.

Lesson 02 Demo 10

Detect a Cycle in a Linked List

Objective: To determine whether a linked list has a cycle in it. A linked list has a cycle if there is a node in the list that can be reached again by continuously following the next pointer

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

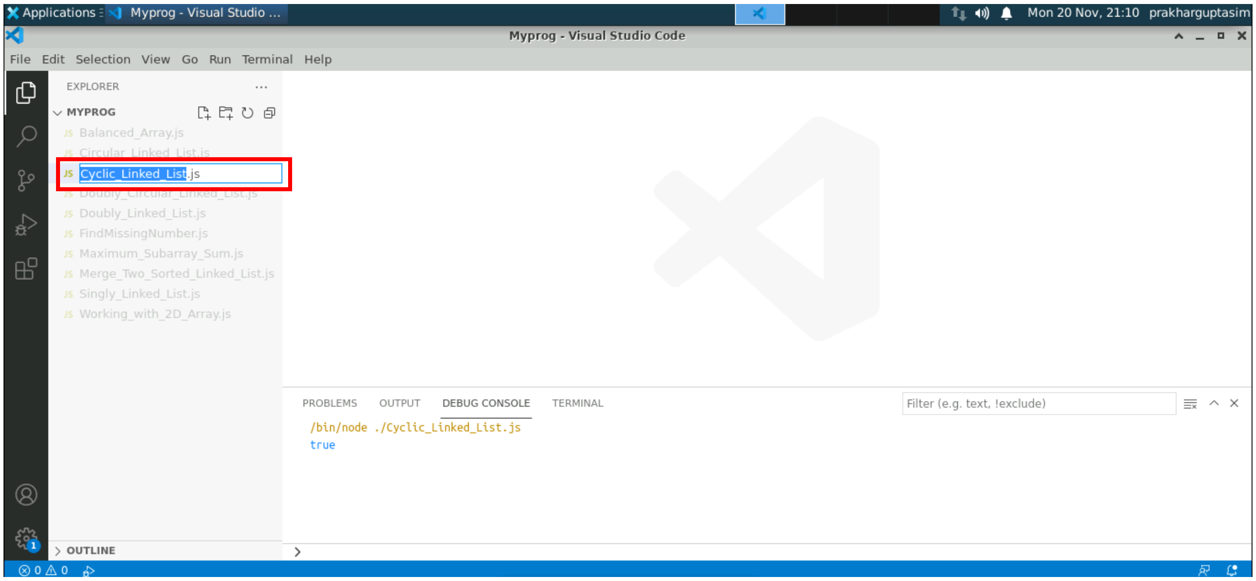
Prerequisites: Perform demo 01 of lesson 02

Steps to be followed:

1. Create and execute the JS file

Step 1: Create and execute the JS file

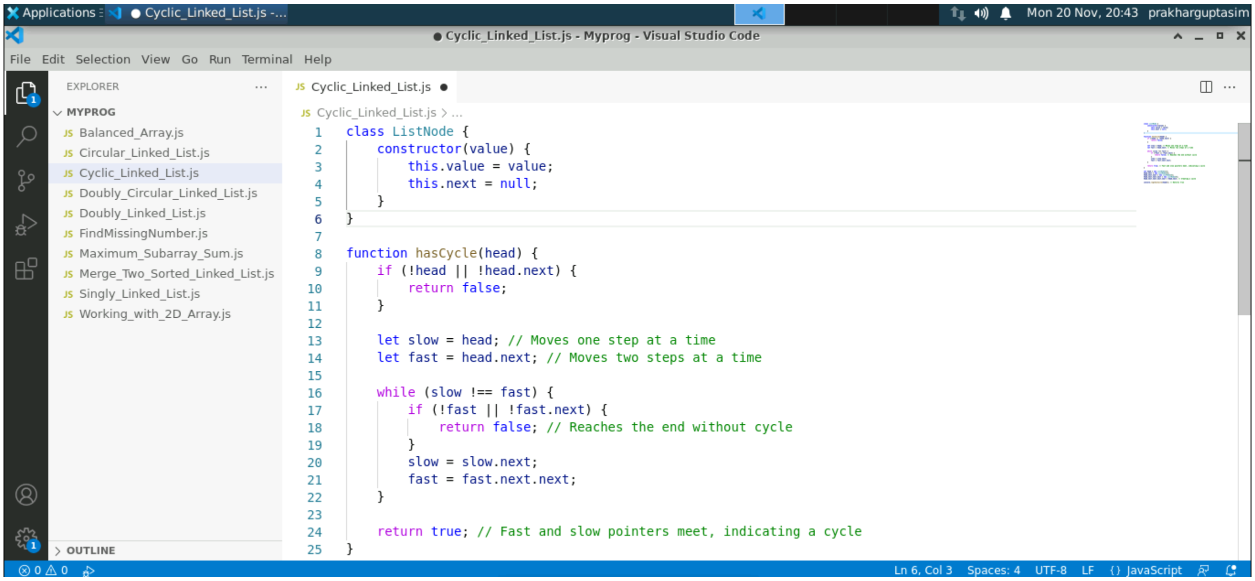
1.1 Create a JavaScript file named **Cyclic_Linked_List.js** as shown below:



```
/bin/node ./Cyclic_Linked_List.js
true
```

