**THE DOOMED DICE CHALLENGE**

**GIVEN:** consider two six-sided faces namely die a and die b with faces numbered from 1 to 6. Both the dice are rolled together.

1. **HOW MANY TOTAL COMBINATIONS ARE POSSIBLE? SHOW THE MATH ALONG WITH THE CODE?**

The total number of combinations when two dice are rolled together can be calculated using the formula:

TotalCombinations=(NumberOfFaces)^(NumberOfDice)

For six-sided dice, each die has 6 faces.

NumberOfFaces=6

NumberOfDice=2

If two dice are rolled together,then

TotalCombinations=6^2

=36

**CODE:**

def calculate\_total\_combinations(number\_of\_faces, number\_of\_dice):

total\_combinations = number\_of\_faces \*\* number\_of\_dice

return total\_combinations

def main():

number\_of\_faces = 6

number\_of\_dice = 2

# Calculate total combinations

total\_combinations = calculate\_total\_combinations(number\_of\_faces, number\_of\_dice)

print("Total Combinations:", total\_combinations)

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

main()

**SCREENSHOT:**



1. **CALCULATE AND DISPLAY THE DISTRIBUTION OF ALL POSSIBLE COMBINATIONS THAT CAN BE OBTAINED WHEN ROLLING BOTH DIE A AND DIE B TOGETHER.**

NumberOfFaces=6

TotalCombinations=36

**DISTRIBUTION:**

In the context of rolling two six-sided dice, the distribution refers to the possible sums that can be obtained and their associated probabilities.

**CODE:**

def calculate\_combination\_distribution(sides):

distribution\_list = []

for i in range(1, sides + 1):

for j in range(1, sides + 1):

total = i + j

combination\_map = {

"Die A": i,

"Die B": j,

"Sum": total

}

distribution\_list.append(combination\_map)

return distribution\_list

def main():

sides = 6

# Calculate and display combination distribution

combination\_distribution = calculate\_combination\_distribution(sides)

print("Distribution:")

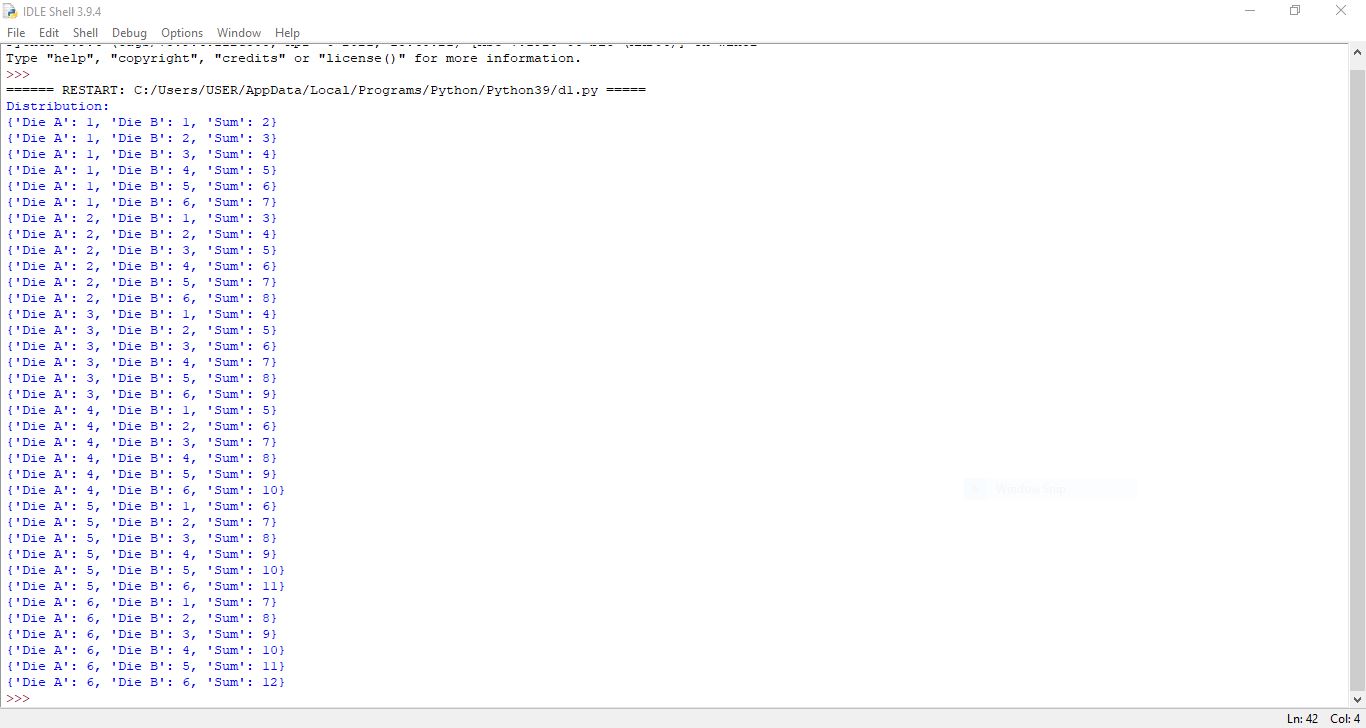
for combination\_map in combination\_distribution:

print(combination\_map)

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

main()

**SCREENSHOT:**



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.**

Probability is calculated by dividing the number of favorable outcomes by the total number of possible outcomes.

**CODE:**

def calculate\_sum\_probability(sides):

distribution = [0] \* 11

for i in range(1, sides + 1):

for j in range(1, sides + 1):

total = i + j

distribution[total - 2] += 1

probability\_map = {}

for i in range(2, 13):

probability\_map[i] = f"{distribution[i - 2]}/{36}"

return probability\_map

def main():

sides = 6

# Calculate and display sum probabilities

sum\_probability = calculate\_sum\_probability(sides)

print("Probability of Sums:")

for sum\_value, prob in sum\_probability.items():

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

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

main()

**SCREENSHOT:**

****

**FULL INPUT:**

def calculate\_total\_combinations(number\_of\_faces, number\_of\_dice):

total\_combinations = number\_of\_faces \*\* number\_of\_dice

return total\_combinations

def main():

number\_of\_faces = 6

number\_of\_dice = 2

# Calculate total combinations

total\_combinations = calculate\_total\_combinations(number\_of\_faces, number\_of\_dice)

print("Total Combinations:", total\_combinations)

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

main()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

def calculate\_combination\_distribution(sides):

distribution\_list = []

for i in range(1, sides + 1):

for j in range(1, sides + 1):

total = i + j

combination\_map = {

"Die A": i,

"Die B": j,

"Sum": total

}

distribution\_list.append(combination\_map)

return distribution\_list

def main():

sides = 6

# Calculate and display combination distribution

combination\_distribution = calculate\_combination\_distribution(sides)

print("Distribution:")

for combination\_map in combination\_distribution:

print(combination\_map)

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

main()

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

def calculate\_sum\_probability(sides):

distribution = [0] \* 11

for i in range(1, sides + 1):

for j in range(1, sides + 1):

total = i + j

distribution[total - 2] += 1

probability\_map = {}

for i in range(2, 13):

probability\_map[i] = f"{distribution[i - 2]}/{36}"

return probability\_map

def main():

sides = 6

# Calculate and display sum probabilities

sum\_probability = calculate\_sum\_probability(sides)

print("Probability of Sums:")

for sum\_value, prob in sum\_probability.items():

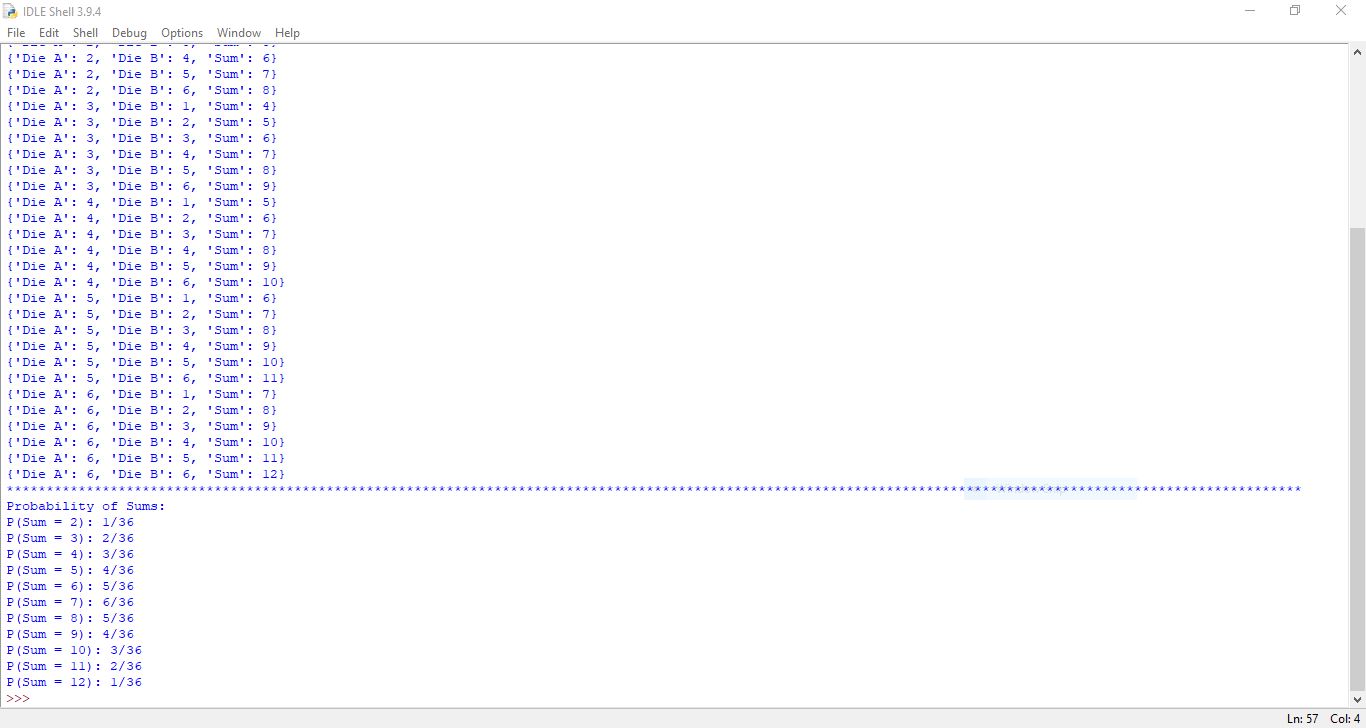
print(f"P(Sum = {sum\_value}): {prob}")

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

main()

**SCREENSHOT:**

****

****

**PART-B**

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.

CODE:

def calculate\_scaling\_factor(die\_a, die\_b):

sum\_a = sum(die\_a)

sum\_b = sum(die\_b)

return sum\_b / sum\_a

def generate\_alternative\_dice(length):

return [1, 2, 3, 4] \* (length // 4) + [1, 2, 3, 4][:length % 4]

def undoom\_dice(die\_a, die\_b):

target\_sum = sum(die\_b)

alternative\_die\_a = generate\_alternative\_dice(len(die\_a))

scaling\_factor = calculate\_scaling\_factor(alternative\_die\_a, die\_b)

new\_die\_a = [min(4, round(spots \* scaling\_factor)) for spots in alternative\_die\_a]

new\_die\_b = [min(6, spots) for spots in die\_b]

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

original\_dice = [1, 2, 3, 4, 5, 6]

new\_die\_a, new\_die\_b = undoom\_dice(original\_dice, original\_dice)

print("Modified Dice Values:")

print("New Die A:", new\_die\_a)

print("New Die B:", new\_die\_b)

OUTPUT SCREENSHOT:

