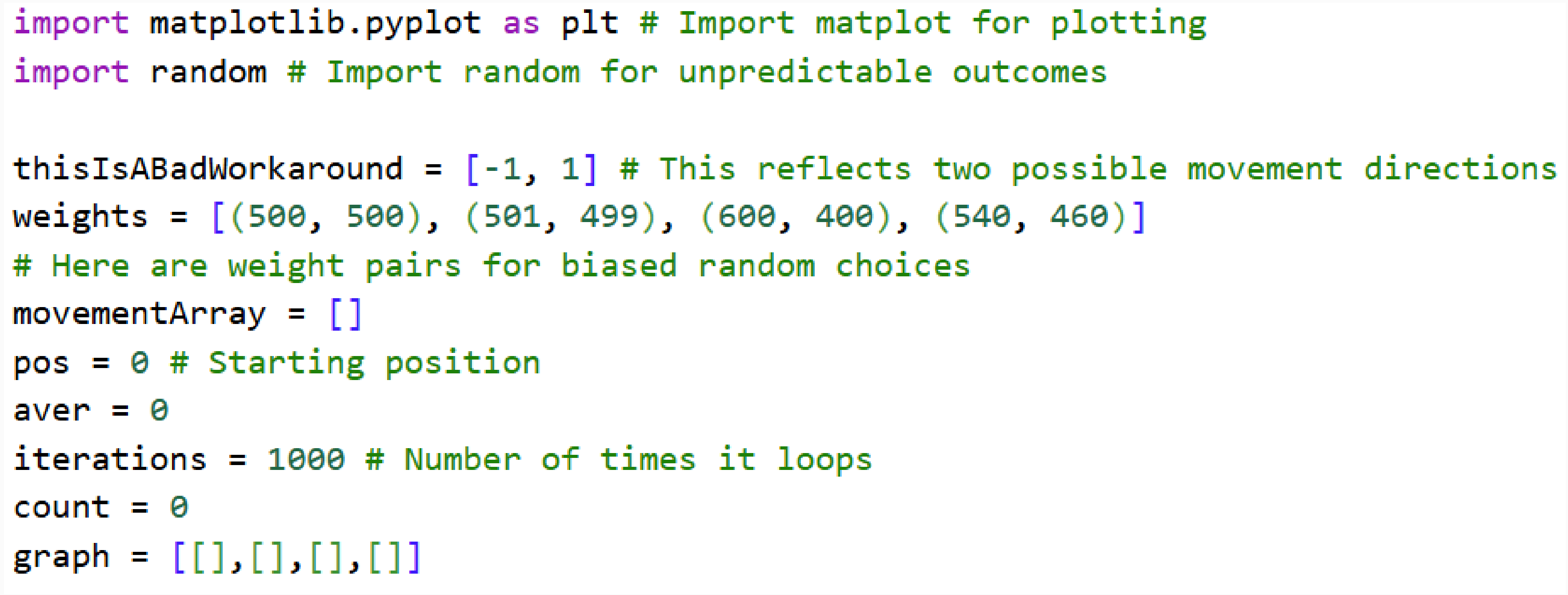
Random Walk and Gambler's Ruin

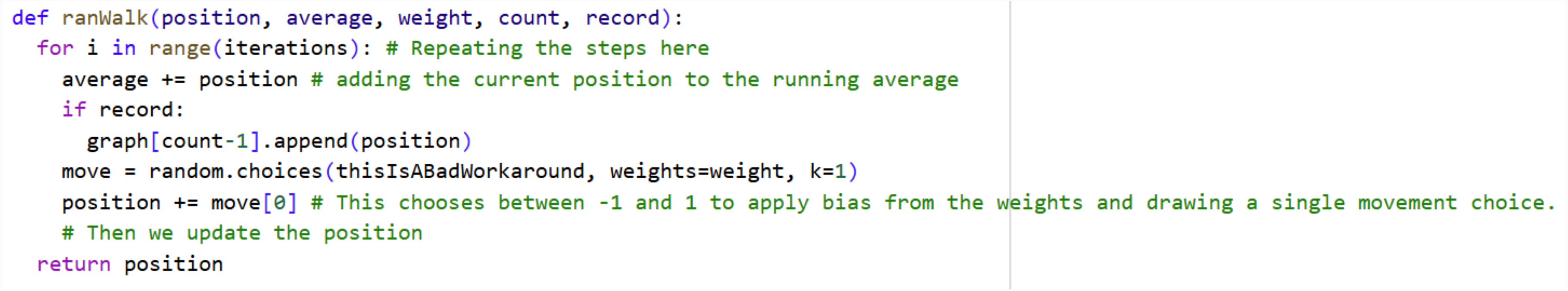
By
Evan Mayo
Henry Ahmed
Carter Cheaney
Wyatt Keller

Goal and Terms

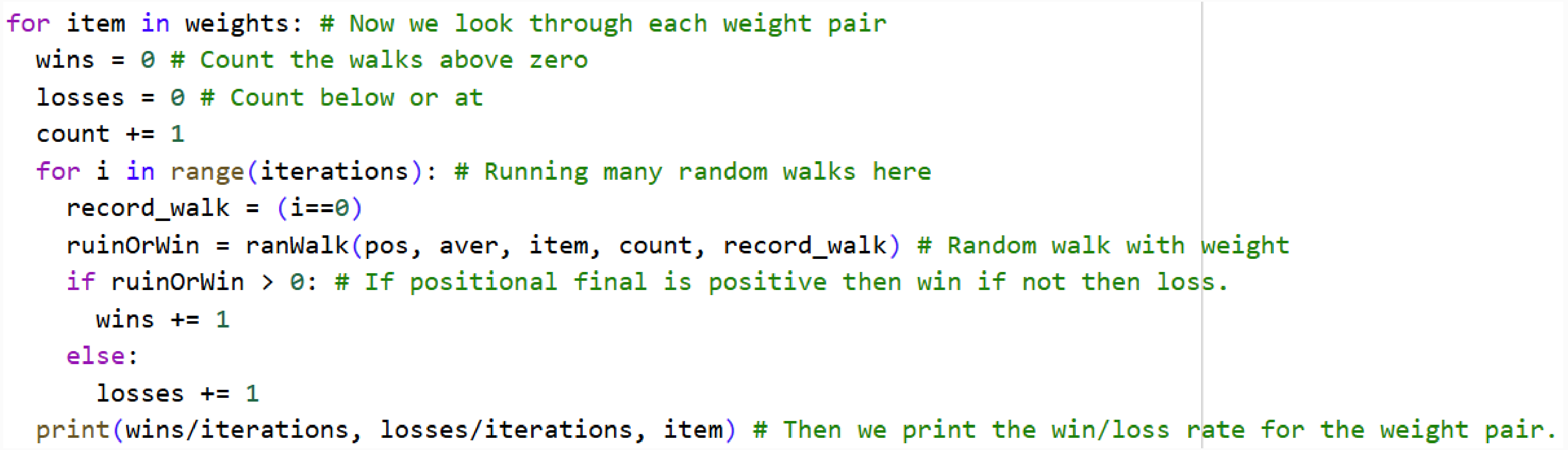
- Model a Gambler's Fortune as a Random Walk
- Random Walk: random steps forwards or backwards, with weights for each way, represented as a tuple
- Biased Walk: a random walk with weights not (500, 500)
- Simulation: iterating through n different walks, resulting in a win or loss
- Win/Loss: Ending with more/less than $0 at the end of the game



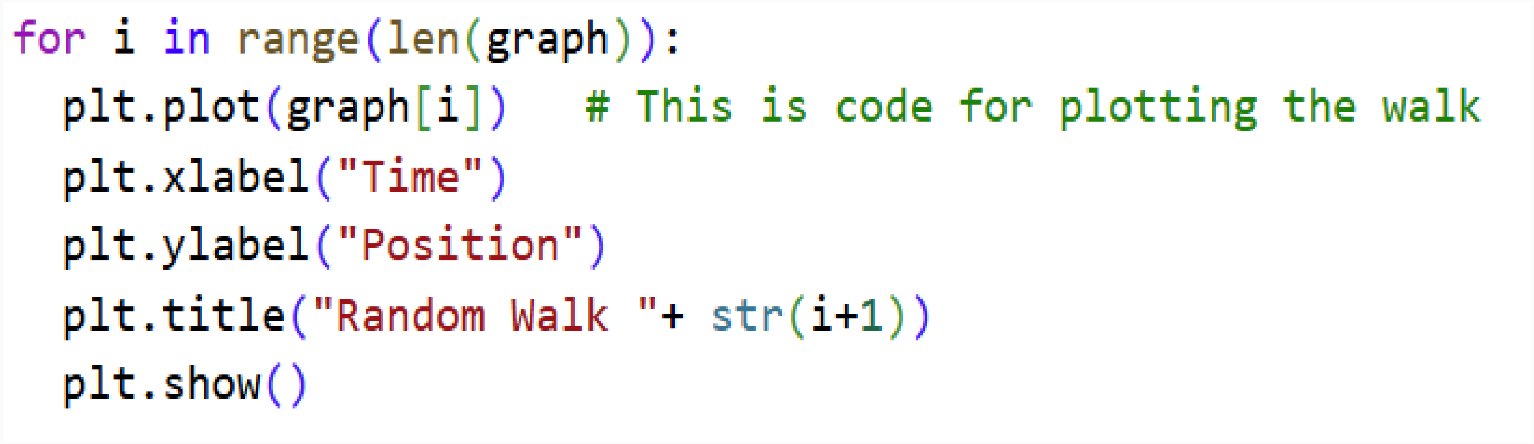
```python
import matplotlib.pyplot as plt # Import matplot for plotting
import random # Import random for unpredictable outcomes

thisIsABadWorkaround = [-1, 1] # This reflects two possible movement directions
weights = [(500, 500), (501, 499), (600, 400), (540, 460)]
# Here are weight pairs for biased random choices
movementArray = []
pos = 0 # Starting position
aver = 0
iterations = 1000 # Number of times it loops
count = 0
graph = [[],[],[],[]]
```



```python
def ranWalk(position, average, weight, count, record):
  for i in range(iterations): # Repeating the steps here
    average += position # adding the current position to the running average
    if record:
      graph[count-1].append(position)
    move = random.choices(thisIsABadWorkaround, weights=weight, k=1)
    position += move[0] # This chooses between -1 and 1 to apply bias from the weights and drawing a single movement choice.
    # Then we update the position
  return position
```



```python
for item in weights: # Now we look through each weight pair
  wins = 0 # Count the walks above zero
  losses = 0 # Count below or at
  count += 1
  for i in range(iterations): # Running many random walks here
    record_walk = (i==0)
    ruinOrWin = ranWalk(pos, aver, item, count, record_walk) # Random walk with weight
    if ruinOrWin > 0: # If positional final is positive then win if not then loss.
      wins += 1
    else:
      losses += 1
  print(wins/iterations, losses/iterations, item) # Then we print the win/loss rate for the weight pair.
```

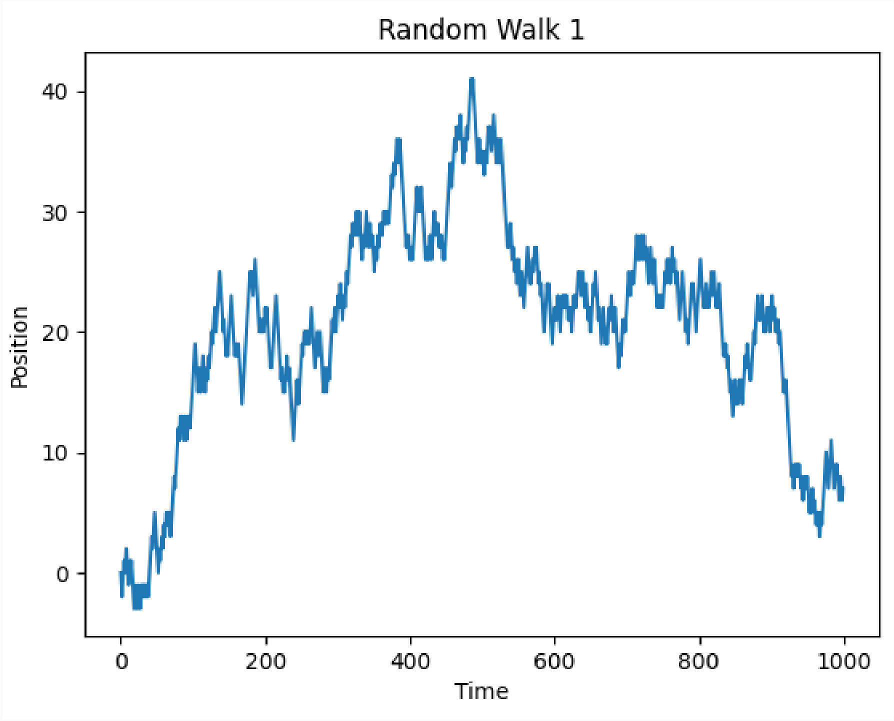


```python
for i in range(len(graph)):
  plt.plot(graph[i])   # This is code for plotting the walk
  plt.xlabel("Time")
  plt.ylabel("Position")
  plt.title("Random Walk "+ str(i+1))
  plt.show()
```

Random Walk 1

The simulation on the right is standard, with weights (500, 500). This would translate to a 0.5 chance for both a step forward and a step back.

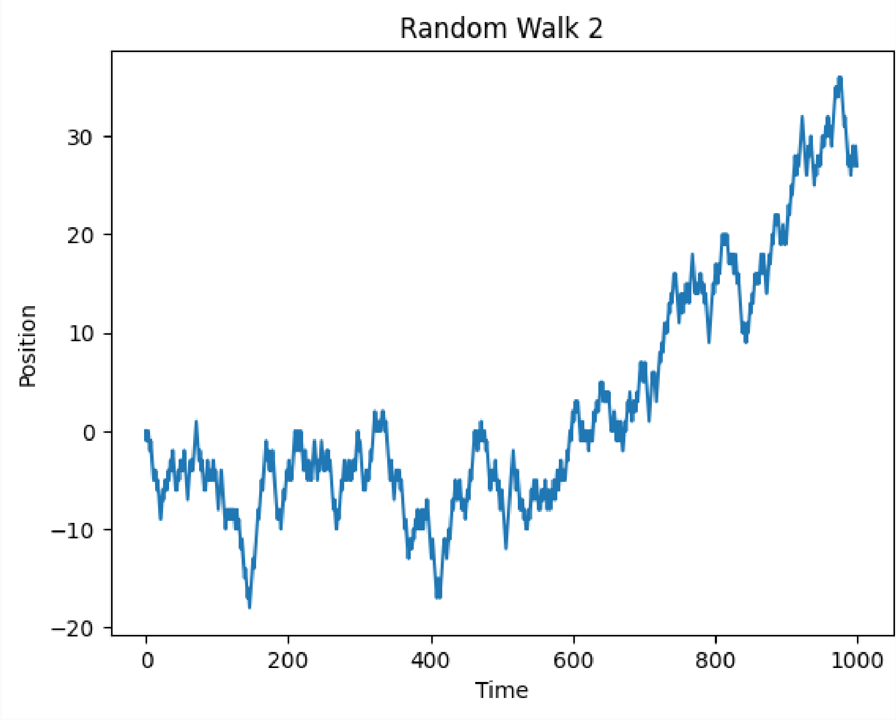
For the 1000 simulations we ran, the probability of a win/loss was (0.475, 0.525). It is important to note that the graph depicted is of one simulation, since this is a equal weighted walk, the results vary greatly.



Random Walk 2

The simulation on the right is biased, with weights (501, 499). This would translate to a 0.501 chance for a step back and 0.499 for a step forward.

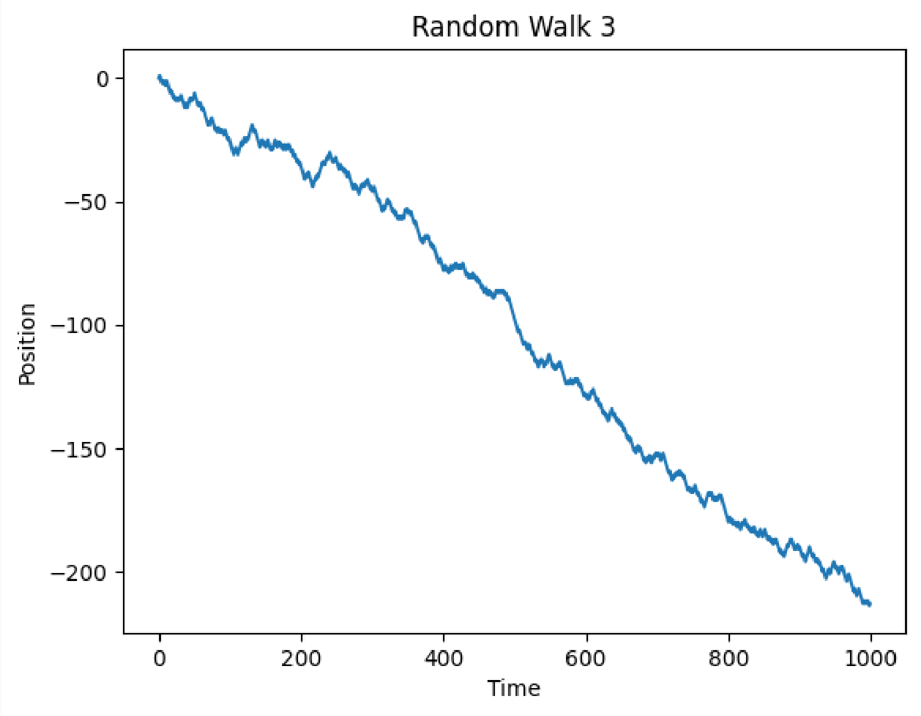
For the 1000 simulations we ran, the probability of a win/loss was (0.546, 0.454). Though since probabilistically the difference is not that great, the results still vary greatly.



Random Walk 3

The simulation on the right is biased, with weights (600, 400). This would translate to a 0.6 chance for a step back and 0.4 for a step forward.

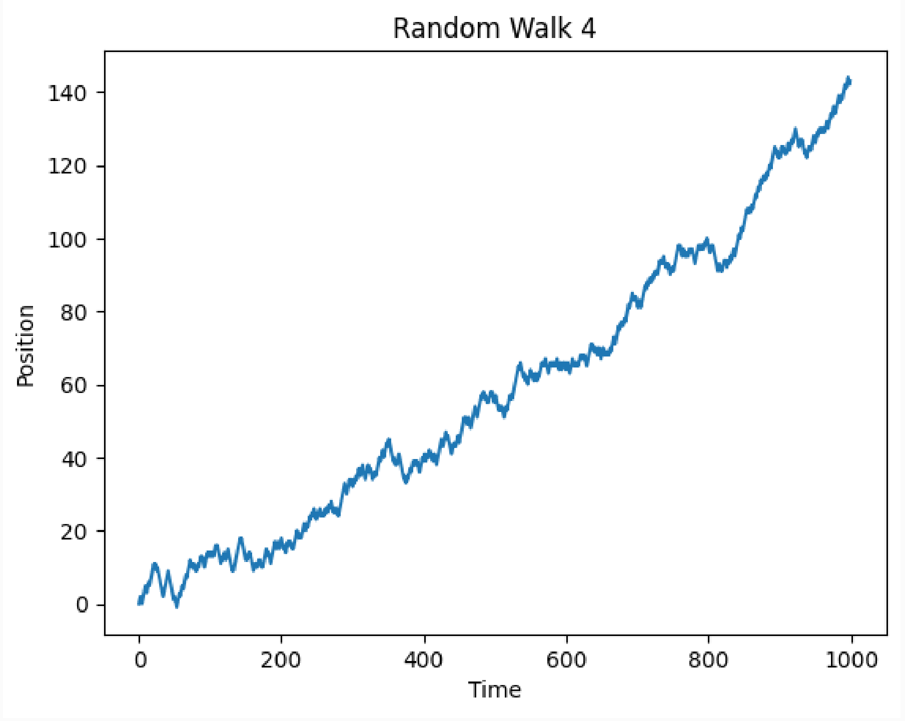
For the 1000 simulations we ran, the probability of a win/loss was (0, 1). Due to the heavy penalty, it was almost guaranteed with many steps that the gambler would lose.



Random Walk 4

The simulation on the right is biased, with weights (420, 580). This would translate to a 0.42 chance for a step back and 0.58 for a step forward.

For the 1000 simulations we ran, the probability of a win/loss was (1, 0). Due to the heavy positive bias, it was almost guaranteed with many steps that the gambler would win.



Thank You!