Doomed Dice Challenge: Part A - Code and Logic Explanation

# Code Overview:

1. print\_the\_matrix (message, matrix):
   * This function prints a given message and then displays each row of a provided matrix.
2. count\_the\_occurances\_of\_sums\_in\_matrix(matrix):
   * This function takes a matrix as input, iterates through its elements, and creates a dictionary to count the occurrences of each element. The keys represent the sums, and the values represent the occurrences.
3. probability\_calculator (occurances\_of\_sum\_all\_possible\_combinations, total\_combinations\_of\_dice\_when\_rolled\_together):
   * This function calculates and prints the probability of each sum in the given combinations. It utilizes the count of occurrences obtained from the previous function and the total number of combinations.
4. part\_A (dice\_A, dice\_B):
   * The main function for Part A of the experiment.
   * It initializes variables and matrices to store combinations and sums of dice rolls.
   * Iterates through each value in `dice\_A` and `dice\_B`, calculates the sum of pairs, and keeps track of the total number of combinations.
   * Prints the total combinations, the distribution matrix of all possible combinations, the distribution matrix of sums, and the probabilities of each sum.

# Mathematical Explanation:

Let's denote:

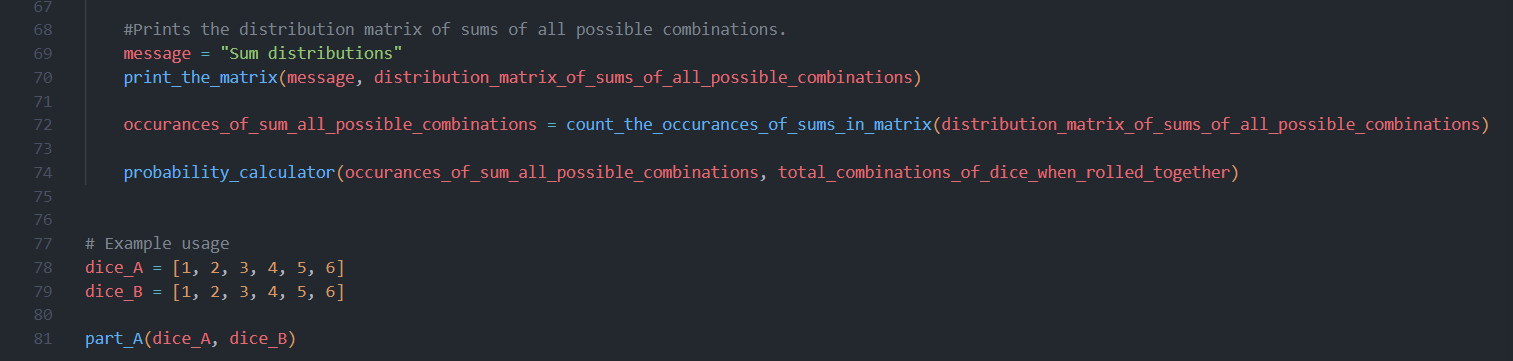
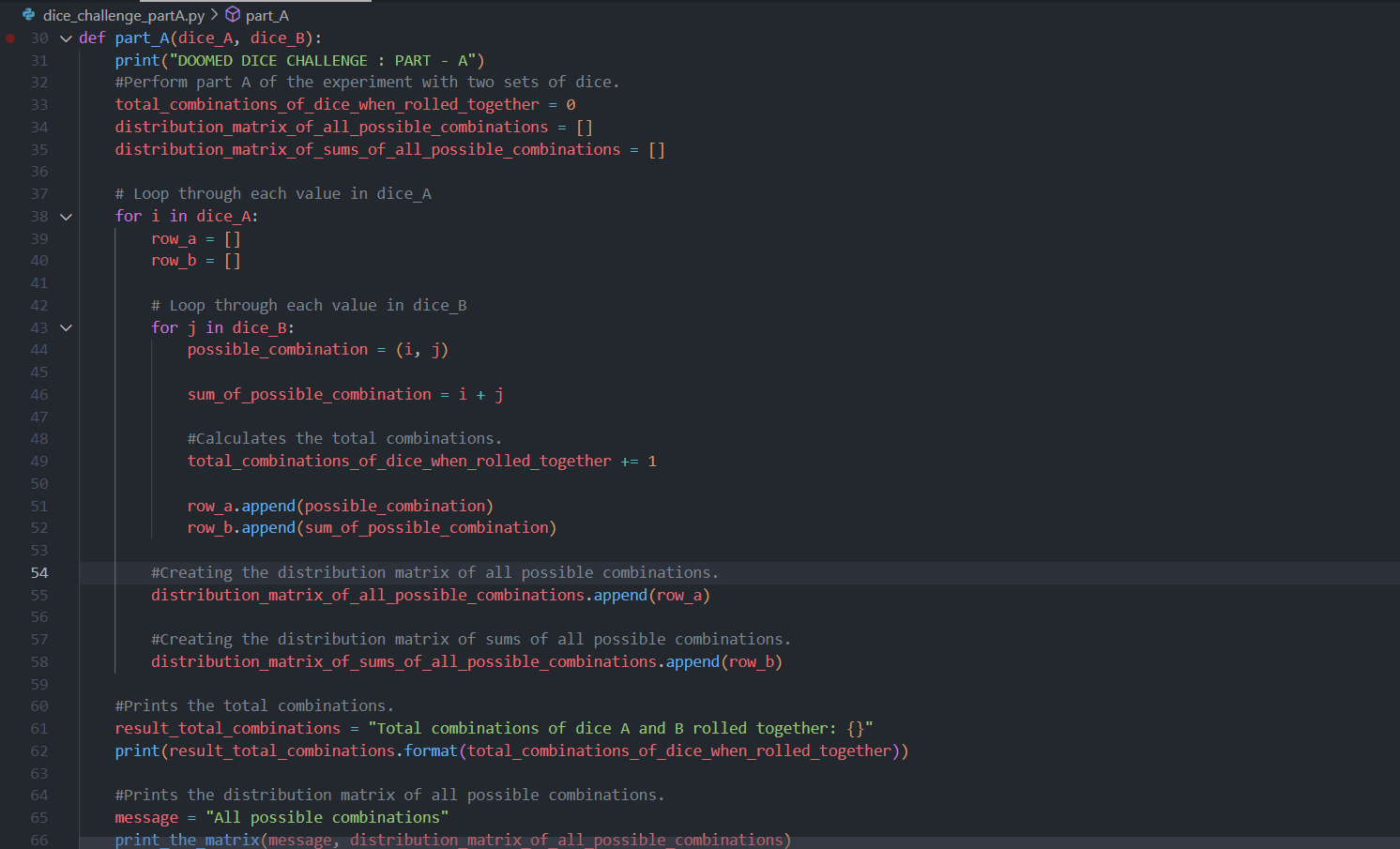
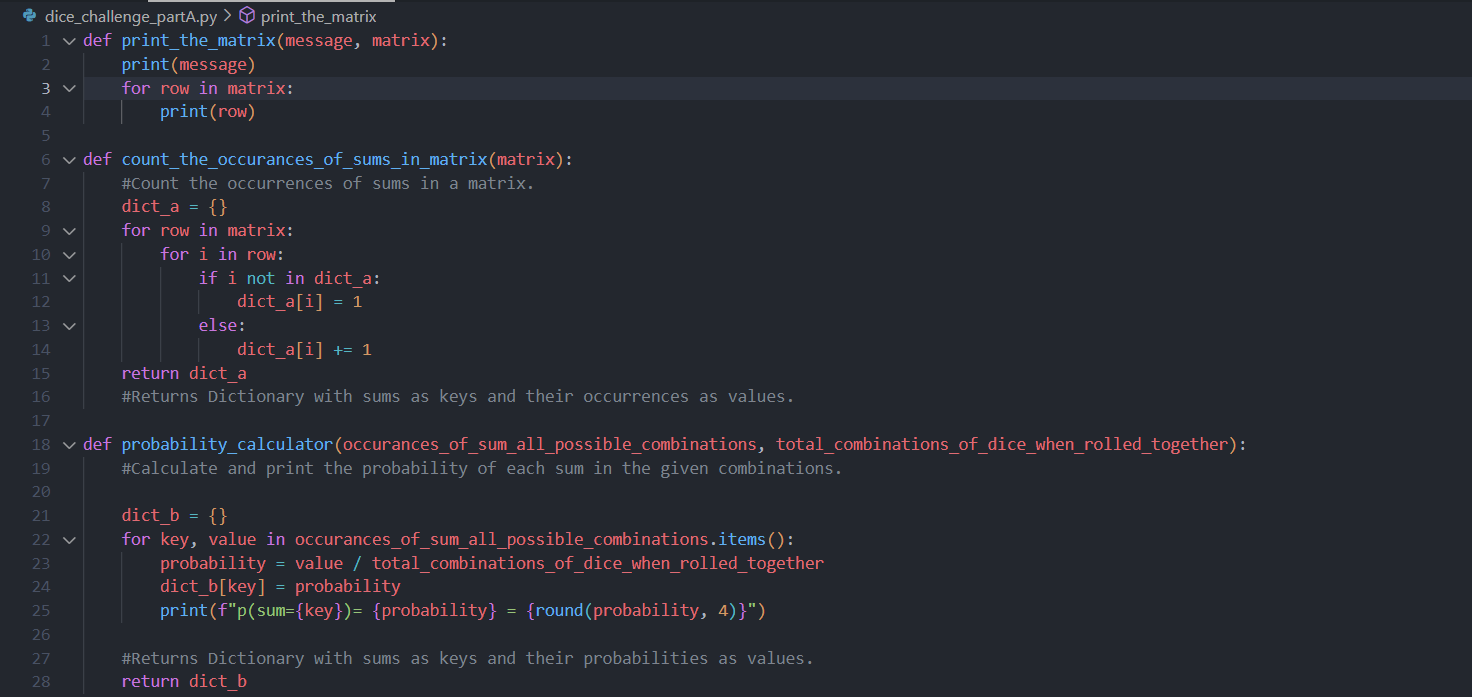
***A*** as the set of outcomes of dice A,

***B*** as the set of outcomes of dice B.

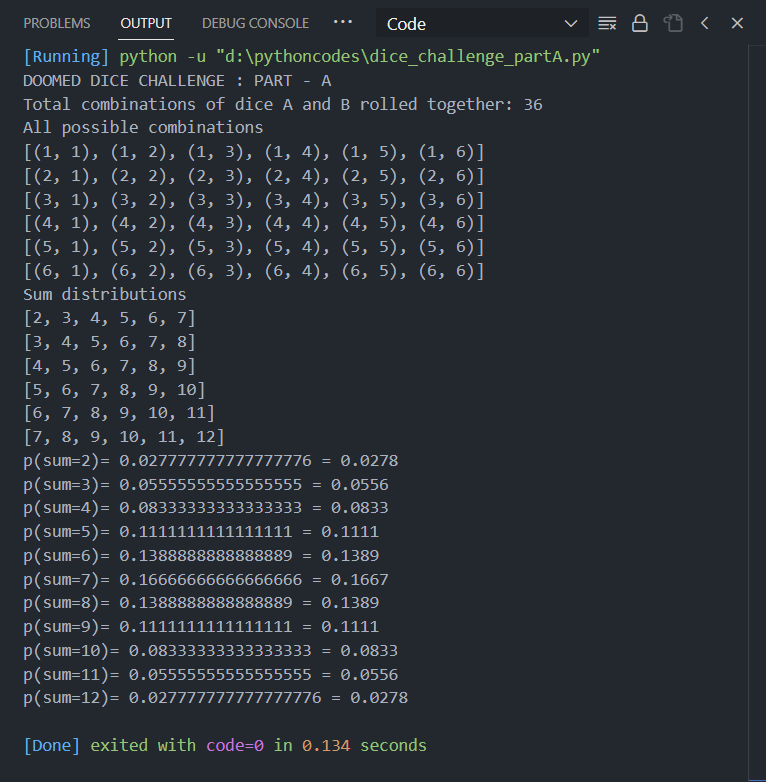
1. Total Combinations:
   * The total number of combinations when rolling both dice is the product of the sizes of sets ***A*** and ***B***.
   * Total combinations ***= A \* B***
2. Distribution Matrix of All Possible Combinations:
   * Each cell in this matrix represents a ***pair (i, j),*** where ***(i)*** is an outcome from ***A*** and ***(j)*** is an outcome from ***B***.
3. Distribution Matrix of Sums of All Possible Combinations:
   * Each cell in this matrix represents the ***sum (i + j)*** for the corresponding ***pair (i, j).***
4. Counting Occurrences:
   * The `count\_the\_occurances\_of\_sums\_in\_matrix` function counts the occurrences of each sum in the matrix.
5. Probability Calculation:
   * The probability of a ***sum (k)*** is given by the number of ***occurrences of (k)*** divided by the total number of combinations.
   * Probability of ***sum (k)*** = *(Number of occurrences of k) / (Total combinations)*

# Conclusion:

The provided code simulates rolling two dice, calculates the probabilities of different sums, and presents the results in a structured manner. The logic is grounded in fundamental principles of probability and combinatorics.

CODE INPUT:   


CODE OUTPUT:



Doomed Dice Challenge: Part B- Code and Logic Explanation

# Code Overview:

The provided Python code defines functions to calculate the sum distribution of two dice, compute the probability of specific sums, generate all possible combinations of dice based on certain restrictions, and finally transform the dice according to specified rules. Here's a step-by-step breakdown:

1. **sumOfArray(array)**

* This function calculates the sum of elements in an array.

1. **sum\_Distribution (Dice\_A, Dice\_B)**

* This function generates a 2D array representing the sum distribution of two dice. It initializes `sum\_Array` as a 2D array filled with zeros and calculates the sum for each combination of outcomes from `Dice\_A` and `Dice\_B`.

1. **single\_probability (Dice\_A, Dice\_B, sum)**

* Given two dice and a specific sum, this function calculates the probability of obtaining that sum based on the generated sum distribution.

1. **all\_probabilities (Dice\_A, Dice\_B)**

* This function calculates the probabilities for all possible sums using the `single\_probability` function and returns a dictionary with sum values as keys and their corresponding probabilities.

1. **posibilities\_Calc (curr, free\_space, input\_values, posibilities, fixed\_values, repetition=True)**

* This recursive function calculates all possible combinations of dice outcomes based on specified restrictions such as free space, input values, fixed values, and repetition. It uses backtracking to explore different combinations.

1. **transform (dice\_a, dice\_b)**

* The main function applies the transformation rules to generate new dice. It calculates the original probabilities, defines fixed and input values for both dice, calculates all possible combinations using `posibilities\_Calc`, and iterates through the combinations to find a match with the original probabilities.

# Mathematical Explanation:

**1. Sum Distribution:**

- The sum distribution of two dice is calculated by considering all possible combinations of outcomes from the two dice.

**2. Probability Calculation:**

- The probability of obtaining a specific sum is calculated by dividing the number of occurrences of that sum by the total number of possible combinations.

**3. Combinations Calculation:**

- The `posibilities\_Calc` function uses recursive backtracking to generate all possible combinations of outcomes for dice based on specified restrictions.

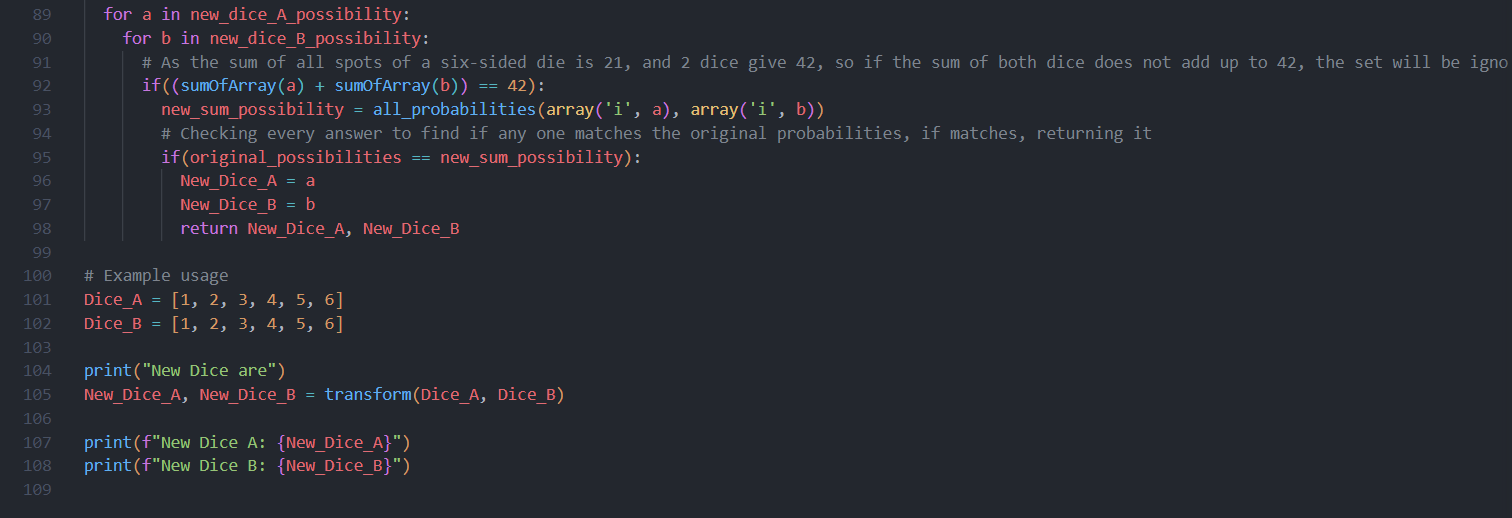
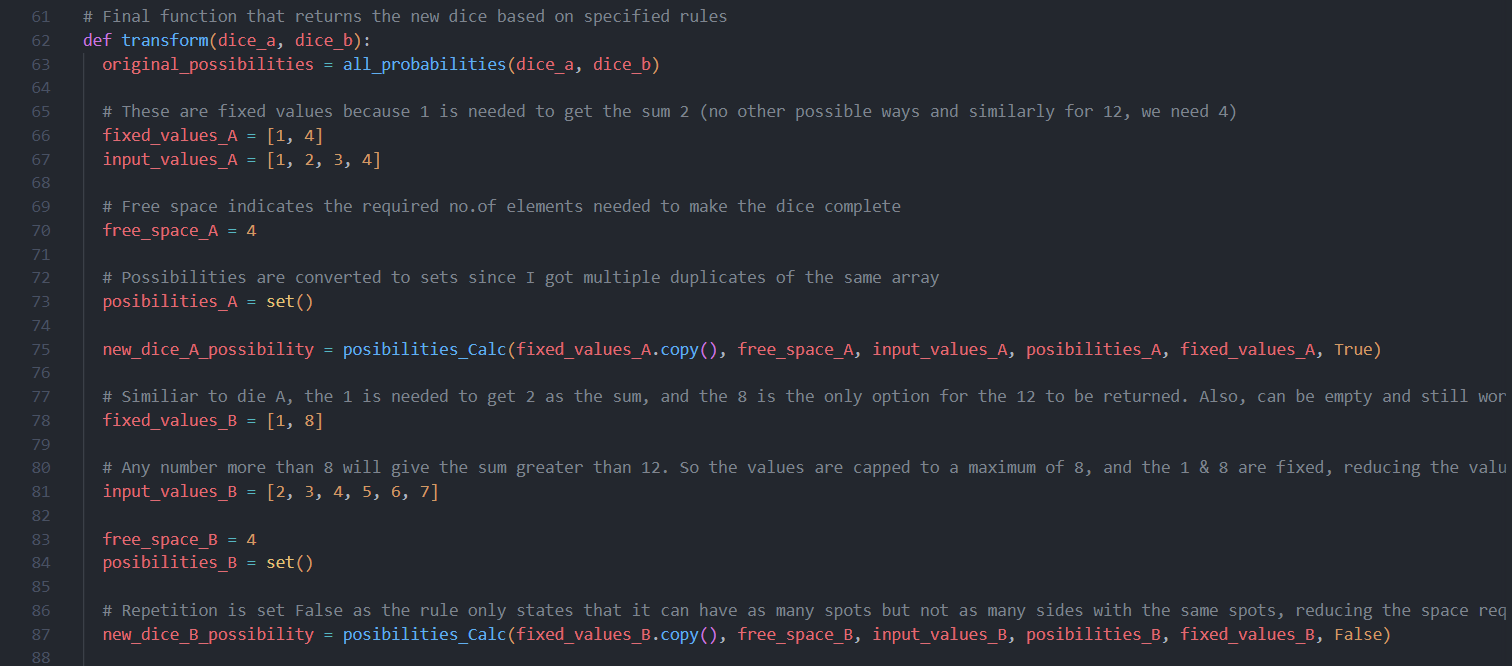
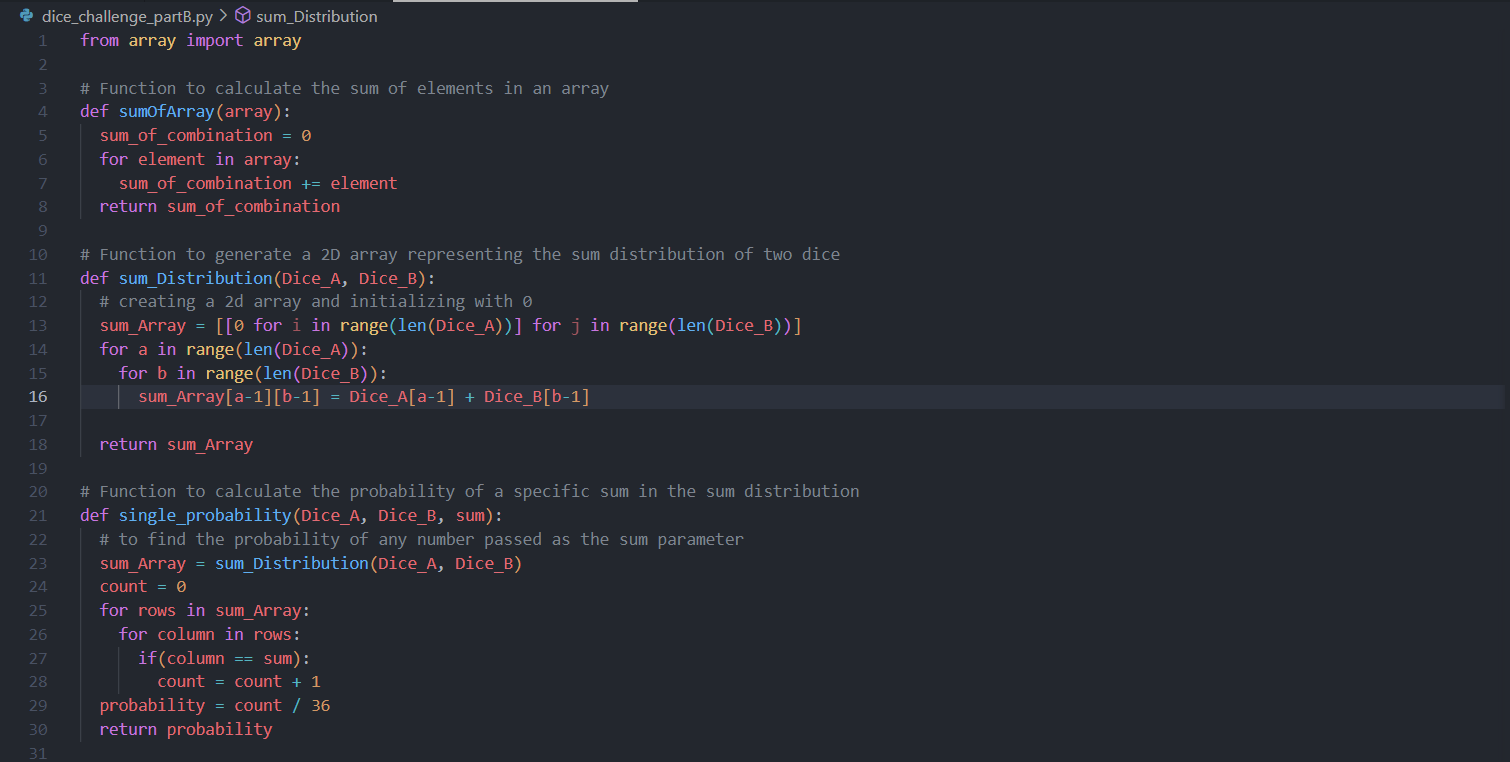
**4. Dice Transformation:**

- The `transform` function applies specific rules to transform the original dice into new dice that satisfy the given conditions, including constraints on sums and probabilities.

# Conclusion:

The code provides a comprehensive solution to the transformation of two dice, involving the calculation of sum distributions, probabilities, and the generation of possible outcomes. The backtracking algorithm efficiently explores different combinations to find the transformed dice that match the original probabilities.

CODE INPUT:



CODE OUTPUT:

