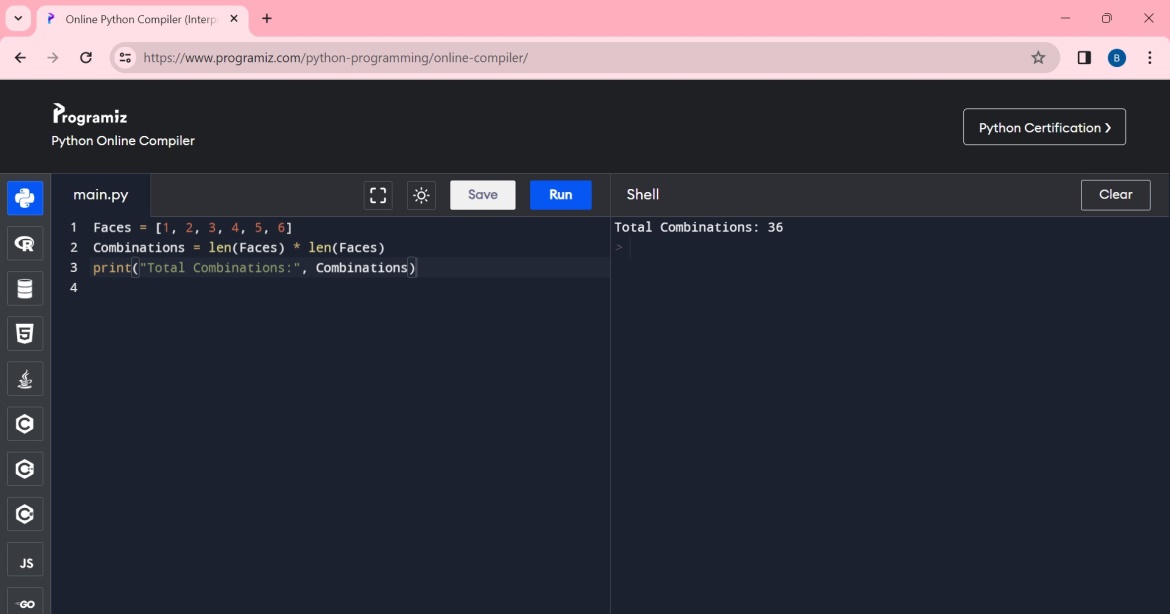
**PART A :** **QUESTION 1:**How many total combinations are possible? Show the math along with the code!

**LOGIC:**  
 1. Both dice have 6 faces, each with an equal probability of landing on any face. To count the total combinations, we consider every possible pairing of faces from Die A and Die B.

2. Die A has 6 possible outcomes. Die B also has 6 possible outcomes. Therefore, the total number of possible combinations is 6 \* 6 = 36.

**Code and Output:**



**Code Explanation:** 1. First line creates a list representing the possible values on a single die.

2. Second line  calculates the total number of possible combinations when rolling two dice. It does this by multiplying the length of the “Faces” list by itself.

3. Multiplying  “len(faces)” by itself essentially creates a 6\*6 grid of possible outcomes, where each cell represents a unique pairing of faces from Die A and Die B.

4. Third lie prints the calculated number of combinations, which will be 36.

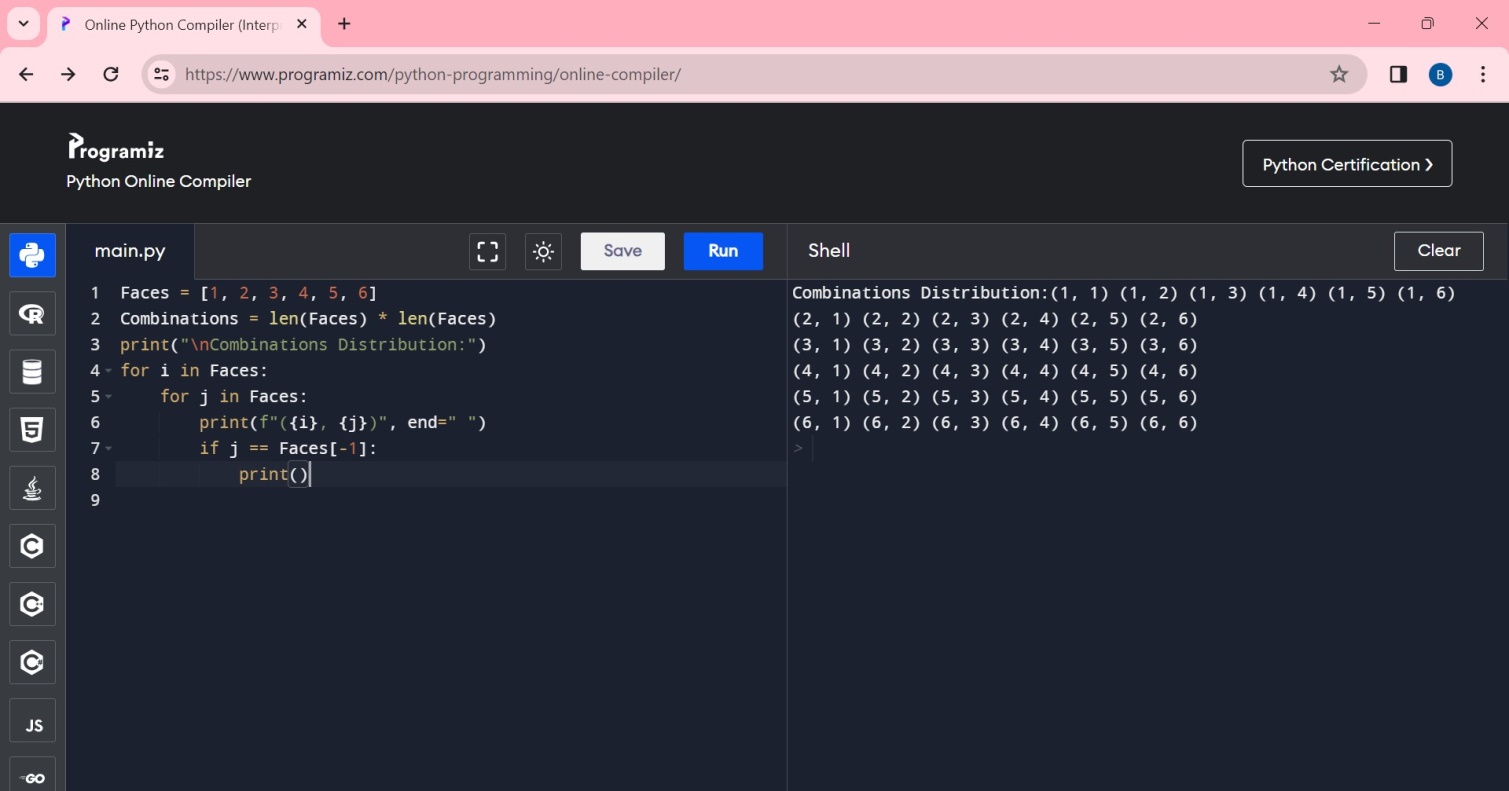
**QUESTION 2:** Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code!

**LOGIC:  
 1. Representing Dice Faces**

**2. Calculating Total Combinations**

**3. Printing Combinations Distribution and creating Nested Loops to Generate Combinations**

**CODE AND OUTPUT:**



**CODE EXPLANATION:**

1. First line creates a list representing the possible values on a single die.

2. Second line calculates the total number of possible combinations when rolling two dice. It does this by multiplying the length of the “Faces” list by itself.

3. Multiplying “len(faces)” by itself essentially creates a 6\*6 grid of possible outcomes, where each cell represents a unique pairing of faces from Die A and Die B.

4. Third line Prints a label to indicate the upcoming output.

5**.** The outer loop (for i in Faces) iterates through each possible value on Die A.

6. The inner loop (for j in Faces) iterates through each possible value on Die B.

7. Sixth line prints the current combination (face values on Die A and Die B) within parentheses and a space. The end=" " argument prevents a newline, keeping combinations on the same line.

8. Seventh line Checks if it's the last value in the inner loop. If so, print() adds a newline to start a new row of combinations, creating a visual grid.

**QUESTION 3:** Calculate the Probability of all Possible Sums occurring among the number of combinations from (2). **Example**: P(Sum = 2) = 1/X as there is only one combination possible to obtain Sum = 2. Die A = Die B = 1.

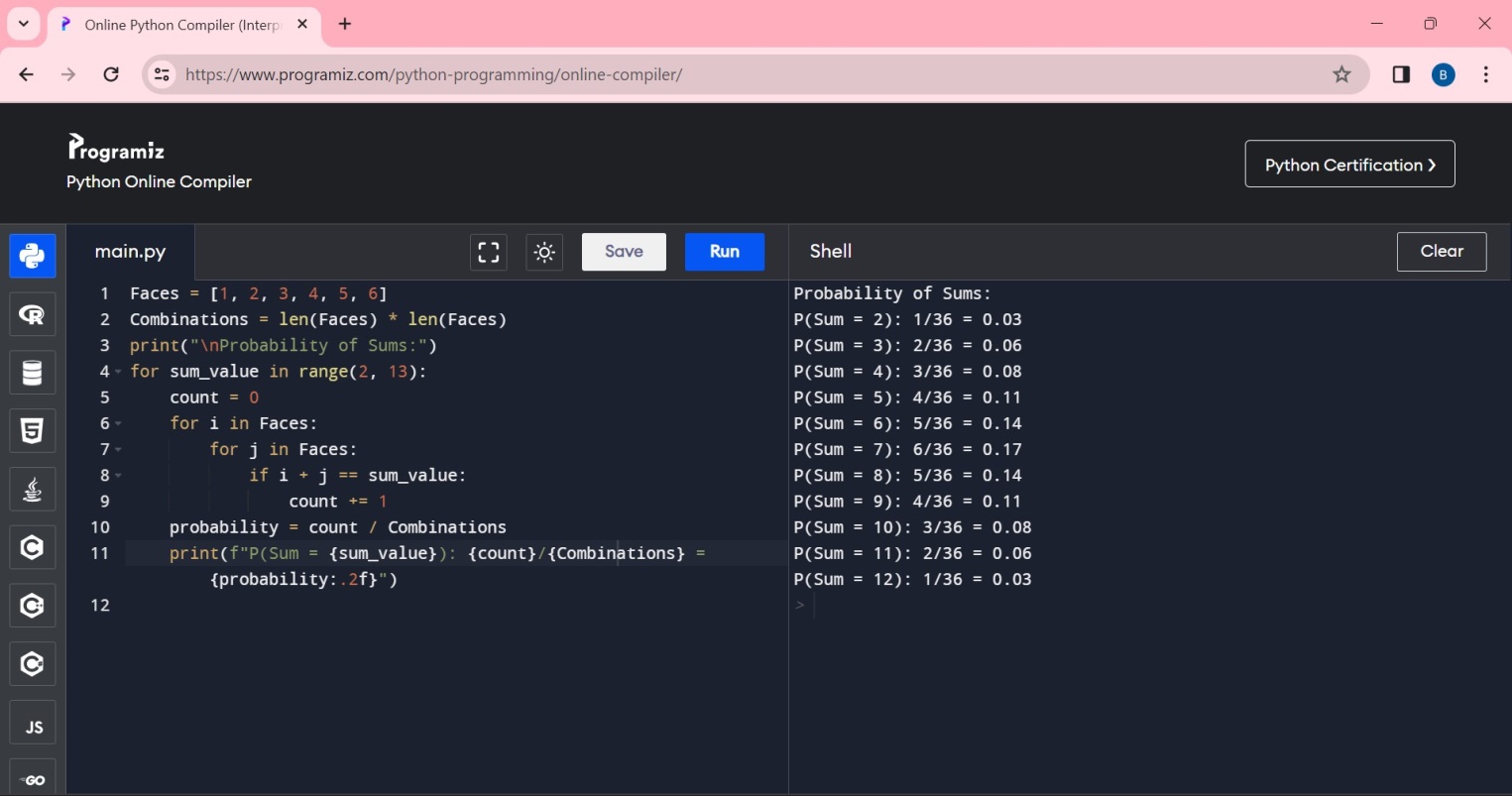
**LOGIC:**

**1.** Defines the possible face values.

**2.** Calculates the total combinations (36).

3. Prints the probability with two decimal places and **using Nested Loops to Generate output**.

**CODE AND OUTPUT:**



**EXPLANATION:**

1.First line creates a list containing the possible face values on a single die.

2.Second line calculates the total number of possible combinations by multiplying the length of the Faces list by itself (6 \* 6 = 36).

3.Third line prints a label to indicate the upcoming output.

4.Fourth line iterates through each possible sum value from 2 to 12.

5.count = 0: Sets a counter to 0 to track the number of combinations that result in the current sum\_value.

6.The outer loop (for i in Faces) iterates through each possible value on Die A.

7.The inner loop (for j in Faces) iterates through each possible value on Die B.

8. 8th line  Checks if the sum of the current combination (values on Die A and Die B) matches the target sum\_value.

9.count += 1 increments the counter.

10.probability = count / Combinations: Calculates the probability of the current sum\_value by dividing the number of favorable outcomes (count) by the total possible outcomes (Combinations)

11.Finally 11th line prints the probability with two decimal places, along with the count of favorable outcomes and the total combinations.

**PART-B:**

**QUESTION:** Now comes the real challenge. You were happily spending a lazy afternoon playing your board game with your dice when suddenly the mischievous Norse God Loki ( You love Thor too much & Loki didn’t like that much ) appeared.Loki dooms your dice for his fun removing all the “Spots” off the dice.

No problem! You have the tools to re-attach the “Spots” back on the Dice.

However, Loki has doomed your dice with the following conditions:

Die A cannot have more than 4 Spots on a face. Die A may have multiple faces with the same number of spots.Die B can have as many spots on a face as necessary i.e. even more than 6.But in order to play your game, the probability of obtaining the Sums must remain the same!

So if you could only roll P(Sum = 2) = 1/X, the new dice must have the spots reattached such that those probabilities are not changed.

Input:

Die\_A = [1, 2, 3, 4, 5, 6] & Die B = Die\_A = [1, 2, 3, 4, 5, 6]

Output:

A Transform Function undoom\_dice that takes (Die\_A, Die\_B) as input &

outputs New\_Die\_A = [?, ?, ?, ?, ?, ?],New\_Die\_B = [?, ?,

?, ?, ?, ?] where,No New\_Die A[x] > 4

**LOGIC:**

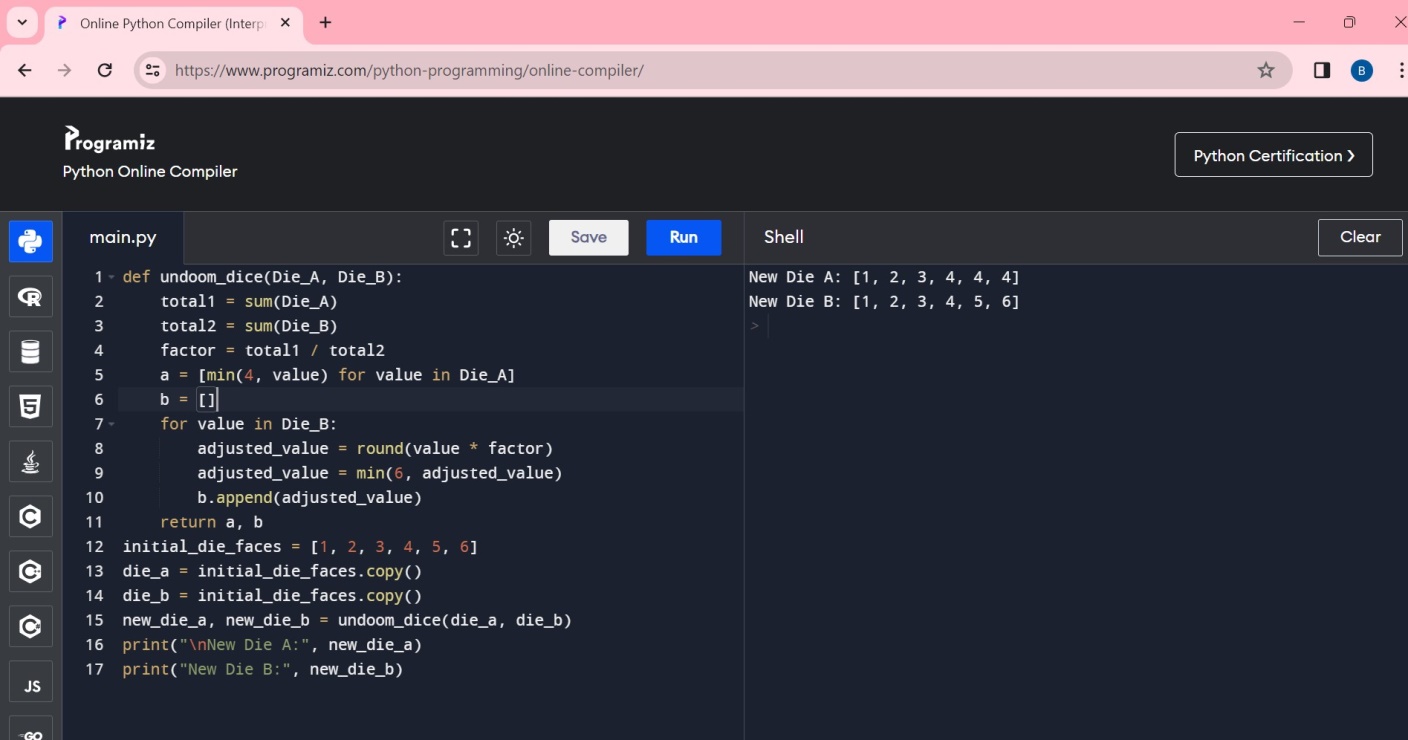
1. The code calculates the sum of both dice (total1 and total2).

2. It then computes a scaling factor (factor) by dividing total1 by total2.

3. It modifies Die A by setting all face values exceeding 4 to 4 using a list comprehension.

4. For Die B, it iterates through each face value, scales it by the factor, rounds it to the nearest whole number, then caps it at 6.

**CODE AND OUTPUT:**



**CODE EXPLANATION:**

1.First line defines a function named undoom\_dice that takes two lists representing dice faces as input and returns modified lists for the adjusted dice.

2. Calculating Sums and Factor:

total1 = sum(Die\_A): Calculates the total sum of face values on Die A.

total2 = sum(Die\_B): Calculates the total sum of face values on Die B.

factor = total1 / total2: Calculates a scaling factor by dividing the sums of the two dice.

3.Fifth line creates a new list a for Die A by ensuring no face value exceeds 4. It uses a list comprehension to iterate through each value in Die\_A and apply the min(4, value) function, effectively capping values at 4.

4.Sixth line initializes an empty list b to store the modified face values for Die B.

5.Seventh line iterates through each face value in Die\_B.

6.8th line scales the current face value by the factor and rounds it to the nearest whole number.

7.9th line Caps the adjusted value at 6 to ensure it doesn't exceed 6.

8.10th line appends the adjusted value to the list b.

9. return a, b: Returns the modified lists a (representing the new Die A) and b (representing the new Die B) as a tuple.

10.The remaining lines demonstrate how to use the function

11.initial\_die\_faces = [1, 2, 3, 4, 5, 6]: Defines the initial face values for both dice.

12.die\_a and die\_b are created as copies of the initial values.

13.new\_die\_a, new\_die\_b = undoom\_dice(die\_a, die\_b): Calls the function to adjust the dice and stores the results.

14.Last two lines prints the modified dice faces.