**Problem Statement: The Doomed Dice Challenge**

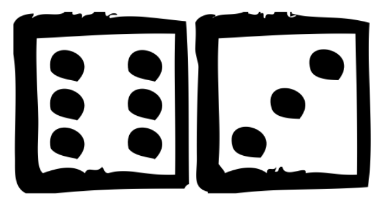
The below problems must be solved & implemented in Python/Java/Ruby/C++/Go

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.

1. 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 to Part-A:**

Let us take an approach where we use the concepts of basic probability to understand the problem and arrive at a solution.

We have two six-sided dice and they are represented as follows Die A(or)B = [1, 2, 3, 4, 5, 6]

Here, let us consider the event of rolling two dice together. So the total number of possibilities or the Sample Space(S) for the event is as follows:

S = { (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) }

**Therefore, the total number of possibilities are:** n(S) = 36

The above sample is obtained simply by the cartesian product to the event sets for the two six-sided dice:

Die A = {1, 2, 3, 4, 5, 6} and Die B = {1, 2, 3, 4, 5, 6}

A X B = S

In the cartesian product we get ordered pairs as (x,y) where x belongs to set A and y belongs to set B and we exhaust all possible combinations to obtain the cartesian product.

We can easily find the total number of combinations possible by finding the cardinality for the cartesian product, as the cardinality of the cartesian product is nothing but the product of the cardinalities of the involved sets.

So, n(S = A**X**B) = n(A) x n(B) = 6 x 6 = 36

**Probabilities of all possible Sums:**

We can see from the above sample space that the lowest possible sum(2) and highest possible sum(12) have only one combination each.

They are, P(Sum = 2) = 1/36 where, Die A = Die B = 1

P(Sum = 12) = 1/36 where, Die A = Die B = 6

The number of possible sums range from 2 through 12 and their probabilities are as follows:

P(Sum = 2) = 1/36 P(Sum = 8) = 5/36

P(Sum = 3) = 2/36 = 1/18 P(Sum = 9) = 4/36 = 1/9

P(Sum = 4) = 3/36 = 1/12 P(Sum = 10) = 3/36 = 1/12

P(Sum = 5) = 4/36 = 1/9 P(Sum = 11) = 2/36 = 1/18

P(Sum = 6) = 5/36 P(Sum = 12) = 1/36

P(Sum = 7) = 6/36 = 1/6

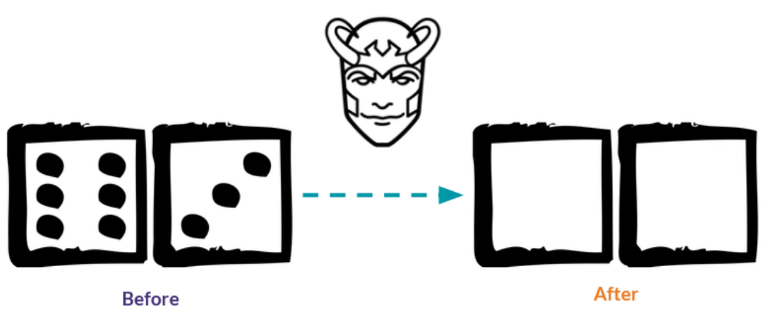
**Part-B (25-30 Minutes):**

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 to Part-B:**

In this part of the challenge, we have to transform the given dice so as to meet the required conditions.

The dice must be altered in such a way that the original probabilities associated with the Sums do not change.

The first thing that comes to mind while attempting to solve this problem is the maximum and minimum sums possible which are 12 and 2 respectively. Adhering to the given conditions it is safe to say that this is how we should attempt to alter the dice:

Die\_A = [ 1, \_, \_, \_, \_, 4 ]

Die\_B = [ 1, \_, \_, \_, \_, 8 ]

As the n(Sum = 2) = 1 and for n(Sum = 12) = 1 which is the only possible way the above conditions are satisfied.

We build on from here:

So the original dice had the following configuration:

Where Die\_A = [1, 2, 3, 4, 5, 6] and Die\_B = [1, 2, 3, 4, 5, 6]

| faces | **1** | **2** | **3** | **4** | **5** | **6** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | 2 | 3 | 4 | 5 | 6 | 7 |
| **2** | 3 | 4 | 5 | 6 | 7 | 8 |
| **3** | 4 | 5 | 6 | 7 | 8 | 9 |
| **4** | 5 | 6 | 7 | 8 | 9 | 10 |
| **5** | 6 | 7 | 8 | 9 | 10 | 11 |
| **6** | 7 | 8 | 9 | 10 | 11 | 12 |

So after the faces of the dice get wiped and we have to rearrange the dots so that they meet the given conditions of keeping the probabilities of the sums unchanged.

The given conditions were:

* Die\_A cannot have more than 4 spots on a face i.e., the highest number of dots on a face on Die\_A is not greater than 4 ( Die\_A[x] <= 4 )
* Die\_A can have repeating faces
* Die B can have as many spots on a face as necessary i.e. even more than 6

Let’s go back to the assumption I made for the new dice and try to arrange the dots on the new dice:

I started by initially setting the dice as:

Die\_A = [1, \_, \_, \_, \_, 4] and Die\_B = [1, \_, \_, \_, \_, 8]

The only way I see possible to get a solution to this problem is by brute forcing the solution, where we need to generate all the possible configurations from the above start condition and find the combination which gives us the same distribution of sums and meets all the conditions listed.

I made use of two functions all\_dice() and all\_pairs(dice) in which:

**all\_dice() →** in this function I use a list to store all the combinations that are possible for the dice. I use list comprehension to create the list where [1, \_, \_, \_, \_, \_] the first face of the die is fixed to 1 and all the other faces are tried within the range of 2 through 8. This is because we know that the sum of 2 is possible with only one combination which is 1 + 1, so all dice must have 1 as a face. From this function we get a list of lists.

**all\_pairs() →** in this function again I make use of list comprehension to make a list of tuples this time where, each tuple is a pair of dice and I use the condition A <= B to include the pairs as in the above assumption you can see that the New\_Die\_A cannot be greater than New\_Die\_B as the maximum number allowed on Die\_A is 4 so we can eliminate the combinations of (B, A) just to make it easier.

**distribution\_of\_sums() →** in this function the inputs are just the original pair of dice and the output for this function is a sorted list of sums for all combinations when the dice are rolled.

(Note: we use a list here and not a set as we need to keep track of the frequency of the sums)

After I’ve defined these functions within the **undoom\_dice()** function which takes as parameters the two original dice.

In the **undoom\_dice()** function I make use of two variables which help in checking the conditions, they are:

*standard\_pair* → this is a tuple of Die\_A and Die\_B i.e., (Die\_A, Die\_B)

*standard\_sums* → this variable stores the sums list / the distributions using the method distribution\_of\_sums(Die\_A, Die\_B)

Then we proceed to the main part of the program where the new pair gets generated. The *new\_pair* variable is a list of tuples created with the following conditions:

All pairs that are generated using **all\_pairs(all\_dice())** are checked for the distributions they produce using **distribution\_of\_sums(A, B)** == *standard\_sums* (keep in mind that standard\_sums contains the distribution we want to remain unchanged) and another check to see if the pair is or is not equal to the original dice pair, as we want a configuration other than the original one.

Finally, the function returns the new pair which is the only pair that can be formed with the given conditions.

Here is the logic for the distributions of the new configuration:

To generate the new dice we will be using the distributions we found out for the sums above, as the conditions require for these distributions to remain unchanged.

| faces | **1** | **2** | **2** | **3** | **3** | **4** |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | 2 | 3 | 3 | 4 | 4 | 5 |
| **3** | 4 | 5 | 5 | 6 | 6 | 7 |
| **4** | 5 | 6 | 6 | 7 | 7 | 8 |
| **5** | 6 | 7 | 7 | 8 | 8 | 9 |
| **6** | 7 | 8 | 8 | 9 | 9 | 10 |
| **8** | 9 | 10 | 10 | 11 | 11 | 12 |

This is a good place to start from as the sum=2 and sum=12 have appeared once, now to ensure that they do not occur again we cannot have any more faces with 1; also we cannot place any more 4 numbered faces on Die\_A as 12 will occur more than once in that way.

Next up, we can add the dots to the remaining faces moving forward in a similar fashion by keeping track of the distributions.

From the new table we can see that with the following configuration of dice:  
New\_Die\_A = [1, 2, 2, 3, 3, 4] and New\_Die\_B = [1, 3, 4, 5, 6, 8]

All the requirements are met and the distributions remain unchanged.