```
console.clear();

/* ================================================================================

     🔁 JavaScript Loops Interview Challenges – Sharpen Your Logic! 🔍

================================================================================ */

/*

📌 3 Challenges Using While Loop

*/

// ✖️ Factorial of a Number

console.log("➤ Factorial using while loop:");

// Initialize a variable 'number' with the value 6. This is the number for which we want to calculate the factorial

let number = 6;

// Initialize a variable 'factorial' with 1. We'll use this variable to store the result as we multiply step-by-step

let factorial = 1;

// Store the original value of 'number' in a temporary variable 'temp'. This helps if you want to use the original number
later (e.g., for printing)

let temp = number;

// Start a while loop that runs as long as 'number' is greater than 1

while (number > 1) {

    // Multiply the current 'factorial' value by 'number'

    // This builds the factorial step by step: e.g., 6 * 5 * 4 * 3 * 2 * 1
```

```
38
39      factorial = factorial * number;
40
41      // Decrement 'number' by 1 on each iteration. So the loop proceeds from 6 → 5 → 4 → 3 → 2 → 1
42
43      number--;
44  }
45
46  // You can print it to the console like this:
47
48  console.log(`The factorial of ${temp} is ${factorial}`);
49
50  // 🔁 Reverse a Number
51
52  console.log("\n➤ Reverse a number using while loop:");
53
54  // Initialize the variable 'original' with the number you want to reverse
55
56  let original = 789432;
57
58  // Initialize 'reversed' with 0. This variable will store the reversed digits step by step
59
60  let reversed = 0;
61
62  // Start a while loop that continues as long as 'original' is greater than 0
63
64  while (original > 0) {
65
66      // Step 1: Extract the last digit of 'original'. original % 10 gives the remainder when divided by 10, which is the last
    digit
67
68      let digit = original % 10;
69
70      // Step 2: Add this digit to the end of the 'reversed' number. Multiply 'reversed' by 10 to shift its digits to the left,
    then add 'digit'
71
72      reversed = reversed * 10 + digit;
73
```

```
    // Step 3: Remove the last digit from 'original'. Use Math.floor() to drop the decimal after dividing by 10

    original = Math.floor(original / 10);
}

// Finally, print the reversed number

console.log(`Reversed Number = ${reversed}`);

// 🔷 Armstrong Number Check

// An Armstrong number is a number that is equal to the sum of its own digits raised to the power of the number of digits.

// For example: 9474 is a 4-digit number, and 9⁴ + 4⁴ + 7⁴ + 4⁴ = 9474

console.log("\n➤ Armstrong number using while loop:");

// Set the candidate number to check whether it's an Armstrong number

let candidate = 9474;

// Convert the number to a string and get its length. This tells us how many digits the number has, which is the "power"

let power = candidate.toString().length;

// Create a copy of the original number to perform digit-wise operations

let copy = candidate;

// Initialize a variable to hold the sum of digits raised to the given power

let armSum = 0;

// Use a while loop to process each digit of the number

while (copy > 0) {

    // Step 1: Extract the last digit using modulo operator
```

```
112
113     let digit = copy % 10;
114
115     // Step 2: Raise the digit to the power of total number of digits and add it to armSum
116
117     armSum = armSum + digit ** power;
118
119     // Step 3: Remove the last digit by dividing by 10 and flooring the result
120
121     copy = Math.floor(copy / 10);
122 }
123
124 // After the loop ends, armSum contains the sum of digits raised to the power Compare it with the original candidate to check
    if it's an Armstrong number
125
126 console.log(armSum === candidate ? `${candidate} is an Armstrong number.` : `${candidate} is NOT an Armstrong number.`);
127
128 /*
129
130 📍 3 Challenges Using Do-While Loop
131
132 */
133
134 // 🗓️ Leap Year Range (2000–2024)
135
136 console.log("\n➤ Leap years between 2000 to 2024 using do...while:");
137
138 // Initialize the variable 'year' with 2000. We'll check from this year up to 2024
139
140 let year = 2000;
141
142 // Start a do...while loop that runs at least once, and continues until year <= 2024
143
144 do {
145
146     // Check if the current year is a leap year. A year is a leap year if:
147
148     /*
```

```
    It is divisible by 4 AND NOT divisible by 100

    OR

    It is divisible by 400

    */

    if ((year % 4 === 0 && year % 100 !== 0) || year % 400 === 0) {

        // If the condition is true, print the year as a leap year

        console.log(year);
    }

    // Move to the next year

    year++;

} while (year <= 2024); // Continue looping until year becomes greater than 2024


// 📊 Count Digits of a Number

console.log("\n➤ Count digits using do...while:");

// Initialize the variable 'num' with the number whose digits we want to count

let num = 987654321;

// Initialize a counter to 0. This will be used to count the number of digits

let count = 0;

// Start a do...while loop — this loop will run at least once

do {
```

```
187
188     // Remove the last digit from 'num' by dividing it by 10 and taking the floor
189
190     // Example: 987654321 → 98765432 → 9876543 → ... → 0
191
192     num = Math.floor(num / 10);
193
194     // Increase the count after removing a digit
195
196     count++;
197
198 } while (num > 0); // Repeat the loop until the number becomes 0
199
200 console.log(`Digit Count = ${count}`);
201
202 // 🧡 GCD (Greatest Common Divisor)
203
204 console.log("\n➤ GCD using do...while:");
205
206 // Initialize two numbers 'a' and 'b' whose GCD we want to find
207
208 let a = 48, b = 180;
209
210 // Use a do...while loop to implement the Euclidean algorithm
211
212 do {
213
214     // Store the current value of 'b' in a temporary variable
215
216     let temp = b;
217
218     // Update 'b' to be the remainder of a divided by b. This effectively keeps reducing the problem
219
220     b = a % b;
221
222     // Update 'a' to the previous value of 'b' (stored in temp)
223
224     a = temp;
```

```
225
226 } while (b !== 0); // Repeat until the remainder becomes 0
227
228 // When b becomes 0, 'a' holds the GCD of the original two numbers
229
230 console.log(`GCD = ${a}`);
231
232 /*
233
234 📍 3 Challenges Using For Loop
235
236 */
237
238 // 🔢 Fibonacci Series of N Terms
239
240 console.log("\n➤ Fibonacci Series (10 terms) using for loop:");
241
242 // Set the number of Fibonacci terms you want to print
243
244 let terms = 10;
245
246 // Initialize the first two Fibonacci numbers
247
248 let f1 = 0, f2 = 1;
249
250 // Use a for loop to generate 'terms' number of Fibonacci values
251
252 for (let i = 0; i < terms; i++) {
253
254     // Calculate the next term by adding the previous two terms
255
256     let f3 = f1 + f2;
257
258     // Update f1 and f2 for the next iteration3
259
260     f1 = f2;
261
262     f2 = f3;
```

```
}

// 🔍 Prime Number Check

console.log("\n➤ Prime check using for loop:");

// Set the number you want to check for primality

let primeCheck = 37;

// Assume the number is prime initially

let isPrime = true;

// Use a for loop starting from 2 up to the square root of the number

// No need to check beyond sqrt(n) because if a number is divisible

// by anything greater than its square root, the pair factor must be less than it

for (let i = 2; i <= Math.sqrt(primeCheck); i++) {

    // If primeCheck is divisible by i, it's not a prime number

    if (primeCheck % i === 0) {

        // Set isPrime to false and break out of the loop

        isPrime = false;

        break;
    }
}

// Print the result based on isPrime

console.log(isPrime ? `${primeCheck} is Prime.` : `${primeCheck} is NOT Prime.`);

```

```
// 💯 Perfect Number Check

console.log("\n➤ Perfect number check using for loop:");

// Set the number you want to check

let perfectNum = 28;

// Initialize a variable to store the sum of proper divisors

let sum = 0;

// Loop from 1 up to (but not including) the number itself

// We're looking for all numbers that divide 'perfectNum' exactly

for (let i = 1; i < perfectNum; i++) {

    // If 'i' is a divisor of 'perfectNum' (i.e., no remainder)

    if (perfectNum % i === 0) {

        // Add the divisor to the running sum

        sum = sum + i;
    }
}

// After the loop, check if the sum of divisors equals the original number. If yes, it's a Perfect Number

console.log(sum === perfectNum ? `${perfectNum} is a Perfect Number.` : `${perfectNum} is NOT a Perfect Number.`);
```