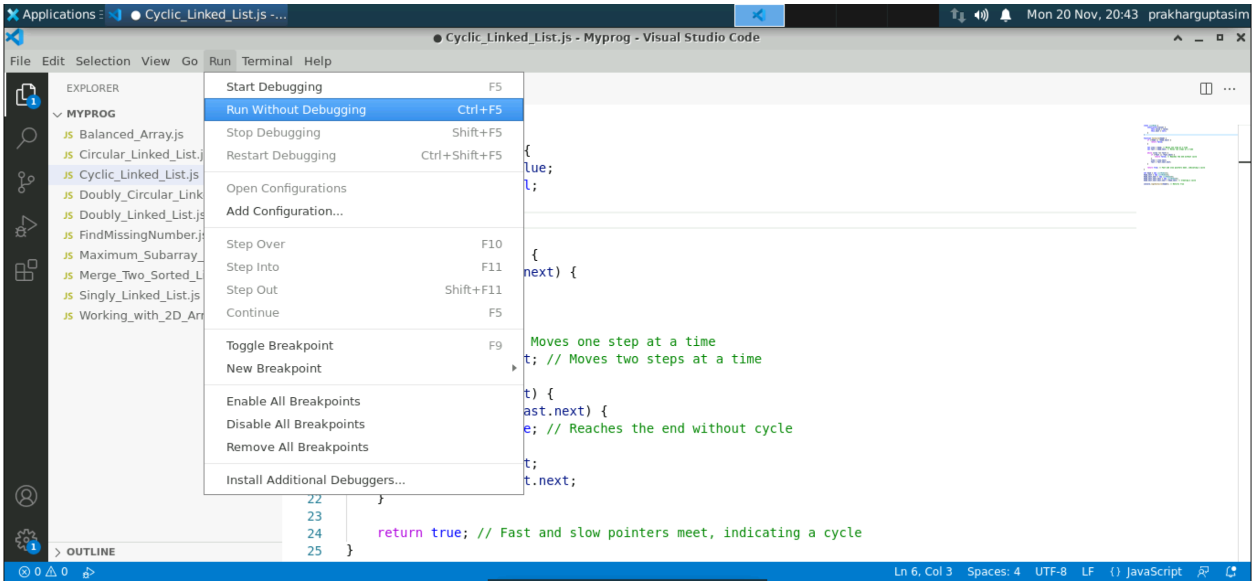
1.2 Write the code given below in the file created in step 1.1:

```
     class ListNode {
     constructor(value) {
      this.value = value;
      this.next = null;
      }
     }
    function hasCycle(head) {
    if (!head || !head.next) {
      return false;
     }
    let slow = head; // Moves one step at a time
    let fast = head.next; // Moves two steps at a time
    while (slow !== fast) {
      if (!fast || !fast.next) {
          return false; // Reaches the end without cycle
      }
      slow = slow.next;
      fast = fast.next.next;
  }

   return true; // Fast and slow pointers meet, indicating a cycle
  }
let head = new ListNode(1);
head.next = new ListNode(2);
head.next.next = new ListNode(3);
head.next.next.next = new ListNode(4);
head.next.next.next.next = head.next; // Creating a cycle
console.log(hasCycle(head)); // Returns true
```

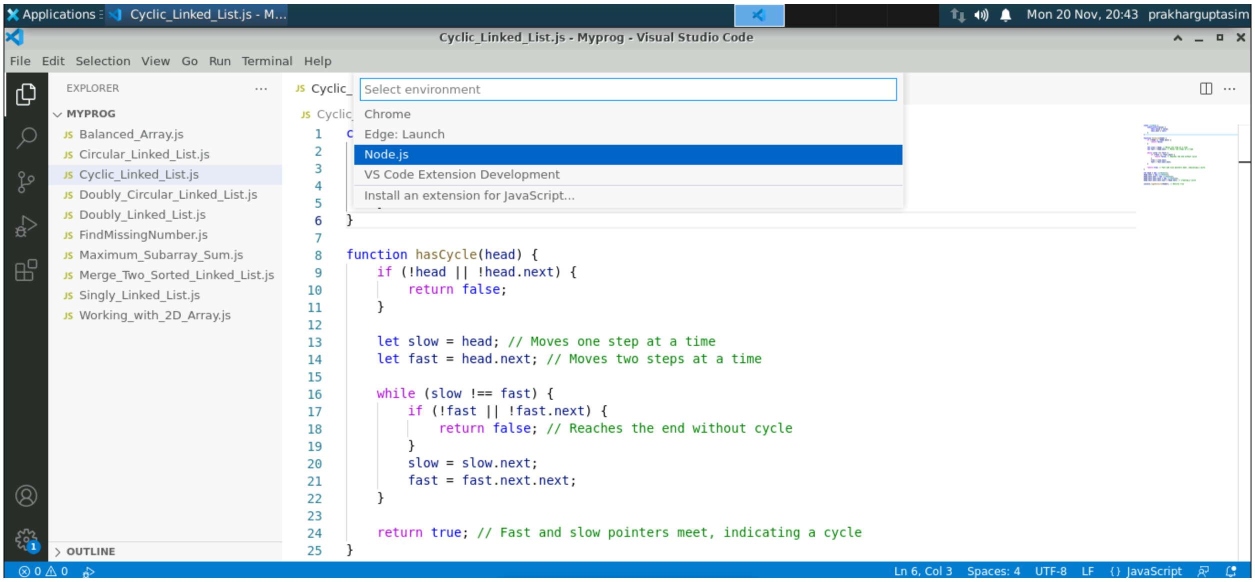

Applications
Cyclic_Linked_List.js - Myprog - Visual Studio Code
Mon 20 Nov, 20:43 prakharguptasim
File Edit Selection View Go Run Terminal Help
EXPLORER
MYPROG
Balanced_Array.js
Circular_Linked_List.js
Cyclic_Linked_List.js
Doubly_Circular_Linked_List.js
Doubly_Linked_List.js
FindMissingNumber.js
Maximum_Subarray_Sum.js
Merge_Two_Sorted_Linked_List.js
Singly_Linked_List.js
Working_with_2D_Array.js
OUTLINE
Ln 6, Col 3 Spaces: 4 UTF-8 LF {} JavaScript

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

function hasCycle(head) {
    if (!head || !head.next) {
        return false;
    }

    let slow = head; // Moves one step at a time
    let fast = head.next; // Moves two steps at a time

    while (slow !== fast) {
        if (!fast || !fast.next) {
            return false; // Reaches the end without cycle
        }
        slow = slow.next;
        fast = fast.next.next;
    }

    return true; // Fast and slow pointers meet, indicating a cycle
}
```

1.3 Save the code and click on **Run->Run Without Debugging** to check the output in the debug console



.

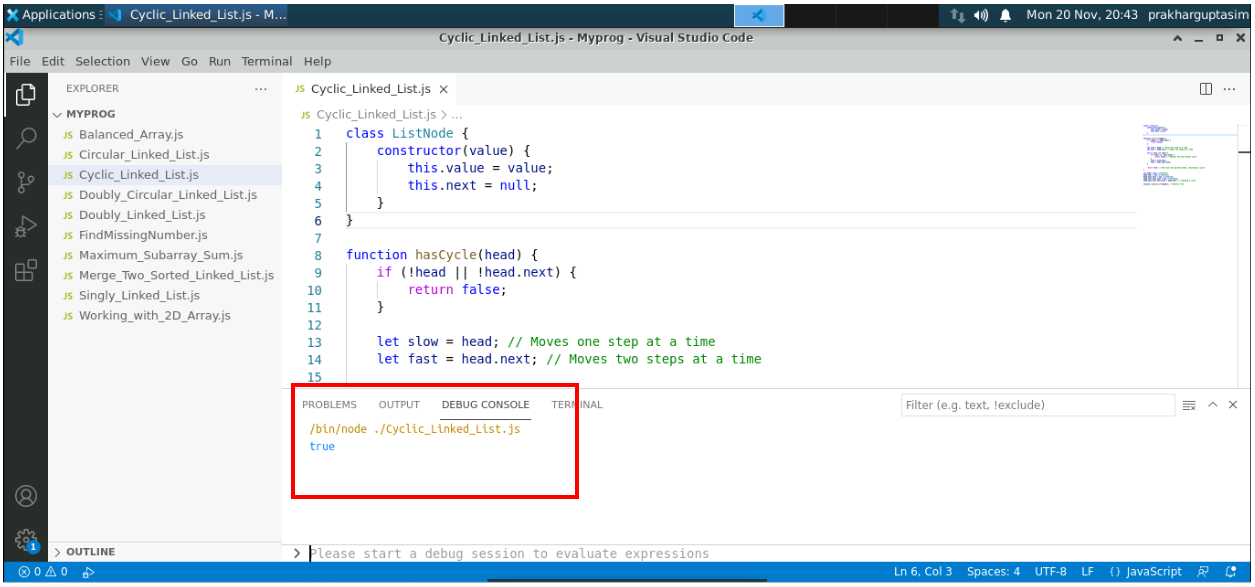


Select environment

- Chrome
- Edge: Launch
- Node.js
- VS Code Extension Development
- Install an extension for JavaScript...

```
}

function hasCycle(head) {
    if (!head || !head.next) {
        return false;
    }

    let slow = head; // Moves one step at a time
    let fast = head.next; // Moves two steps at a time

    while (slow !== fast) {
        if (!fast || !fast.next) {
            return false; // Reaches the end without cycle
        }
        slow = slow.next;
        fast = fast.next.next;
    }

    return true; // Fast and slow pointers meet, indicating a cycle
}
```

- Now you can see the output in the debug console as shown below:



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

function hasCycle(head) {
    if (!head || !head.next) {
        return false;
    }

    let slow = head; // Moves one step at a time
    let fast = head.next; // Moves two steps at a time
```

```
/bin/node ./Cyclic_Linked_List.js
true
```

Explanation:

1. **Two-Pointer Approach:** The function uses two pointers, slow and fast. The slow pointer advances one node at a time, while the fast pointer moves two nodes at a time.
2. **Cycle Detection Logic:** If there's a cycle, slow and fast will eventually meet inside the cycle. If there's no cycle, fast will reach the end of the list (null).
3. **Edge Case Handling:** Before starting the loop, the function checks if the list is empty or has only one node, in which case it returns false as a cycle is impossible.
4. **Return Value:** The function returns true if a cycle is detected (when slow and fast meet) and false otherwise (when fast reaches the end of the list).

By following the above steps, you have successfully determined whether a linked list contains a cycle using Floyd's Tortoise and Hare algorithm. This method is advantageous as it requires no additional memory and operates in linear time complexity.