Student Name : **Sujith Aksay .M**

Roll Number : **200701263**

Department : B.E Computer Science and Engineering

Year : 4th Year

Email : 200701263@rajalakshmi.edu.in , sujithaksay21@gmail.com

College Name : Rajalakshmi Engineering College , Thandalam , Chennai.

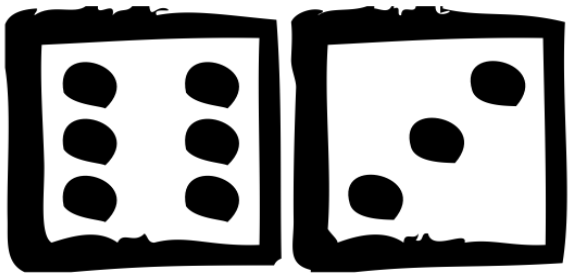
GitHub Link for assessment : https://github.com/sujithaksay/Doomed-Dice/tree/main

SECURIN INDIA ASSESSMENT

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 one 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.

ANSWER FOR PART – A : (Python code is given after the logic)

Logic to the Problem Statement :

The problem statement is to calculate the total combinations that are possible by rolling the dice , distribution of all possible combinations that can be obtained when rolling both Die A and Die B together and the Probability of all Possible Sums occurring among the number of combinations from (2).

1) The total number of combinations can be calculated by using the formula:

= (Number of faces of a die)^(Number of dice used)

This is nothing but Number of faces of a die raised to the power of value of number of dice used .

The logic to this problem is given by a few steps , First initialize the number of faces in a dice as 6 in a variable '**no_of_faces_in_a_die'** (as a die has 6 faces as fixed value) . Get the number of dice used as the input from the user and store it in a variable '**no_of_dice'.** By using a user defined function '**def combinations(x)'** initialize a local variable '**a'** equal to 6 (no of faces in a die). Using a for loop calculate the total combinations by multiplying 6 with '**a**' variable and storing it in the '**a'** variable. The for loop will run x-1 times . where x is the no of dice used and returns the value of **'a'**. For Example if no of dice used is 2 , then x value is 2 , so that the for loop will run x-1 time which is equal to 2-1 = 1 time . So 6 is multiplied once with variable 'a' which has a value 6, which becomes as 36 and it gets stored in the **'a'** variable . The 36 value is returned and printed (**NOTE : The user defined function is used to calculate the combinations because it mentioned not to use any library functions , Otherwise pow() built-in function can also be used to optimize the code.**)

Steps for calculating total combinations :

1) Initialize the value of no of faces and number of dice used.
2) Pass the number of dice used value to the function.
3) The for loop iterates x-1 times where x is the number of dice used value and multiplies the a variable with 6 and returns the value.

2) Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together.

The logic to this problem is also given by using a user defined function **'def print_dice_combinations(num_dice , current_combination)'** where '**num_dice**' is the number of dice used. The number of dice used is taken as input from the user and passed to this variable '**num_dice**'. The variable **'current_combination'** is an empty list passed to the function. Here a recursive and a backtracking algorithm is used in order to print the combinations of the dice. First an initial call is made **print_dice_combinations(2, [])** is called with **num_dice = 2** and an empty **current_combination ([]).** Then in the first iteration, the function enters a loop from 1 to 6 (inclusive) representing the possible values on a single die. For each value (let's say 1), it calls itself with num_dice - 1 **(i.e., print_dice_combinations(1, [1]))**. It appends the current die value **(1)** to **current_combination**, making it **[1].** Likewise the second iteration takes places and makes it as **[1,1].** The base case is reached when **num_dice** becomes 0. In this case, it prints the current combination [1, 1]. Now backtracking is done and all other combinations is also printed .

Steps for displaying all combinations :

Initial Call:

print_dice_combinations(2, []) is called with num_dice = 2 and an empty current_combination ([]).

First Iteration (First Die):

- The function enters a loop from 1 to 6 (inclusive) representing the possible values on a single die.
- For each value (let's say 1), it calls itself with num_dice - 1 (i.e., print_dice_combinations(1, [1])).
- It appends the current die value (1) to current_combination, making it [1].

Second Iteration (Second Die):

- Within the recursive call (print_dice_combinations(1, [1])), it enters the loop from 1 to 6 again for the second die.
- For each value (let's say 1), it calls itself with num_dice - 1 (i.e., print_dice_combinations(0, [1, 1])).

- It appends the current die value (1) to current_combination, making it
- [1, 1].

Base Case Reached:

The base case is reached when num_dice becomes 0. In this case, it prints the current combination [1, 1].

Backtracking:

Since the recursive call print_dice_combinations(0, [1, 1]) has finished, the function backtracks to the previous state (print_dice_combinations(1, [1])).
It continues the loop for the second die with the next value (2).

Recursive Calls and Backtracking:

The function continues this process, exploring all possible combinations for the second die for each value of the first die.
After exploring all combinations for the second die with the first die set to 1, it backtracks to the state with the first die set to 2 and repeats the process.

Output:

The function prints all possible combinations of dice values for the given number of dice.

3) Calculating Probability of all Possible Sums occurring among the number of combinations from (2).

The logic to this is given b y using a dictionary in python . First the key value is the sum of the combinations of dice value and the probability is given dividing the number of occurences of that sum divided by total number of combinations. For example for sum 3 the combinations are [1,2] and [2,1] , so 2 occurences for sum 3 . Therefore the probability is determined by 2 divided by the total combinations which is 36. So 2/36 is equal to 0.0555555556 . Liewise the unique sum value is taken as key and occurences are taken using count() function and it is divided by total and stored in the dictionary .

Steps for calculating Probability of all Possible Sums occurring among the number of combinations from (2).

- Initialize and define a list **total_sum** for finding all the sums with

repeating values of all combinations . [a + b for a in DieA for b in DieB].

- Using a dictionary each key value is given with a probability where key value is the sum of the each combinations. {sum_val: total_sum.count(sum_val) / len(total_sum) for sum_val in set(total_sum)}
- Finally print the dictionary using a for loop by accessing the dictionary by using its key value from 2 to 12 .

NOTE : All the above three problems statements are solved using a single python program by using the above logic

CODE FOR PART A : Coding done in Python using VS CODE

```
def combinations(x):
    a=6
    for i in range(1,x,1):
        a=a*6
    return a

def print_dice_combinations(num_dice, current_combination):
    if num_dice == 0:
        print(current_combination)
    else:
        for i in range(1, 7):
            print_dice_combinations(num_dice - 1, current_combination + [i])

#no_of_faces_in_a_die is the variable that represents the number of faces in a die . The Number of faces in a die is 6 . So no = 6

no_of_faces_in_a_die = 6
DieA = [1,2,3,4,5,6]
DieB = [1,2,3,4,5,6]

print("The number of spots or value on each faces of the dice is given as follows :")
print("\n[1,2,3,4,5,6]")
print("\nThe number of faces in a die = ",no_of_faces_in_a_die)

#no_of_dice is the variable that represents the number of dice used

no_of_dice = int(input("\nEnter the number of dice used :"))

print("\nThe number of dice used = ",no_of_dice)
```

```
print("\nThe Formula for the calculation of total combinations possible is given
by :")
print("\n(Number   of   faces   in   a   die)^(Number   of   dice   used)   i.e
no_of_faces_in_a_die^no_of_dice")

total_no_of_combinations = combinations(no_of_dice)

print("\nThe     total     number     of     possible     combinations     =
",total_no_of_combinations)
print("\nAll possible combinations are :")
print_dice_combinations(no_of_dice, [])

print("\nProbabilities are :")

total_sum = [a + b for a in DieA for b in DieB]
probabilities = {sum_val: total_sum.count(sum_val) / len(total_sum) for sum_val
in set(total_sum)}
for i in range(2,12+1):
    print("The Probability of Sum ",i,"=",probabilities[i])
```

INPUT AND OUTPUT :

Here number of dices used is the only input from the user .

```
The number of spots or value on each faces of the dice is given as follows :

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

The number of faces in a die =  6

Enter the number of dice used :2

The number of dice used =  2

The Formula for the calculation of total combinations possible is given by :

(Number of faces in a die)^(Number of dice used) i.e no_of_faces_in_a_die^no_of_dice

The total number of possible combinations =  36
```

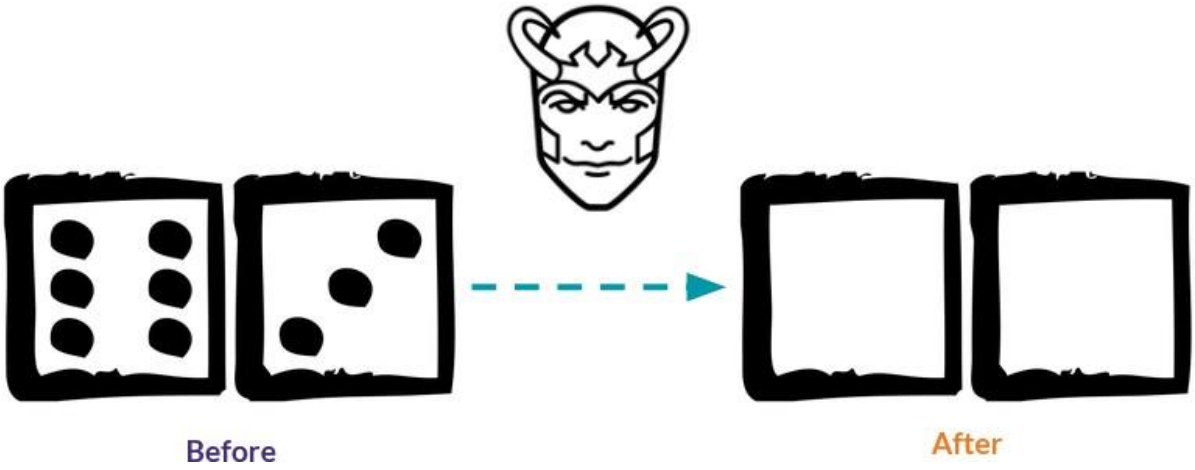
```
All possible combinations are :
[1, 1]
[1, 2]
[1, 3]
[1, 4]
[1, 5]
[1, 6]
[2, 1]
[2, 2]
[2, 3]
[2, 4]
[2, 5]
[2, 6]
[3, 1]
[3, 2]
[3, 3]
[3, 4]
[3, 5]
[3, 6]
[4, 1]
[4, 2]
[4, 3]
[4, 4]
[4, 5]
[4, 6]
[5, 1]
[5, 2]
[5, 3]
[5, 4]
[5, 5]
[5, 6]
[6, 1]
[6, 2]
[6, 3]
[6, 4]
[6, 5]
[6, 6]
```

```
Probabilities are :
The Probability of Sum  2 = 0.02777777777777776
The Probability of Sum  3 = 0.05555555555555555
The Probability of Sum  4 = 0.08333333333333333
The Probability of Sum  5 = 0.1111111111111111
The Probability of Sum  6 = 0.1388888888888889
The Probability of Sum  7 = 0.16666666666666666
The Probability of Sum  8 = 0.1388888888888889
The Probability of Sum  9 = 0.1111111111111111
The Probability of Sum  10 = 0.08333333333333333
The Probability of Sum  11 = 0.05555555555555555
The Probability of Sum  12 = 0.02777777777777776
```

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 (Youlove 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`

ANSWER FOR PART – B :

1) This method can be solved using combination method with logic:

METHOD :

Solving this problem by applying the logic of values on the faces and probability distribution .

LOGIC for this :

If we look at the problem's combinations for the lowest value that is 2 and the highest value that is 12,

Number of occurences of 2 = 1 time = [1,1]

Number of occurences of 12 = 1 time = [4,8]

Since 2 comes only once with a combination of [1,1] both the new dice values New_Die_A and New_Die_B must contain value 1 . Also the sum 12 occurs only once with a combination of 4+8=12 that is [4,8]. Therefore the New_Die_A must contain 4 only once and New_Die_B must contain 8 only once. 4 must be present in New_Die_A only once because if there are more than one 4 in New_Die_A then the occurrence of sum 12 will occur more than one time. Also 8 must be present in New_Die_B only once because if there are more than one 8 in New_Die_B then the occurrence of sum 12 will occur more than one time. So the initial dice values are

New_Die_A = [1, 4, a1, a2, a3, a4]

New_Die_B = [1, 8, b1, b2, b3, b4]

We need to find the values of a1 , a2 , a3 , a4 and b1 , b2 , b3 , b4 by combinations method.

The variables a1 , a2 , a3 , a4 will be repetitive combinations of 2 and 3 . That is a1 , a2 , a3 , a4 will be either 2 or 3.

So the combinations for this variables a1 , a2 , a3 , a4 are given as :

2,2,2,2

3,2,2,2

3,3,2,2

3,3,3,2

3,3,3,3

Also b1 ,b2 , b3 , b4 will be a combinations of any four values within 2,3,4,5,6,7 because the New_Die_B will have elements which are distinct . So totally there will be only 15 combinations. Eg : [4,5,6,7]….

Trying out the 15 combinations with the above 5 combinations may lead to 75 different combinations.

So therefore upcoming repetitive iterations using for loop…the combination 2,2,3,3 for a1, a2, a3, a4 with the combination 3,4,5,6 for b1 ,b2, b3, b4 will be the perfect answer for this problem

Therefore the answer is :

New_Die_A = [1,2,2,3,3,4]
New_Die_B = [1,3,4,5,6,8]

Logic for the code :

First get the Dice values from the users as user input for both the variables **Die_A** and **Die_B** using a for loop .

Then create two user defined functions **def calculate_sums(die_a, die_b)** and **def undoom_dice(Die_A, Die_B) .**

The **calculate_sums()** function is used to calculate the sum of combinations of the dice values i.e [2,3,4,5,6,7,8,9,10,11,12] with repetition.

The **original_sums** variables is used to take only the unique sum values.

Create a dictionary named as **original_probabilities** to store the probabilities with the sum as the key and probability distribution as the key value.

Now the **New_Die_A** and **New_Die_B** are to be calculated with combinations . As we said **New_Die_A** will have only one 1 and only one 4 in it for sure and **New_Die_B** will contain 1 once and 8 once. The remaining four values of both **New_Die_A** and **New_Die_B** must be found .

First **New_Die_A** is found using these five combinations **[2,2,2,2], [3,2,2,2],[3,3,2,2],[3,3,3,2],[3,3,3,3]** for those four variables a1,a2,a3,a4 using nested for loops .

The **New_Die_B** is found by checking the each probabilistic values with 15 more combinations . Likewise the loops are run for 75 times and checked with the probabilistic distribution variable **original_probability** which finally detects the values as

New_Die_A = [1, 2, 2, 3, 3, 4]

New_Die_B = [1, 3, 4, 5, 6, 8]

CODE :

```
def calculate_sums(die_a, die_b):
    return [a + b for a in die_a for b in die_b]

def undoom_dice(Die_A, Die_B):
    # Calculate original probabilities
    original_sums = calculate_sums(Die_A, Die_B)
       original_probabilities   =   {sum_val:   original_sums.count(sum_val)   /
len(original_sums) for sum_val in set(original_sums)}

    # Initialize new dice configurations
    New_Die_A = [1,4,0,0,0,0]
    New_Die_B = [1,8,0,0,0,0]
    a1,a2,a3,a4=2,2,2,2
    for i in range(0,5):
        if(i==1):
            a1+=1
        if(i==2):
            a2+=1
        if(i==3):
            a3+=1
        if(i==4):
            a4+=1
        New_Die_A[2]=a1
        New_Die_A[3]=a2
        New_Die_A[4]=a3
        New_Die_A[5]=a4
        for x in range(2,8):
            for y in range(2,8):
                C = [2,3,4,5,6,7]
                if(x==y):
                    continue
                else:
                     D = C
                     D.remove(x)
                     D.remove(y)
                     New_Die_B[2] = D[0]
                     New_Die_B[3] = D[1]
                     New_Die_B[4] = D[2]
                     New_Die_B[5] = D[3]
```

```
                    new_sums = calculate_sums(New_Die_A, New_Die_B)
                      new_probabilities = {sum_val: new_sums.count(sum_val) /
len(new_sums) for sum_val in set(new_sums)}
                    # Check if the probabilities match
                    if new_probabilities == original_probabilities:
                        return New_Die_A, New_Die_B
    return [], []

# Example usage
Die_A = []
Die_B = Die_A
print("Enter the Die face values for Die A and Die B :")
for z in range(0,6):
    Die_A.append(int(input()))
Die_B = Die_A
print("Die A : ",Die_A)
print("Die B : ",Die_B)
print("\n")
New_Die_A, New_Die_B = undoom_dice(Die_A, Die_B)
print("New_Die_A:", New_Die_A)
print("\nNew_Die_B:", New_Die_B)
```

INPUT AND OUTPUT :

1,2,3,4,5,6 is the input to the Dice values A and B.

```
Enter the Die face values for Die A and Die B :
1
2
3
4
5
6
Die A :  [1, 2, 3, 4, 5, 6]
Die B :  [1, 2, 3, 4, 5, 6]


New_Die_A: [1, 4, 3, 3, 2, 2]

New_Die_B: [1, 8, 3, 4, 5, 6]
```