Use case

In [33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

It is a function to Create an array of the given shape and populate it with random samples from a uniform distribution

take paramters as dimensions of returen array

In [9]:
```
np.random.rand(5,2)
```

Out[9]:
```
array([[0.36200677, 0.74081302],
       [0.6664724 , 0.56885827],
       [0.69842479, 0.67093925],
       [0.45554915, 0.00605685],
       [0.86534559, 0.58610249]])
```

produce integer numbers, take high and low values as a range and the size of returen array

In [16]:
```
np.random.randint(0,4 , size = 10)
```

Out[16]:
```
array([1, 0, 2, 3, 1, 1, 0, 2, 0, 2])
```

In [30]:
```
coins= np.random.randint(0,2)
if coins:
    print("head")
else:
    print("tail")
```

```
head
```

Determine your next move

In the Empire State Building bet, your next move depends on the number of eyes you throw with the dice.

In [31]:
```
# Starting step
step = 50

# Roll the dice
dice = np.random.randint(1,7)

# Finish the control construct
if dice <= 2 :
    step = step - 1
elif dice == 3 or dice == 4 :
    step = step + 1
else:
    step = step + np.random.randint(1,7)

print(dice)
print(step)
```

```
3
51
```

I have already written Python code that determines the next step based on the previous step. Now it's time to put this code inside a for loop so that we can simulate a random walk.

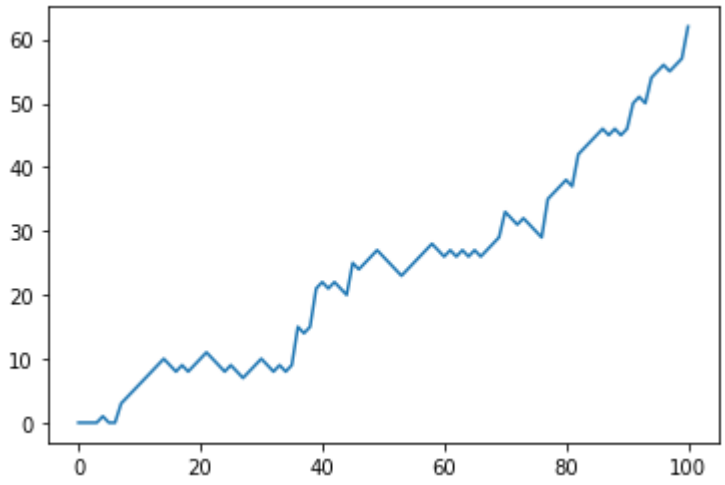
I calculate the location in the Empire State Building after 100 dice throws. However, there's something we haven't thought about - you can't go below

In [32]:
```
# Initialize random_walk
random_walk = [0]

for x in range(100) :
    step = random_walk[-1]
    dice = np.random.randint(1,7)

    if dice <= 2:
        # Replace below: use max to make sure step can't go below 0
        step = max(0, step - 1)
    elif dice <= 5:
        step = step + 1
    else:
        step = step + np.random.randint(1,7)

    random_walk.append(step)

print(random_walk)
```

```
[0, 0, 0, 0, 1, 0, 0, 3, 4, 5, 6, 7, 8, 9, 10, 9, 8, 9, 8, 9, 10, 11, 10, 9, 8,
9, 8, 7, 8, 9, 10, 9, 8, 9, 8, 9, 15, 14, 15, 21, 22, 21, 22, 21, 20, 25, 24, 2
5, 26, 27, 26, 25, 24, 23, 24, 25, 26, 27, 28, 27, 26, 27, 26, 27, 26, 27, 26,
27, 28, 29, 33, 32, 31, 32, 31, 30, 29, 35, 36, 37, 38, 37, 42, 43, 44, 45, 46,
45, 46, 45, 46, 50, 51, 50, 54, 55, 56, 55, 56, 57, 62]
```

Visualize the walk

In [34]:
```
# Plot random_walk
plt.plot(random_walk)

# Show the plot
plt.show()
```



In [ ]:

Simulate multiple walks

A single random walk is one thing, but that doesn't tell you if you have a good chance at winning the bet.

To get an idea about how big your chances are of reaching 60 steps, you can repeatedly simulate the random walk and collect the results.

