In [30]:

```
# 1. Flowchart: Print_values
# Ask users to input 3 numbers to assign a,b,c
a=float(input("please input the 1st number as the value of a:"))
b=float(input("please input the 2nd number as the value of b:"))
c=float(input("please input the 3rd number as the value of c:"))
# Compare number size for a,b,c
if(a>b):
    if(b>c):
        # Based on the number size of a,b,c, assign x,y,z
        x=a
        y=b
        z=c
        # Output the value of (x+y-10*z)
        print("x+y-10*z = ", x+y-10*z)
    elif(a>c):
        x=a
        y=c
        z=b
        print("x+y-10*z = ", x+y-10*z)
    else:
        x=c
        y=a
        z=b
        print("x+y-10*z = ", x+y-10*z)
elif(b>c):
    print(None)
else:
    x=c
    y=b
    z=a
    print("x+y-10*z = ", x+y-10*z)
```

```
please input the 1st number as the value of a:10
please input the 2nd number as the value of b:5
please input the 3rd number as the value of c:1
x+y-10*z =  5.0
```

In [29]:

```
# 2. Continuous celing function
from math import *
import numpy as np
# Ask users to input the range and the item number to create a random list
M=int(input("Please input the range of positive intergers (>0):"))
N=int(input("Please input the number of positive intergers:"))
list=np.random.randint(1,M+1,N)
print("The random list is shown below:\n",list)
# Define a function F(x) to get the value of F(x) = F(ceil(x/3)) + 2x, where F(1) = 1
y=0
def F(x):
    if(x==1):
        return 1
    else:
        y=F(ceil(x/3))+2*x
        return y
# Use a "for" loop and the function F(x) to get the value of F(list[i])
for i in range(N):
    print ("x=", list[i],",F(x)=", F(list[i]))
```

```
Please input the range of positive intergers (>0):50
Please input the number of positive intergers:9
The random list is shown below:
 [40 49 27  5 40  4 42 15 24]
x= 40 ,F(x)= 123
x= 49 ,F(x)= 149
x= 27 ,F(x)= 79
x= 5 ,F(x)= 15
x= 40 ,F(x)= 123
x= 4 ,F(x)= 13
x= 42 ,F(x)= 127
x= 15 ,F(x)= 45
x= 24 ,F(x)= 71
```

In [28]:

```
# 3. Dice Rolling
# 3.1 Find_number_of_ways
# Set the total number of dices(D)
D=10
# Define a function to calculate the probability of sum X for D dices
def Prob(D,X):
    prob=0
    # For each dice, the probability to toss “0/1/2/3/4/5/6” is (1/6).
    # When sum X is equal to the number of dices(D), the probability is (1/6)^D
    if X==D:
        return pow(1/6,D)
    # When sum X is smaller than D or larger than 6D, the probability is 0
    if X<D or X>6*D:
        return 0
    else:
    # The probability of sum X for D dices is equal to the probability of
    # sum (X-a) for (D-1) dices, where "a" is the potint of the last dice, belonging to 1-6.
    # Inspired from https://blog.csdn.net/yue_luo_/article/details/95517498
        for i in range(6):
            prob += Prob(D-1, X-i-1)
        return prob/6
# Define a function to calculate the number of ways to reach sum X
def Find_number_of_ways(X):
    # The total number of ways is (6^10). The number of ways for each sum(X) is probability*(6^10).
    print("The number of ways is",round(Prob(D,X)*(6**10)))
# Ask users to input a sum X and find its number of ways
x=int(input("Please input a sum X:"))
Find_number_of_ways(x)

# 3.2 List: Number_of_ways
# Use a "for" loop to calculate 51 times to get Number_of_ways for sum X, 10-60 respectively.
# Consume around 2.5 min
Number_of_ways=[]
for i in range(10,61):
    ways_number=round(Prob(D,i)*(6**10))
    Number_of_ways.append(ways_number)
print("Number of ways are listed below\n", Number_of_ways)
```

```
Please input a sum X:13
The number of ways is 220
Number of ways are listed below
 [1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 38347
0, 576565, 831204, 1151370, 1535040, 1972630, 2446300, 2930455, 3393610, 3801535, 41
21260, 4325310, 4395456, 4325310, 4121260, 3801535, 3393610, 2930455, 2446300, 19726
30, 1535040, 1151370, 831204, 576565, 383470, 243925, 147940, 85228, 46420, 23760, 1
1340, 4995, 2002, 715, 220, 55, 10, 1]
```

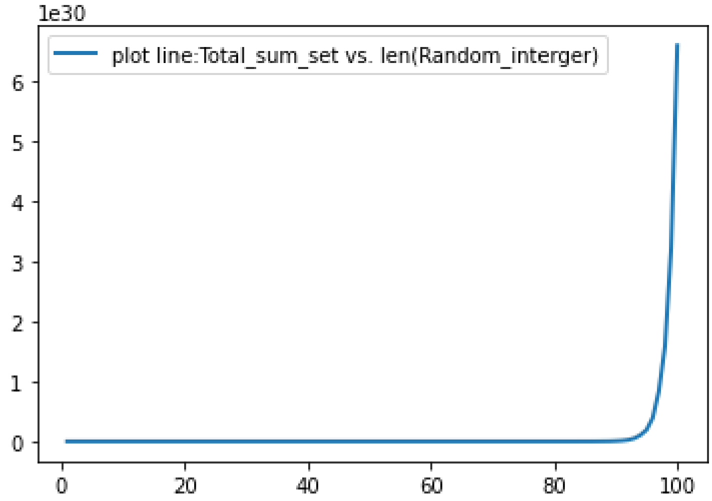
In [27]:

```
# 4.Dynamic programming
import numpy as np
import math
# 4.1 Random_integer function
# Ask users to decide the size of an array and create an array
N=int(input("Please input the number of elements to create an array:"))
Random_interger=np.random.randint(0,11,N)
print(Random_interger)

# 4.2 Sum_averages function
# Define a function to calculate the sum of subset average(SA)
def Sum_averages(array):
    # For an array with N elements, sum of all subset average with n elements (n<=N) is
    # equal to (array.sum() * C(N-1,n-1)/n).Here use a "for" loop to achieve the sum of C(N-1,n-1)/n
    # for all subsets, whose element number increasing from 1 to N
    # Discussed with Wenting Yuan and inspired from https://www.geeksforgeeks.org/sum-average-subset
    SA=0
    for i in range(0,len(array)):
        SA+=(math.factorial(len(array)-1)/math.factorial(len(array)-1-i)/math.factorial(i))/(i+1)
    SA=SA*array.sum()
    return SA
print("The Sum_averages for each subset is:\n", Sum_averages(Random_interger))

# 4.3 Total_sum_set function
import matplotlib.pyplot as plt
Total_sum_set=[]
# Use a "for" loop to calculate Sum_averages for random list with N increasing from 1 to 100
# Each result of Sum_averages(np.random.randint(0,11,j+1)) appended in the Total_sum_set
for j in range(100):
    sum=Sum_averages(np.random.randint(0,11,j+1))
    Total_sum_set.append(sum)
print("Sum_averages with element number increasing from 1 to 100 are shown below:\n",Total_sum_set)
# Plot Total_sum_set
x=np.arange(1,101,1)
y=Total_sum_set
plt.plot(x, y, ls="-", lw=2, label="plot line:Total_sum_set vs. len(Random_interger)")
plt.legend()
plt.show()
```

```
Please input the number of elements to create an array:5
[5 5 9 5 6]
The Sum_averages for each subset is:
 186.0
Sum_averages with element number increasing from 1 to 100 are shown below:
 [7.0, 12.0, 28.0, 63.75, 155.0, 283.5, 489.85714285714283, 1211.25, 2668.5555555555
56, 5626.5, 8374.09090909091, 17744.999999999996, 49776.07692307692, 57340.5, 18567
9.66666666666, 245756.25, 693905.2941176471, 859246.5, 2621435.0, 5347732.5, 1088521
2.333333332, 21924765.68181818, 46319699.52173913, 91575631.875, 144955141.92, 28392
2112.6923077, 830161496.6296295, 1370938216.6071432, 2776918505.172414, 4617089838.9
00001, 13092722880.096773, 23890755578.4375, 41387866665.72727, 97521022127.02942, 1
85542587181.80002, 376048247688.74994, 683480201044.973, 1540763004706.8157, 2551430
828552.4873, 6542094185261.251, 9332440157704.244, 20419501658692.5, 43366784202508.
93, 84362528531171.9, 164975611349852.03, 304421306333803.0, 838436100840246.0, 1495
335813775354.5, 2665395697831513.0, 4661225614328459.0, 1.0243481505391714e+16, 2.33
84074988269876e+16, 4.622562636395377e+16, 8.373359307185142e+16, 1.860396064251957
8e+17, 3.1010500291322554e+17, 6.85179227518543e+17, 1.3914569883186086e+18, 2.960467
931320971e+18, 6.302637558517427e+18, 1.0659798829479703e+19, 2.179393553869717e+19,
4.6116860184273895e+19, 8.81984951024238e+19, 1.8333225648640567e+20, 3.957665092177
6854e+20, 6.916152404949014e+20, 1.4496970636750558e+21, 3.39633966249864e+21, 5.801
764536096992e+21, 1.0874745351397e+22, 2.289035975724314e+22, 4.864684376873936e+22,
1.0618943010128495e+23, 1.863760638572553e+23, 4.0065551423083514e+23, 8.18379978537
8254e+23, 1.4065386939747132e+24, 2.9611031151320355e+24, 5.984182807092415e+24, 1.2
41760841875767e+25, 2.317599254188046e+25, 4.812398684345463e+25, 1.0293139835575988
e+26, 1.961588812249996e+26, 3.5986629048993615e+26, 7.932776688524129e+26, 1.716235
0544638217e+27, 2.82361604466216e+27, 6.712385990791839e+27, 1.2352192919462915e+28,
2.4382036425925093e+28, 5.026302783163007e+28, 1.0388160670088383e+29, 1.90981570692
27936e+29, 3.895384656951331e+29, 8.347544545317335e+29, 1.584563250285287e+30, 3.22
03447066404013e+30, 6.591783121186792e+30]
```



In [24]:

```
# 5. Path counting
# 5.1 Create a matrix
import numpy as np
# Ask users to decide the size of a matrix and create a matrix
N = int(input("Please input the number of rows:"))
M = int(input("Please input the number of columns:"))
arr=np.random.randint(2,size=(N,M))
arr[0,0]=1
arr[N-1,M-1]=1
print(arr)

# 5.2 Count_path function
# Define a function Count_path(matrix,a,b) to count the pathway for a specific matrix,
# where "a" is row number, "b" is column number.
# Regard each element as a point, the number of pathways to a point is equal to
# the pathway number of its upper point plus that of its left point, because
# only moving either rightward or downward is allowed.
# Inspired from https://www.geeksforgeeks.org/count-number-of-ways-to-reach-destination-in-a-maze/?r
def Count_path(matrix,a,b):
    # Substract 1 from each element for counting purposes, and the matrix is
    # refilled by -1 or 0, which were 0 or 1 before, respectively.
    matrix=np.add(matrix,-1)
    # Initialize the leftmost column
    for i in range(a):
        # If meet a blockage, break the loop
        if(matrix[i,0] != 0):
            break
        # If meet an access, each element plus 1, which means
        # the number of pathways from entrance[0,0] to the element[i,0] is 1
        else:
            matrix[i,0] +=1
    # Initialize the uppermost row, whose solution is similar to the leftmost column
    for j in range(b):
        if(matrix[0,j] != 0):
            break
        else:
            matrix[0,j] +=1
    # Since the pathway number of a point is equal to the pathway number of
    # its upper point plus that of its left point,
    # two "for" loops were used to count the pathways for each point, up to the last element[a-1,b-1
    for i in range(1,a,1):
        for j in range(1,b,1):
            if(matrix[i,j] != 0):
                continue
            if(matrix[i-1,j] >0):
                matrix[i,j] += matrix[i-1,j]
            if(matrix[i,j-1] >0):
                matrix[i,j] += matrix[i,j-1]
    Count_path = matrix[a-1,b-1]
    # Return the pathway number of the last element[a-1,b-1]
    if (Count_path >= 0):
        return Count_path
    # If the value of last element[a-1,b-1] is smaller than zero, return 0
    else:
        return 0
# Print the pathway number of the matrix in 5.1
print(Count_path(arr,N,M))

# 5.3 Count_path function
```

```python
# Calculate the mean pathway number for 1000 runs by refilling new random matrix
sum_path=0
for k in range(1000):
    sum_path += Count_path(np.random.randint(2,size=(10,8)),10,8)
print("The mean of Count_path for matrixes with size(10,8) from the 1000 runs is\n", sum_path/1000)
```

```
Please input the number of rows:6
Please input the number of columns:7
[[1 1 1 1 0 0 1]
 [0 0 0 1 1 0 1]
 [1 0 0 1 0 1 0]
 [1 0 1 0 1 0 1]
 [0 1 0 0 0 0 1]
 [1 0 0 0 1 0 1]]
0
The mean of Count_path for matrixes with size(10,8) from the 1000 runs is
 0.05
```