**What are the chances of winning a bet (Dice).**

The game involves walking up the empire state building to the top and we play this with a friend. We can throw the dice 100 times. If it is a 1 or 2 then we go down one step. If it is a 3, 4 or 5 then we go up one step. If you throw a six, we get to throw again and walk up the resulting number of steps. We can not go lower than ground floor – 0. There is also a 0.1% chance of falling down the stairs when you make a move, which means we start again from 0. The bet is that I will reach 60 steps high. We want to work out the odds of winning this bet. We are going to simulate the process thousands of times and determine the fraction of times 60 steps is achieved. We need to randomly generate our die numbers. In numpy package we have a function called rand and starts from a random seed. If we set the seed at 100 and call rand twice, we get two random numbers. Now set the seed back to 100 and call rand twice, we get the exact same two random numbers (pseudo-random) – this enables reproducibility.

First, generate random floats between 0 and 1 using NumPy.

import numpy as np

np.random.seed(100)

print(np.random.rand())

Output = 0.5434049417909654

Then we need to simulate rolling a dice using randint(). Repeat twice

import numpy as np

np.random.seed(100)

# Use randint() to simulate a dice

dice = np.random.randint(1, 7)

print(dice)

# Use randint() again

dice = np.random.randint(1, 7)

print(dice)

Output = 1 & 1

In the Empire State Building bet, our next move depends on the number after throwing the dice. We can do this by utilising an IF – ELSE construct. Lets start at step 40 and throw dice once to see outcome and which step we end up on.

# Starting step

step = 40

# Roll the dice

dice = np.random.randint(1, 7)

# Finish the control construct

if dice <= 2 :

    step = step - 1

elif 3 <=dice <= 5 :

    step = step + 1

else :

    step = step + np.random.randint(1,7)

# Print out dice and step

print(dice, step)

Output = Dice = 4 and Step = 41

Now we want to see what the outcome of a random walk over 100 steps would give as an outcome. We will use a FOR loop to build up pattern of the walk by appending the steps to the walk. Also use an IF-ELSE statement to update each step.

# Initialize random\_walk

random\_walk = [0]

# Complete the \_\_\_

for x in range(100) :

    # Set step: last element in random\_walk

    step = random\_walk[-1]

    # Roll the dice

    dice = np.random.randint(1,7)

    # Determine next step

    if dice <= 2:

        step = step - 1

    elif dice <= 5:

        step = step + 1

    else:

        step = step + np.random.randint(1,7)

    # append next\_step to random\_walk

    random\_walk.append(step)

# Print random\_walk

print(random\_walk)

In our Empire State Building game we cannot go below 0 and we solved this by using max(). If you pass max() two arguments, the biggest one gets returned.

# 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)

Now we want to visualise this random walk using MatPlotLib and build a line plot.

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Plot random\_walk

plt.plot(random\_walk)

# Adding a title to the plot

plt.title("The random walk of one set of 100 steps")

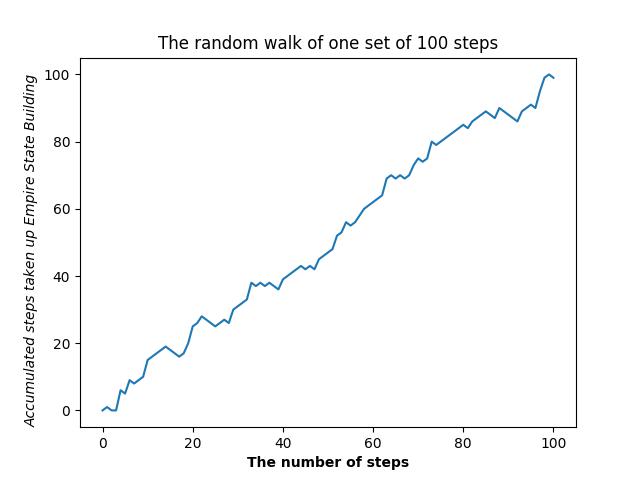
# Adding and styling axis labels

plt.xlabel(‘The number of steps’, fontweight='bold')

plt.ylabel(Accumulated steps taken up Empire State Building', style='italic', loc='bottom')

# Show the plot

plt.show()



We now need to consider distribution to be able to answer the question “What is the chance that we will reach 60 steps high? We need to run the simulation thousands of times and record the final step value. This creates a distribution of final steps from which we can calculate chances.

Now let us simulate multiple random walks 10 times by creating a variable all\_walks to store the outcomes.

# Initialise an empty list to store all walks

all\_walks = []

# Simulate random walk 10 times

for i in range(10) :

    # Code from before

    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)

We have created all\_walks as a list of lists, each sub-list represents a single random walk. We need to convert these list of lists into a numpy array and make interesting plots. You can see from the first graph that the results shows the result at each step of the 10 walks – therefore we need to transpose the values to correcly plot 10 walks over 100 steps.

# Convert all\_walks to NumPy array: np\_aw

np\_aw = np.array(all\_walks)

# Plot np\_aw and show

plt.plot(np\_aw)

plt.ylabel("Total number of steps after 100 throws")

plt.xlabel("The resultant of 10 walks")

plt.title("The 10 random walks for 100 steps")

plt.show()

# Clear the figure

plt.clf()

# Transpose np\_aw: np\_aw\_t

np\_aw\_t = np.transpose(np\_aw)

plt.xlabel("Total number of steps taken")

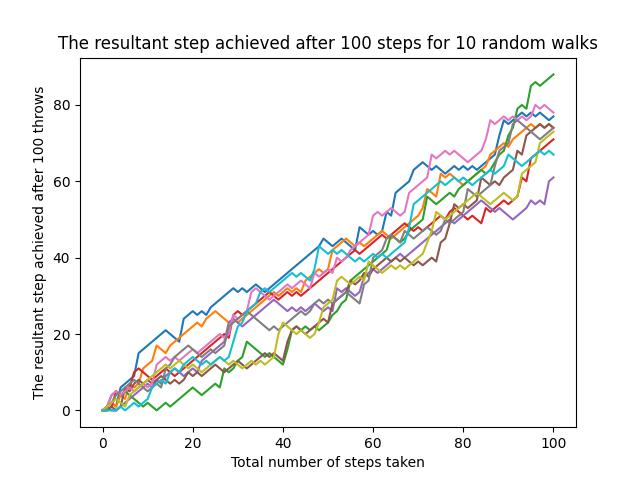
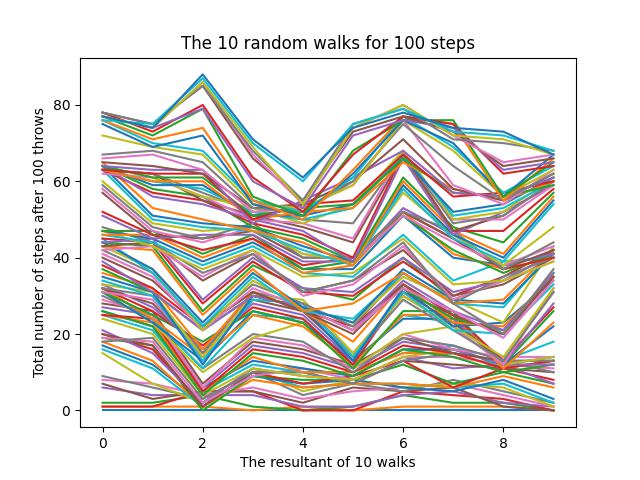
plt.ylabel("The resultant step achieved after 100 throws")

plt.title("The resultant step achieved after 100 throws for 10 random walks")

# Plot np\_aw\_t and show

plt.plot(np\_aw\_t)

plt.show()



Then we wanted to add a jeopardy and implement clumsiness and we have a 0.5% chance of falling down. To do this we generated another random float between 0 and 1. If this value ever is less than or equal to 0.005 the overall accumulated score is reset to 0.

# clear the plot so it doesn't get cluttered if you run this many times

plt.clf()

# Simulate random walk 20 times

all\_walks = []

for i in range(20) :

    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() <= 0.005 :

            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))

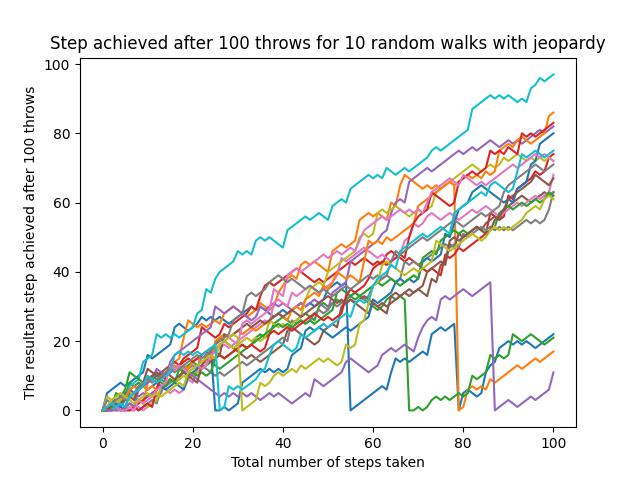
plt.plot(np\_aw\_t)

plt.xlabel("Total number of steps taken")

plt.ylabel("The resultant step achieved after 100 throws")

plt.title("Step achieved after 100 throws for 10 random walks with jeopardy")

plt.show()



We still have to solve the million-dollar question – “What are the odds that we will reach 60 steps high on the Empire State Building? To do this we need to know the end points of all the random walks simulated and then visualise them as a histogram. We need to have lots of repeat simulations.

# Simulate random walk 500 times

all\_walks = []

for i in range(1000) :

    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() <= 0.005 :

            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)

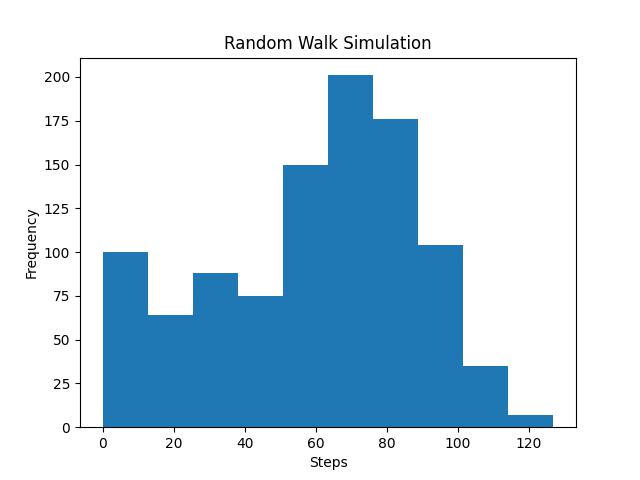
#Add axis titles and a main title

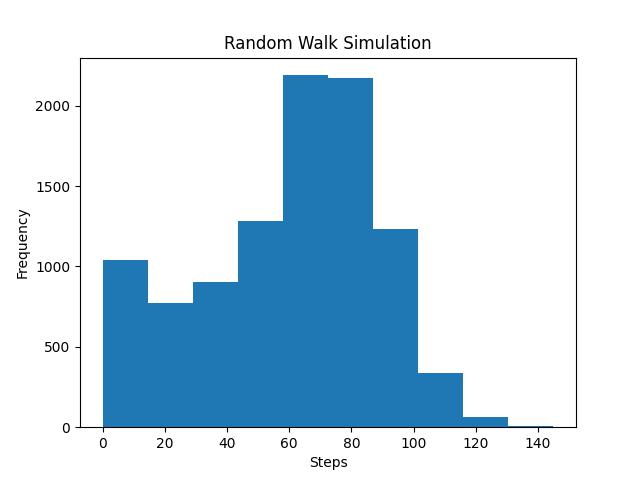
plt.xlabel("Steps")

plt.ylabel("Frequency")

plt.title("Random Walk Simulation")

plt.show()





Now we need to calculate the odds. The Histogram was created from a NumPy array of ends that contains 10000 integers. These integers represent the end point of each random walk. To calculate the chance the odds that end point is greater than or equal to 60 – we counted the number of integers in ends that were greater than or equal to 60 and divided that number by 10000 (total number of simulations).

probability = np.count\_nonzero(ends >= 60) / len(ends)

percentage = probability \* 100

print(f"The estimated chance of reaching at least 60 steps high (with 5% jeopardy) is approximately{percentage: }%")

The estimated chance of reaching at least 60 steps high (with 5% jeopardy) is approximately 57.13%.

The estimated chance of reaching at least 60 steps high (with 1% jeopardy) is approximately 77.63%.