**Problem Statement: The Doomed Dice Challenge**

The below problems must be solved & implemented in Python/Java/Ruby/C++/Go

You are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to 6.

You can only roll both the dice together & your turn is guided by the obtained sum.

Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9

You may represent Dice as an Array or Array-like structure.

Die A = [1, 2, 3, 4, 5, 6] where the indices represent the 6 faces of the die & the value on each face.

**Part-A (15-20 Minutes):**

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

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!

Hint: A 6 x 6 Matrix.

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.

**Solution:**

faces = [1, 2, 3, 4, 5, 6]

total = len(faces) \*\* 2

print("Total Combinations:", total)

combinations = [(i, j) for i in faces for j in faces]

print("\nCombinations Distribution:")

for i in range(0, total, len(faces)):

print(combinations[i:i + len(faces)])

print("\nProbability of Sums:")

for i in range(2, 13):

count = sum(1 for face in combinations if sum(face) == i)

probability = count / total

print(f"P(Sum = {i}): {count}/{total} = {probability:.2f}")

**Explanation:**

**1.Initializtion of faces:**

faces = [1, 2, 3, 4, 5, 6]

**2.Calculation of total combinations:**

total = len(faces) \*\* 2

print("Total Combinations:", total)

The variable total is assigned the value of the square of the length of the faces list, which represents the total number of possible combinations when two dice are rolled.

**3.Generating combinations:**

combinations = [(i, j) for i in faces for j in faces]

This line creates a list called combinations containing tuples, where each tuple represents a possible combination of two dice rolls.

**4. Printing Combinations Distribution:**

print("\nCombinations Distribution:")

for i in range(0, total, len(faces)):

print(combinations[i:i + len(faces)])

This part of the code prints the combinations in a formatted way, separating each set of combinations corresponding to a unique value of the first die.

**5. Calculating Probability of Sums:**

print("\nProbability of Sums:")

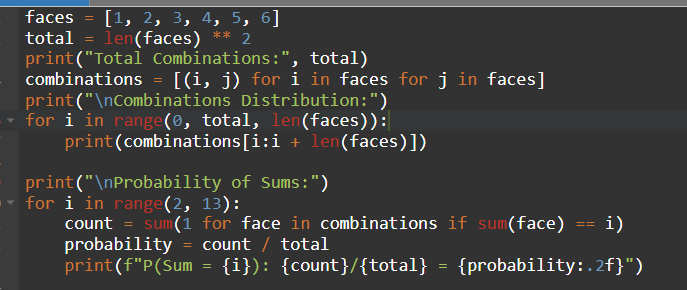
for i in range(2, 13):

count = sum(1 for face in combinations if sum(face) == i)

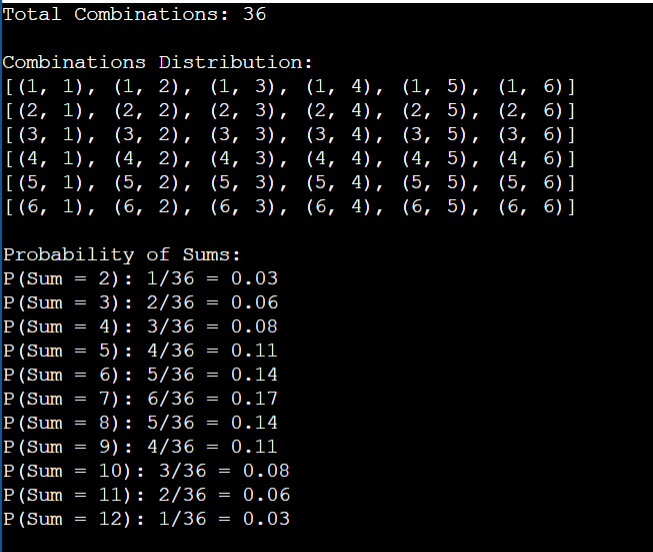
probability = count / total

print(f"P(Sum = {i}): {count}/{total} = {probability:.2f}")

This section calculates and prints the probability of each possible sum when two dice are rolled. It iterates over possible sums from 2 to 12, counts the occurrences of each sum in the combinations list, calculates the probability, and prints the result.



**Output:**



**Part-B (25-30 Minutes):**

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

**Solution:**

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

scaling\_factor = sum(die\_a) / sum(die\_b)

a = [min(4, spots) for spots in die\_a]

b = [min(6, round(spots \* scaling\_factor)) for spots in die\_b]

return a, b

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

die\_b = die\_a

new\_die\_a, new\_die\_b = undoom\_dice(die\_a, die\_b)

print("\nNew Die A:", new\_die\_a)

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

**Explanation:**

**1. Function Definition:**

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

Here, a function named undoom\_dice is defined. It takes two parameters, die\_a and die\_b, which represent the spots on two dice.

**2. Scaling Factor Calculation:**

scaling\_factor = sum(die\_a) / sum(die\_b)

The code calculates a scaling factor by dividing the sum of spots on die\_a by the sum of spots on die\_b.

**3. Creating New Dice:**

a = [min(4, spots) for spots in die\_a]

b = [min(6, round(spots \* scaling\_factor)) for spots in die\_b]

Two new lists a and b are created. For a, each element is set to the minimum value between 4 and the corresponding spot on die\_a. For b, each element is set to the minimum value between 6 and the result of rounding the corresponding spot on die\_b multiplied by the scaling factor.

**4. Returning New Dice:**

return a, b

The function returns the new dice, represented by the lists a and b.

**5. Dice Initialization:**

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

die\_b = die\_a

Two dice, die\_a and die\_b, are initialized with the same values.

**6. Calling the Function:**

new\_die\_a, new\_die\_b = undoom\_dice(die\_a, die\_b)

The function undoom\_dice is called with die\_a and die\_b as arguments, and the returned values are assigned to new\_die\_a and new\_die\_b.

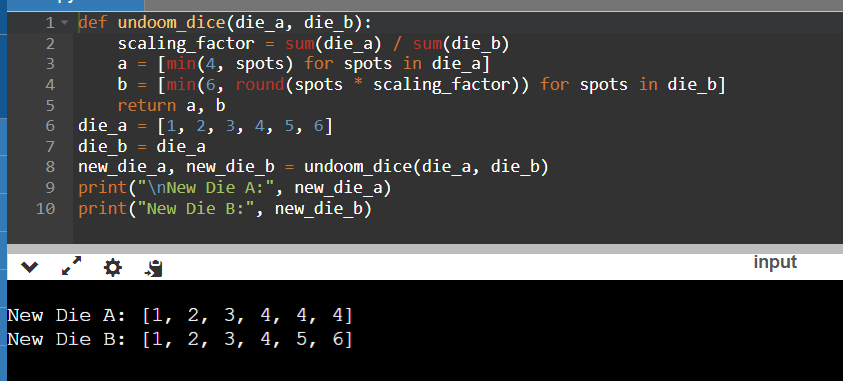
**7. Printing the Result:**

print("\nNew Die A:", new\_die\_a)

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

The code prints the new values of die\_a and die\_b after applying the transformation defined in the undoom\_dice function.

**Output:**

****