

**TASK SUBMISSION**

NAME: CHEVVA JAIVIGNESH

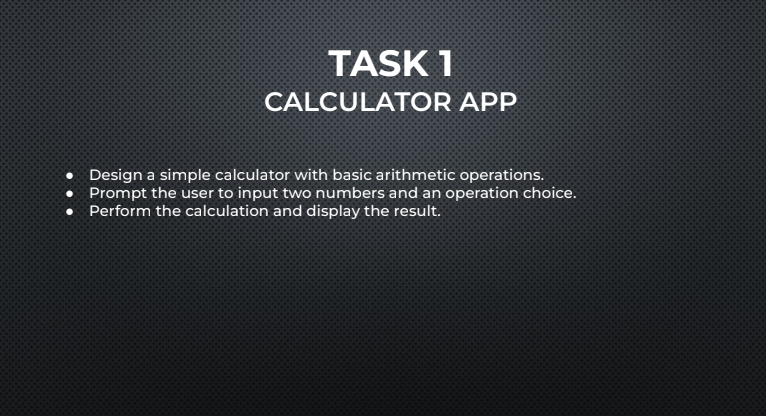
INTERNSHIP : PYTHON DEVELOPMENT

DATE: 20-06-2024

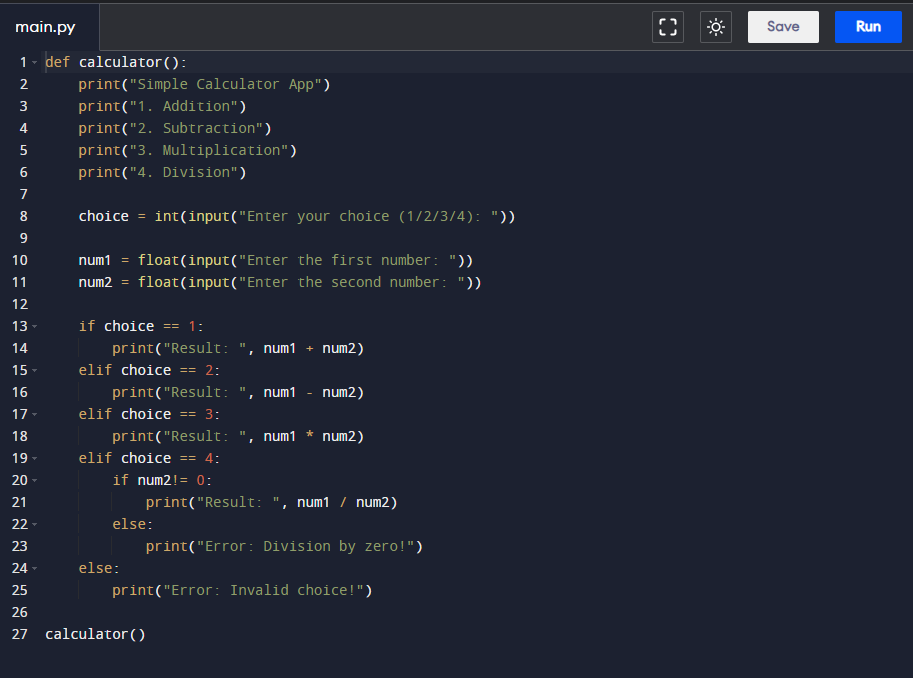
GMAIL: [jaivigneshchevva24@gmail.com](mailto:jaivigneshchevva24@gmail.com)







**PROGRAM:**

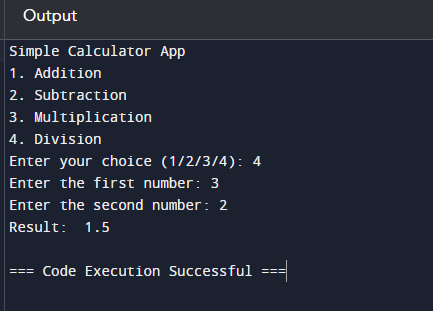
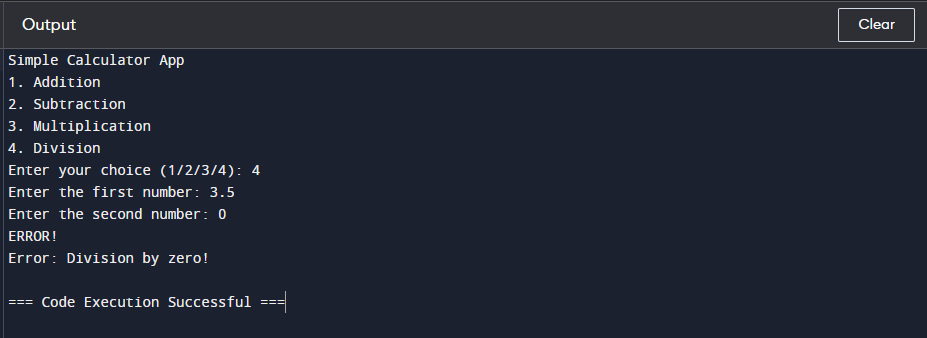


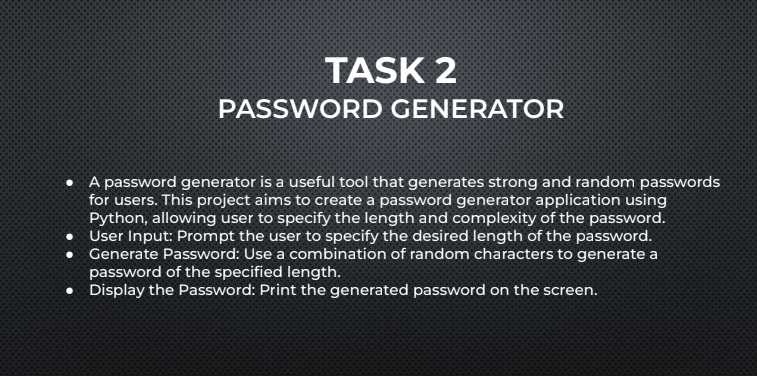


**Here's how the code works:**

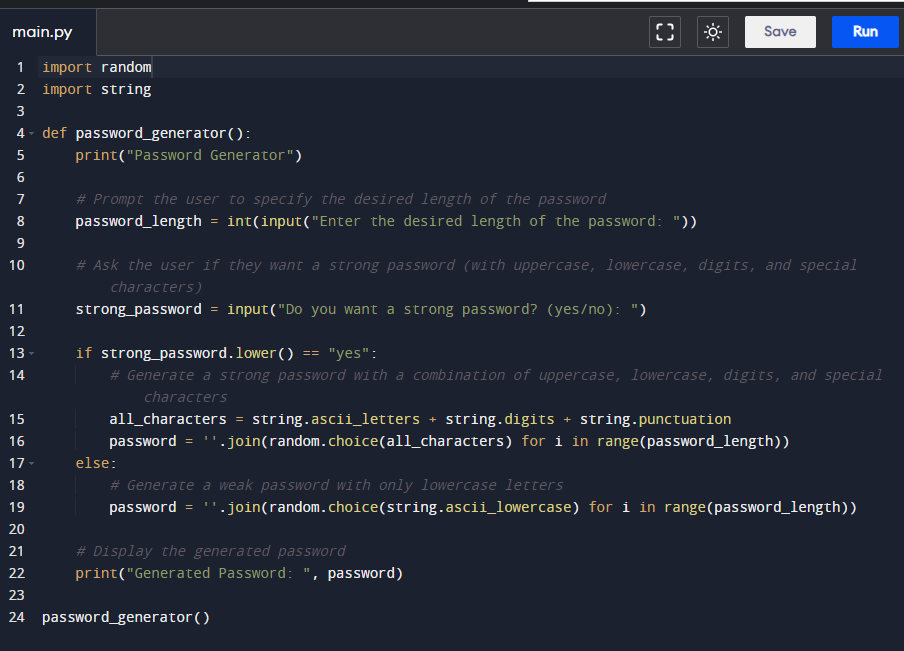
1. The **calculator()** function is defined to contain the calculator app's logic.
2. The app displays a menu with four options: Addition, Subtraction, Multiplication, and Division.
3. The user is prompted to enter their choice (1, 2, 3, or 4).
4. The user is then prompted to enter two numbers.
5. Based on the user's choice, the app performs the corresponding arithmetic operation:
   * If the user chooses Addition (1), the app adds the two numbers.
   * If the user chooses Subtraction (2), the app subtracts the second number from the first.
   * If the user chooses Multiplication (3), the app multiplies the two numbers.
   * If the user chooses Division (4), the app divides the first number by the second, but checks for division by zero and displays an error message if necessary.
6. The app displays the result of the calculation.
7. If the user enters an invalid choice, the app displays an error message.

**OUTPUT:** WE HAVE 2 OUT PUT THERE





**PROGRAM:**

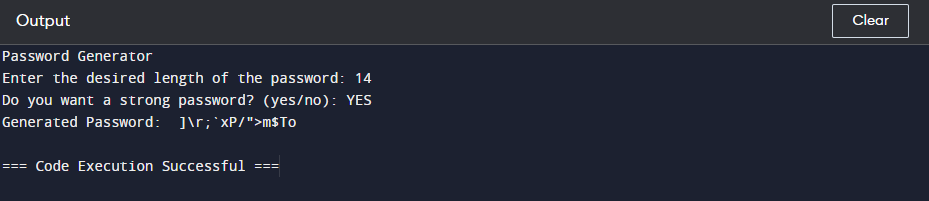




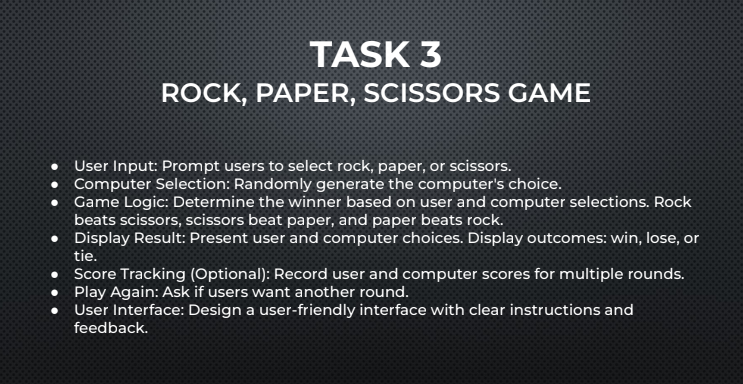
Here's how the code works:

1. The **password\_generator()** function is defined to contain the password generator's logic.
2. The app displays a message and prompts the user to specify the desired length of the password.
3. The user is asked if they want a strong password (with uppercase, lowercase, digits, and special characters).
4. If the user chooses a strong password, the app generates a password using a combination of:
   * **string.ascii\_letters**: Uppercase and lowercase letters (A-Z, a-z)
   * **string.digits**: Digits (0-9)
   * **string.punctuation**: Special characters (!, @, #, $, etc.) The **random.choice()** function is used to select a random character from the combined string, and this is repeated for the specified password length.
5. If the user chooses a weak password, the app generates a password using only lowercase letters.
6. The generated password is displayed on the screen.

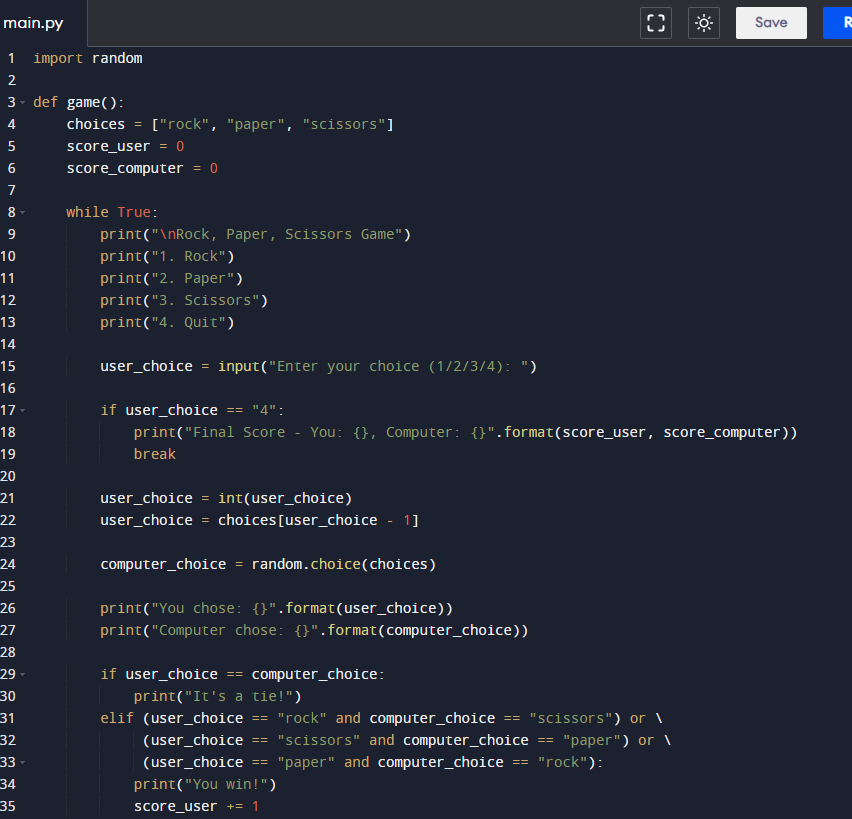
**OUTPUT:**

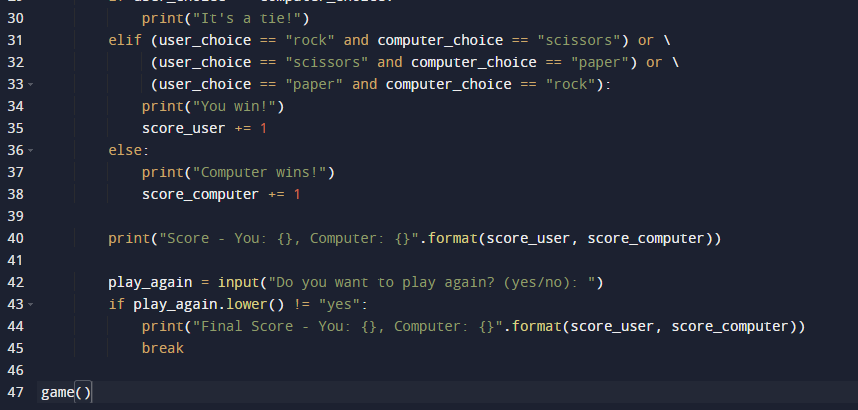




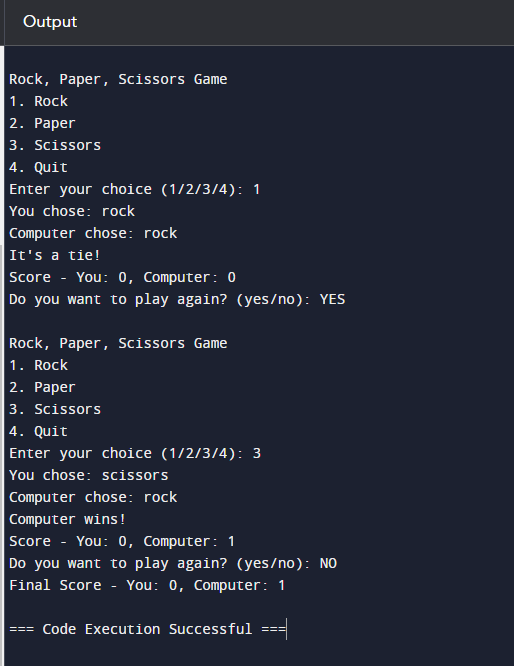


**PROGRAM:**

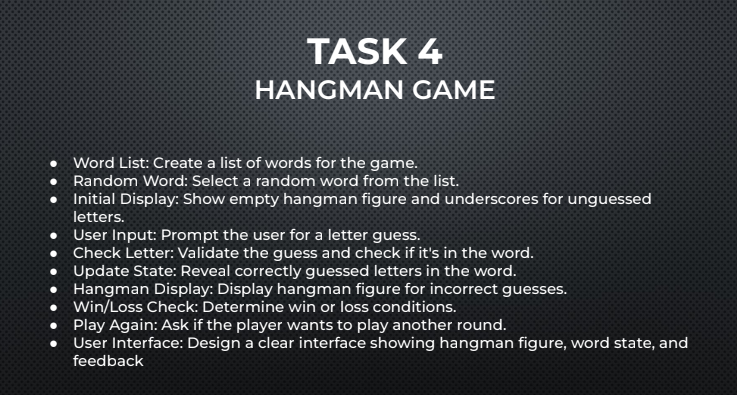




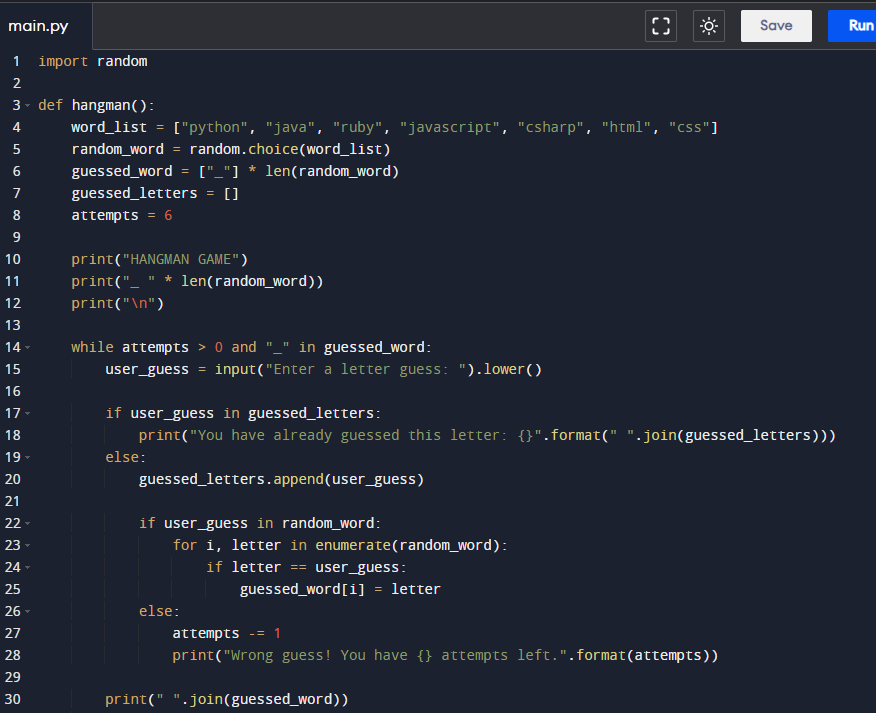
**OUTPUT:**

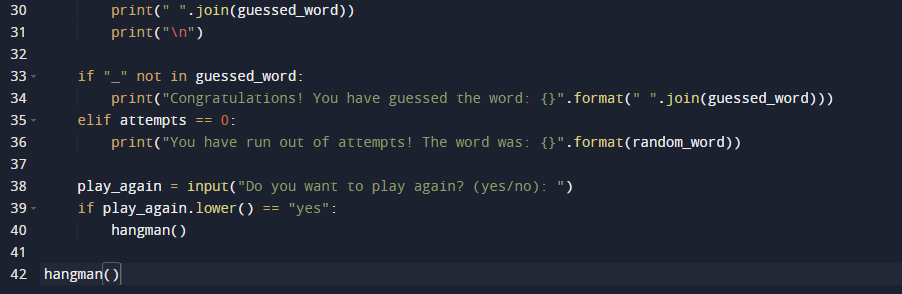
****





**PROGRAM:**

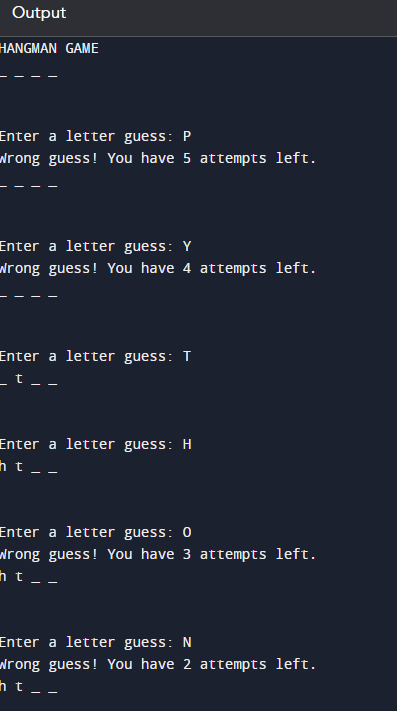


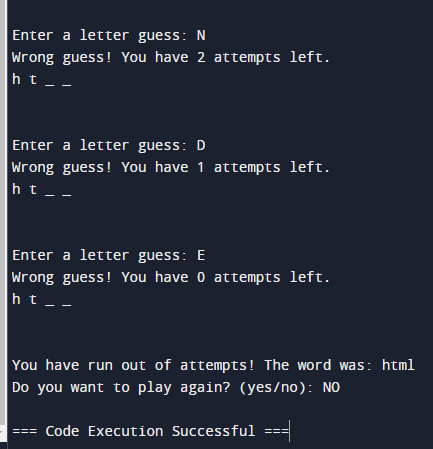


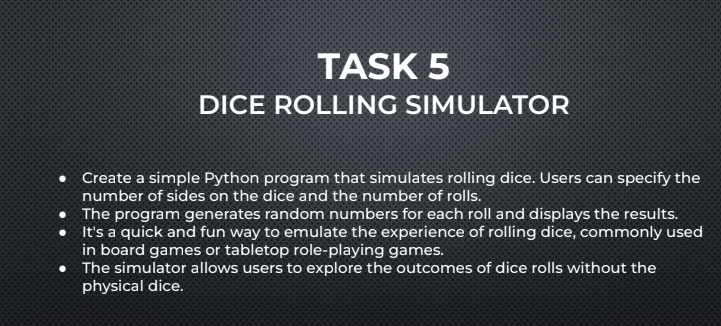
Here's how the code works:

1. The **hangman()** function is defined to contain the game's logic.
2. The game creates a list of words (**word\_list**) and selects a random word (**random\_word**).
3. The game initializes the guessed word (**guessed\_word**) as a list of underscores with the same length as the random word.
4. The game initializes the list of guessed letters (**guessed\_letters**) and the number of attempts (**attempts**).
5. The game displays the hangman figure (initially empty), the guessed word, and the number of attempts.
6. The game enters a loop that continues until the user has guessed the word or run out of attempts.
7. The user is prompted to enter a letter guess.
8. The game checks if the letter has already been guessed and displays a message if it has.
9. The game adds the letter to the list of guessed letters.
10. The game checks if the letter is in the random word and updates the guessed word accordingly.
11. The game decrements the number of attempts if the letter is not in the random word.
12. The game displays the hangman figure, the guessed word, and the number of attempts.
13. The game checks if the user has guessed the word or run out of attempts.
14. The game asks if the user wants to play again. If they respond with "yes", the game starts over. Otherwise, the game exits.

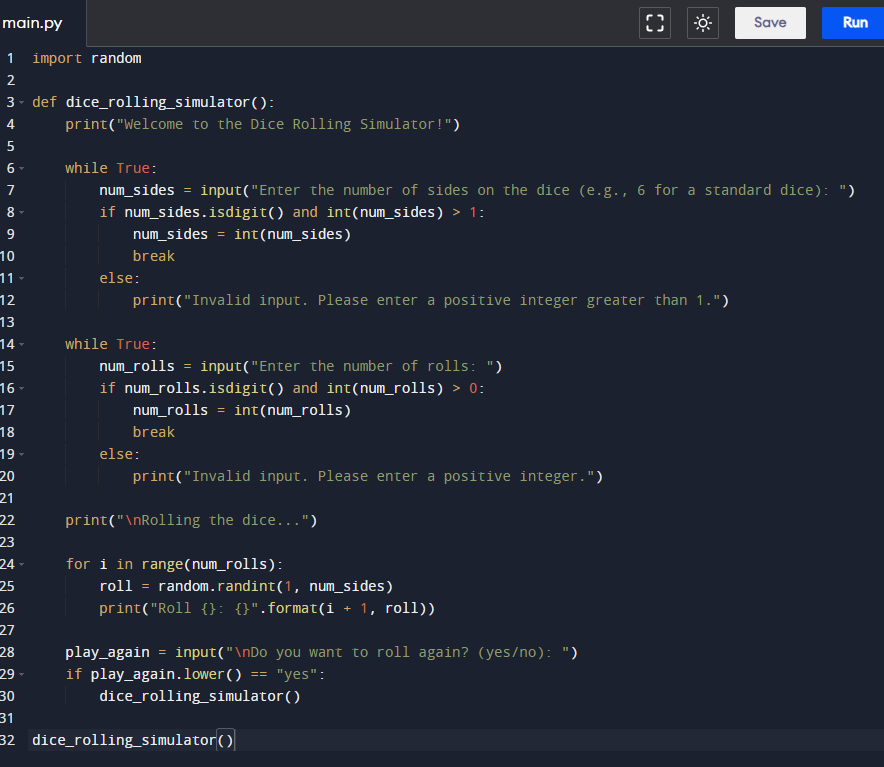
**OUTPUT:**







**PROGRAM:**

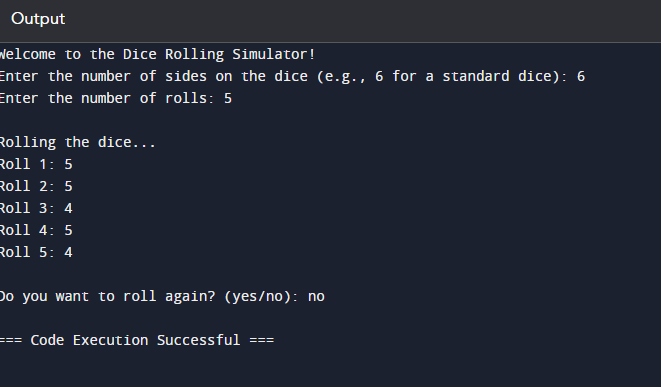
****



Here's how the code works:

1. The **dice\_rolling\_simulator()** function is defined to contain the game's logic.
2. The program welcomes the user and prompts them to enter the number of sides on the dice.
3. The program checks if the input is a positive integer greater than 1 and prompts the user to enter a valid input if not.
4. The program prompts the user to enter the number of rolls and checks if the input is a positive integer.
5. The program generates random numbers for each roll using **random.randint(1, num\_sides)** and displays the results.
6. The program asks if the user wants to roll again. If they respond with "yes", the program starts over. Otherwise, the program exits.

**OUTPUT:**





**THANK YOU**