1. Rock, Paper, Scissors

Create a program that allows the user to play a **best-of-3 Rock, Paper, Scissors** game against the computer. The game should repeatedly ask the user to input their choice—**Rock, Paper, or Scissors**—while the computer randomly generates its own choice using Math.random(). The program should then compare both choices and determine the winner of each round based on the standard rules: **Rock beats Scissors, Scissors beats Paper, and Paper beats Rock**. If both the user and the computer choose the same option, the round is considered a tie, and the user must replay that round. The game continues until one player wins three rounds first, making them the overall winner. Once a winner is determined, the program should display the final score and announce the winner. After completing a match, the program should ask the user if they want to **start a new game** or **exit**. If the user chooses to play again, the game resets, and a new match begins. If they choose to exit, the program terminates. Input validation should be included to ensure the user enters a valid choice before proceeding with each round.

Example output:

Rock Paper Scissor - Best of 3

[1] New Game

[2] Exit

> Enter your choice: 1

Playing game...

========================================================

Your score: 0

Computer score: 0

[1] Rock

[2] Paper

[3] Scissor

> Enter your choice: 2

Your choice: Paper

Computer choice: Paper

- Draw -

========================================================

Playing game...

========================================================

Your score: 0

Computer score: 0

[1] Rock

[2] Paper

1. Number Guessing Game

Write a program that lets the user play a **number guessing game** with a limit of **ten attempts**. The program should randomly generate a secret number between **1 and 50**. The user will be prompted to guess the number, and after each guess, the program should indicate whether the guess is **too high, too low, or correct**. If the user correctly guesses the number within three attempts, the program should display a congratulatory message and end the game. If the user fails to guess the number after three attempts, they lose, and the program should reveal the correct number. Once the game ends, whether the user wins or loses, the program should ask if they want to **play again** or **exit**. If the user chooses to play again, the game resets, and a new number is generated. If they choose to exit, the program should terminate. Input validation should ensure the user enters a valid number within the correct range before proceeding with each guess.

Example output:

Number Guessing Game

[1] Play Game

[2] Exit Game

> Enter your choice: 1

Playing game...

===================================================

You only have 10 tries.

> Enter your guess number(0-50): 51

Invalid guess. Try again.

Playing game...

===================================================

You only have 10 tries.

> Enter your guess number(0-50): -1

Invalid guess. Try again.

Playing game...

===================================================

You only have 10 tries.

> Enter your guess number(0-50): 45

Your guess is too high.

===================================================

Playing game...

===================================================

You only have 9 tries.

> Enter your guess number(0-50): 34

Your guess is too low.

===================================================

Playing game...

===================================================

You only have 8 tries.

> Enter your guess number(0-50): 40

Your guess is too high.

===================================================

Playing game...

===================================================

You only have 7 tries.

> Enter your guess number(0-50): 38

Your guess is too high.

===================================================

Playing game...

1. Reverse Number

Create a program that a computer generates random number and reverses its digits using a **while loop** and the **modulus (%) operator**. Extract each digit one by one to construct the reversed number. To achieve this, the program should repeatedly extract the last digit of the number using the modulus operator (% 10) and add it to the reversed number by shifting its existing digits to the left (multiplying the reversed number by 10 before adding the extracted digit). After extracting a digit, the program should remove it from the original number using integer division (/ 10). This process should continue until the number becomes **zero**, ensuring that all digits are reversed. If the compute generates a **negative number**, the program should preserve the negative sign while reversing the digits. Once the reversed number is obtained, the program should display the result to the user in a clear format. After displaying the reversed number, the program should ask if the user wants to **reverse another number or exit**. If the user chooses to continue, then do the same process again. If they choose to exit, the program should terminate.

Example output:

Reverse Number Program

[1] New number

[2] Exit

> Enter choice: 1

> Enter a number: 32124

Original Number: 32124

Reverse Number: 42123

Reverse Number Program

[1] New number

[2] Exit

> Enter choice: 1

> Enter a number: -31

Original Number: -31

Reverse Number: -13

Reverse Number Program

[1] New number

[2] Exit

> Enter choice: 3

1. Coin Flip Simulation

Create a program that simulates flipping a coin. The program should generate a random integer number between 1 and 10, and if the value is **less than 5**, it should display "Heads"; otherwise, it should display "Tails". The program should run **continuously in a loop**, allowing the user to flip the coin multiple times. After each flip, the program should prompt the user with a question asking if they want to flip again. If the user wants to flip again, the program will perform another coin flip. If the user wants to exit, the program should display a farewell message and exit. The program must also **validate user input**. If the user enters anything else, the program should display an error message and ask for a valid input until the user provides a correct response.

Example output:

- Coin Flip Simulation -

[1] Flip coin

[2] Exit

> Enter your choice: 1

Flipping coin...

The random number is: 4

Coin Flip Result: Heads

- Coin Flip Simulation -

[1] Flip coin

[2] Exit

> Enter your choice: 1

Flipping coin...

The random number is: 6

Coin Flip Result: Tails

- Coin Flip Simulation -

[1] Flip coin

[2] Exit

> Enter your choice: 3

Invalid choice.

1. Number Lottery Game

Create a program that simulates a **Number Lottery Game**, where the computer generates a random three-digit number between **100 and 999**. The program then prompts the user to enter their own **three-digit number** as a guess. Once the input is received, the program compares the user’s number with the randomly generated number and determines the result based on how many digits match. If all three digits match exactly in the same positions, the player **wins the jackpot**. If two digits match, they **win a smaller prize**. If only one digit matches, they receive a **consolation prize**. If none of the digits match, the program displays "Better luck next time!". To ensure valid input, the program checks whether the user’s number falls within the **100 to 999** range. If the input is invalid, the user is prompted to enter a valid three-digit number. After displaying the results, the program asks whether the user wants to play again. If the user chooses to continue, the game repeats with a **new random number**. If they decide to exit, the program ends with a thank-you message. The game runs in a loop, allowing multiple attempts until the user chooses to stop.

Example output:

- Number Lottery Game -

[1] Play Game

[2] Exit Game

> Enter your choice: 1

Enter 3 digit number: 118

Better luck next time

The lottery number is: 991

- Number Lottery Game -

[1] Play Game

[2] Exit Game

> Enter your choice: 2

Thanks for playing the game.

1. Random Traffic Light Simulation

Create a program that simulates a **random traffic light system**, where the computer randomly selects one of the three standard traffic light colors: **Red, Yellow, or Green**. The program will display the corresponding traffic signal instruction. If the selected color is **Red**, the program will display **"Stop"**, indicating that vehicles must come to a complete halt. If the color is **Yellow**, it will display **"Get Ready"**, signaling that vehicles should prepare to move but not proceed yet. If the color is **Green**, the program will display **"Go"**, allowing vehicles to move forward. The program will continuously allow the user to check the traffic light multiple times by generating a new random color each time. After each display, the program will ask whether the user wants to check the light again. If the user chooses to continue, a new random color is generated and displayed. If the user decides to exit, the program will end with a farewell message.

Example output:

Random Traffic Light Simulation -

[1] Check traffic light

[2] Exit program

> Enter your choice: 3

Invalid choice.

- Random Traffic Light Simulation -

[1] Check traffic light

[2] Exit program

> Enter your choice: 1

===============================================

Traffic lights: Red, Yellow, Green

Traffic light: Green -> Go

Checking ended. Drive Safely!

===============================================

- Random Traffic Light Simulation -

[1] Check traffic light

[2] Exit program

> Enter your choice: 2

Thanks for using the program.

1. Random Math Quiz Generator

Create a program that generates a **random multiplication quiz** for the user. The program will randomly select two numbers between **1 and 10** and present a multiplication question in the format **"X × Y = ?"**, where **X** and **Y** are the randomly chosen numbers. The user must enter their answer, and the program will determine whether it is **correct or incorrect**. If the answer is correct, the program will display **"Correct!"** and add a point to the user's score. If the answer is incorrect, the program will display **"Wrong! The correct answer is [correct result]."** The quiz consists of **five rounds**, and after answering all five questions, the program will display the **final score** out of 5. After completing the quiz, the user will be given the option to **play again or exit**. If the user chooses to play again, the quiz restarts with a new set of random questions. If the user chooses to exit, the program will display a **thank-you message** and terminate.

Example output:

- Math Quiz Generator -

[1] Start quiz

[2] Exit quiz

> Enter your choice: 3

Invalid choice.

- Math Quiz Generator -

[1] Start quiz

[2] Exit quiz

> Enter your choice: 1

==============================================

Round 1

Question 1: What is 10 × 1? 10

Correct.

==============================================

Round 2

Question 2: What is 8 × 2? 16

Correct.

==============================================

Round 3

Question 3: What is 1 × 4? 4

Correct.

==============================================

Round 4

Question 4: What is 1 × 1? 1

Correct.

==============================================

Round 5

Question 5: What is 9 × 2? 18

Correct.

Quiz over! Your final score is 5.

1. Dice Roller Simulator

Create a program that simulates the rolling of a **six-sided dice**. Each time the user rolls the dice, the program will generate a **random number between 1 and 6**, simulating an actual dice roll. After displaying the result, the program will ask the user if they want to **roll again or exit**. If the user chooses to roll again, the program will generate another random number and repeat the process. If the user chooses to exit, the program will display a **goodbye message** and terminate.

Example output:

- Roll Dice Simulator -

[1] Roll Dice

[2] Exit.

> Enter your choice: 3

Invalid choice.

- Roll Dice Simulator -

[1] Roll Dice

[2] Exit.

> Enter your choice: 1

You rolled a 3.

- Roll Dice Simulator -

[1] Roll Dice

[2] Exit.

> Enter your choice: 1

You rolled a 2.

- Roll Dice Simulator -

[1] Roll Dice

[2] Exit.

> Enter your choice: 2

Thanks for playing.

1. Random Walk Simulator

Create a program that simulates a random walk for a virtual character starting at position **0**. The character will take exactly **10 steps**, and each step's direction is determined randomly. The program should generate a random number for each step. If the generated number is **less than 5**, the character moves one step to the **left (-1)**; otherwise, the character moves one step to the **right (+1)**. After completing all **10 steps**, the program should display the character's final position. Once the walk is completed, the program should ask the user whether they want to **perform another random walk or exit**. If the user chooses to continue, the simulation should restart; otherwise, the program should terminate.

Example output:

- Random Walk Simulator -

[1] Walk

[2] Exit

> Enter your choice: 3

Invalid choice.

- Random Walk Simulator -

[1] Walk

[2] Exit

> Enter your choice: 1

Step 1: Moved left (-1), Position: -1

Step 2: Moved right (+1), Position: 0

Step 3: Moved left (-1), Position: -1

Step 4: Moved right (+1), Position: 0

Step 5: Moved left (-1), Position: -1

Step 6: Moved right (+1), Position: 0

Step 7: Moved right (+1), Position: 1

Step 8: Moved left (-1), Position: 0

Step 9: Moved left (-1), Position: -1

Step 10: Moved right (+1), Position: 0

Final position: 0

- Random Walk Simulator -

[1] Walk

[2] Exit

> Enter your choice: 2

Thanks for using the program.

1. Random Color Picker

Create a program that randomly selects a color from a predefined list of five colors: **Red, Blue, Green, Yellow, and Purple**. The user is then prompted to guess which color was selected. After the user inputs their guess, the program compares it with the randomly chosen color. If the user’s guess matches the randomly selected color, the program prints **"You got it!"** to indicate a correct answer. If the guess is incorrect, the program displays **"Wrong! The correct color was X."**, replacing **X** with the correct color. The program should allow the user to play multiple rounds, asking after each attempt whether they would like to continue playing or exit the game.

Example output:

- Random Color Picker -

[1] Play game

[2] Exit game

> Enter your choice: 1

Choose a color

[1] Red [2] Blue [3] Green [4] Yellow [5] Purple

> Enter your choice: 3

Wrong! The color is Purple.

- Random Color Picker -

[1] Play game

[2] Exit game

> Enter your choice: 1

Choose a color

[1] Red [2] Blue [3] Green [4] Yellow [5] Purple

> Enter your choice: 5

Wrong! The color is Green.

- Random Color Picker -

[1] Play game

[2] Exit game

> Enter your choice: 2

Thanks for playing the game.