Logic to the Problem Statement

Part A: Original Dice

1. **Total Combinations**

   - Each die has 6 faces.
   - When rolling two dice, each face of die A can be combined with any face of die B.
   - Therefore, the total number of combinations is 6 (faces in die A) * 6 (faces in die B) = 36.

2. **Distribution of Combinations (Sum Frequencies)**

   - Iterate through all possible sums (2 to 12, representing the sum of two dice rolls).
   - For each sum `i+j`, count the number of combinations that result in that sum.
   - A combination contributes to the count if the sum of the faces on die A and die B equals `i+j`.
   - Store these counts in an array `freq`.

3. **Probability of Each Sum**

   - Divide the frequency of each sum (`freq[i+j]`) by the total number of combinations (36).
   - This gives the probability of obtaining that sum when rolling the dice.

Solution for the Problem Statement

Part A: Original Dice

1. **Total Combinations Calculation**

   - Since each die has 6 faces, the total combinations when rolling two dice are 6×6=36
     (i.e) 6×6=36.

2. **Distribution of Combinations**

   - Generate all possible combinations by iterating over each face of Die A and Die B and display them in a matrix format.

3. **Probability Calculation**

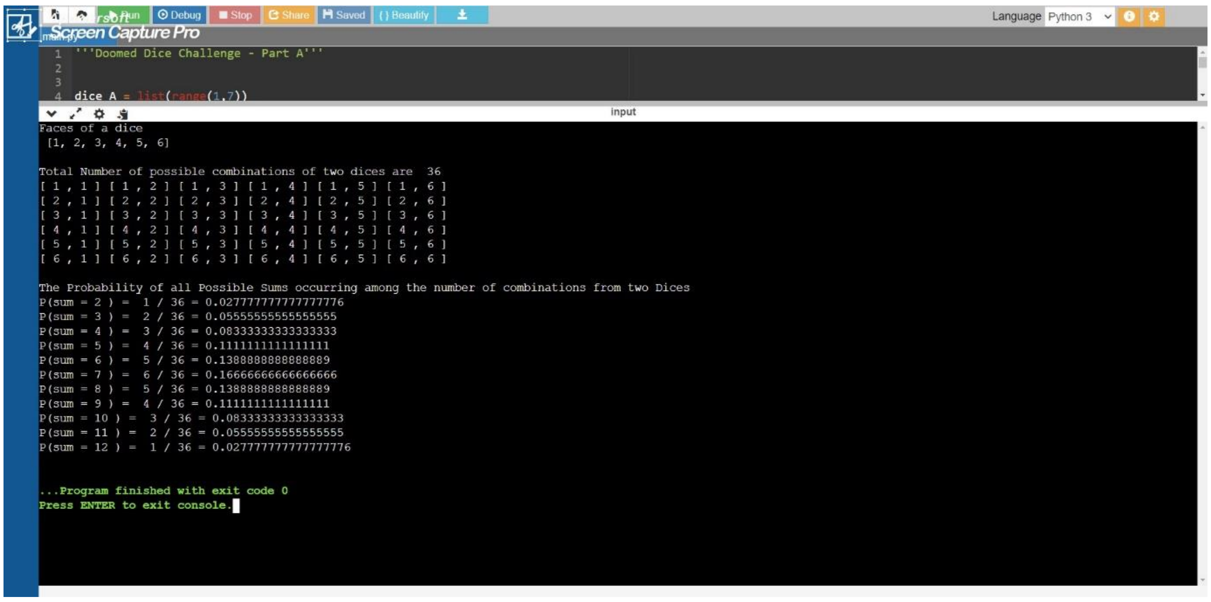
   - Calculate the frequencies of sums occurring from the combinations.

- Divide the frequency of each sum by the total combinations (36) to get the probability of each sum.

Code for Part A

```
'''Doomed Dice Challenge - Part A'''

dice_A = list(range(1,7))

dice_B = list(range(1,7))

print("Faces of a dice\n",dice_A)

print("\nTotal Number of possible combinations of two dices are
",len(dice_A)*len(dice_B))

for i in dice_A:

    for j in dice_B:

        print("[",i,",",j,"]",end=" ")

    print()

print("\nThe Probability of all Possible Sums occurring among the number of
combinations from two Dices")

freq = {}

for i in dice_A:

    for j in dice_B:

        if (i+j in freq):

            freq[i+j] += 1

        else:

            freq[i+j] = 1

for key, value in freq.items():

        print("P(sum=",key,")=",value,"/",(len(dice_A)*len(dice_B)),"=",(value/(len(dice
_A)*len(dice_B))))
```

Output :



```
'''Doomed Dice Challenge - Part A'''


dice_A = list(range(1,7))
```

```
Faces of a dice
 [1, 2, 3, 4, 5, 6]

Total Number of possible combinations of two dices are  36
[ 1 , 1 ] [ 1 , 2 ] [ 1 , 3 ] [ 1 , 4 ] [ 1 , 5 ] [ 1 , 6 ]
[ 2 , 1 ] [ 2 , 2 ] [ 2 , 3 ] [ 2 , 4 ] [ 2 , 5 ] [ 2 , 6 ]
[ 3 , 1 ] [ 3 , 2 ] [ 3 , 3 ] [ 3 , 4 ] [ 3 , 5 ] [ 3 , 6 ]
[ 4 , 1 ] [ 4 , 2 ] [ 4 , 3 ] [ 4 , 4 ] [ 4 , 5 ] [ 4 , 6 ]
[ 5 , 1 ] [ 5 , 2 ] [ 5 , 3 ] [ 5 , 4 ] [ 5 , 5 ] [ 5 , 6 ]
[ 6 , 1 ] [ 6 , 2 ] [ 6 , 3 ] [ 6 , 4 ] [ 6 , 5 ] [ 6 , 6 ]

The Probability of all Possible Sums occurring among the number of combinations from two Dices
P(sum = 2 ) =  1 / 36 = 0.027777777777777776
P(sum = 3 ) =  2 / 36 = 0.05555555555555555
P(sum = 4 ) =  3 / 36 = 0.08333333333333333
P(sum = 5 ) =  4 / 36 = 0.1111111111111111
P(sum = 6 ) =  5 / 36 = 0.1388888888888889
P(sum = 7 ) =  6 / 36 = 0.16666666666666666
P(sum = 8 ) =  5 / 36 = 0.1388888888888889
P(sum = 9 ) =  4 / 36 = 0.1111111111111111
P(sum = 10 ) =  3 / 36 = 0.08333333333333333
P(sum = 11 ) =  2 / 36 = 0.05555555555555555
P(sum = 12 ) =  1 / 36 = 0.027777777777777776


...Program finished with exit code 0
Press ENTER to exit console.
```

Logic to the Problem Statement

Part B: Doomed Dice

Undoom Dice

- Design a function undoom_dice to reattach spots on the dice.
- For Die A, set spots such that no face has more than 4 spots.
- For Die B, maintain the same faces as before.
- Ensure that the probabilities of obtaining the sums remain unchanged after reattaching the spots.

Solution for the Problem Statement

Part B: Doomed Dice

Undoom Dice

This code tackles the challenge of re-attaching "spots" (values) to die A while maintaining the original probability distribution of sums when rolling both dice. Here's a breakdown of the solution:

Data Structures

list1: Represents the possible values for the modified die A (1 to 4).

list2: Represents the possible values for die B (1 to 11, can have higher values).

orgnl: An array containing the original probability distribution (provided).

comb1: Stores all possible combinations of values for the modified die A.

comb2: Stores all possible combinations of values for die B.

Functions

find_comb_B(ind, temp): This recursive function generates all possible combinations of values for die B using backtracking. It iterates through list2 and adds elements to a temporary list (temp). When the temporary list reaches size 6 (number of faces), it's added to comb2.

find_comb_A(ind, temp): Similar to find_comb_B, this function generates all possible combinations for the modified die A using list1.

total_comb(dice_A, dice_B): Calculates the total number of combinations by multiplying the lengths of dice_A and dice_B.

distribution(dice_A, dice_B): This function calculates the distribution of sums for a given pair of dice (A and B). It iterates through each die and calculates the sum of all possible combinations, storing the frequency of each sum in an array (res).

probabilities(dice_A, dice_B): This function checks if the probability distribution for the given dice (A and B) matches the original distribution (orgnl). It first verifies if the sum of maximum values in both dice is 12 (ensuring all possible sums can be achieved). Then, it calculates the distribution using distribution and the total number of combinations using total_comb. Finally, it iterates through each possible sum and compares the calculated frequency with the original probability from orgnl. If any mismatch is found, it returns False, indicating the distribution is not the same. Otherwise, it returns True.

Overall Approach

Generate All Combinations: The code first generates all possible combinations of values for both the modified die A (comb1) and die B (comb2) using the find_comb_A and find_comb_B functions.

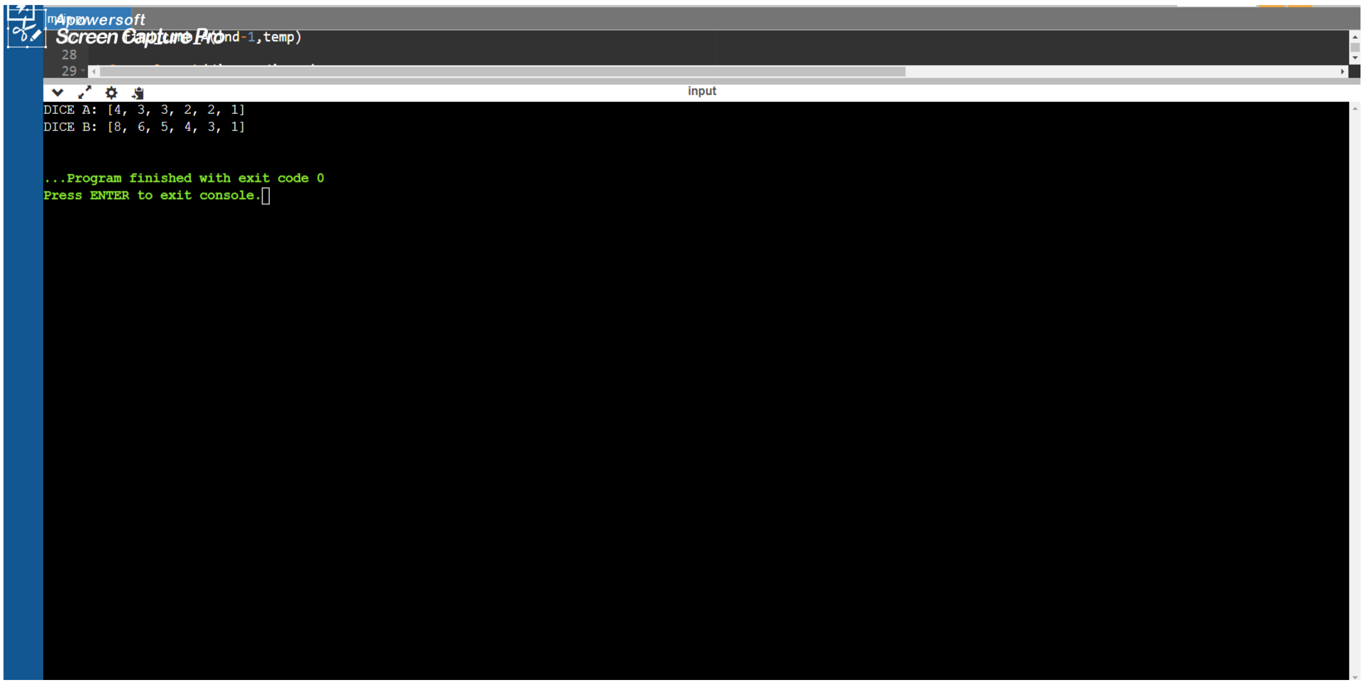
Check Probability Distribution: It then iterates through each combination of die A (dice_A) and die B (dice_B) from the generated lists.

Verify Match: For each pair, the probabilities function calculates the distribution of sums and compares it with the original distribution (orgnl). If the distributions match (meaning the probabilities of each sum remain the same), the code prints both dice_A and dice_B as a solution.

Code for Part B

```
'''Doomed Dice Challenge - Part B'''

list1 = list(range(1,5))

list2 = list(range(1,12))

orgnl = [0.0, 0.0, 0.027777777777777776, 0.05555555555555555, 0.08333333333333333, 0.1111111111111111, 0.1388888888888889, 0.16666666666666666, 0.1388888888888889, 0.1111111111111111, 0.08333333333333333, 0.05555555555555555, 0.027777777777777776]

comb1 = []

comb2 = []

def find_comb_B(ind,temp):

    if len(temp)==6:

        comb2.append(temp)

        return

    if ind<0:

        return

    find_comb_B(ind-1,temp+[list2[ind]])

    find_comb_B(ind-1,temp)

def find_comb_A(ind,temp):

    if len(temp)==6:

        comb1.append(temp)

        return

    if ind<0:

        return

    find_comb_A(ind,temp+[list1[ind]])

    find_comb_A(ind-1,temp)

def total_comb(dice_A,dice_B):
    return len(dice_A)*len(dice_B)
def distribution(dice_A,dice_B):
    res = [0 for i in range(max(dice_A)+max(dice_B)+1)]
    for i in dice_A:
        for j in dice_B:
            res[i+j] += 1
    return res
def probabilities(dice_A,dice_B):
    if max(dice_A) + max(dice_B) == 12:
        dist = distribution(dice_A,dice_B)
        ln = total_comb(dice_A,dice_B)
        for event in range(max(dice_A)+max(dice_B)+1):
            if dist[event]:
                if orgnl[event] != dist[event]/ln:
                    return False
        return True
    return False
find_comb_B(len(list2)-1,[])
find_comb_A(len(list1)-1,[])
for dice_A in comb1:
    for dice_B in comb2:
        if probabilities(dice_A,dice_B):
            print("DICE A:",dice_A)
            print("DICE B:",dice_B)
```

Output :



```
DICE A: [4, 3, 3, 2, 2, 1]
DICE B: [8, 6, 5, 4, 3, 1]


...Program finished with exit code 0
Press ENTER to exit console.
```