**SECURIN ASSESSMENT**

**SARVASETTY SAI SATHVIKA**

**40731087**

**CSE-AI**

[**sathvikasathvi322@gmail.com**](mailto:sathvikasathvi322@gmail.com)

**PART-A**

**Problem statement:** The Doomed Dice Challenge

1. How many total combinations are possible? Show the math along with the code!

**Logic**

Total Combinations: The product of the number of faces on each die yields the total number of combinations. As there are six faces on each dice:

No.of faces on Die A × No.of faces on Die B = Total Combinations

Total Combinations: 6 × 6 = 36

**Code**

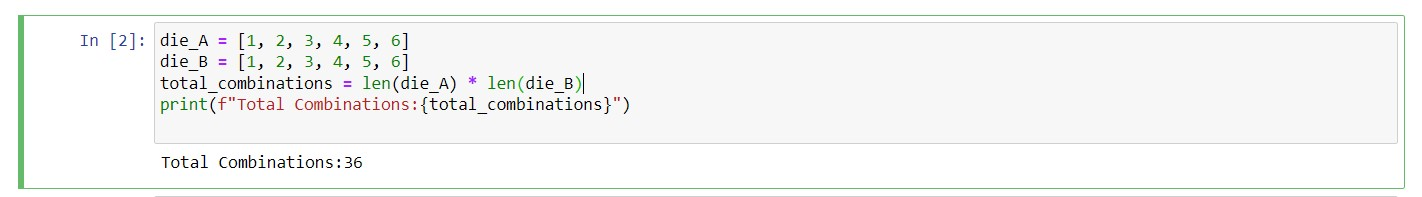
die\_A = [1, 2, 3, 4, 5, 6]

die\_B = [1, 2, 3, 4, 5, 6]

total\_combinations = len(die\_A) \* len(die\_B)

print(f"Total Combinations:{total\_combinations}")

**code and output screenshot**



2.Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code!

**Logic**

A 6x6 matrix can be used to illustrate the distribution of all possible outcomes when rolling both Die A and Die B simultaneously. The total that is achieved by rolling Die A with face i and Die B with face j is represented by each element (i, j) in the matrix. We cycle through every face on Die A (i) and cycle through every face on Die B (j) for every face. When rolling both Die A and Die B together, the combination is represented by the sum of the faces (i + j). We create a 6x6 matrix called distribution\_matrix, in which the sum of the results of rolling Die A with face i and Die B with face j is represented by each entry (i, j). The distribution of all potential combinations is then shown by printing the matrix.

**Code**

die\_A = [1, 2, 3, 4, 5, 6]

die\_B = [1, 2, 3, 4, 5, 6]

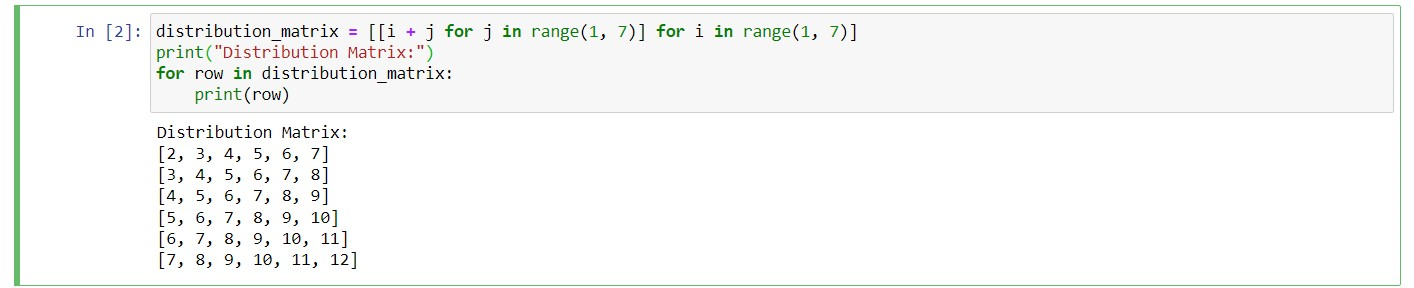
distribution\_matrix = [[i + j for j in die\_B] for i in die\_A]

print("Distribution Matrix:")

for row in distribution\_matrix:

print(row)

**code and output screenshot**



3. Calculate the Probability of all Possible Sums occurring among the number of combinations from (2).

**Logic**

By dividing the number of possibilities that lead to each sum by the total number of combinations, the probability of each sum is computed.

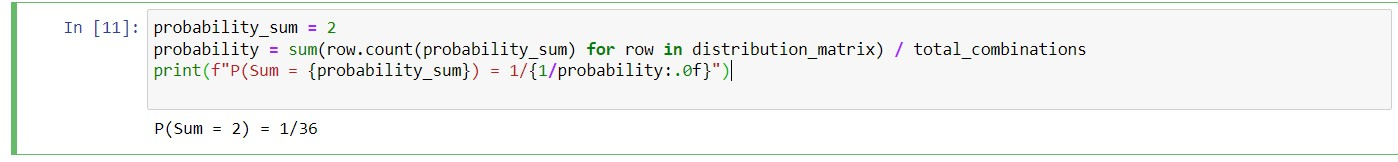
**Code**

probability\_sum = 2

probability = sum(row.count(probability\_sum) for row in distribution\_matrix) / total\_combinations

print(f"\nExample: P(Sum = {probability\_sum}) = 1/{1/probability:.0f}")

**code and output screenshot**



**PART-B**

**Logic**

Finding a transformation function called undoom\_dice that alters Die A and Die B's faces while maintaining the probability distribution of the sums is necessary to solve this puzzle. Based on the above limitations, we can compute the new faces using the original distribution.

**Code**

def undoom\_dice(die\_A, die\_B):

total\_combinations = len(die\_A) \* len(die\_B)

original\_probabilities = [0] \* 11

for face\_A in die\_A:

for face\_B in die\_B:

original\_probabilities[face\_A + face\_B - 2] += 1

new\_die\_A = [0] \* len(die\_A)

for face\_A in die\_A:

for face\_B in die\_B:

new\_die\_A[face\_A - 1] += original\_probabilities[face\_A + face\_B - 2]

max\_spots\_A = max(new\_die\_A)

scale\_factor\_A = 4 / max\_spots\_A

new\_die\_A = [min(4, int(face \* scale\_factor\_A)) for face in new\_die\_A]

new\_die\_B = [0] \* len(die\_B)

for face\_A in die\_A:

for face\_B in die\_B:

new\_die\_B[face\_B - 1] += original\_probabilities[face\_A + face\_B - 2]

return new\_die\_A, new\_die\_B

die\_A = [1, 2, 3, 4, 5, 6]

die\_B = [1, 2, 3, 4, 5, 6]

new\_die\_A, new\_die\_B = undoom\_dice(die\_A, die\_B)

print("New Die A:", new\_die\_A)

print("New Die B:", new\_die\_B)

**code and output and screenshot**

