Tasks:

Task 1: Concatenate Two Strings Without Using Methods

Description: Manually append the characters of one string to another, character by character, without using any built-in string methods.

Task 2: Create and Display an Object Manually

Description: Create an object to store details of a student (name, age, grade). Display the student's details by manually accessing object properties (no methods/functions).

```
ndex1.js > ...
1   var students = {
2       'name': "John",
3       "age": 20,
4       'grade': "A"
5
6   };
7   console.log(`
8       Name: ${students[`name`]}
9       Age: ${students[`age`]}
10      Grade: ${students[`grade`]}`
11  );
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] node "d:\my files\daily_activity\day40\index1.js"

    Name: John
    Age: 20
    Grade: A

[Done] exited with code=0 in 0.158 seconds
```

Task 3: Count Occurrences of a Specific Character in a String Without Using Methods

Description: Manually count the number of occurrences of a specific character in a given string.

```
//string = prompt('enter any string here:');
string = 'banana';
//char_to_count = promp("enter the character you want to count:");
char_to_count = 'a';
count = 0;
for(i of string){
    if(i=='a'){
        count++;
    }
}
console.log(`Result: ${count} (The character "${char_to_count}" appears ${count} times in "${string}")`)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[Running] node "d:\my files\daily_activity\day40\index2.js"
Result: 3 (The character "a" appears 3 times in "banana")

[Done] exited with code=0 in 0.155 seconds
```

Task 4: Create an Object and Retrieve Properties Without Methods

Description: Create an object representing a car (make, model, year, color) and manually retrieve and display the properties.

```
ex3.js > ...
var car = {
    'Make': "Toyota",
'Model': "Corolla",
'Year': 2020,
'Color': "Blue"

}
console.log(`
    Make: ${car['Make']}
    Model: ${car['Model']}
    Year: ${car['Year']}
    Color: ${car['Color']}`)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] node "d:\my files\daily_activity\day40\index3.js"

    Make: Toyota
    Model: Corolla
    Year: 2020
    Color: Blue

[Done] exited with code=0 in 0.134 seconds
```

Task 5: Manually Reverse a String

Description: Write a program to reverse a string character by character without using any built-in string methods or functions.

dex4.js

```
1   string = 'abcdef';
2   reverse = '';
3   for(i=string.length-1;i>=0;i--){
4       reverse += string[i];
5   }
6   console.log(`string before : "${string}"`);
7   console.log(`string after reversing : "${reverse}"`);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] node "d:\my files\daily_activity\day40\index4.js"
string before : "abcdef"
string after reversing : "fedcba"

[Done] exited with code=0 in 0.135 seconds
```

Global execution context:

The global execution context is the top-level context in a JavaScript program. It represents the global scope, encompassing the entire program and all its components. This context sets the stage for the entire code execution process and plays a pivotal role in managing global variables and functions.

Creating the Global Context

The global execution context is automatically created when a JavaScript program starts running. Here's what happens during its creation:

1. Initialization: When your JavaScript code is loaded by a web browser or executed by a JavaScript runtime environment (like Node.js), the global execution context is initialized.
2. Global Object: A global object (also known as the window object in a browser environment or the global object in Node.js) is created. This object serves as the global scope, and all global variables and functions become properties and methods of this object.
3. This Binding: In the global context, the this keyword is bound to the global object. This means that when you refer to this at the top level of your code, it points to the global object.
4. Outer Environment: The outer (or parent) environment of the global context is set to null since there is no outer context beyond the global scope.
5. Creation of Global Variables and Functions: Any global variables and functions declared in your code are hoisted and initialized within the global context.