Day 19 of Training

JavaScript: Understanding and Manipulating Arrays

Today's session introduced a fundamental data structure in JavaScript: **Arrays**. Arrays are essential for storing collections of related items, offering a structured and efficient way to manage multiple pieces of data under a single variable name.

What are Arrays?

We learned that an array is a linear collection of items, meaning elements are stored in a sequential order. Unlike objects, where data is stored in key-value pairs (using descriptive keys), array elements are accessed by their numerical **index**, starting from zero. This makes arrays ideal for scenarios where the order of items matters, or when dealing with homogeneous data (items of the same type), though JavaScript arrays can technically store different data types.

Key characteristics of arrays:

- **Collection of Items:** Arrays group multiple values together.
- **Indexed Access:** Each item in an array has an associated numerical index, starting from 0. For an array with N elements, the indices range from 0 to N-1.
- **Creation:** Arrays are created using square brackets ([]) and items are separated by commas (e.g., let fruits = ["apple", "banana", "cherry"];).
- **length Property:** Like strings, arrays have a built-in length property that returns the total number of items in the array.
- **Mutable Nature:** A crucial distinction from strings is that arrays are **mutable**. This means that once an array is created, its contents (elements) can be changed directly, unlike strings which require creating a new string for any modification.

Accessing and Modifying Array Elements

We practiced accessing individual array elements using their index within square brackets (e.g., marks[0] to get the first mark). Attempting to access an index that doesn't exist in the array results in undefined. Critically, we learned how to modify array elements directly by assigning a new value to a specific index (e.g., marks[0] = 90;).

Looping Over Arrays

To perform operations on every item in an array, we revisited loops. We focused on using for loops and for-of loops for efficient iteration:

- **for Loop:** We used a traditional for loop, iterating from index 0 up to (but not including) array.length. This provides direct access to the index, allowing for both reading and modifying elements.
- **for-of Loop:** This more modern loop simplifies iteration by directly providing each *value* of the array in successive iterations. It's concise and readable for simply accessing elements.

Essential Array Methods

The session introduced several built-in array methods that provide powerful functionalities for manipulating arrays:

- **push():** Adds one or more elements to the *end* of an array and returns the new length of the array. This method modifies the original array.
- **pop():** Removes the *last* element from an array and returns that removed element. This method also modifies the original array.
- **toString():** Converts an array into a single string, with array elements separated by commas. This method *does not* change the original array; it returns a new string.
- **concat():** Joins two or more arrays and returns a *new* array containing all the combined elements. The original arrays remain unchanged.
- **unshift():** Adds one or more elements to the *beginning* of an array and returns the new length of the array. This method modifies the original array.
- **shift():** Removes the *first* element from an array and returns that removed element. This method modifies the original array.
- **slice(start, end):** Returns a *new* array containing a portion of the original array, from the start index up to (but not including) the end index. This method *does not* modify the original array. It's useful for creating shallow copies or extracting sub-arrays.

- **splice(startIndex, deleteCount, item1, item2, ...):** A highly versatile method that can **add, remove, and replace** elements in an array. It modifies the original array and returns an array of the deleted elements.
    - startIndex: The index at which to start changing the array.
    - deleteCount: The number of elements to remove from startIndex.
    - item1, item2, ...: (Optional) Elements to add to the array at startIndex.

Practice Exercises:

1. **Calculate Average Marks:** Given an array of student marks (e.g., [85, 97, 44, 37, 76, 60]), write a program to calculate and print the average marks of the class.
2. **Apply Discount:** You have an array representing the prices of five items (e.g., [250, 645, 300, 900, 50]). Each item has a 10% discount. Write a program to update the array with the final prices after applying the discount to each item.