():
    tot_sols = [len(Find_expression(i)) for i in range(101)]
    return tot_sols
```

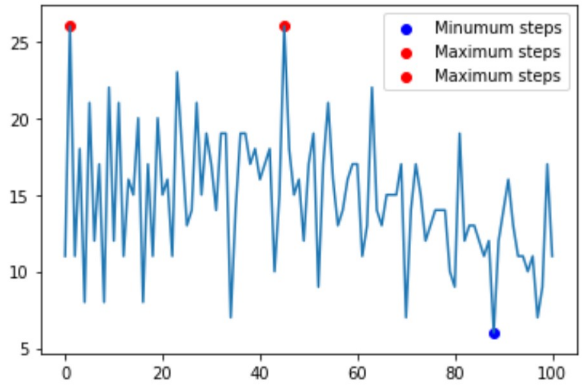
In [107]:
```
tot_sols = Total_solutions()
```

In [108]:
```
x = np.arange(101)
pltdata = np.array(tot_sols)
plt.plot(x,pltdata)

sols_min = [np.argwhere(pltdata==pltdata.min()),pltdata.min()]
sols_max = [np.argwhere(pltdata==pltdata.max()),pltdata.max()]

plt.scatter(sols_min[0],sols_min[1],c='b',label='Minumum steps')
[plt.scatter(sols_max[0][i],sols_max[1],c='r',label='Maximum steps') for i in range(2)]
plt.legend()
```

Out[108]:
```
<matplotlib.legend.Legend at 0x7f3449e35e50>
```



In [110]:
```
sols_min,sols_max
```

Out[110]:
```
([array([[88]]), 6],
 [array([[ 1],
         [45]]), 26])
```

In [ ]: