Part-A Code:

import numpy as np

# Number of faces on each die

num\_faces = 6

# Total combinations

total\_combinations = num\_faces \* num\_faces

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

# Number of faces on each die

num\_faces = 6

# Initialize a 6x6 matrix to hold the distribution

distribution\_matrix = np.zeros((num\_faces, num\_faces), dtype=int)

# Fill the matrix with counts of sums

for die\_a in range(1, num\_faces + 1):

    for die\_b in range(1, num\_faces + 1):

        sum\_ab = die\_a + die\_b

        distribution\_matrix[die\_a - 1][die\_b - 1] = sum\_ab

print("Distribution matrix of sums:")

print(distribution\_matrix)

# Initialize a dictionary to hold the counts of each sum

sum\_counts = {i: 0 for i in range(2, 13)}

# Calculate the counts for each sum

for die\_a in range(1, num\_faces + 1):

    for die\_b in range(1, num\_faces + 1):

        sum\_ab = die\_a + die\_b

        sum\_counts[sum\_ab] += 1

# Calculate probabilities

total\_combinations = num\_faces \* num\_faces

probabilities = {sum\_val: count / total\_combinations for sum\_val, count in sum\_counts.items()}

print("Sum counts:", sum\_counts)

print("Probabilities of each sum:")

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

    print(f"P(Sum = {sum\_val}) = {prob:.4f}")

import numpy as np

# Number of faces on each die

num\_faces = 6

# Total combinations

total\_combinations = num\_faces \* num\_faces

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

# Number of faces on each die

num\_faces = 6

# Initialize a 6x6 matrix to hold the distribution

distribution\_matrix = np.zeros((num\_faces, num\_faces), dtype=int)

# Fill the matrix with counts of sums

for die\_a in range(1, num\_faces + 1):

for die\_b in range(1, num\_faces + 1):

sum\_ab = die\_a + die\_b

distribution\_matrix[die\_a - 1][die\_b - 1] = sum\_ab

print("Distribution matrix of sums:")

print(distribution\_matrix)

# Initialize a dictionary to hold the counts of each sum

sum\_counts = {i: 0 for i in range(2, 13)}

# Calculate the counts for each sum

for die\_a in range(1, num\_faces + 1):

for die\_b in range(1, num\_faces + 1):

sum\_ab = die\_a + die\_b

sum\_counts[sum\_ab] += 1

# Calculate probabilities

total\_combinations = num\_faces \* num\_faces

probabilities = {sum\_val: count / total\_combinations for sum\_val, count in sum\_counts.items()}

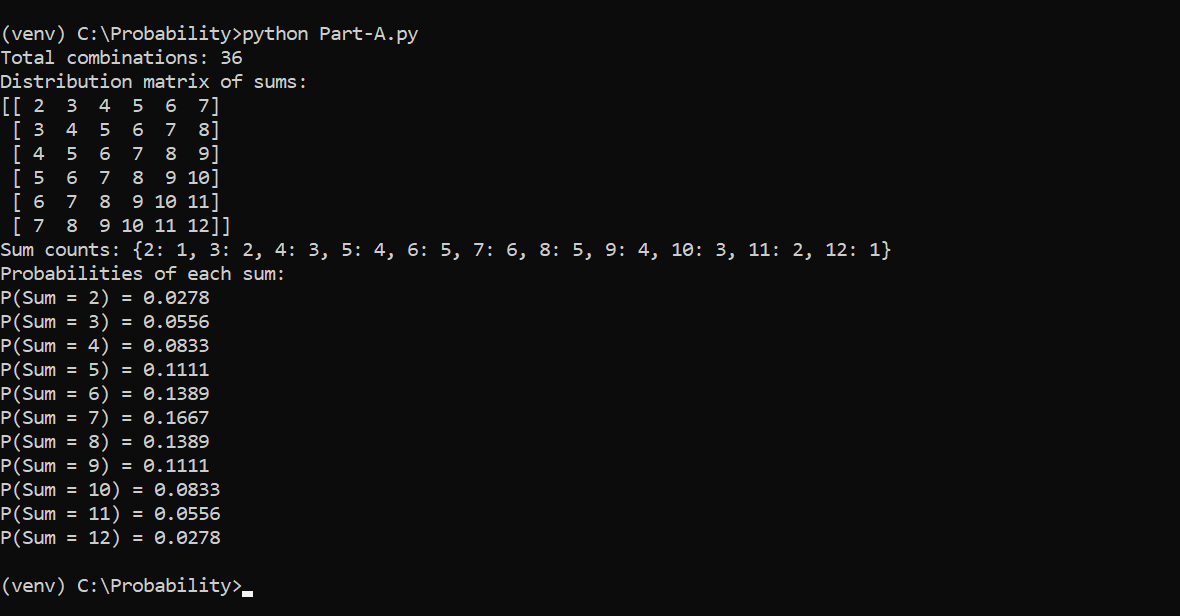
print("Sum counts:", sum\_counts)

print("Probabilities of each sum:")

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

print(f"P(Sum = {sum\_val}) = {prob:.4f}")

Output Screenshot:



Part B:

from collections import defaultdict

def calculate\_combinations(die\_a, die\_b):

    count = 0

    for i in die\_a:

        for j in die\_b:

            count += 1

    return count

def generate\_all\_combinations(die\_a, die\_b):

    combinations = []

    for i in die\_a:

        for j in die\_b:

            combinations.append([i, j])

    return combinations

def calculate\_probabilities(die\_a, die\_b):

    combinations = []

    for i in die\_a:

        for j in die\_b:

            combinations.append(i + j)

    count = defaultdict(int)

    probabilities = {}

    total\_combinations = len(combinations)

    for i in combinations:

        count[i] += 1

    for i in range(2, 13):

        count\_i = count.get(i, 0)

        probability\_i = count\_i / total\_combinations

        probabilities[i] = {"count": count\_i, "probability": round(probability\_i, 5)}

    return probabilities

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

    new\_die\_a = [min(i, 4) for i in die\_a]

    new\_die\_b = die\_b  # This function does not modify Die\_B, as there is no constraint on Die\_B

    return sorted(new\_die\_a), sorted(new\_die\_b)

def main():

    die\_a = range(1, 7)

    die\_b = range(1, 7)

    # Calculate original probabilities

    original\_probabilities = calculate\_probabilities(die\_a, die\_b)

    # Generate all possible combinations of the original dice

    all\_combinations = generate\_all\_combinations(die\_a, die\_b)

    # Transform dice according to the constraints

    new\_die\_a, new\_die\_b = undoom\_dice(list(die\_a), list(die\_b))

    # Calculate modified probabilities

    modified\_probabilities = calculate\_probabilities(new\_die\_a, new\_die\_b)

    # Print results

    print("Total Combinations:", calculate\_combinations(die\_a, die\_b))

    print("All Combinations:")

    for combination in all\_combinations:

        print(combination)

    print("\nOriginal Probabilities:")

    for sum\_combination, probability in original\_probabilities.items():

        print(f"Sum: {sum\_combination}, Count: {probability['count']}, Probability: {probability['probability']}")

    print("\nModified Probabilities:")

    for sum\_combination, probability in modified\_probabilities.items():

        print(f"Sum: {sum\_combination}, Count: {probability['count']}, Probability: {probability['probability']}")

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

    main()

from collections import defaultdict

def calculate\_combinations(die\_a, die\_b):

count = 0

for i in die\_a:

for j in die\_b:

count += 1

return count

def generate\_all\_combinations(die\_a, die\_b):

combinations = []

for i in die\_a:

for j in die\_b:

combinations.append([i, j])

return combinations

def calculate\_probabilities(die\_a, die\_b):

combinations = []

for i in die\_a:

for j in die\_b:

combinations.append(i + j)

count = defaultdict(int)

probabilities = {}

total\_combinations = len(combinations)

for i in combinations:

count[i] += 1

for i in range(2, 13):

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

probability\_i = count\_i / total\_combinations

probabilities[i] = {"count": count\_i, "probability": round(probability\_i, 5)}

return probabilities

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

new\_die\_a = [min(i, 4) for i in die\_a]

new\_die\_b = die\_b # This function does not modify Die\_B, as there is no constraint on Die\_B

return sorted(new\_die\_a), sorted(new\_die\_b)

def main():

die\_a = range(1, 7)

die\_b = range(1, 7)

# Calculate original probabilities

original\_probabilities = calculate\_probabilities(die\_a, die\_b)

# Generate all possible combinations of the original dice

all\_combinations = generate\_all\_combinations(die\_a, die\_b)

# Transform dice according to the constraints

new\_die\_a, new\_die\_b = undoom\_dice(list(die\_a), list(die\_b))

# Calculate modified probabilities

modified\_probabilities = calculate\_probabilities(new\_die\_a, new\_die\_b)

# Print results

print("Total Combinations:", calculate\_combinations(die\_a, die\_b))

print("All Combinations:")

for combination in all\_combinations:

print(combination)

print("\nOriginal Probabilities:")

for sum\_combination, probability in original\_probabilities.items():

print(f"Sum: {sum\_combination}, Count: {probability['count']}, Probability: {probability['probability']}")

print("\nModified Probabilities:")

for sum\_combination, probability in modified\_probabilities.items():

print(f"Sum: {sum\_combination}, Count: {probability['count']}, Probability: {probability['probability']}")

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

main()

Output:

