In [1]:

```
def Print_values(a,b,c):
    if a>b:
        if b>c:
            print(a,b,c)
        else:
            if a>c:
                print(a,c,b)
            else:
                print(c,a,b)
    else:
        if b<c:
            print(c,b,a)
#根据流程图，b最大时无输出结果
Print_values(4,13,9)
Print_values(4,2,9)
```

```
9 4 2
```

In [2]:

```
#2.1 creat two matrix
import numpy as np
# 生成5*10的矩阵
M1 = np.random.randint(0,50,size=(5,10))
print( M1 )
# 生成10*5的矩阵
M2 = np.random.randint(0,50,size=(10,5))
print( M2 )

#2.2 A function of matrix multiplication
def Matrix_multip(A, B):
# 获取矩阵的行列个数
    row_A = A.shape[0]
    col_A = A.shape[1]
    row_B = B.shape[0]
    col_B = B.shape[1]
# 判断两矩阵能否做乘法
    if col_A == row_B:
# 建立一个0数组
        result =[[0 for i in range(row_A)] for j in range(col_B)]
        for i in range(0,row_A):
            for j in range(0,col_B):
                for k in range(0,row_B):
                    result[i][j] += A[i][k] * B[k][j]
        print(result)

Matrix_multip(M1, M2)

#用numpy中的dot函数验证计算结果是否正确
print(np.dot(M1,M2))
```

```
[[ 6  8 42 28 25  2  5 29 45 45]
 [41 32  2 26 47 19 47 26  6 20]
 [23 45 28 40 38  8 37  5 30 41]
 [39 11 34 38 16  3 13  2 14 43]
 [ 2  9 15 13  7  7 28  4 36 38]]
[[ 3 47 13 27 26]
 [ 4 22 42 14 33]
 [33 47 13 25 17]
 [ 1 28 22 19  8]
 [ 7 26 25 30 13]
 [37 45 29 40 38]
 [34 18  2 35  7]
 [41 29 31 34 30]
 [45 36 21 24 22]
 [ 8 12 27  5  4]]
[[5457, 7047, 5328, 5152, 3834], [4469, 7586, 5767, 7042, 5018], [4916, 8238, 6581,
6561, 4910], [3042, 6600, 4277, 4453, 3212], [3898, 4230, 3225, 3462, 2325]]
[[5457 7047 5328 5152 3834]
 [4469 7586 5767 7042 5018]
 [4916 8238 6581 6561 4910]
 [3042 6600 4277 4453 3212]
 [3898 4230 3225 3462 2325]]
```

In [3]:

```
#3.pascal triangle
def Pascal_triangle(k):
        result = []
        for i in range(k):
            result.append([1]*(i+1))
            for j in range(1, i):
                result[i][j] = result[i-1][j-1] + result[i-1][j]
        #函数返回第K行的值
        print(result[k-1])

Pascal_triangle(50)
Pascal_triangle(100)
```

```
[1, 49, 1176, 18424, 211876, 1906884, 13983816, 85900584, 450978066, 2054455634, 821
7822536, 29135916264, 92263734836, 262596783764, 675248872536, 1575580702584, 334810
8992991, 6499270398159, 11554258485616, 18851684897584, 28277527346376, 390499187164
24, 49699896548176, 58343356817424, 63205303218876, 63205303218876, 58343356817424,
49699896548176, 39049918716424, 28277527346376, 18851684897584, 11554258485616, 6499
270398159, 3348108992991, 1575580702584, 675248872536, 262596783764, 92263734836, 29
135916264, 8217822536, 2054455634, 450978066, 85900584, 13983816, 1906884, 211876, 1
8424, 1176, 49, 1]
[1, 99, 4851, 156849, 3764376, 71523144, 1120529256, 14887031544, 171200862756, 1731
030945644, 15579278510796, 126050526132804, 924370524973896, 6186171974825304, 38000
770702498296, 215337700647490344, 1130522928399324306, 5519611944537877494, 25144898
858450330806, 107196674080761936594, 428786696323047746376, 1613054714739084379224,
5719012170438571889976, 19146258135816088501224, 60629817430084280253876, 1818894522
90252840761628, 517685364210719623706172, 1399667836569723427057428, 359914586546500
3098147672, 8811701946483283447189128, 20560637875127661376774632, 45764000431735762
419272568, 97248500917438495140954207, 197443926105102399225573693, 3832735036157870
10261407757, 711793649572175876199757263, 1265410932572757113244012912, 215461861492
1181030658724688, 3515430371713505892127392912, 5498493658321124600506947888, 824774
0487481686900760421832, 11868699725888281149874753368, 1639010914527429301649370703
2, 21726423750712434928840495368, 27651812046361280818524266832, 3379665916777489877
8196326128, 39674339023040098565708730672, 44739148260023940935799206928, 4846741061
5025936013782474172, 50445672272782096667406248628, 50445672272782096667406248628, 4
8467410615025936013782474172, 44739148260023940935799206928, 39674339023040098565708
730672, 33796659167774898778196326128, 27651812046361280818524266832, 21726423750712
434928840495368, 16390109145274293016493707032, 11868699725888281149874753368, 82477
40487481686900760421832, 5498493658321124600506947888, 3515430371713505892127392912,
2154618614921181030658724688, 1265410932572757113244012912, 711793649572175876199757
263, 383273503615787010261407757, 197443926105102399225573693, 972485009174384951409
54207, 45764000431735762419272568, 20560637875127661376774632, 881170194648328344718
9128, 3599145865465003098147672, 1399667836569723427057428, 51768536421071962370617
2, 181889452290252840761628, 60629817430084280253876, 19146258135816088501224, 57190
12170438571889976, 1613054714739084379224, 428786696323047746376, 107196674080761936
594, 25144898858450330806, 5519611944537877494, 1130522928399324306, 215337700647490
344, 38000770702498296, 6186171974825304, 924370524973896, 126050526132804, 15579278
510796, 1731030945644, 171200862756, 14887031544, 1120529256, 71523144, 3764376, 156
849, 4851, 99, 1]
```

In [4]:

```
#4 add or double
def Least_moves(x):
    if x == 1:
        step = 0
    elif x >= 2:
        step = 0
        for i in range (100):
            if x >= 2:
                #判断能否除以2，若余数为0继续除以2，并计数一次
                if x % 2 == 0:
                    x = x / 2
                    step += 1
                    continue
                #若除以2余数为1则减去1，并计数一次
                elif x % 2 == 1:
                    x = x - 1
                    step += 1
                    continue
                #若x<2，跳出循环
                else:
                    break
    print(step)

Least_moves(2)
Least_moves(5)
Least_moves(6)
```

```
1
3
3
```

In [8]:

```
#5.1 find expression
def Find_expression(var):
# 建立运算符的LIST
    operator = ['+','-','']
# sep是用于后续join操作的空值
    sep = ''
    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):
                                    # 建立string类型的数字数组
                                    num = ["1","2","3","4","5","6","7","8","9"]
                                    # 在num数组各数字间插入运算符
                                    num.insert(1,operator[i])
                                    num.insert(3,operator[j])
                                    num.insert(5,operator[k])
                                    num.insert(7,operator[l])
                                    num.insert(9,operator[m])
                                    num.insert(11,operator[n])
                                    num.insert(13,operator[o])
                                    num.insert(15,operator[p])
                                    # join函数，利用sep将num数组中的元素连接为字符串
                                    expression = sep.join(num)
                                    # 利用eval函数计算字符串运行结果
                                    res = eval(expression)
                                    if res == var:
                                        print(str(expression) + "=" + str(res))
                                        continue


Find_expression(100)
```

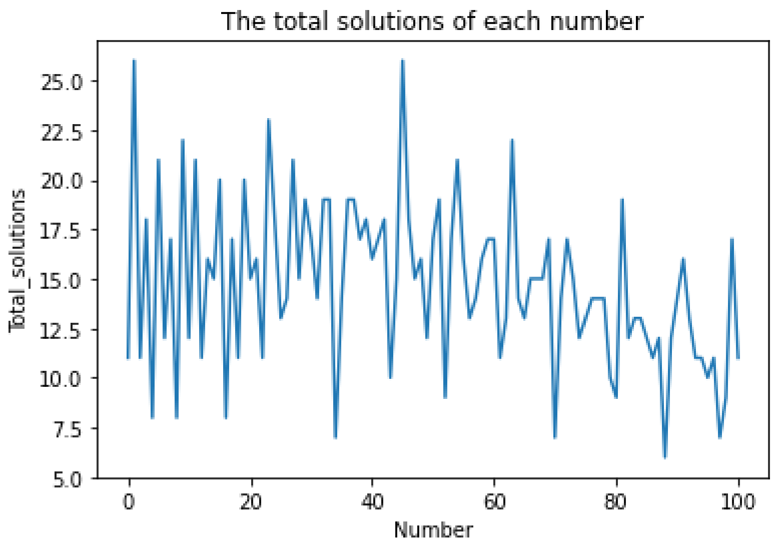
```
1+2+3-4+5+6+78+9=100
1+2+34-5+67-8+9=100
1+23-4+5+6+78-9=100
1+23-4+56+7+8+9=100
12+3+4+5-6-7+89=100
12+3-4+5+67+8+9=100
12-3-4+5-6+7+89=100
123+4-5+67-89=100
123+45-67+8-9=100
123-4-5-6-7+8-9=100
123-45-67+89=100
```

In [7]:

```
#5.2Total solution
def Find_expression(var):
# 建立运算符的LIST
    operator = ['+','-','']
# sep是用于后续join操作的空值
    sep = ''
    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):
                                    # 建立string类型的数字数组
                                    num = ["1","2","3","4","5","6","7","8","9"]
                                    # 在num数组各数字间插入运算符
                                    num.insert(1,operator[i])
                                    num.insert(3,operator[j])
                                    num.insert(5,operator[k])
                                    num.insert(7,operator[l])
                                    num.insert(9,operator[m])
                                    num.insert(11,operator[n])
                                    num.insert(13,operator[o])
                                    num.insert(15,operator[p])
                                    # join函数，利用sep将num数组中的元素连接为字符串
                                    expression = sep.join(num)
                                    # 利用eval函数计算字符串运行结果
                                    res = eval(expression)
                                    if res == var:
                                        # 计算解的个数
                                        count += 1
                                        continue
    return count
import matplotlib.pyplot as plt

# 生成一个长度为101的数组
Total_solutions = [0 for i in range(101)]
for i in range(101):
    Total_solutions[i]=Find_expression(i)
number = range(0,101)

# 画图
plt.plot(number, Total_solutions)

plt.xlabel("Number")
plt.ylabel("Total_solutions")

plt.title("The total solutions of each number")
# 画出每个数字的解的个数的图像
plt.show

# 找到解的个数的最大值/最小值
solution_max = max(Total_solutions)
solution_min = min(Total_solutions)

# 获取解的个数的最大值对应的数字
arr_max = []
```

```
for i in range(101):
    if Total_solutions[i] == solution_max:
        arr_max.append(i)

# 获取解的个数的最小值对应的数字
arr_min = []
for i in range(101):
    if Total_solutions[i] == solution_min:
        arr_min.append(i)


print("The lowest number of solutions is  "+str(arr_max) +", and the number of solution is "+str(sol
print("The largest number of solutions is  "+str(arr_min) +", and the number of solution is "+str(sc
```

```
The lowest number of solutions is  [1, 45], and the number of solution is 6
The largest number of solutions is  [88], and the number of solution is 26
```



In [ ]: