Problem Statement: The Doomed Dice Challenge

# PART-A

## PROGRAM:

# Total Combinations n=int(input("Total number of dice:")) total\_combinations =6\*\*2

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

## EXPLANATION:

To get the total combinations of a dice, the formula is 6^n ,where n is the number of dice.Here, the input is n , which is number of dice. The output is total combinations.

## OUTPUT:

Total number of dice:2 Total combinations:36

## PROGRAM:

# Distribution of Combinations distribution\_matrix = [[0] \* 6 for \_ in range(6)]

for i in range(6):

for j in range(6):

distribution\_matrix[i][j] = (i + 1) + (j + 1)

# Display the distribution matrix print("Distribution Matrix:")

for row in distribution\_matrix:

print(row)

## EXPLANATION:

The first line creates a 6x6 matrix filled with zeros. The matrix will be used to store the sums of rolling two dice, where each die has values from 1 to 6. The nested for loops iterate over each element of the 6x6 matrix. The values in the matrix are filled by adding 1 to the values of i and j. Since the indices start from 0, adding 1 ensures that the values range from 2 to 12, representing the possible sums when

rolling two six-sided dice. The result is a matrix that shows all possible combinations of sums when rolling two dice.

## OUTPUT:

Distribution Matrix:

[2, 3, 4, 5, 6, 7]

[3, 4, 5, 6, 7, 8]

[4, 5, 6, 7, 8, 9]

[5, 6, 7, 8, 9, 10]

[6, 7, 8, 9, 10, 11]

[7, 8, 9, 10, 11, 12]

## PROGRAM

# Probability of Sums

probability\_sums = [row.count(sum\_val) / total\_combinations for sum\_val in range(2, 13)]

# Display the probability of each sum print("\nProbability of Sums:")

for sum\_val, prob in zip(range(2, 13), probability\_sums):

print(f"P(Sum = {sum\_val}) = {prob:.2f}")

## EXPLANATION:

This program calculates and displays the probability of getting each sum when rolling two six-sided dice.

probability\_sums = [row.count(sum\_val) / total\_combinations for sum\_val in range(2, 13)]: This list comprehension calculates the probability of each sum value by dividing the count of occurrences of that sum in each row by the total number of combinations. It iterates over the sums ranging from 2 to 12.

The next part of the program prints a header "Probability of Sums:" and then iterates through the calculated probabilities and displays them in a formatted string. The zip function is used to iterate over pairs of sum values and their corresponding probabilities.

## OUTPUT:

Probability of Sums: P(Sum = 2) = 0.00

P(Sum = 3) = 0.00

P(Sum = 4) = 0.00

P(Sum = 5) = 0.00

P(Sum = 6) = 0.00

P(Sum = 7) = 0.03

P(Sum = 8) = 0.03

P(Sum = 9) = 0.03

P(Sum = 10) = 0.03

P(Sum = 11) = 0.03

P(Sum = 12) = 0.03

# PART-B

## PROGRAM:

def calculate\_probability\_distribution(die\_a, die\_b):

probability\_sums = [0] \* 11 for face\_a in die\_a:

for face\_b in die\_b:

probability\_sums[face\_a + face\_b - 2] += 1 total\_combinations = len(die\_a) \* len(die\_b)

return [prob / total\_combinations for prob in probability\_sums]

def undoom\_dice(die\_a, die\_b):

original\_distribution = calculate\_probability\_distribution(die\_a, die\_b)

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

# Calculate new spots for Die A for i in range(len(die\_a)):

new\_die\_a[i] = original\_distribution[i] \* len(die\_b) - sum(new\_die\_b[j] for j in range(len(die\_b)))

# Check if the generated combination satisfies the condition if all(spots <= 4 for spots in new\_die\_a):

return new\_die\_a, new\_die\_b else:

return None

# Example usage 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)

if new\_die\_a is not None:

print("New Die A:", new\_die\_a) print("New Die B:", new\_die\_b)

else:

print("No valid combination found.")

## EXPLANATION:

calculate\_probability\_distribution function:

* This function takes two lists, die\_a and die\_b, representing the faces of two dice.
* It calculates the probability distribution of the sums of rolling these two dice.
* The function initializes a list probability\_sums to store the count of each possible sum.
* Nested loops iterate through all combinations of faces from die\_a and die\_b, updating the count for each sum.
* The total number of combinations is calculated, and the probability for each sum is computed.
* The function returns a list representing the probability distribution of the sums. undoom\_dice function:
* This function takes two lists of faces for two dice, die\_a and die\_b.
* It calculates the original probability distribution using calculate\_probability\_distribution.
* It initializes new lists, new\_die\_a and new\_die\_b, to store the reattached spots for each die.
* It calculates new spots for Die A based on the probability distribution, ensuring that the conditions are met.
* If the conditions are satisfied, it returns the new configurations for Die A and Die B; otherwise, it returns None.

## OUTPUT:

New Die A:

[0.16666666666666666, 0.3333333333333333, 0.5, 0.6666666666666666, 0.8333333333333334, 1.0]

New Die B:

[0, 0, 0, 0, 0, 0]

)