JAVA SCRIPT

**TASK-11**

// A minus 1 results in NaN (Not a Number) because a string cannot be subtracted from a number.

console.log('A' - 1); ANSWER:NaN

// A plus 1 concatenates the string 'A' and the number 1, resulting in the string 'A1'.

console.log('A' + 1); ANSWER:A1

// 2 + '2' results in the string '22'. Then '22' + '2' results in '222'.

console.log(2 + '2' + '2'); ANSWER:222

// 'hello' + 'world' concatenates to 'helloworld'. Then 'helloworld' + 89 concatenates to 'helloworld89'.

console.log('hello' + 'world' + 89); ANSWER: helloworld89

// 'hello' - 'world' is an invalid operation, resulting in NaN. NaN + 89 remains NaN.

console.log('hello' - 'world' + 89); ANSWER:NaN

// 'hello' + 78 concatenates the string 'hello' and the number 78.

console.log('hello' + 78); ANSWER:hello78

// '78' - 90 converts '78' to the number 78. 78 - 90 is -12. Then -12 + '2' concatenates to '-122'.

console.log('78' - 90 + '2'); ANSWER:-122

// 2 - '2' converts '2' to the number 2. 2 - 2 is 0. Then 0 + 90 is 90.

console.log(2 - '2' + 90); ANSWER:90

// '90' / 2 converts '90' to the number 90. 90 / 2 is 45. Then 89 - 45 is 44.

console.log(89 - '90' / 2); ANSWER:44

// true == false evaluates to false. Then false > 2 coerces false to 0, so the expression becomes 0 > 2, which is false.

console.log((true == false) > 2); ANSWER:false

**TASK:12**

let str = ' gfuh ieiuei ';

let firstFive = str.substring(0, 5);

console.log(firstFive); // ' gfuh'

let str = 'hduej dij';

let len = str.length;

let upperCaseStr = str.toUpperCase();

console.log(`Original string: "${str}"`);

console.log(`Length: ${len}`); // Length: 9

console.log(`Uppercase string: "${upperCaseStr}"`); // Uppercase string: "HDUEJ DIJ"

let str = ' biji jdo ';

let result = str.toLowerCase().trim();

console.log(result); // 'biji jdo' // Example: replacing "world" with "everyone"

let originalStr = 'Hello world, this is a beautiful world.';

let replacedStr = originalStr.replace('world', 'everyone');

console.log(replacedStr); // 'Hello everyone, this is a beautiful world.'

// Example using regex to replace all occurrences

let originalStrAll = 'Hello world, this is a beautiful world.';

let replacedStrAll = originalStrAll.replace(/world/g, 'everyone');

console.log(replacedStrAll); // 'Hello everyone, this is a beautiful everyone.'

let result = 89 + 'hello' + 90 / 9;

console.log(89 + 'hello' + 90 / 9); // '89hello10'

**TASK:13**

// 1. Create objects for animal and car

// Objects are collections of key-value pairs, representing entities with properties.

// Animal Object

const animal = {

species: 'Lion',

name: 'Simba',

age: 5,

habitat: 'Savannah',

sound: function() {

return 'Roar!';

}

};

console.log("1. Animal Object:", animal);

console.log(" Animal's sound:", animal.sound());

// Car Object

const car = {

make: 'Toyota',

model: 'Camry',

year: 2022,

color: 'Silver',

isElectric: false,

start: function() {

return 'Engine started!';

}

};

console.log(" Car Object:", car);

console.log(" Car action:", car.start());

// 2. Find duplicates in a string (using array) - Optional

// This function converts the string to an array of characters, then uses a frequency map

// to count occurrences and identify characters that appear more than once.

function findDuplicatesInString(str) {

const charArray = str.split(''); // Convert string to an array of characters

const charCounts = {}; // Object to store character frequencies

const duplicates = new Set(); // Use a Set to store unique duplicate characters

for (const char of charArray) {

// Ignore spaces if not considered a character for duplication

if (char === ' ') {

continue;

}

charCounts[char] = (charCounts[char] || 0) + 1;

if (charCounts[char] > 1) {

duplicates.add(char); // Add to duplicates set if count is > 1

}

}

return Array.from(duplicates); // Convert Set back to an Array

}

const myStringWithDuplicates = "hello world";

const foundDuplicates = findDuplicatesInString(myStringWithDuplicates);

console.log(`\n2. Duplicates in "${myStringWithDuplicates}":`, foundDuplicates); // Expected: ['l', 'o']

// 3. Reverse a string (use array method)

// This function takes a string, splits it into an array of characters,

// reverses the array, and then joins the array back into a string.

function reverseString(str) {

// Split the string into an array of characters

const charArray = str.split('');

// Reverse the array in place

charArray.reverse();

// Join the array back into a string

return charArray.join('');

}

const originalString = "JavaScript";

const reversedStr = reverseString(originalString);

console.log(`\n3. Original string: "${originalString}"`);

console.log(` Reversed string: "${reversedStr}"`); // Expected: "tpircSavaJ"

// 4. Find the highest and lowest value in an array

// Using Math.max() and Math.min() with the spread operator (...)

// is an efficient way to find the highest and lowest values in a numeric array.

const numbers = [10, 5, 20, 15, 25, 3];

const highestValue = Math.max(...numbers);

const lowestValue = Math.min(...numbers);

console.log(`\n4. Array: [${numbers}]`);

console.log(` Highest value: ${highestValue}`); // Expected: 25

console.log(` Lowest value: ${lowestValue}`); // Expected: 3

// 5. Sorting of an array - Optional

// The .sort() method sorts the elements of an array in place and returns the sorted array.

// For numbers, a compare function is necessary to ensure correct numerical sorting.

const unsortedNumbers = [3, 1, 4, 1, 5, 9, 2, 6];

const unsortedStrings = ["banana", "apple", "cherry", "date"];

// Sorting numbers (ascending)

const sortedNumbersAsc = [...unsortedNumbers].sort((a, b) => a - b);

console.log(`\n5. Original numbers: [${unsortedNumbers}]`);

console.log(` Sorted numbers (ascending): [${sortedNumbersAsc}]`); // Expected: [1, 1, 2, 3, 4, 5, 6, 9]

// Sorting numbers (descending)

const sortedNumbersDesc = [...unsortedNumbers].sort((a, b) => b - a);

console.log(` Sorted numbers (descending): [${sortedNumbersDesc}]`); // Expected: [9, 6, 5, 4, 3, 2, 1, 1]

// Sorting strings (alphabetical)

const sortedStrings = [...unsortedStrings].sort();

console.log(` Original strings: ["${unsortedStrings.join('", "')}"]`);

console.log(` Sorted strings (alphabetical): ["${sortedStrings.join('", "')}"]`); // Expected: ["apple", "banana", "cherry", "date"]

// 6. Display first 3 elements in an array in UI

// This part requires HTML to display elements.

// The JavaScript will dynamically create and append list items to a designated HTML element.

// For demonstration purposes, we'll assume a <div> with id="app" exists in the HTML.

// HTML structure for the UI display (conceptual, will be rendered by the immersive)

/\*

<div id="app" class="p-4 bg-gray-100 rounded-lg shadow-md">

<h2 class="text-xl font-bold mb-2">First 3 Array Elements</h2>

<ul id="firstThreeElementsList" class="list-disc pl-5">

<!-- Elements will be inserted here by JavaScript -->

</ul>

</div>

\*/

const myArrayForUI = ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry', 'Fig'];

// Function to display elements in the UI

function displayFirstThreeElements(arr, elementId) {

const listContainer = document.getElementById(elementId);

if (!listContainer) {

console.error(`Element with ID '${elementId}' not found for UI display.`);

return;

}

listContainer.innerHTML = ''; // Clear existing content

// Iterate over the first 3 elements and create list items

for (let i = 0; i < Math.min(3, arr.length); i++) {

const listItem = document.createElement('li');

listItem.textContent = arr[i];

listItem.className = 'text-gray-800'; // Tailwind class for text color

listContainer.appendChild(listItem);

}

console.log(`\n6. Displaying first 3 elements of [${myArrayForUI.join(', ')}] in UI (check preview).`);

}

// Ensure the DOM is loaded before trying to access elements

document.addEventListener('DOMContentLoaded', () => {

// Create a div and ul for the UI display if they don't exist

let appDiv = document.getElementById('app');

if (!appDiv) {

appDiv = document.createElement('div');

appDiv.id = 'app';

appDiv.className = 'p-4 bg-gray-100 rounded-lg shadow-md mt-4';

document.body.appendChild(appDiv);

}

let h2 = document.createElement('h2');

h2.className = 'text-xl font-bold mb-2';

h2.textContent = 'First 3 Array Elements';

appDiv.appendChild(h2);

let ul = document.createElement('ul');

ul.id = 'firstThreeElementsList';

ul.className = 'list-disc pl-5';

appDiv.appendChild(ul);

displayFirstThreeElements(myArrayForUI, 'firstThreeElementsList');

});

// 7. Remove 4th (index) element and add 2 elements there

// The .splice() method changes the contents of an array by removing or replacing existing elements

// and/or adding new elements in place.

// Syntax: array.splice(start, deleteCount, item1, item2, ...)

// - start: The index at which to start changing the array.

// - deleteCount: The number of elements to remove from 'start'.

// - item1, item2, ...: The elements to add to the array, starting at 'start'.

let fruits = ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry', 'Fig', 'Grape'];

console.log(`\n7. Original array: [${fruits.join(', ')}]`);

// Remove 1 element at index 4 ('Elderberry') and add 'Kiwi' and 'Lemon'

// Index 4 is the 5th element ('Elderberry')

fruits.splice(4, 1, 'Kiwi', 'Lemon');

console.log(` Array after splice: [${fruits.join(', ')}]`);

// Expected: ['Apple', 'Banana', 'Cherry', 'Date', 'Kiwi', 'Lemon', 'Fig', 'Grape']

**TASK:14**

// 1. Create parameterized method/function to multiply 3 numbers.

// This function accepts three parameters and returns their product.

function multiplyThreeNumbers(num1, num2, num3) {

return num1 \* num2 \* num3;

}

// 2. Create parameterized method to divide 2 numbers

// This function accepts two parameters and returns the quotient.

function divideTwoNumbers(dividend, divisor) {

// Add a check to prevent division by zero

if (divisor === 0) {

console.error("Error: Cannot divide by zero.");

return null;

}

return dividend / divisor;

}

// Example usage of the math functions:

console.log("1. Multiplication of 2, 5, and 10:", multiplyThreeNumbers(2, 5, 10)); // Output: 100

console.log("2. Division of 20 by 4:", divideTwoNumbers(20, 4)); // Output: 5

console.log("2. Division by zero:", divideTwoNumbers(10, 0)); // Output: Error message and null

// 3. Create a method to add background color to a div element

// This function takes the ID of a DOM element and a color string, then sets the background style.

function setElementBackgroundColor(elementId, color) {

const element = document.getElementById(elementId);

if (element) {

element.style.backgroundColor = color;

console.log(`3. Background color of element '${elementId}' set to '${color}'.`);

} else {

console.error(`3. Element with ID '${elementId}' not found.`);

}

}

// 4. Write function to generate random background color of body (hint: Math.random())

// This function generates a random hexadecimal color code (e.g., #RRGGBB) and applies it

// to the <body> element's background.

function setRandomBodyBackgroundColor() {

// Generate a random number (0 to 16777215) and convert it to a hex string.

// Ensure the hex string is 6 digits long by padding with zeros.

const randomColor = '#' + Math.floor(Math.random() \* 16777215).toString(16).padStart(6, '0');

document.body.style.backgroundColor = randomColor;

console.log(`4. Body background color set to random color: ${randomColor}`);

}

// Example usage for 3 and 4 (requires HTML elements to exist):

// We can call setRandomBodyBackgroundColor() here, but we need HTML for setElementBackgroundColor.

document.addEventListener('DOMContentLoaded', () => {

// We assume an HTML structure like <div id="myDiv"> exists for demonstration of task 3.

// For this example, we'll temporarily create a div element to demonstrate.

const tempDiv = document.createElement('div');

tempDiv.id = 'myDiv';

tempDiv.style.width = '100px';

tempDiv.style.height = '100px';

tempDiv.style.border = '1px solid black';

tempDiv.textContent = 'Demo Div';

document.body.appendChild(tempDiv);

setElementBackgroundColor('myDiv', 'lightblue');

setRandomBodyBackgroundColor();

});

// 5. Onload of your webpage, display modal (bootstrap 4 or 5)

// This requires jQuery and Bootstrap JS/CSS to be loaded in the HTML.

// The Bootstrap modal typically relies on jQuery to function correctly.

// We use the 'DOMContentLoaded' event listener to ensure the HTML is loaded before trying to show the modal.

document.addEventListener('DOMContentLoaded', () => {

// Check if Bootstrap and jQuery are available (optional but good practice)

if (typeof $ !== 'undefined' && typeof $.fn.modal !== 'undefined') {

console.log("5. Bootstrap and jQuery detected. Displaying modal on page load.");

// Use jQuery's Bootstrap modal method to show the modal by its ID

$('#myBootstrapModal').modal('show');

} else {

// If jQuery/Bootstrap are not loaded, we log an error.

console.warn("5. jQuery and/or Bootstrap JS not loaded. Modal cannot be displayed.");

}

});

// Note: To make this work, your HTML must include the necessary Bootstrap CSS and JS files,

// as well as the HTML structure for the modal itself, e.g.:

/\*

<div class="modal fade" id="myBootstrapModal" tabindex="-1" role="dialog" aria-labelledby="modalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="modalLabel">Welcome!</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

This modal is displayed automatically using JavaScript on page load.

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>

</div>

</div>

</div>

</div>

\*/

**TASK:15**

// 1. Check if a given number is a multiple of 3

// The modulo operator (%) returns the remainder of a division.

// If a number is a multiple of 3, the remainder when divided by 3 will be 0.

function isMultipleOfThree(number) {

if (typeof number !== 'number') {

console.error("Input must be a number.");

return false;

}

return number % 3 === 0;

}

// Example with 10900

const numberToCheck = 10900;

console.log(`1. Is ${numberToCheck} a multiple of 3?`, isMultipleOfThree(numberToCheck)); // Expected: false

// Example with a number that is a multiple of 3

const anotherNumber = 123; // 1 + 2 + 3 = 6, which is divisible by 3

console.log(`1. Is ${anotherNumber} a multiple of 3?`, isMultipleOfThree(anotherNumber)); // Expected: true

// 2. Check if a particular sub-word exists in a string

// The `String.prototype.includes()` method is the most straightforward way to

// determine whether one string may be found within another string. It returns true or false.

function subwordExists(mainString, subword) {

if (typeof mainString !== 'string' || typeof subword !== 'string') {

console.error("Inputs must be strings.");

return false;

}

return mainString.includes(subword);

}

// Example usage with 'js'

const mainPhrase = "i am learning js";

const subwordToFind = "js";

console.log(`\n2. Does '${subwordToFind}' exist in '${mainPhrase}'?`, subwordExists(mainPhrase, subwordToFind)); // Expected: true

// Example with a sub-word that does not exist

const subwordToFind2 = "python";

console.log(`2. Does '${subwordToFind2}' exist in '${mainPhrase}'?`, subwordExists(mainPhrase, subwordToFind2)); // Expected: false

// 3. Calculate "complex interest" based on your formula: ((p/r)\*t) / 100

// This function uses the `prompt()` method to take input from the user.

function calculateComplexInterest() {

// Prompt the user for input and convert the strings to numbers

const principal = parseFloat(prompt("Enter the principal amount (p):"));

const rate = parseFloat(prompt("Enter the rate (r):"));

const time = parseFloat(prompt("Enter the time (t):"));

// Check for valid number inputs and prevent division by zero

if (isNaN(principal) || isNaN(rate) || isNaN(time)) {

console.error("\n3. Invalid input. Please enter valid numbers.");

return null; // Return null or another value to indicate an error

}

if (rate === 0) {

console.error("\n3. Error: Rate (r) cannot be zero, as it will cause division by zero.");

return null;

}

// Apply the user's specific formula

const result = ((principal / rate) \* time) / 100;

// Display the result

console.log(`\n3. Calculating complex interest with p=${principal}, r=${rate}, t=${time}`);

console.log(` Result using your formula ((p/r)\*t) / 100 is: ${result}`);

return result;

}

// Note: The `prompt()` function is asynchronous and will pause script execution.

// To run this function, uncomment the line below and try it in a browser environment.

// calculateComplexInterest();

**TASK 16:**

// 1. Display even numbers up to a user-specified number (n)

// This function uses `prompt` to get a number from the user.

function displayEvenNumbersUpToN() {

const n = parseInt(prompt("1. Enter a number to display even numbers up to:"));

if (isNaN(n)) {

console.error("Invalid input. Please enter a valid number.");

return;

}

console.log(`\n1. Even numbers up to ${n}:`);

for (let i = 0; i <= n; i++) {

if (i % 2 === 0) {

console.log(i);

}

}

}

// 2. Check if a character is a vowel or a consonant

// This function prompts for a character and checks if it is a vowel.

function checkVowelOrConsonant() {

const char = prompt("2. Enter a single letter to check if it's a vowel or consonant:").toLowerCase();

// Check if the input is a single character

if (char.length !== 1 || !/[a-z]/.test(char)) {

console.error("Invalid input. Please enter a single letter.");

return;

}

const vowels = ['a', 'e', 'i', 'o', 'u'];

if (vowels.includes(char)) {

console.log(`\n2. The character '${char}' is a vowel.`);

} else {

console.log(`\n2. The character '${char}' is a consonant.`);

}

}

// 3. Count of even and odd numbers from 1 to 999

// This function uses a loop to iterate and count even and odd numbers.

function countEvenAndOddFrom1To999() {

let evenCount = 0;

let oddCount = 0;

for (let i = 1; i <= 999; i++) {

if (i % 2 === 0) {

evenCount++;

} else {

oddCount++;

}

}

console.log("\n3. Counting from 1 to 999:");

console.log(` Total Even numbers: ${evenCount}`);

console.log(` Total Odd numbers: ${oddCount}`);

}

// 4. Count occurrence of a particular character in a string (using loops)

// This function takes a string and a character, then iterates through the string

// to count the occurrences of that character.

function countCharacterOccurrence(str, char) {

let count = 0;

for (let i = 0; i < str.length; i++) {

if (str[i] === char) {

count++;

}

}

return count;

}

const exampleString = "hello";

const characterToFind = "l";

const count = countCharacterOccurrence(exampleString, characterToFind);

console.log(`\n4. The character '${characterToFind}' appears ${count} times in the string '${exampleString}'.`);

// 5. Sum and average of array elements

// This function takes an array, calculates the sum of its elements using a loop,

// and then determines the average.

function calculateSumAndAverage(arr) {

if (arr.length === 0) {

console.error("The array is empty.");

return { sum: 0, average: 0 };

}

let sum = 0;

for (let i = 0; i < arr.length; i++) {

sum += arr[i];

}

const average = sum / arr.length;

console.log(`\n5. Array: [${arr}]`);

console.log(` Sum of elements: ${sum}`);

console.log(` Average of elements: ${average}`);

return { sum, average };

}

const myArr = [1, 9, 8];

calculateSumAndAverage(myArr);

// 6. Find the largest number in an array (using loops)

// This function iterates through an array to find the largest number without using built-in methods like Math.max().

function findLargestNumber(arr) {

if (arr.length === 0) {

return "Array is empty.";

}

let largest = arr[0]; // Assume the first element is the largest

for (let i = 1; i < arr.length; i++) {

if (arr[i] > largest) {

largest = arr[i]; // Update largest if a bigger number is found

}

}

return largest;

}

const numbersArray = [10, 5, 25, 1, 30, 15];

const largestNumber = findLargestNumber(numbersArray);

console.log(`\n6. Array: [${numbersArray}]`);

console.log(` The largest number is: ${largestNumber}`);

// To run the interactive functions (1 and 2), you can uncomment the lines below in your browser console:

// displayEvenNumbersUpToN();

// checkVowelOrConsonant();

**TASK 17:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Display Cards in a Row</title>

<style>

/\* Basic styling for the container to hold cards \*/

.cards-container {

display: flex; /\* Enables Flexbox for row layout \*/

justify-content: space-around; /\* Distributes space between and around items \*/

align-items: flex-start; /\* Aligns items to the start of the cross-axis \*/

gap: 20px; /\* Space between cards \*/

padding: 20px;

background-color: #f0f0f0;

border-radius: 8px;

flex-wrap: wrap; /\* Allows cards to wrap to the next line on smaller screens \*/

}

/\* Styling for individual cards \*/

.card {

flex: 1; /\* Allows cards to grow and shrink \*/

min-width: 250px; /\* Minimum width for cards before wrapping \*/

max-width: 30%; /\* Max width, useful when combined with flex: 1 \*/

background-color: white;

border: 1px solid #ddd;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

padding: 20px;

text-align: center;

box-sizing: border-box; /\* Include padding and border in the element's total width and height \*/

}

.card h3 {

color: #333;

margin-bottom: 10px;

}

.card p {

color: #666;

font-size: 0.9em;

}

/\* Responsive adjustments \*/

@media (max-width: 768px) {

.card {

max-width: 45%; /\* Two cards per row on medium screens \*/

}

}

@media (max-width: 480px) {

.card {

max-width: 90%; /\* One card per row on small screens \*/

}

.cards-container {

flex-direction: column; /\* Stack cards vertically \*/

align-items: center; /\* Center stacked cards \*/

}

}

</style>

</head>

<body>

<div id="cards-wrapper" class="cards-container">

</div>

<script>

// Get the container element where cards will be placed

const cardsWrapper = document.getElementById('cards-wrapper');

// Data for the cards (optional, for demonstration)

const cardData = [

{ title: 'Card One', content: 'This is the content for the first card. It contains some descriptive text.' },

{ title: 'Card Two', content: 'Here is some information for the second card. It demonstrates dynamic creation.' },

{ title: 'Card Three', content: 'And finally, the third card with its unique content to fill the row.' }

];

// Loop 3 times to create 3 cards

for (let i = 0; i < 3; i++) {

// Create a new div element for each card

const cardElement = document.createElement('div');

cardElement.className = 'card'; // Add the 'card' class for styling

// Create and append a title (h3) for the card

const cardTitle = document.createElement('h3');

cardTitle.textContent = cardData[i].title; // Use data from array

cardElement.appendChild(cardTitle);

// Create and append a paragraph (p) for the card's content

const cardContent = document.createElement('p');

cardContent.textContent = cardData[i].content; // Use data from array

cardElement.appendChild(cardContent);

// Append the created card to the main container

cardsWrapper.appendChild(cardElement);

}

console.log("3 cards have been dynamically added to the webpage.");

</script>

</body>

</html>

**TASK 18:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>User Registration Form</title>

<!-- Tailwind CSS CDN for styling -->

<script src="https://cdn.tailwindcss.com"></script>

<style>

/\* Custom styles for error messages \*/

.error-message {

color: #ef4444; /\* Tailwind red-500 \*/

font-size: 0.875rem; /\* text-sm \*/

margin-top: 0.25rem; /\* mt-1 \*/

display: none; /\* Hidden by default \*/

}

.input-error {

border-color: #ef4444; /\* Tailwind red-500 \*/

}

</style>

</head>

<body class="font-sans bg-gray-100 flex items-center justify-center min-h-screen p-4">

<div class="bg-white p-8 rounded-lg shadow-lg max-w-md w-full">

<h2 class="text-2xl font-bold text-center mb-6 text-gray-800">Registration Form</h2>

<form id="registrationForm" class="space-y-4">

<!-- Name Field -->

<div>

<label for="name" class="block text-sm font-medium text-gray-700 mb-1">Name:</label>

<input type="text" id="name" name="name" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="Enter your full name">

<div id="nameError" class="error-message"></div>

</div>

<!-- Email Field -->

<div>

<label for="email" class="block text-sm font-medium text-gray-700 mb-1">Email:</label>

<input type="email" id="email" name="email" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="your.email@example.com">

<div id="emailError" class="error-message"></div>

</div>

<!-- Contact Field -->

<div>

<label for="contact" class="block text-sm font-medium text-gray-700 mb-1">Contact No.:</label>

<input type="tel" id="contact" name="contact" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="e.g., 1234567890" pattern="[0-9]{10}">

<div id="contactError" class="error-message"></div>

</div>

<!-- Age Field -->

<div>

<label for="age" class="block text-sm font-medium text-gray-700 mb-1">Age:</label>

<input type="number" id="age" name="age" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="e.g., 30">

<div id="ageError" class="error-message"></div>

</div>

<!-- Designation Field -->

<div>

<label for="designation" class="block text-sm font-medium text-gray-700 mb-1">Designation:</label>

<input type="text" id="designation" name="designation" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500" placeholder="e.g., Software Engineer">

<div id="designationError" class="error-message"></div>

</div>

<!-- Multiple Files Field -->

<div>

<label for="files" class="block text-sm font-medium text-gray-700 mb-1">Upload Documents (Multiple):</label>

<input type="file" id="files" name="files" multiple class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500 file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-blue-50 file:text-blue-700 hover:file:bg-blue-100">

<div id="filesError" class="error-message"></div>

</div>

<!-- Date of Birth Field -->

<div>

<label for="dob" class="block text-sm font-medium text-gray-700 mb-1">Date of Birth:</label>

<input type="date" id="dob" name="dob" class="w-full p-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500">

<div id="dobError" class="error-message"></div>

</div>

<!-- Submit Button -->

<button type="submit" class="w-full bg-blue-600 text-white p-3 rounded-md font-semibold hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:ring-offset-2">

Submit Form

</button>

<!-- Success Message Display -->

<div id="successMessage" class="hidden bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded relative" role="alert">

<strong class="font-bold">Success!</strong>

<span class="block sm:inline">Form submitted successfully.</span>

</div>

</form>

</div>

<script>

document.addEventListener('DOMContentLoaded', () => {

const form = document.getElementById('registrationForm');

// Helper function to display error messages

function displayError(element, message) {

const errorDiv = document.getElementById(element.id + 'Error');

element.classList.add('input-error');

errorDiv.textContent = message;

errorDiv.style.display = 'block';

}

// Helper function to clear error messages

function clearError(element) {

const errorDiv = document.getElementById(element.id + 'Error');

element.classList.remove('input-error');

errorDiv.textContent = '';

errorDiv.style.display = 'none';

}

// Validation functions for each field

function validateName() {

const nameInput = document.getElementById('name');

const nameValue = nameInput.value.trim();

if (nameValue === '') {

displayError(nameInput, 'Name is required.');

return false;

}

// Regex to allow letters, spaces, hyphens, and apostrophes

if (!/^[a-zA-Z\s'-]+$/.test(nameValue)) {

displayError(nameInput, 'Name can only contain letters, spaces, hyphens, and apostrophes.');

return false;

}

clearError(nameInput);

return true;

}

function validateEmail() {

const emailInput = document.getElementById('email');

const emailValue = emailInput.value.trim();

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (emailValue === '') {

displayError(emailInput, 'Email is required.');

return false;

}

if (!emailRegex.test(emailValue)) {

displayError(emailInput, 'Please enter a valid email address.');

return false;

}

clearError(emailInput);

return true;

}

function validateContact() {

const contactInput = document.getElementById('contact');

const contactValue = contactInput.value.trim();

// Regex for 10 digits only

const contactRegex = /^[0-9]{10}$/;

if (contactValue === '') {

displayError(contactInput, 'Contact number is required.');

return false;

}

if (!contactRegex.test(contactValue)) {

displayError(contactInput, 'Please enter a 10-digit contact number.');

return false;

}

clearError(contactInput);

return true;

}

function validateAge() {

const ageInput = document.getElementById('age');

const ageValue = parseInt(ageInput.value.trim(), 10);

if (isNaN(ageValue) || ageInput.value.trim() === '') {

displayError(ageInput, 'Age is required and must be a number.');

return false;

}

if (ageValue < 18 || ageValue > 120) {

displayError(ageInput, 'Age must be between 18 and 120.');

return false;

}

clearError(ageInput);

return true;

}

function validateDesignation() {

const designationInput = document.getElementById('designation');

const designationValue = designationInput.value.trim();

if (designationValue === '') {

displayError(designationInput, 'Designation is required.');

return false;

}

clearError(designationInput);

return true;

}

function validateFiles() {

const filesInput = document.getElementById('files');

// filesInput.files is a FileList object

if (filesInput.files.length === 0) {

displayError(filesInput, 'Please select at least one file.');

return false;

}

clearError(filesInput);

return true;

}

function validateDOB() {

const dobInput = document.getElementById('dob');

const dobValue = dobInput.value; // Value is in 'YYYY-MM-DD' format

if (dobValue === '') {

displayError(dobInput, 'Date of Birth is required.');

return false;

}

const dobDate = new Date(dobValue);

const today = new Date();

let age = today.getFullYear() - dobDate.getFullYear();

const monthDifference = today.getMonth() - dobDate.getMonth();

if (monthDifference < 0 || (monthDifference === 0 && today.getDate() < dobDate.getDate())) {

age--; // Decrement age if birthday hasn't occurred yet this year

}

if (age < 18) {

displayError(dobInput, 'You must be at least 18 years old.');

return false;

}

clearError(dobInput);

return true;

}

// Event listener for form submission

form.addEventListener('submit', function(event) {

event.preventDefault(); // Prevent default form submission

// Clear any previous success message

document.getElementById('successMessage').classList.add('hidden');

// Run all validation functions

const isNameValid = validateName();

const isEmailValid = validateEmail();

const isContactValid = validateContact();

const isAgeValid = validateAge();

const isDesignationValid = validateDesignation();

const isFilesValid = validateFiles();

const isDobValid = validateDOB();

// Check if all fields are valid

if (isNameValid && isEmailValid && isContactValid && isAgeValid && isDesignationValid && isFilesValid && isDobValid) {

// If all valid, you can process the form data here

console.log('Form is valid and ready for submission!');

// Example: Display success message

document.getElementById('successMessage').classList.remove('hidden');

// In a real application, you would send this data to a server:

// const formData = new FormData(form);

// fetch('/submit-form', { method: 'POST', body: formData })

// .then(response => response.json())

// .then(data => console.log(data))

// .catch(error => console.error('Error:', error));

// Optional: Reset the form after successful submission

// form.reset();

// Clear all error messages after reset (if any were showing)

// Array.from(form.elements).forEach(element => clearError(element));

} else {

console.log('Form has validation errors. Please correct them.');

}

});

// Add blur event listeners for real-time validation feedback (optional)

document.getElementById('name').addEventListener('blur', validateName);

document.getElementById('email').addEventListener('blur', validateEmail);

document.getElementById('contact').addEventListener('blur', validateContact);

document.getElementById('age').addEventListener('blur', validateAge);

document.getElementById('designation').addEventListener('blur', validateDesignation);

document.getElementById('files').addEventListener('change', validateFiles); // 'change' for file input

document.getElementById('dob').addEventListener('blur', validateDOB);

});

</script>

</body>

</html>

**TASK 19:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Movie Records & Callback Demo</title>

<!-- Tailwind CSS CDN for styling -->

<script src="https://cdn.tailwindcss.com"></script>

<style>

/\* Custom styles for the movie cards \*/

.movie-cards-container {

display: grid; /\* Use CSS Grid for responsive layout \*/

grid-template-columns: repeat(auto-fill, minmax(200px, 1fr)); /\* Responsive columns \*/

gap: 20px; /\* Space between cards \*/

padding: 20px;

background-color: #f8f8f8;

border-radius: 8px;

}

.movie-card {

background-color: white;

border: 1px solid #e0e0e0;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

overflow: hidden; /\* Ensures image corners are rounded \*/

text-align: center;

display: flex;

flex-direction: column;

justify-content: space-between; /\* Pushes footer to bottom \*/

transition: transform 0.2s ease-in-out;

}

.movie-card:hover {

transform: translateY(-5px); /\* Simple hover effect \*/

}

.movie-card-poster {

width: 100%;

height: 300px; /\* Fixed height for posters \*/

object-fit: cover; /\* Ensures image covers the area without distortion \*/

border-bottom: 1px solid #eee;

}

.movie-card-info {

padding: 15px;

flex-grow: 1; /\* Allows info section to take available space \*/

display: flex;

flex-direction: column;

justify-content: center;

}

.movie-card-title {

font-size: 1.1em;

font-weight: bold;

color: #333;

margin-bottom: 5px;

}

.movie-card-year {

font-size: 0.9em;

color: #666;

}

/\* Styling for the callback demo section \*/

#callback-demo-container {

padding: 20px;

margin-top: 30px;

background-color: #e6f7ff; /\* Light blue background \*/

border: 1px solid #b3e0ff;

border-radius: 8px;

text-align: center;

}

</style>

</head>

<body class="font-sans text-gray-800">

<div class="container mx-auto p-4">

<h1 class="text-3xl font-bold text-center my-6 text-blue-700">Movie Records</h1>

<p class="text-center text-sm text-gray-600 mb-4">

(Powered by OMDB API. Remember to replace `YOUR\_OMDB\_API\_KEY` in the JavaScript with your actual key.)

</p>

<div id="movieRecordsContainer" class="movie-cards-container">

<!-- Movie cards will be dynamically loaded here -->

<p class="text-center col-span-full text-gray-500" id="loadingMessage">Loading movies...</p>

</div>

<div id="callback-demo-container">

<h2 class="text-2xl font-bold mb-4 text-green-700">Callback Function Demonstration</h2>

<p class="mb-4 text-gray-700">Click the button below to enter some data, which will then be processed by a callback function.</p>

<button id="triggerCallbackBtn" class="bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded-full shadow-lg transition duration-300 ease-in-out">

Enter Data for Callback

</button>

<div id="callbackOutput" class="mt-4 p-3 bg-white border border-gray-300 rounded-md text-left">

<p class="text-gray-600">Callback Output will appear here:</p>

<pre id="callbackResult" class="whitespace-pre-wrap text-gray-800 font-mono text-sm"></pre>

</div>

</div>

</div>

<script>

document.addEventListener('DOMContentLoaded', async () => {

// --- 1. Display Movie Records in UI Cards ---

const OMDB\_API\_KEY = 'YOUR\_OMDB\_API\_KEY'; // IMPORTANT: Replace with your actual OMDB API key

const movieRecordsContainer = document.getElementById('movieRecordsContainer');

const loadingMessage = document.getElementById('loadingMessage');

/\*\*

\* Fetches movie data from the OMDB API and displays it as cards in the UI.

\* @param {string} searchTerm - The movie title to search for.

\*/

async function fetchAndDisplayMovies(searchTerm) {

if (OMDB\_API\_KEY === 'YOUR\_OMDB\_API\_KEY') {

movieRecordsContainer.innerHTML = `

<p class="text-center col-span-full text-red-600 font-bold">

Error: Please replace 'YOUR\_OMDB\_API\_KEY' in the JavaScript code with a valid key from omdbapi.com.

</p>

`;

return;

}

loadingMessage.style.display = 'block'; // Show loading message

movieRecordsContainer.innerHTML = ''; // Clear previous content

const apiUrl = `https://www.omdbapi.com/?s=${encodeURIComponent(searchTerm)}&apikey=${OMDB\_API\_KEY}`;

try {

const response = await fetch(apiUrl);

const data = await response.json();

if (data.Response === "True" && data.Search) {

data.Search.forEach(movie => {

// Create card element

const movieCard = document.createElement('div');

movieCard.className = 'movie-card';

// Create poster image

const poster = document.createElement('img');

poster.className = 'movie-card-poster';

// Use a placeholder image if no poster is available

poster.src = movie.Poster !== 'N/A' ? movie.Poster : `https://placehold.co/300x450/cccccc/333333?text=No+Poster`;

poster.alt = `${movie.Title} Poster`;

movieCard.appendChild(poster);

// Create info container

const infoDiv = document.createElement('div');

infoDiv.className = 'movie-card-info';

// Create title

const title = document.createElement('h3');

title.className = 'movie-card-title';

title.textContent = movie.Title;

infoDiv.appendChild(title);

// Create year

const year = document.createElement('p');

year.className = 'movie-card-year';

year.textContent = `Year: ${movie.Year}`;

infoDiv.appendChild(year);

movieCard.appendChild(infoDiv);

movieRecordsContainer.appendChild(movieCard);

});

} else {

movieRecordsContainer.innerHTML = `

<p class="text-center col-span-full text-gray-600">No movies found for "${searchTerm}".</p>

`;

}

} catch (error) {

console.error("Error fetching movie data:", error);

movieRecordsContainer.innerHTML = `

<p class="text-center col-span-full text-red-600">Failed to load movies. Please check your API key or network connection.</p>

`;

} finally {

loadingMessage.style.display = 'none'; // Hide loading message

}

}

// Call the function to display movies on page load with a default search term

fetchAndDisplayMovies('Avengers'); // You can change this search term

// --- 2. Create a function to get data from user and display in a callback function ---

const triggerCallbackBtn = document.getElementById('triggerCallbackBtn');

const callbackResultPre = document.getElementById('callbackResult');

/\*\*

\* Prompts the user for data and then passes that data to a provided callback function.

\* @param {function(string): void} callback - The function to call with the user's data.

\*/

function getUserDataAndCall(callback) {

const userData = prompt("Please enter some data for the callback function:");

// Check if the user cancelled the prompt

if (userData !== null) {

callback(userData);

} else {

callbackResultPre.textContent = "User cancelled the input.";

}

}

/\*\*

\* A simple callback function to display the received data.

\* @param {string} data - The data received from the user.

\*/

function displayUserDataInUI(data) {

callbackResultPre.textContent = `Received data: "${data}"`;

console.log("Callback executed with data:", data);

}

// Attach event listener to the button to trigger the callback demo

triggerCallbackBtn.addEventListener('click', () => {

getUserDataAndCall(displayUserDataInUI);

});

});

</script>

</body>

</html>

**TASK 20:**// --- Array Operations ---

// Sample array for tasks 1, 2, and 4

const numbersArray = [15, 22, 18, 30, 12, 45, 90, 89, 56, 45];

console.log("Original Array:", numbersArray);

// 1. Get the value of the first element in an array that has a value greater than 20

// The `find()` method returns the value of the first element in the array that satisfies the provided testing function.

const firstElementGreaterThan20 = numbersArray.find(element => element > 20);

console.log("1. First element greater than 20:", firstElementGreaterThan20); // Expected output: 22

// 2. Get the value of the first element in an array that has a value less than 20

const firstElementLessThan20 = numbersArray.find(element => element < 20);

console.log("2. First element less than 20:", firstElementLessThan20); // Expected output: 15

// 3. Filter data based on a property (id) in an array of objects

const users = [

{ id: 101, name: 'Alice', age: 30 },

{ id: 102, name: 'Bob', age: 25 },

{ id: 103, name: 'Charlie', age: 35 },

{ id: 104, name: 'David', age: 40 }

];

// To find a single object by a specific ID, `find()` is the most efficient method.

// We will find the user with the id 103 as an example.

const userWithId103 = users.find(user => user.id === 103);

console.log("3. User found with ID 103:", userWithId103);

// 4. Check if an element is odd or even in an array [90, 89, 56, 45]

// This function iterates through the array and checks each number.

function checkOddOrEvenInArray(arr) {

console.log("\n4. Checking if elements in array are odd or even:");

arr.forEach(number => {

if (number % 2 === 0) {

console.log(` ${number} is an even number.`);

} else {

console.log(` ${number} is an odd number.`);

}

});

}

const customArray = [90, 89, 56, 45];

checkOddOrEvenInArray(customArray);

// --- Class Definitions ---

// 5. Create a class Book

// A class is a blueprint for creating objects with specific properties and methods.

class Book {

constructor(bookType, pages, pageType, author) {

this.bookType = bookType; // Property to store the type of book

this.numberOfPages = pages;

this.typeOfPages = pageType;

this.author = author;

}

// Method to return the type of book (as requested)

type\_of\_book() {

return this.bookType;

}

displayInfo() {

console.log(`\n5. Book Information:`);

console.log(` - Type of Book: ${this.type\_of\_book()}`);

console.log(` - Author: ${this.author}`);

console.log(` - Pages: ${this.numberOfPages}`);

console.log(` - Page Type: ${this.typeOfPages}`);

}

}

// Create an instance of the Book class

const myBook = new Book('Fantasy', 450, 'Glossy', 'J.R.R. Tolkien');

myBook.displayInfo();

// 6. Create a class Animal

class Animal {

constructor(gender, name, disease) {

this.gender = gender;

this.name = name;

this.disease = disease;

}

walk() {

console.log(`\n6. Animal action: ${this.name} is walking.`);

}

eat() {

console.log(` Animal action: ${this.name} is eating.`);

}

climb() {

console.log(` Animal action: ${this.name} is climbing.`);

}

displayDetails() {

console.log(` Animal Details:`);

console.log(` - Name: ${this.name}`);

console.log(` - Gender: ${this.gender}`);

console.log(` - Disease: ${this.disease || 'None'}`);

}

}

// Create an instance of the Animal class

const myPet = new Animal('Female', 'Luna', 'Arthritis');

myPet.displayDetails();

myPet.walk();

myPet.eat();

myPet.climb();

**MINI PROJECT: 2**

<https://guileless-faloodeh-62c790.netlify.app/>