Lesson 03 Demo 07

Implementing Array Mutability, Immutability, and Advanced Destructuring

Objective: To implement advanced array operations in JavaScript, including mutability, immutability, and destructuring for optimized code efficiency

Tools required: Visual Studio Code and Node.js

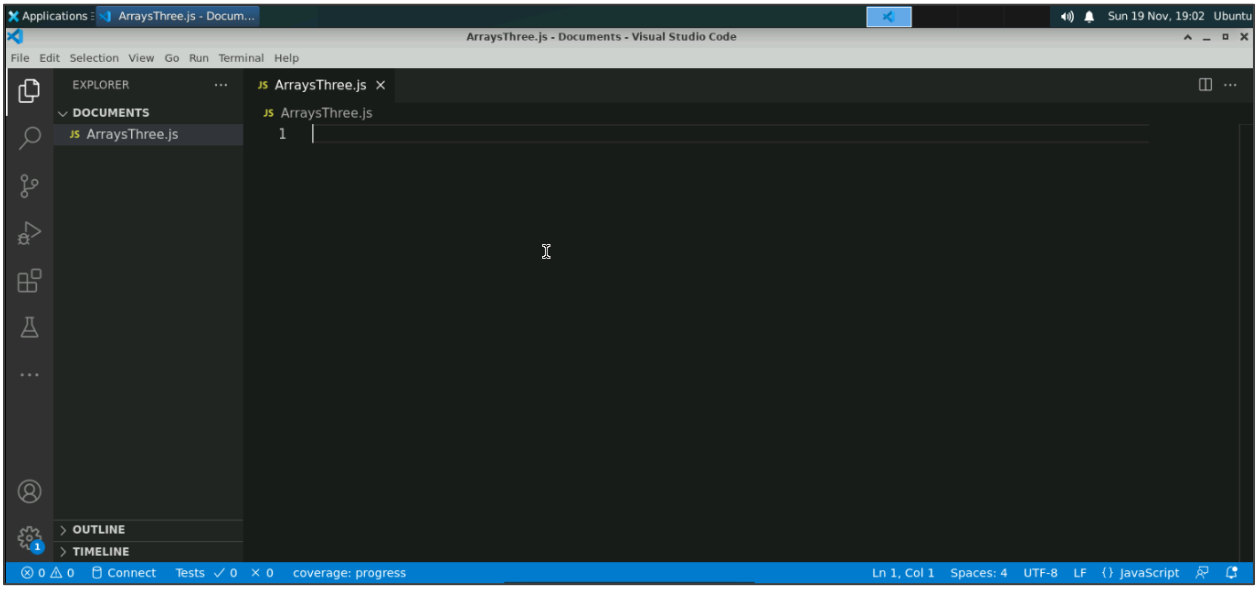
Prerequisites: A basic understanding of array mutability, immutability, and destructuring in JavaScript

Steps to be followed:

1. Create and execute the JS file

Step 1: Create and execute the JS file

1.1 Open the Visual Studio Code editor and create a JavaScript file named **ArraysThree.js**

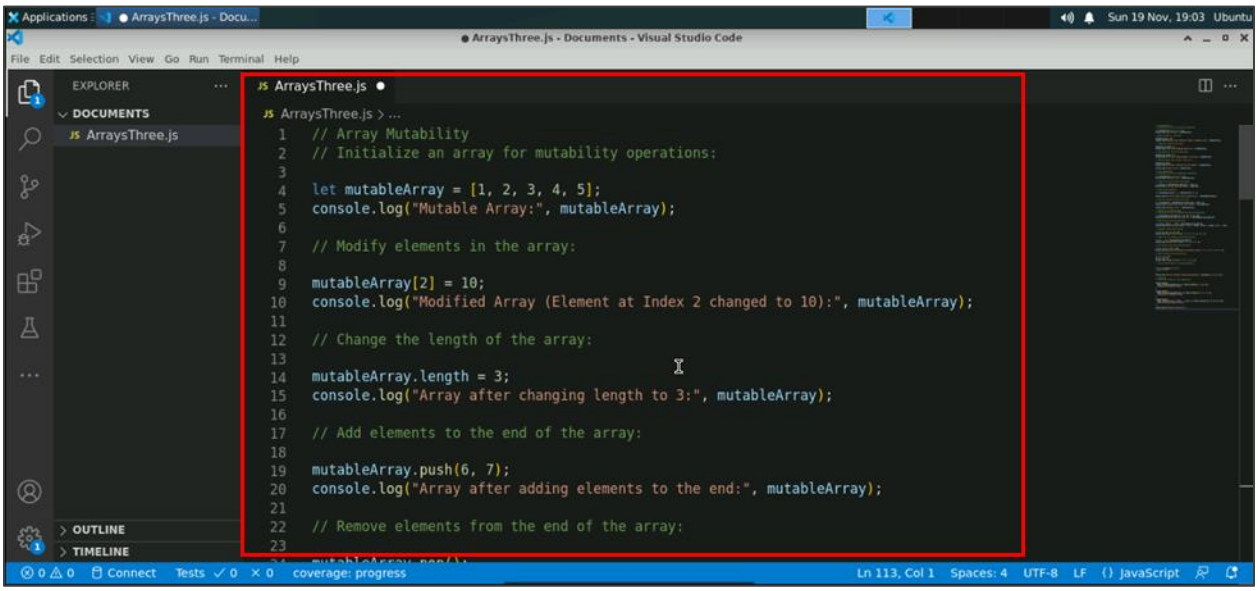


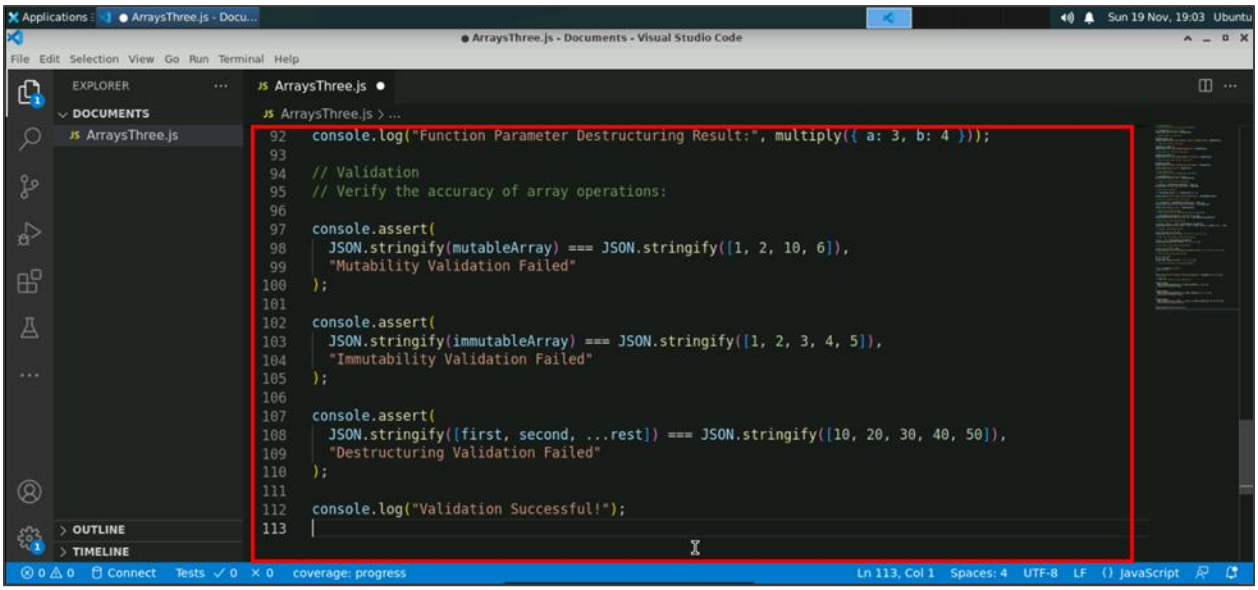
1.2 Enter the code below into the **ArraysThree.js** file:

```
// Array Mutability
// Initialize an array for mutability operations:

let mutableArray = [1, 2, 3, 4, 5];
console.log("Mutable Array:", mutableArray);

// Modify elements in the array:

mutableArray[2] = 10;
console.log("Modified Array (Element at Index 2 changed to 10):", mutableArray);

// Change the length of the array:

mutableArray.length = 3;
console.log("Array after changing length to 3:", mutableArray);

// Add elements to the end of the array:

mutableArray.push(6, 7);
console.log("Array after adding elements to the end:", mutableArray);

// Remove elements from the end of the array:

mutableArray.pop();
console.log("Array after removing the last element:", mutableArray);

console.log("Mutable Array:", mutableArray);

// Array Immutability
// Initialize an array for immutability operations:

let immutableArray = [1, 2, 3, 4, 5];
console.log("Immutable Array:", immutableArray);

// Create a new array using array methods:

let newArray = immutableArray.map(item => item * 2);
console.log("New Array using map method:", newArray);

// Avoid direct mutations:

let withoutModification = [...immutableArray, 6, 7];
```

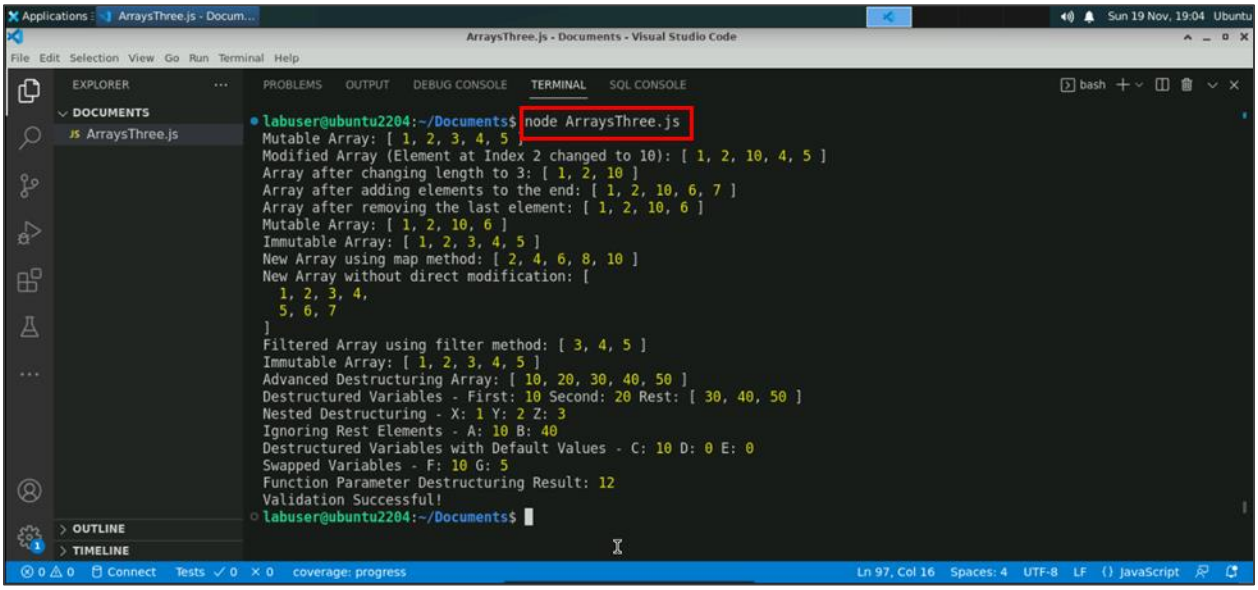
```
console.log("New Array without direct modification:", withoutModification);

// Use array methods returning new arrays:

let filteredArray = immutableArray.filter(item => item > 2);
console.log("Filtered Array using filter method:", filteredArray);

console.log("Immutable Array:", immutableArray);

// Advanced Array Destructuring
// Initialize an array for advanced destructuring operations:

let advancedDestructuringArray = [10, 20, 30, 40, 50];
console.log("Advanced Destructuring Array:", advancedDestructuringArray);

// Perform basic array destructuring:

let [first, second, ...rest] = advancedDestructuringArray;
console.log("Destructured Variables - First:", first, "Second:", second, "Rest:", rest);

// Implement nested destructuring:

let [x, [y, z]] = [1, [2, 3]];
console.log("Nested Destructuring - X:", x, "Y:", y, "Z:", z);

// Ignore the rest elements during destructuring:

let [a, , , b] = advancedDestructuringArray;
console.log("Ignoring Rest Elements - A:", a, "B:", b);

// Set default values during destructuring:

let [c, d = 0, e = 0] = [10];
console.log("Destructured Variables with Default Values - C:", c, "D:", d, "E:", e);

// Swap variables using destructuring:

let f = 5, g = 10;
[f, g] = [g, f];
console.log("Swapped Variables - F:", f, "G:", g);

// Utilize destructuring in function parameters:
```

```
function multiply({ a, b }) {
  return a * b;
}

console.log("Function Parameter Destructuring Result:", multiply({ a: 3, b: 4 }));

// Validation
// Verify the accuracy of array operations:

console.assert(
  JSON.stringify(mutableArray) === JSON.stringify([1, 2, 10, 6]),
  "Mutability Validation Failed"
);

console.assert(
  JSON.stringify(immutableArray) === JSON.stringify([1, 2, 3, 4, 5]),
  "Immutability Validation Failed"
);

console.assert(
  JSON.stringify([first, second, ...rest]) === JSON.stringify([10, 20, 30, 40, 50]),
  "Destructuring Validation Failed"
);
console.log("Validation Successful!");
```





1.3 Save the file and execute the command given below in the terminal:
node ArraysThree.js



The provided code shows how to change elements and lengths in an array. It emphasizes not changing things directly using methods like map and filter. Advanced array destructuring techniques show flexibility, and validation is included to ensure the operations are accurate.

By following the above steps, you have successfully demonstrated advanced array operations, including mutability, immutability, and destructuring, to ensure precise and efficient programming.