.

Lesson 02 Demo 06

Implementing CRUD Operations on a Doubly Linked List

Objective: To create a doubly linked list in JavaScript with CRUD functionalities such as node addition, traversal, value modification, and deletion to enhance your understanding of bidirectional data structures

Tools required: Visual Studio Code (VS Code) and JavaScript

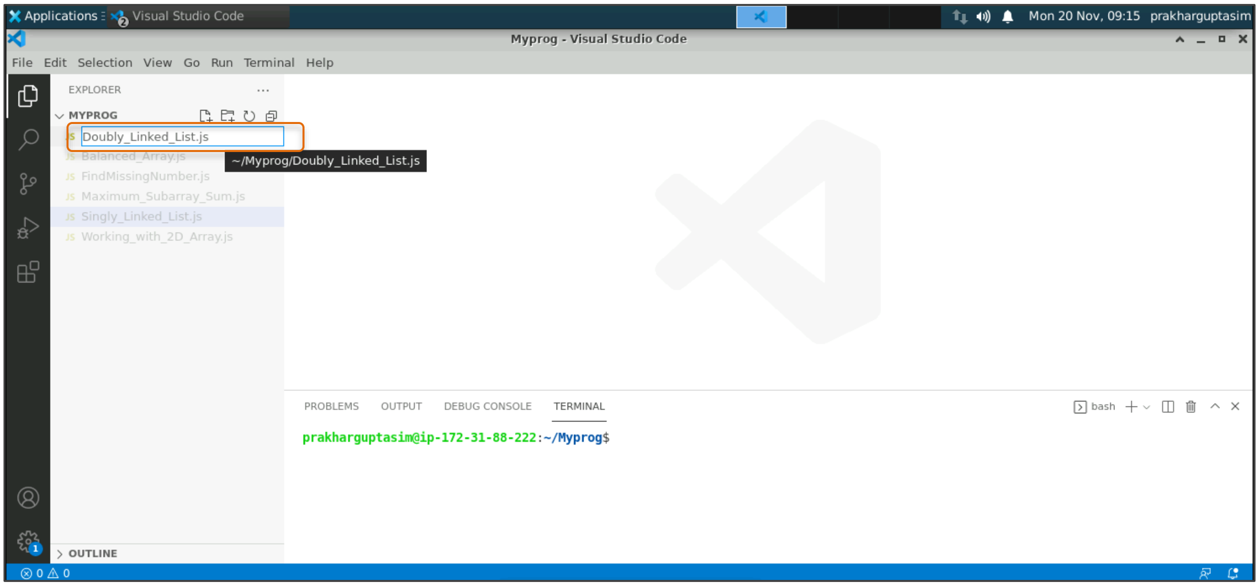
Prerequisites: Completion of Lesson 02 Demo 01

Steps to be followed:

1. Create a JavaScript file and execute it

Step 1: Create a JavaScript file and execute it

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



```
prakharguptasim@ip-172-31-88-222:~/Myprog$
```

.

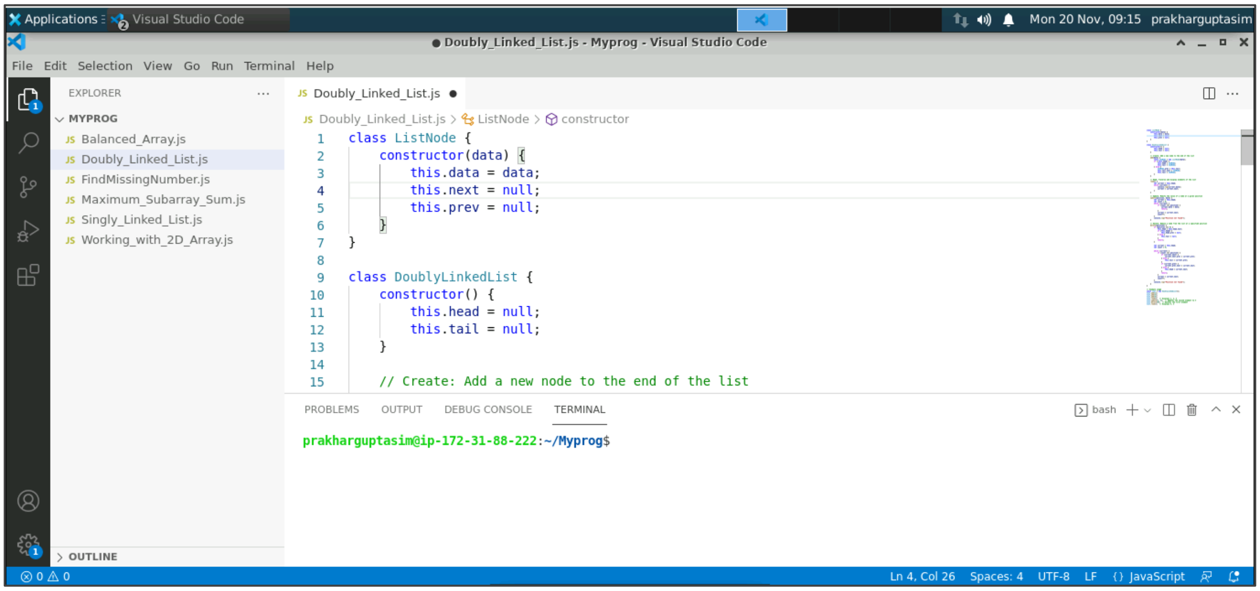
1.2 Add the following code to the file:

```
class ListNode {
  constructor(data) {
    this.data = data;
    this.next = null;
    this.prev = null;
  }
}

class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  // Create: Add a new node to the end of the list
  add(data) {
    const newNode = new ListNode(data);
    if (!this.head) {
      this.head = newNode;
      this.tail = newNode;
    } else {
      newNode.prev = this.tail;
      this.tail.next = newNode;
      this.tail = newNode;
    }
  }

  // Read: Traverse and display elements of the list
  read() {
    let current = this.head;
    while (current) {
      console.log(current.data);
      current = current.next;
    }
  }
```

```
// Update: Modify the value of a node at a given position
update(position, data) {
  let current = this.head;
  let count = 0;
  while (current) {
    if (count === position) {
      current.data = data;
      return;
    }
    current = current.next;
    count++;
  }
  console.log("Position not found");
}

// Delete: Remove a node from the list at a specified position
delete(position) {
  if (position === 0) {
    this.head = this.head.next;
    if (this.head) {
      this.head.prev = null;
    } else {
      this.tail = null;
    }
    return;
  }

  let current = this.head;
  let count = 0;

  while (current) {
    if (count === position) {
      if (current.next) {
        current.next.prev = current.prev;
      } else {
        this.tail = current.prev;
      }
      if (current.prev) {
```

.

```
          current.prev.next = current.next;
        } else {
          this.head = current.next;
        }
        return;
      }
      current = current.next;
      count++;
    }
    console.log("Position not found");
  }
}

// Example usage
const list = new DoublyLinkedList();
list.add(1);
list.add(2);
list.add(3);
list.read();  // Displays 1, 2, 3
list.update(1, 4);  // Updates the second element to 4
list.delete(0);  // Deletes the first element
list.read();  // Displays 4, 3
```

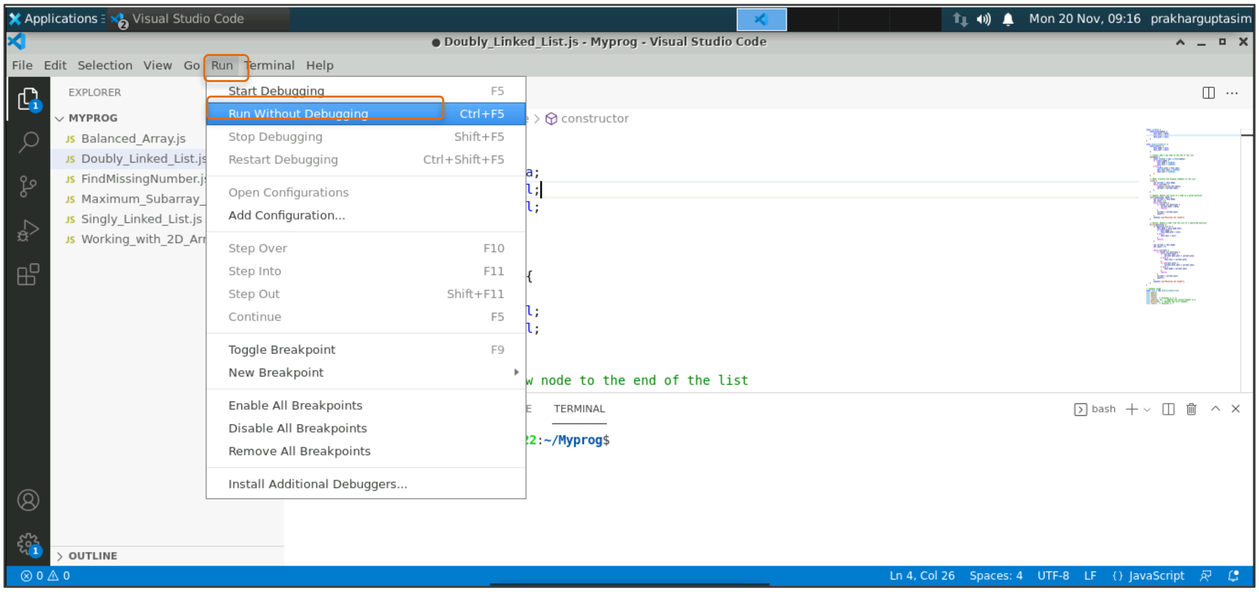


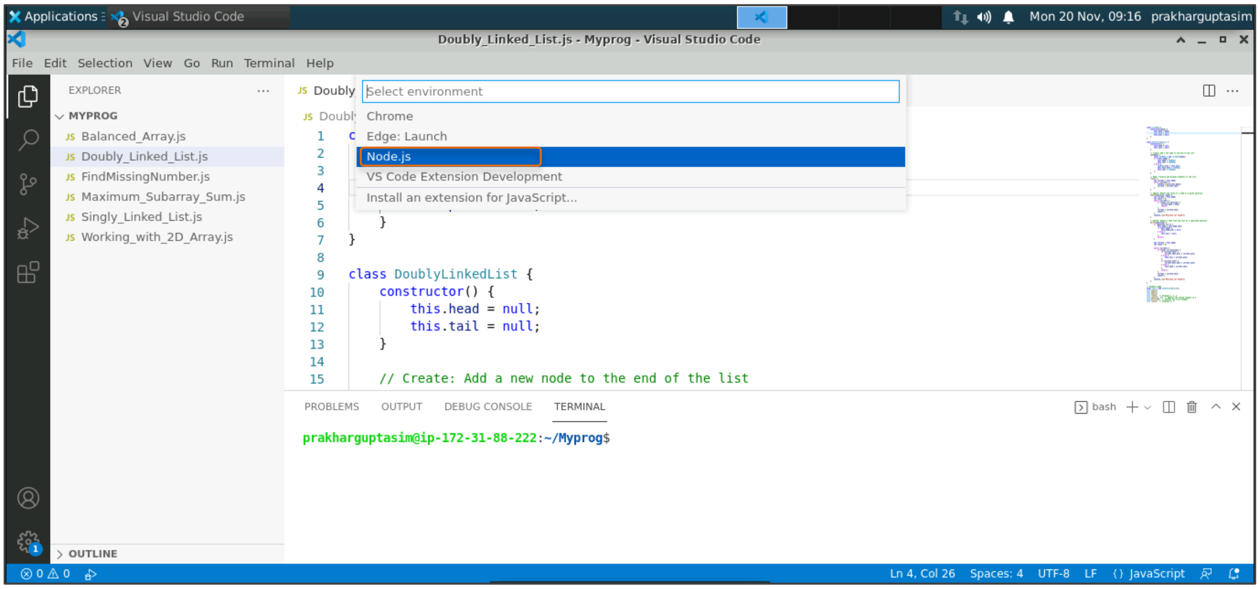
```
class ListNode {
    constructor(data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

class DoublyLinkedList {
    constructor() {
        this.head = null;
        this.tail = null;
    }

    // Create: Add a new node to the end of the list
```

```
prakharguptasim@ip-172-31-88-222:~/Myprog$
```

.

1.3 Click **Run** and then **Run Without Debugging**. Select **Node.js** to check the output in the DEBUG CONSOLE.



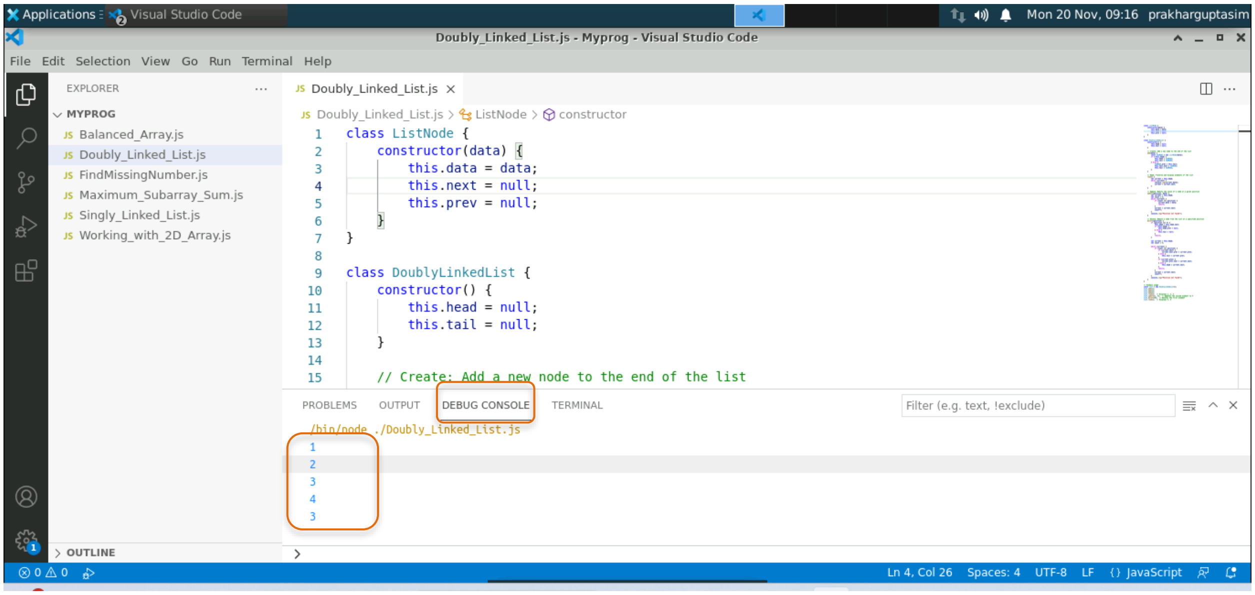


```
    }
}

class DoublyLinkedList {
    constructor() {
        this.head = null;
        this.tail = null;
    }

    // Create: Add a new node to the end of the list
```

```
prakharguptasim@ip-172-31-88-222:~/Myprog$
```

.

1.4 View the output in the **DEBUG CONSOLE** as shown below:



Applications
Visual Studio Code
Mon 20 Nov, 09:16 prakharguptasim
Doubly_Linked_List.js - Myprog - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
MYPROG
Balanced_Array.js
Doubly_Linked_List.js
FindMissingNumber.js
Maximum_Subarray_Sum.js
Singly_Linked_List.js
Working_with_2D_Array.js
OUTLINE
Doubly_Linked_List.js
Doubly_Linked_List.js > ListNode > constructor
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, !exclude)
Ln 4, Col 26 Spaces: 4 UTF-8 LF {} JavaScript

```
class ListNode {
    constructor(data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

class DoublyLinkedList {
    constructor() {
        this.head = null;
        this.tail = null;
    }

    // Create: Add a new node to the end of the list
```

```
/bin/node ./Doubly_Linked_List.js
1
2
3
4
3
```

By following these steps, you have successfully performed CRUD operations on a doubly linked list using JavaScript. This includes adding new nodes, traversing the list, updating node values, and deleting nodes, which are key operations for managing and manipulating bidirectional data structures.