**SECURIN ASSIGNMENT**

**PROBLEM STATEMENT: THE DOOMED DICE CHALLENGE**

GIVEN:

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

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

**PART-A:**

1. How many total combinations are possible?

**Logic:**

When rolling two dice, each die has 6 sides, so there are 6 possible outcomes for each die. To find the total combinations, we multiply the number of outcomes for each die. Since there are 6 outcomes for each die and we have 2 dice, the total combinations would be: 6 × 6 = 36 combinations.

**Code:**

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

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

no\_of\_sides\_A=len(Die\_A)

no\_of\_sides\_B=len(Die\_B)

total\_combinations = no\_of\_sides\_A\*no\_of\_sides\_B

print("Total combinations possible:", total\_combinations)

**Output:**



**Explaination:**

Step 1: Declare 2 arrays Die\_A and Die\_B with dice faces.

Step 2: Calculate the length of both the arrays to know the number of sides of each dice(i.e, 6).

Step 3: To find total combinations, multiply no\_of\_sides\_A and no\_of\_sides\_B and assign it to a variable total\_combinations.

Step 4: print the total combinations.

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.

**Math concept used:**

When rolling two six-sided dice (Die A and Die B) together, each die has 6 faces, numbered from 1 to 6. To calculate the distribution of all possible combinations, we need to consider all possible pairs of outcomes when rolling both dice.

Since each die has 6 faces, there are a total of 6×6=36 possible combinations. We will represent these combinations in a 6x6 matrix, where each cell in the matrix represents a unique combination of outcomes.

**Logic:**

Step 1: Import the itertools module which provides functions for working with iterators.

Step 2: Two lists Die\_A and Die\_B are declared with the 6 faces of the dice (1,2,3,4,5,6).

Step 3: Create a 6 x 6 matrix and initialize it with zeros. Assign it to variable matrix.

Step 4: Use 2 nested for loops for iterating through Die\_A and Die\_B to populate the matrix with tuples of form (val\_1,val\_2) , representing all possible combinations.

Step 5: print the 2-dimensional matrix in the matrix form with necessary formatting required by iterating through each row and through column / value of each row.

**Code:**

import itertools

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

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

matrix = [[0] \* 6 for \_ in range(6)]

for i, val\_1 in enumerate(Die\_A):

for j, val\_2 in enumerate(Die\_B):

matrix[i][j] = (val\_1, val\_2)

count=1

print("[",end=" ")

for row in matrix:

print("[",end=" ")

for value in row:

count=count+1

print(value,end=" ")

if(count !=37):

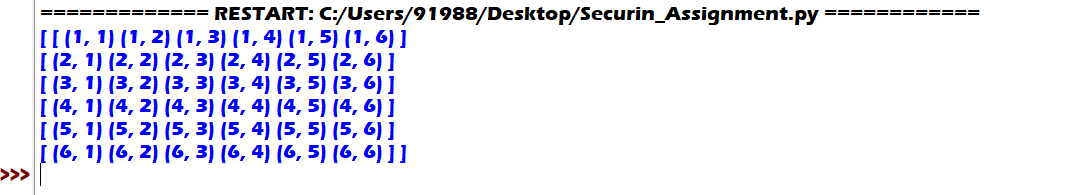
print("]")

else:

print("]",end=" ")

print("]")

**Output:**



**Explaination:**

In order to find the all possible combinations that can be obtained when rolling 2 dice, we declare 2 lists with values. Then a zero matrix is created which is later populated with tuples which represents all possible combinations. Once the matrix is created, I displayed it with necessary formatting as required to display 2D matrix. As output I would get 6 x 6 matrix with 6 rows and 6 columns displaying all possible combinations in tuple format.

3. Calculate the probability of all possible sums occurring among the number of combinations from (2).

**Logic:**

Step 1: import the itertools library.

Step2: Initialize the 2 lists Die\_A and Die\_B.

Step 3: Generate all possible combinations using product() function of itertools and store it in variable combinations.

Step 4: Calculate the total no of combinations using a len() function and store it in a variable total\_Combinations which will be used further.

Step 5: Initialize a dictionary to store no of times each sum vaue appears in a variable sumValueCount.

Step 6: Count the occurances of each sum and store it in above created dictionary.

Step 7: Now calculate the probability of each sum occurring using the formula probability[event]=no of times event occurs / total no of events. Store all probability values in a dictionary probabilities.

Step 8: Print the probabilities using the format function.

**Code:**

import itertools

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

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

combinations = list(itertools.product(Die\_A, Die\_B))

total\_combinations = len(combinations)

sumValueCount = {}

for combo in combinations:

sum\_val = sum(combo)

sumValueCount[sum\_val] = sumValueCount.get(sum\_val, 0) + 1

probabilities = {}

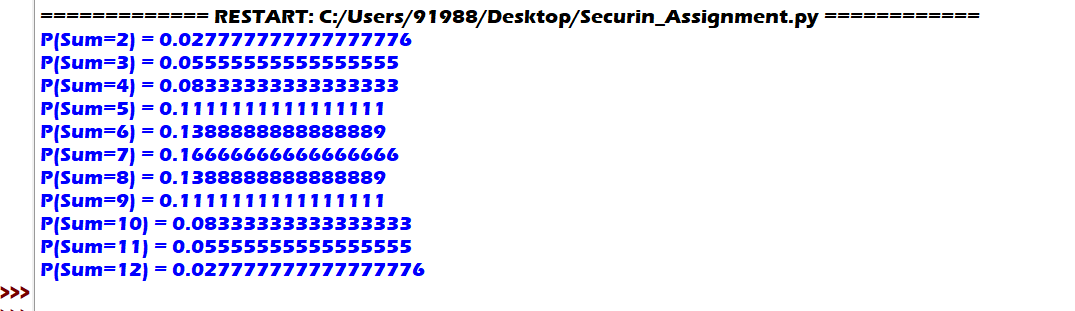
for sum\_val, count in sumValueCount.items():

probabilities[sum\_val] = count / total\_combinations

for sum\_val, prob in probabilities.items():

print(f"P(Sum={sum\_val}) = {prob}")

**Output:**



**Explaination:**

To calculate and display probabilities of all possible sums, I have used a dictionary. I will create a dictionary that stores the “sum” as key and “no of times it occurs” as value. Later I will create another dictionary which replaces the “count” values of the previous dictionary with corresponding probabilities. Once I have the all required probabilities, I will display them using format function in python.

**PART B:**

Conditions to be met:

* 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.

**Logic:**

Step 1: Define a function undoom\_dice() and initialize two empty lists newDie\_A and newDie\_B in it.

Step 2: Multiply lengths of Die\_A and Die\_B to calculate total no of combinations and store it in variable all\_combinations.

Step 3: Use product() function of itertools library to determine the sum frequency of all possible combinations.

Step 4: Use counter() function to count the no of times each sum value occurs.

Step 5: Now among the sums find the maximum frequency.

Step 6: To reattach spots to Die\_A , iterate over it. Is spot count is less than 4 then append it else set it to 4.

Step 7: reassign Die\_B to newDie\_B as newDie\_B can have any no of spots on a face.

Step 8: return the newDie\_A and newDie\_B using the return statement.

Step 9: After completing the function, call the function by passing Die\_A and Die\_B as arguments. Store the result in 2 variables and print them.

**Code:**

import itertools

from collections import Counter

def undoom\_dice(Die\_A, Die\_B):

newDie\_A = []

newDie\_B = []

all\_combinations = len(Die\_A) \* len(Die\_B)

sum\_value\_frequencies = []

for pair in itertools.product(Die\_A, Die\_B):

sum\_value\_frequencies+=[sum(pair)]

sum\_value\_counter = Counter(sum\_value\_frequencies)

max\_sum\_frequency = max(sum\_freq for sum\_freq in sum\_value\_counter.values())

for dot in Die\_A:

if dot <= 4:

newDie\_A.append(dot)

else:

newDie\_A.append(4)

newDie\_B = Die\_B

return newDie\_A, newDie\_B

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

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

newDie\_A, newDie\_B = undoom\_dice(Die\_A, Die\_B)

print(“After dooming dices, the new dices: “)

print("New Die A:", newDie\_A)

print("New Die B:", newDie\_B)

**Output:**



**Explaination:**

To reconstruct the dice after dooming, we need to consider all conditions. We’ll first define a function that returns the new dices values. We first calculate the sum of each pair of values from both original dices to determine the sum frequency of all possible combinations. I’ll use Counter() to count the no of times each sum value occurs. To reattach spots to newDie\_A, I will iterate through the spots. If spot is less than 4, append it else append “4”. To reattach spots to newDie\_B just keep the original as newDie\_B can have any no of spots on a given face. Also te code satisfies the condition that the probability of obtaining sums remains the same as previous. Return the new dice values to the caller line and print the output.