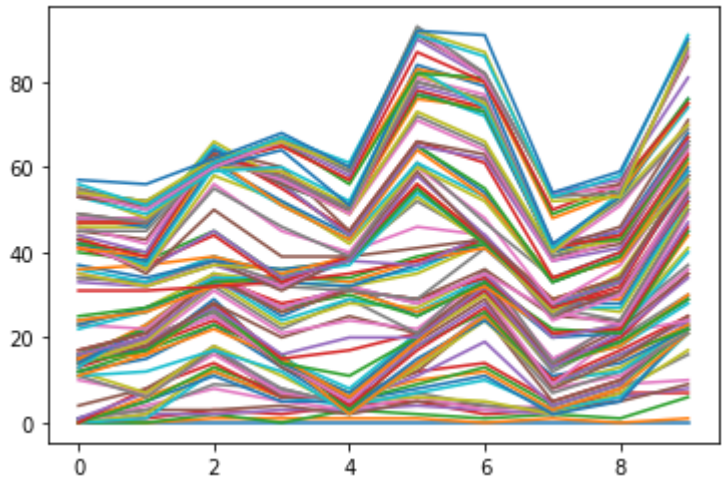
In [35]:
```
# Initialize all_walks (don't change this line)
all_walks = []
# Simulate random walk 10 times
for i in range(10) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)

        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)
        random_walk.append(step)

    # Append random_walk to all_walks
    all_walks.append(random_walk)

# Print all_walks
print(all_walks)
```

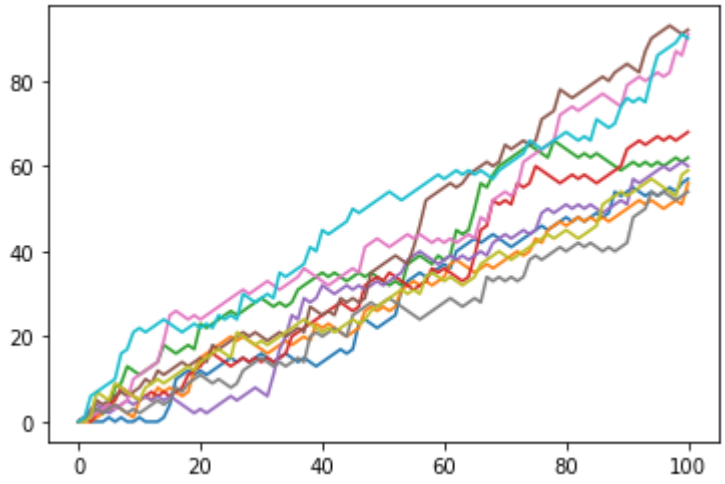
```
[[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 4, 10, 11, 12, 11, 12, 11, 12,
13, 14, 15, 14, 15, 14, 15, 16, 15, 14, 15, 16, 15, 14, 15, 14, 13, 14, 15, 1
6, 17, 16, 17, 23, 24, 23, 22, 23, 24, 25, 31, 33, 34, 35, 34, 35, 36, 37, 3
6, 40, 41, 42, 43, 42, 43, 44, 43, 42, 41, 42, 43, 44, 45, 46, 45, 46, 47, 4
8, 47, 48, 47, 48, 49, 48, 49, 54, 53, 54, 55, 54, 53, 54, 53, 54, 55, 54, 5
6, 57], [0, 0, 0, 1, 2, 3, 4, 3, 2, 1, 5, 6, 5, 8, 7, 8, 7, 6, 7, 12, 15, 16,
17, 18, 19, 20, 19, 20, 19, 18, 17, 16, 17, 18, 19, 18, 19, 20, 19, 23, 22, 2
3, 22, 21, 20, 21, 22, 26, 27, 26, 27, 26, 27, 31, 32, 33, 32, 33, 32, 33, 3
4, 37, 38, 37, 36, 35, 36, 37, 36, 37, 38, 39, 40, 39, 40, 43, 42, 45, 46, 4
7, 46, 47, 48, 47, 48, 47, 48, 47, 48, 49, 50, 51, 52, 51, 52, 51, 50, 51, 5
2, 51, 56], [0, 1, 2, 3, 2, 3, 8, 9, 13, 12, 11, 12, 13, 14, 18, 17, 16, 17,
18, 17, 23, 22, 23, 24, 25, 26, 25, 26, 27, 28, 29, 28, 27, 28, 27, 28, 31, 3
2, 33, 34, 35, 34, 35, 34, 33, 34, 35, 34, 35, 34, 33, 32, 33, 32, 37, 38, 3
9, 38, 37, 39, 38, 39, 45, 44, 45, 50, 56, 55, 58, 60, 61, 62, 63, 64, 65, 6
4, 63, 62, 66, 65, 64, 63, 62, 63, 62, 63, 62, 61, 60, 59, 60, 61, 60, 61, 6
0, 61, 60, 61, 62, 61, 62], [0, 1, 0, 2, 3, 4, 5, 8, 7, 6, 5, 6, 7, 6, 7, 6,
7, 8, 11, 12, 13, 15, 16, 15, 14, 13, 14, 15, 14, 15, 14, 15, 14, 15, 16, 20,
21, 22, 23, 24, 25, 26, 27, 28, 27, 26, 27, 32, 33, 34, 33, 35, 34, 33, 32, 3
1, 32, 33, 36, 35, 36, 35, 34, 33, 34, 39, 45, 46, 52, 51, 52, 51, 56, 55, 5
6, 60, 59, 58, 57, 56, 57, 58, 57, 58, 57, 56, 57, 58, 59, 60, 64, 65, 66, 6
5, 66, 67, 66, 67, 66, 67, 68], [0, 1, 3, 4, 3, 2, 3, 4, 5, 4, 5, 6, 5, 6, 5,
6, 5, 4, 3, 2, 3, 2, 3, 4, 5, 6, 5, 6, 7, 8, 7, 6, 11, 17, 20, 25, 24, 29, 2
8, 29, 32, 31, 30, 31, 32, 31, 32, 31, 32, 33, 32, 33, 34, 35, 38, 39, 40, 3
9, 38, 37, 38, 39, 38, 39, 38, 39, 40, 39, 42, 43, 44, 43, 44, 45, 44, 45, 4
9, 50, 49, 50, 51, 50, 51, 50, 51, 50, 49, 50, 51, 52, 51, 57, 56, 57, 58, 5
9, 60, 59, 60, 61, 60], [0, 1, 2, 5, 4, 5, 6, 7, 6, 7, 8, 10, 9, 12, 11, 12,
13, 14, 13, 14, 15, 14, 18, 17, 18, 19, 20, 21, 20, 21, 20, 19, 20, 21, 20, 2
1, 22, 21, 27, 26, 25, 26, 25, 29, 28, 29, 28, 29, 35, 36, 37, 38, 39, 38, 3
7, 41, 46, 52, 53, 54, 55, 56, 55, 56, 58, 59, 60, 61, 60, 61, 65, 64, 65, 6
6, 65, 66, 71, 72, 73, 78, 77, 76, 77, 78, 79, 80, 81, 80, 82, 83, 84, 83, 8
2, 87, 90, 91, 92, 93, 92, 91, 92], [0, 0, 1, 2, 3, 4, 3, 4, 5, 10, 11, 12, 1
3, 14, 19, 25, 26, 25, 24, 25, 24, 25, 26, 27, 28, 29, 30, 31, 30, 31, 32, 3
3, 32, 31, 32, 33, 34, 36, 35, 34, 33, 32, 33, 34, 35, 36, 35, 41, 42, 43, 4
```

```
2, 41, 42, 43, 44, 43, 44, 43, 42, 43, 42, 43, 42, 43, 44, 43, 48, 47, 52, 5
3, 54, 53, 55, 61, 62, 63, 64, 65, 66, 72, 73, 74, 73, 74, 75, 76, 77, 76, 7
5, 74, 79, 80, 81, 80, 81, 82, 81, 82, 87, 86, 91], [0, 1, 2, 3, 2, 3, 4, 3,
2, 3, 2, 3, 4, 5, 4, 5, 8, 7, 9, 10, 11, 10, 9, 10, 9, 8, 9, 12, 13, 14, 15,
14, 13, 14, 13, 14, 15, 14, 20, 21, 20, 21, 22, 21, 20, 25, 26, 27, 28, 27, 2
8, 29, 28, 27, 26, 25, 24, 25, 26, 27, 28, 29, 28, 27, 28, 29, 28, 34, 33, 3
4, 33, 34, 33, 34, 38, 39, 38, 39, 40, 41, 40, 41, 42, 41, 42, 41, 40, 41, 4
0, 41, 42, 48, 49, 50, 54, 53, 54, 53, 52, 53, 54], [0, 0, 1, 7, 6, 5, 9, 8,
7, 6, 5, 8, 9, 10, 9, 10, 11, 12, 13, 12, 14, 15, 16, 17, 16, 15, 21, 20, 19,
18, 19, 18, 19, 20, 21, 22, 23, 24, 23, 22, 21, 22, 21, 22, 23, 24, 23, 24, 2
6, 27, 28, 29, 30, 31, 30, 31, 30, 34, 35, 34, 33, 34, 33, 32, 33, 34, 37, 3
8, 39, 40, 39, 38, 39, 40, 41, 42, 43, 44, 43, 44, 45, 44, 43, 44, 45, 46, 5
1, 52, 53, 54, 53, 54, 55, 56, 57, 56, 55, 54, 53, 58, 59], [0, 1, 6, 7, 8,
9, 10, 16, 17, 21, 22, 21, 22, 23, 24, 23, 22, 21, 22, 23, 22, 23, 22, 25, 2
4, 25, 24, 30, 29, 28, 29, 30, 29, 35, 34, 35, 36, 37, 41, 40, 45, 44, 45, 4
6, 47, 50, 49, 50, 51, 52, 53, 54, 53, 52, 53, 54, 55, 56, 57, 58, 57, 58, 5
9, 58, 59, 58, 59, 58, 57, 59, 60, 61, 62, 63, 66, 65, 64, 65, 66, 67, 68, 6
7, 66, 67, 66, 71, 70, 69, 70, 74, 76, 75, 76, 75, 81, 86, 87, 88, 89, 91, 9
0]]
```

Visualize all walks

In [37]:
```
# Convert all_walks to NumPy array: np_aw
np_aw = np.array(all_walks)
# Plot np_aw and show
plt.plot(np_aw)
plt.show()
# Clear the figure
plt.clf()
# Transpose np_aw: np_aw_t
np_aw_t = np.transpose(np_aw)
# Plot np_aw_t and show
plt.plot(np_aw_t)
plt.show()
```





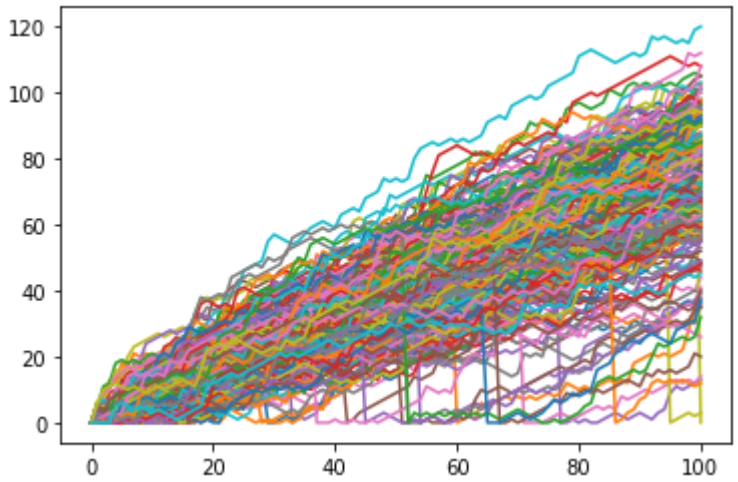
Implement clumsiness

changing the number of times the random walk should be simulated is super-easy. You simply update the range() function in the top-level for loop.

There's still something we forgot! You're a bit clumsy and you have a 0.1% chance of falling down. That calls for another random number generation. Basically, you can generate a random float between 0 and 1. If this value is less than or equal to 0.001, you should reset

step to 0.

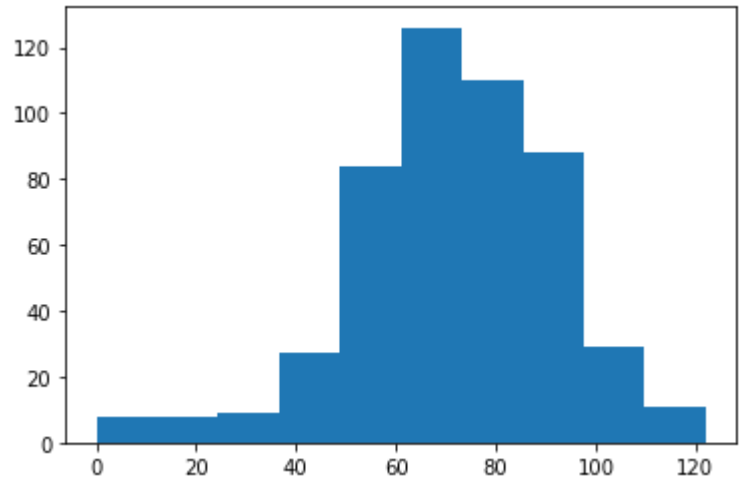
In [38]:
```
# Simulate random walk 250 times
all_walks = []
for i in range(250) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)

        # Implement clumsiness
        if np.random.rand() <= 0.001 :
            step = 0

        random_walk.append(step)
    all_walks.append(random_walk)

# Create and plot np_aw_t
np_aw_t = np.transpose(np.array(all_walks))
plt.plot(np_aw_t)
plt.show()
```



Plot the distribution

Basically, you want to know about the end points of all the random walks you've simulated. These end points have a certain distribution that you can visualize with a histogram.

In [41]:
```
# Simulate random walk 500 times
all_walks = []
for i in range(500) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)
        if np.random.rand() <= 0.001 :
            step = 0
        random_walk.append(step)
    all_walks.append(random_walk)

# Create and plot np_aw_t
np_aw_t = np.transpose(np.array(all_walks))

# Select last row from np_aw_t: ends
ends = np_aw_t[-1,:]

# Plot histogram of ends, display plot
plt.hist(ends)
plt.show()
```



Calculate the odds

The histogram contains 500 integers. Each integer represents the end point of a random walk. To calculate the chance that this end point is greater than or equal to 60, you can count the number of integers in ends that are greater than or equal to 60 and divide that number by 500, the total number of simulations.

In [42]:
```
np.mean(ends >60)
```

Out[42]:
```
0.728
```

In [ ]:
```

```