Array Manipulation in JavaScript

Array manipulation in JavaScript involves performing various operations on arrays to modify, add, remove, or transform their elements. JavaScript provides a rich set of built-in methods for array manipulation. Here are some common array manipulation tasks –

1. **Creating Arrays:** You can create an array in JavaScript using square brackets **[]** or the **Array** constructor.

```
var fruits = ['apple', 'banana', 'cherry'];

var numbers = new Array(1, 2, 3, 4, 5);
```

2. **Accessing Elements:** You can access elements in an array using their index, starting from 0.

```
var firstFruit = fruits[0]; // 'apple'
```

3. **Adding Elements:**

- **push()**: Adds elements to the end of the array.
- **unshift()**: Adds elements to the beginning of the array.

```
fruits.push('orange'); // ['apple', 'banana', 'cherry', 'orange']

fruits.unshift('strawberry'); // ['strawberry', 'apple', 'banana', 'cherry', 'orange']
```

4. **Removing Elements:**

- **pop()**: Removes the last element from the array.
- **shift()**: Removes the first element from the array.

```
fruits.pop(); // Removes 'orange'

fruits.shift(); // Removes 'strawberry'
```

5. **Modifying Elements:** You can directly modify elements in an array using their index.

```
fruits[1] = 'grape'; // Changes 'banana' to 'grape'
```

6. **Iterating through Arrays:**

- **for** loop - Iterating through an array using a **for** loop is a common and straightforward way to access each element of the array by controlling variable to index the array.

```
var  fruits = ['apple', 'banana', 'cherry', 'orange'];

for (var i = 0; i < fruits.length; i++) {

 console.log(fruits[i]);
```

}

In this example:

- **fruits.length** is used to determine the length of the array, which represents the number of elements in the array.
- The **for** loop initializes a loop control variable **i** to 0.
- The loop condition **i < fruits.length** ensures that the loop runs until **i** is less than the length of the array.
- Inside the loop, **fruits[i]** is used to access the current element at the index **i**, and **console.log** is used to print it.

As the loop iterates, it prints each element of the **fruits** array to the console.

- **forEach()** method - The **forEach** loop is a higher-order function in JavaScript that provides a more elegant way to iterate through an array compared to a traditional **for** loop.

```
fruits.forEach((fruit) => {
        console.log(fruit);
});
```

The **forEach** loop automatically handles the iteration and doesn't require you to manually manage an index variable or the array's length.

These are some of the fundamental array manipulation operations in JavaScript. You will learn array manipulation in detail in upcoming weeks.

Coding Challenge with solution –

Problem Statement:

Create a JavaScript function that takes an array of numbers as input and returns a new array containing the squares of those numbers. You need to implement a reusable function that can calculate the squares for any array of numbers provided as an argument.

Example Usage:

```
var numbers = [1, 2, 3, 4, 5];
var squares = calculateSquares(numbers);
console.log(squares); // Output: [1, 4, 9, 16, 25]
```

Your task is to write a function **calculateSquares** that adheres to this problem statement and can be used to calculate the squares of numbers in various scenarios by passing different arrays of numbers as input.

Solution in JavaScript:

```
function calculateSquares(numbers) {
 var squares = [];
 for (var i = 0; i < numbers.length; i++) {
  var square = numbers[i] * numbers[i];
  squares.push(square);
 }
 return squares;
}
// Example usage:
var numbers = [1, 2, 3, 4, 5];
var squares = calculateSquares(numbers);
console.log(squares); // Output: [1, 4, 9, 16, 25]
```

In this implementation:

- We define a function **calculateSquares** that takes an array of **numbers** as its parameter.
- Inside the function, we create an empty array called **squares** to store the squared values.
- We use a **for** loop to iterate through each element of the **numbers** array.
- Inside the loop, we calculate the square of each number by multiplying it by itself (**numbers[i] * numbers[i]**) and store in the **square** variable and then push it into the **squares** array.
- Finally, we return the **squares** array as the result of the function.

You can call this function with any array of numbers, and it will compute and return the squares of those numbers.