In [8]:
```
# Flowchart
# define a function to compute a,b,c
def compute(x,y,z):
    return x+y-10*z

a=10
b=5
c=1
if a>b:
    if b>c:
        result=compute(a,b,c)
    elif a>c:
        result=compute(a,c,b)
    else:
        result=compute(c,a,b)
else:
    if b>c:
        print("Try again")
    else:
        result=compute(c,b,a)
compute(a,b,c)
```

Out[8]:
```
5
```

In [4]:
```
# Continuous celing function
import math
# define the function
def F_ceil(x):
    if x==1:
        return 1
    else:
        return F_ceil(math.ceil(x/3))+2*x

# 定义一个list
ceil_list=[10,33,5,8,6]

# t代入函数
for i in ceil_list:
    print(F_ceil(i))
```

```
33
101
15
23
17
```

In [25]:
```
# Dice rolling
def Find_number_of_ways(n,x):
    # n为骰子数目
    # x为数字和
    # 参考了这个网站的思路和代码，思路和网站的一致，代码是看了之后自己写的
    # https://www.geeksforgeeks.org/dice-throw-dp-30/
    if n>x:
        return 0
    elif n==1:
        return 1
    else:
        return Find_number_of_ways(n-1,x-1)+Find_number_of_ways(n-1,x-2)+Find_numbe
    # 主要思路就是将n个骰子，和为x的分解成n-1个骰子，和为x-1，x-2，……，x-6
    # 然后利用迭代求出最后的和

# 定义一个零数组
```

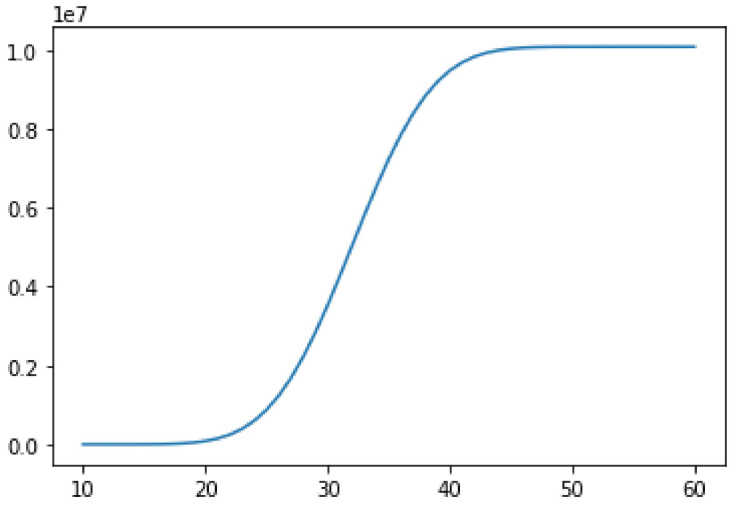
```
Number_of_ways=[0]*51

# 代入函数计算10-60的骰子和种数
for i in range(51):
    Number_of_ways[i]=Find_number_of_ways(10,i+10)
print(Number_of_ways)
```

```
[1, 10, 55, 220, 715, 2002, 4996, 11350, 23815, 46640, 85943, 149942, 248921, 394820, 600380, 877844, 1237313, 1684982, 2221551, 2841120, 3530835, 4271454, 5038848, 5806242, 6546861, 7236576, 7856145, 8392714, 8840383, 9199852, 9477316, 9682876, 9828775, 9927754, 9991753, 10031056, 10053881, 10066346, 10072700, 10075694, 10076981, 10077476, 10077641, 10077686, 10077695, 10077696, 10077696, 10077696, 10077696, 10077696, 10077696]
```

In [27]:
```
# Dice rolling
# plot Number_of_ways 10-60
import matplotlib.pyplot as plt
n=range(10,61)
plt.plot(n, Number_of_ways)
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x23131745880>]
```



In [24]:
```
# Dynamic programming
# 4.1
import random
N=int(input("input a number N: "))
def Random_integer(N):
    Rn=[0]*N
    for i in range(N):
        Rn[i]=round(random.random()*10)
    return Rn
Rn=Random_integer(N)
print(Rn)
```

```
input a number N: 5
[4, 3, 2, 5, 0]
```

In [37]:
```
# Dynamic programming
# 4.2
import math
def Sum_averages(n,S):
    # n为数组元素数
    # S为数组所有元素和
    Sn=0
    for i in range(n):
        Sn=Sn+S*math.factorial(n-1)/math.factorial(i)/math.factorial(n-1-i)/(n-i)
        # 对于有n个元素的数组，假设里面有一个元素为a，a可以是里面的任意一个元素
        # 考虑它的有k个元素的子数组
```

```
        # 那么包含a的有k个元素的子数组一共有C(n-1,k-1)（C代表组合数，n-1是下标，k-1是
        # 因为固定a之后，需要在剩下的n-1个元素中选取k-1个数，因此是C(n-1,k-1)
        # 所以在求有k个元素的子数组的平均值的和时，每个元素都计算了C(n-1,k-1)次
        # 得到k个元素的子数组的平均值的和为：Sum*C(n-1,k-1)/k，Sum代表有n个元素的数组
        # k从n到1，得到上述的循环
    return Sn
A=[1,2,3]
n=len(A)
S=sum(A)
Sum_averages(n,S)
```

Out[37]:
```
14.0
```

In [38]:
```
# Dynamic programming
# 4.3
N=100
Total_sum_averages=[0]*100
for i in range(N):
    K=[0]*(i+1)
    for j in range(i+1):
        K[j]=j+1
    S=sum(K)
    Total_sum_averages[i]=Sum_averages(i+1,S)
print(Total_sum_averages)
```

```
[1.0, 4.5, 14.0, 37.5, 93.0, 220.5, 508.0, 1147.5, 2555.0, 5626.5, 12282.0, 26617.5,
57337.0, 122872.5, 262136.0, 557047.5, 1179639.0, 2490358.5, 5242870.0, 11010037.5,
23068661.0, 48234484.5, 100663284.0, 209715187.5, 436207603.0, 905969650.5, 18790481
78.0, 3892314097.5, 8053063665.0, 16642998256.5, 34359738352.0, 70866960367.5, 14602
8888047.0, 300647710702.5, 618475290606.0, 1271310319597.5, 2611340115949.0, 5360119
185388.5, 10995116277740.0, 22539988369387.5, 46179488366571.0, 94557999988714.5, 19
3514046488554.0, 395824185999337.5, 809240558043113.0, 1653665488175080.5, 337769972
0527848.0, 6896136929411047.0, 1.4073748835532776e+16, 2.8710447624486896e+16, 5.854
6795155816424e+16, 1.193453901253181e+17, 2.431943798780068e+17, 4.953959590107545e+
17, 1.008806316530991e+18, 2.053641430080946e+18, 4.17934045419982e+18, 8.5027960964
755e+18, 1.7293822569102707e+19, 3.5164105890508825e+19, 7.148113328562452e+19, 1.45
26810958046274e+20, 2.951479051793528e+20, 5.995191823955603e+20, 1.2174851088648301
e+21, 2.4718637058708e+21, 5.017514388048999e+21, 1.018260272868767e+22, 2.06603533
62554694e+22, 4.191100253546809e+22, 8.500259669165365e+22, 1.723663766247419e+23,
3.494551197323536e+23, 7.08354972430447e+23, 1.4355994107923713e+24, 2.9089777534477
02e+24, 5.893513370621316e+24, 1.1938142468694465e+25, 2.4178516392292583e+25, 4.896
149569439247e+25, 9.913191720839966e+25, 2.006816860560285e+26, 4.061990753905152e+2
6, 8.22069557337948e+26, 1.6634819277897295e+27, 3.3656494818071265e+27, 6.808670216
069593e+27, 1.3772082937049851e+28, 2.785365088392107e+28, 5.632627178748481e+28, 1.
1389048361425502e+29, 2.3025684730708084e+29, 4.654654547713028e+29, 9.4083442985688
9e+29, 1.9014759003423438e+30, 3.8425658819418204e+30, 7.764359926397905e+30, 1.5687
17617782433e+31, 3.1691265005705744e+31, 6.40163553115256e+31]
```

In [158…
```
# Path counting
# 5.1
import numpy as np
import random
N=int(input("Create a matrix with N rows: "))
M=int(input("Create a matrix with M columns: "))
def random_matrix(N,M):
    matrix=np.empty((N,M))
    # 建立一个N行M列的空数组
    for i in range(N):
        for j in range(M):
            matrix[i,j]=random.choice([0,1])
    matrix[0,0]=1
    matrix[N-1,M-1]=1
    # 先用0和1把数组随机填满，再令左上角和右下角的元素为1
    return matrix
```

```
matrix=random_matrix(N,M)
print(matrix)
```

```
Create a matrix with N rows: 4
Create a matrix with M columns: 5
[[1. 0. 1. 1. 0.]
 [1. 1. 0. 1. 1.]
 [0. 0. 0. 1. 1.]
 [0. 0. 1. 1. 1.]]
```

In [156…

```
# Path counting
# 5.2
def Count_path(A,N,M):
    # A为给定的矩阵
    # N为A的行数，M为A的列数
    # 参考了这个网站的思路和代码，代码是看了之后自己写的
    # https://www.geeksforgeeks.org/count-number-ways-reach-destination-maze/
    for i in range(N):
        for j in range(M):
            A[i][j]=A[i][j]-1
    for i in range(N):
        if A[i][0]==0:
            A[i][0]=1
        else:
            break
    for j in range(1,M):
        if A[0][j]==0:
            A[0][j]=1
        else:
            break
    for i in range(1,N):
        for j in range(1,M):
            if A[i][j]==-1:
                continue
            if A[i-1][j]>0:
                A[i][j]=A[i][j]+A[i-1][j]
            if A[i][j-1]>0:
                A[i][j]=A[i][j]+A[i][j-1]
    return A[N-1][M-1]
    # 主要思路就是到达每个元素的路径数等于到达它的左边和上边的路径数之和

P=[[1,0,1,0],
   [1,1,1,1],
  [1,1,1,0],
  [0,1,1,1],
  [1,0,1,1]]
N=len(P)
M=len(P[0])
print(Count_path(P,N,M))
```

```
10
```

In [157…

```
# Path counting
# 5.3

# 定义一个1000个元素的空数组
Count=np.empty(1000)

# 指定行列数
N=10
M=8

# 代入函数
for i in range(1000):
```

```
    matrix=random_matrix(N,M)
    Count[i]=Count_path(matrix,N,M)
avera=np.mean(Count)
print(avera)
```

```
0.437
```

In [ ]: