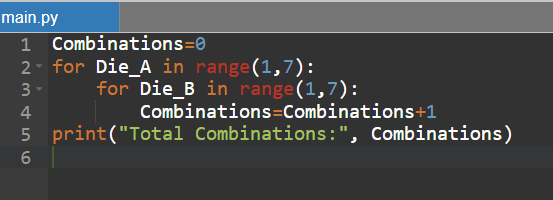
**EXPLANATION:**

**PART A**

**Question 1)**

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

**CODE:**



**EXPLANATION:**

* for Die\_A in range (1, 7): The outer loop iterates over the values of Die\_A from 1 to 6 (inclusive). In each iteration of the outer loop, the inner loop will run.
* for Die\_B in range (1, 7): The inner loop iterates over the values of Die\_B from 1 to 6 (inclusive). For each combination of Die\_A and Die\_B, the variable Combinations is incremented by 1.
* Combinations = Combinations + 1: Inside the inner loop, this line increments the Combinations variable by 1 for each combination of Die\_A and Die\_B.
* After both loops complete execution, the total number of combinations is printed using print ("Total Combinations:", Combinations).

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

**Question 2)**

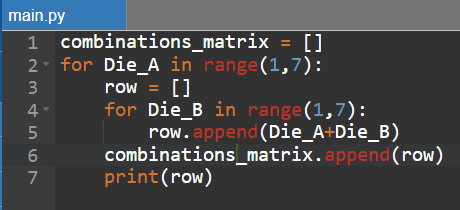
**Calculate and display the distribution of all possible combinations that can be.**

**obtained when rolling both Die A and Die B together. Show the math along with**

**the code!**

**Hint: A 6 x 6 Matrix.**

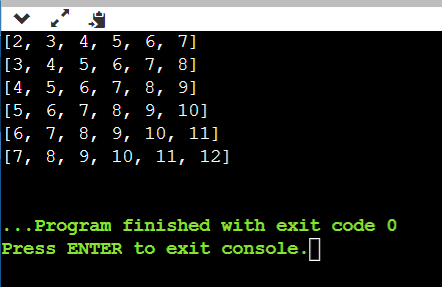
**CODE:**

****

**EXPLANATION:**

* The outer loop for Die\_A in range (1,7): iterates over possible values of the first die (Die\_A). The values of Die\_A range from 1 to 6, inclusive.
* row = []: Initializes an empty list called row to store the combinations for the current value of Die\_A.
* The inner loop for Die\_B in range (1,7): iterates over possible values of the second die (Die\_B). The values of Die\_B also range from 1 to 6, inclusive.
* row.append(Die\_A + Die\_B): Calculates the sum of the current values of Die\_A and Die\_B and appends the result to the row list. This represents one possible combination of the two dice.
* combinations\_matrix.append(row): Appends the row list (representing combinations for the current value of Die\_A) to the combinations\_matrix list.
* print(row): Prints the current row, showing the combinations for the current value of Die\_A.
* The outer loop continues, and the process is repeated for each possible value of Die\_A.

**OUTPUT:**

****

**Question 3)**

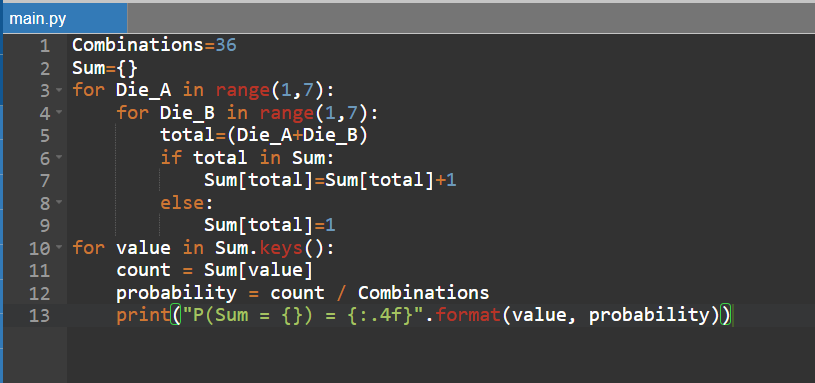
**Calculate the Probability of all Possible Sums occurring among the number of**

**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.**

**CODE:**

****

**EXPLANATION:**

* Combinations = 36: Initializes the total number of possible combinations when rolling two six-sided dice. There are 6 possible outcomes for the first die and 6 possible outcomes for the second die, so the total combinations are 6 \* 6 = 36.
* Sum = {}: Initializes an empty dictionary called Sum to store the counts of each possible sum.
* The outer loop for Die\_A in range (1,7): iterates over the values of the first die (Die\_A), and the inner loop for Die\_B in range(1,7): iterates over the values of the second die (Die\_B).
* total = (Die\_A + Die\_B): Calculates the sum of the values of the two dice for the current iteration.
* Checks if total is already a key in the Sum dictionary:
* If it is, increment the count associated with that sum.
* If it's not, add a new key with the sum and set its count to 1.
* After both loops complete, the code enters another loop for value in Sum.keys(): to iterate over the keys (sums) in the Sum dictionary.
* Inside this loop, it retrieves the count for each sum and calculates the probability of getting that sum by dividing the count by the total number of combinations (Combinations).
* Finally, it prints the probability for each sum with a formatted string, displaying the sum and its corresponding probability.

**OUTPUT:**

**A screenshot of a computer program

Description automatically generated**

**PART B**

**Question 1)**

**CODE:**

**A computer screen shot of a program

Description automatically generated**

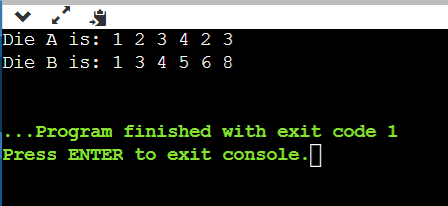
**A computer screen with text and numbers

Description automatically generated**

**EXPLANATION:**

* Die Setup: The initial setup of two dice, dieA and dieB, where dieA has values [1, 2, 3, 4, 0, 0] and dieB has values [1, 3, 4, 5, 6, 0]. The 0 values are placeholders.
* Target Distribution: The target dictionary represents the desired distribution of sums when rolling two six-sided dice.
* Validation Function (validate\_cur\_combination): This function checks if the current combination of dieA and dieB matches the target distribution. It iterates over all possible combinations of rolls and counts the occurrences of each sum.
* Print Function (print\_die): This function prints the current state of dieA and dieB.
* Backtracking Function (backtrack): This recursive function explores different combinations of values for dieA and dieB. It starts with curA and curB both set to 4 (using 0-based indexing), and it iterates over possible values for each die, recursively exploring all combinations.
* Main Execution Block: The script starts the backtracking process with an initial call to backtrack (4, 4)

**OUTPUT:**

****