SecureIn

Problem statement:

Given that there are essentially two dice, dice A and dice B can be taken as [1,2,3,4,5,6] in the grid matrix.

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

To find the total number of combinations, we can simply square the number of possibilities for one die. This is because each roll of one die can be paired with any of the 6 possibilities of the other die.

Total combinations = Possibilities for one die \* Possibilities for the other die

= 6 \* 6

= 36

So, there are 36 total combinations possible.

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!

The function generateHelper creates a list of lists, where each inner list represents a combination of rolls for two six-sided dice (Die A and Die B).

It iterates through all possible values for Die A and Die B, appending each combination as a pair [Die A value, Die B value] to the list.

Finally, it returns the list containing all combinations. When executed in the main function, it prints out all possible combinations of rolls for Die A and Die B.

Code :

*def* generateHelper():

  dieA = range(1, 7)

  dieB = range(1, 7)

  combinations\_list = []

  for i in dieA:

    temp = []

    for j in dieB:

      temp.append([i, j])

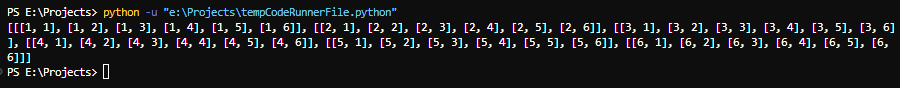
    combinations\_list.append(temp)

    return (combinations\_list)

*#main function*

generateHelper()

Output :



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.

To calculate the probability of all possible sums from 2 to 12, we need to consider all combinations of values from both dice.

The minimum sum occurs when both dice show a value of 1, and the maximum sum occurs when both dice show a value of 6.

We will iterate through all possible combinations of values for both dice, compute the sum, and then calculate the probability for each sum.

*def* calculate(dieA, dieB):

    for i in dieA:

        for j in dieB:

            s = i + j

            sums\_list.append(s)

    counts = {}

    prob = {}

    for i in sums\_list:

        counts[i] = counts.get(i, 0) + 1

    for i in range(2, 13):

        count\_i = counts.get(i, 0)

        prob\_i = count\_i / len(sums\_list)

        prob[i] = {"count": count\_i, "prob": str(round(prob\_i, 3))}

    return prob

if \_\_name\_\_ == "\_\_main\_\_":

    dieA = range(1, 7)

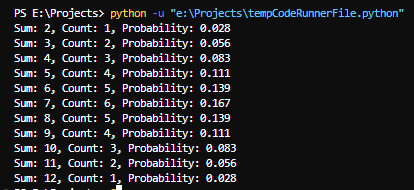
    dieB = range(1, 7)

    sums\_list = []

    probabilities = calculate(dieA, dieB)

    for key, value in probabilities.items():

        print(*f*"Sum: {key}, Count: {value['count']}, Probability: {value['prob']}")

Output :  


**Part B**

Everything was running fine but suddenly The cunning Norse god Loki materialised out of nowhere.

For his own amusement, Loki destroys all of the "Spots" on your dice, dooming it.



To reattach the spots we have some rules

**● 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.**

* The function takes two lists, DieA and DieB, which represent the original values on the two dice.
* It creates a list called combos that contains all possible pairs of values that can occur when rolling the two dice together. For example, if we roll a 1 on the first die and a 2 on the second die, the pair (1, 2) is included in the combos list.
* The combos list is then sorted based on the sum of the two values in each pair. For example, pairs like (1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1) will be sorted together because their sum is 2, followed by pairs with a sum of 3, and so on.
* Two new empty lists, newDieA and newDieB, are created to store the new values for the two dice.
* The function then goes through the sorted combos list one pair at a time:
* If the first value in the pair is less than or equal to 4, and we haven't yet assigned values to all 6 faces of the dice, the function assigns the first value to the corresponding position in newDieA and the second value to the corresponding position in newDieB.
* If the first value in the pair is greater than 4, and we haven't yet assigned values to all 6 faces of the dice, the function assigns the first value to the corresponding position in newDieB and the second value to the corresponding position in newDieA.
* After going through all the pairs in the combos list, the function returns the newDieA and newDieB lists, which represent the new values on the two dice.

Code:

*def* undoomDice(DieA, DieB):

    combos = list(product(range(1, 7), repeat=2))

    combos.sort(key=*lambda* x: sum(x))

    newDieA = [0] \* 6

    newDieB = [0] \* 6

    for i, combo in enumerate(combos):

        if combo[0] <= 4:

            if i < 6:

                newDieA[i] = combo[0]

                newDieB[i] = combo[1]

        else:

            if i < 6:

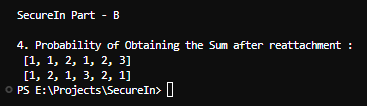
                newDieA[i] = combo[1]

                newDieB[i] = combo[0]

    return newDieA, newDieB

print("4. Probability of Obtaining the Sum after reattachment :","\n",newDieA,"\n",newDieB)

Output :



Final Code: Object Oriented Code

*'''*

*SecureIn*

*Problem Statement: The Doomed Dice Challenge*

*PART - A*

*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, and your turn is guided by the obtained sum.*

*Problem statement:*

*Given that there are essentially two dice, dice A and dice B can be taken as [1,2,3,4,5,6] in the grid matrix.*

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

*To find the total number of combinations, we can simply square the number of possibilities for one die. This is because each roll of one die can be paired with any of the 6 possibilities of the other die.*

*Total combinations = Possibilities for one die \* Possibilities for the other die*

*= 6 \* 6*

*= 36*

*So, there are 36 total combinations possible.*

*'''*

*# Import Libraries*

from itertools import product

from collections import Counter

*class* Dice:

*def* \_\_init\_\_(*self*, num\_sides=6):

*self*.num\_sides = num\_sides

*def* total\_combinations(*self*):

        return *self*.num\_sides \*\* 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!*

*The function distributionCombinations creates a list of lists, where each inner list represents a combination of rolls for two six-sided dice (Die A and Die B).*

*It iterates through all possible values for Die A and Die B, appending each combination as a pair [Die A value, Die B value] to the list.*

*Finally, it returns the list containing all combinations. When executed in the main function, it prints out all possible combinations of rolls for Die A and Die B.*

*'''*

*def* distribution\_combinations(*self*):

        dieA = range(1, 7)

        dieB = range(1, 7)

        combinations\_list = []

        for i in dieA:

            temp = []

            for j in dieB:

                temp.append([i, j])

            combinations\_list.append(temp)

        return combinations\_list

*'''*

*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.*

*To calculate the probability of all possible sums from 2 to 12, we need to consider all combinations of values from both dice.*

*The minimum sum occurs when both dice show a value of 1, and the maximum sum occurs when both dice show a value of 6.*

*We will iterate through all possible combinations of values for both dice, compute the sum, and then calculate the probability for each sum.*

*'''*

*def* probability\_sum(*self*):

        dieA = range(1, *self*.num\_sides + 1)

        dieB = range(1, *self*.num\_sides + 1)

        sums\_list = []

        for i in dieA:

            for j in dieB:

                s = i + j

                sums\_list.append(s)

        counts = {}

        prob = {}

        for i in sums\_list:

            counts[i] = counts.get(i, 0) + 1

        for i in range(2, 2 \* *self*.num\_sides + 1):

            count\_i = counts.get(i, 0)

            prob\_i = count\_i / len(sums\_list)

            prob[i] = {"count": count\_i, "prob": str(round(prob\_i, 3))}

        return prob

*''''*

*PART - B*

*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, then 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*

*'''*

*def* undoom\_dice(*self*):

        combos = list(product(range(1, *self*.num\_sides + 1), repeat=2))

        combos.sort(key=*lambda* x: sum(x))

        new\_die\_a = [0] \* *self*.num\_sides

        new\_die\_b = [0] \* *self*.num\_sides

        for i, combo in enumerate(combos):

            if combo[0] <= 4:

                if i < *self*.num\_sides:

                    new\_die\_a[i] = combo[0]

                    new\_die\_b[i] = combo[1]

            else:

                if i < *self*.num\_sides:

                    new\_die\_a[i] = combo[1]

                    new\_die\_b[i] = combo[0]

        return new\_die\_a, new\_die\_b

*#main function*

if \_\_name\_\_ == "\_\_main\_\_":

    dice = Dice(6)

    print("SecureIn Part - A", "\n")

    print("1. Total Combinations Possible by the Dice :", dice.total\_combinations(), "\n")

    print("2. Distribution of all possible combinations: ", dice.distribution\_combinations(), "\n")

    print("3. Probability of all Possible Sums occurring among the number of combinations:")

    probabilities = dice.probability\_sum()

    for key, value in probabilities.items():

        print(*f*"Sum: {key}, Count: {value['count']}, Probability: {value['prob']}")

    print()

    print("SecureIn Part - B", "\n")

    new\_die\_a, new\_die\_b = dice.undoom\_dice()

    print("4. Probability of Obtaining the Sum after reattachment :", "\n", new\_die\_a, "\n", new\_die\_b)