Lesson 03 Demo 03

Implementing an AVL Tree

Objective: To demonstrate an AVL tree implementation in JavaScript using node insertion and self-balancing rotations to maintain height-balanced binary search tree properties

Tools required: Visual Studio Code and Node.js

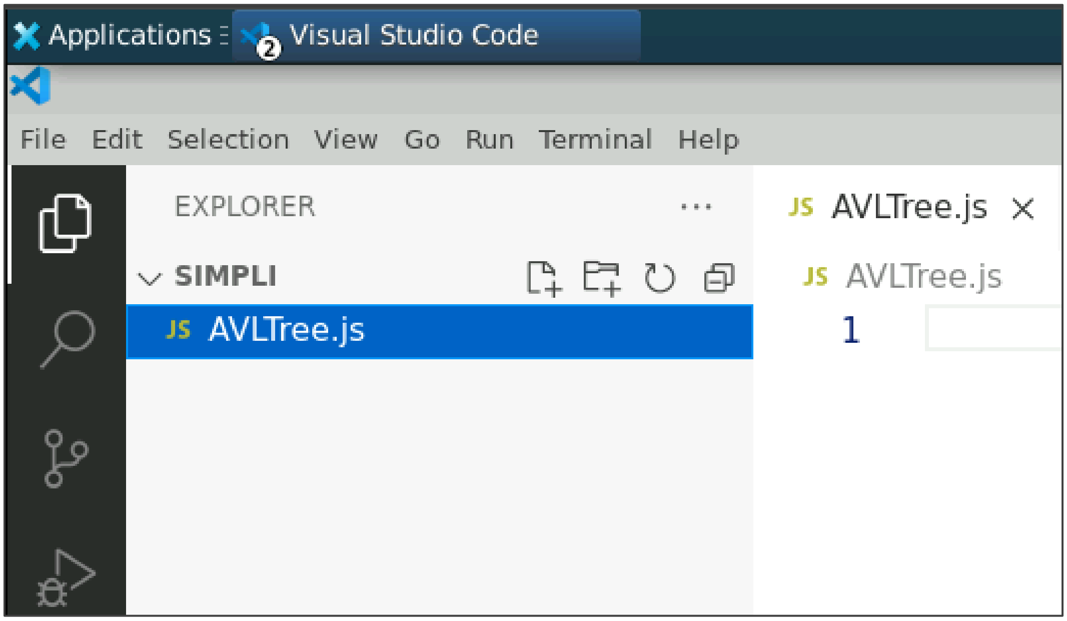
Prerequisites: A basic understanding of an AVL tree and proficiency in JavaScript

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 **AVLTree.js**

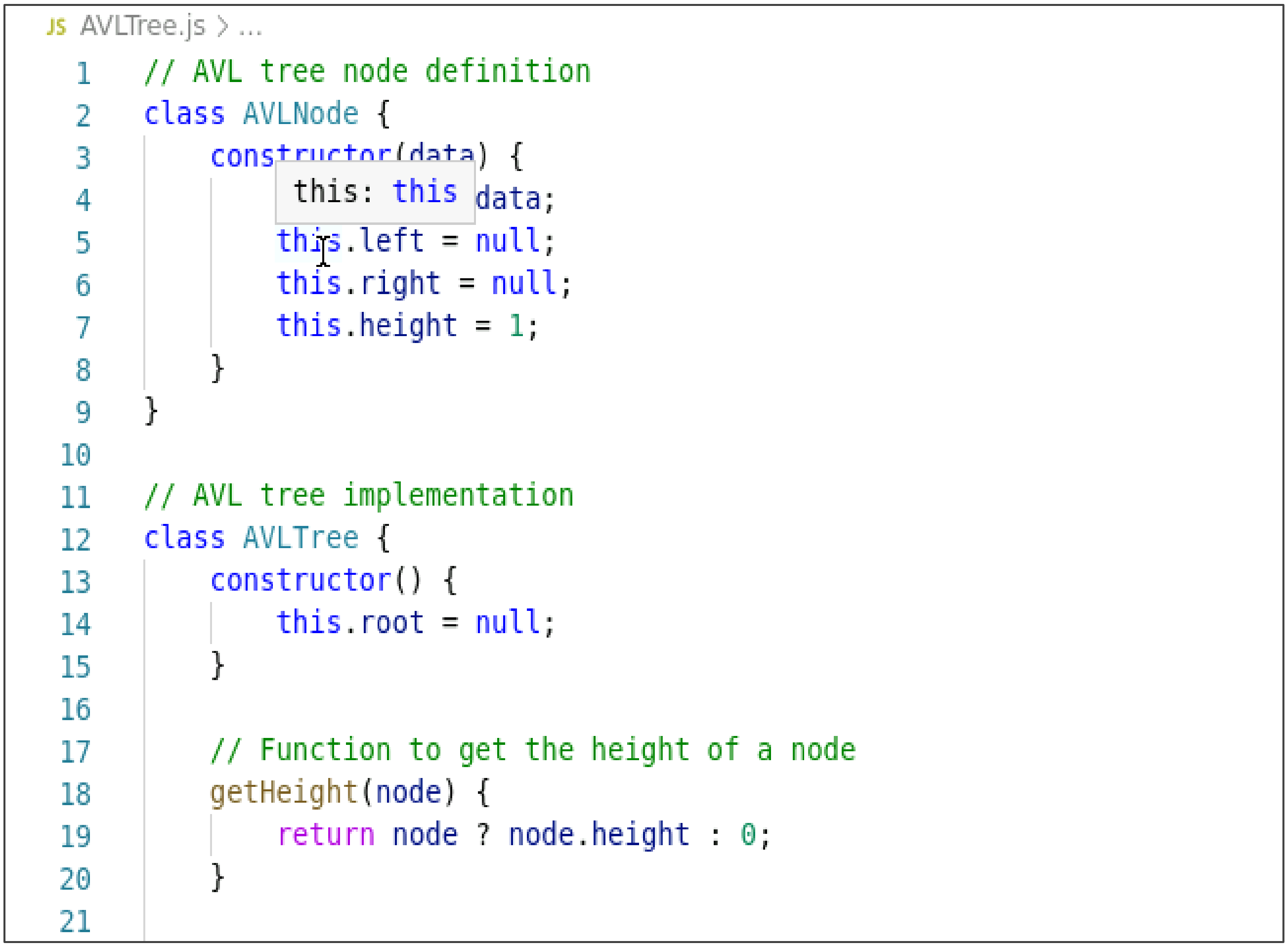


1.2 Add the following code to the file:

```
// AVL tree node definition
class AVLNode {
  constructor(data) {
    this.data = data;
    this.left = null;
    this.right = null;
    this.height = 1;
  }
}

// AVL tree implementation
class AVLTree {
  constructor() {
    this.root = null;
  }

  // Function to get the height of a node
  getHeight(node) {
    return node ? node.height : 0;
  }

  // Function to update the height of a node
  updateHeight(node) {
    if (node) {
      node.height = Math.max(this.getHeight(node.left), this.getHeight(node.right)) +
1;
    }
  }

  // Function to perform right rotation
  rotateRight(y) {
    const x = y.left;
    const T2 = x.right;

    x.right = y;
    y.left = T2;

    this.updateHeight(y);
    this.updateHeight(x);
    return x;
  }
```

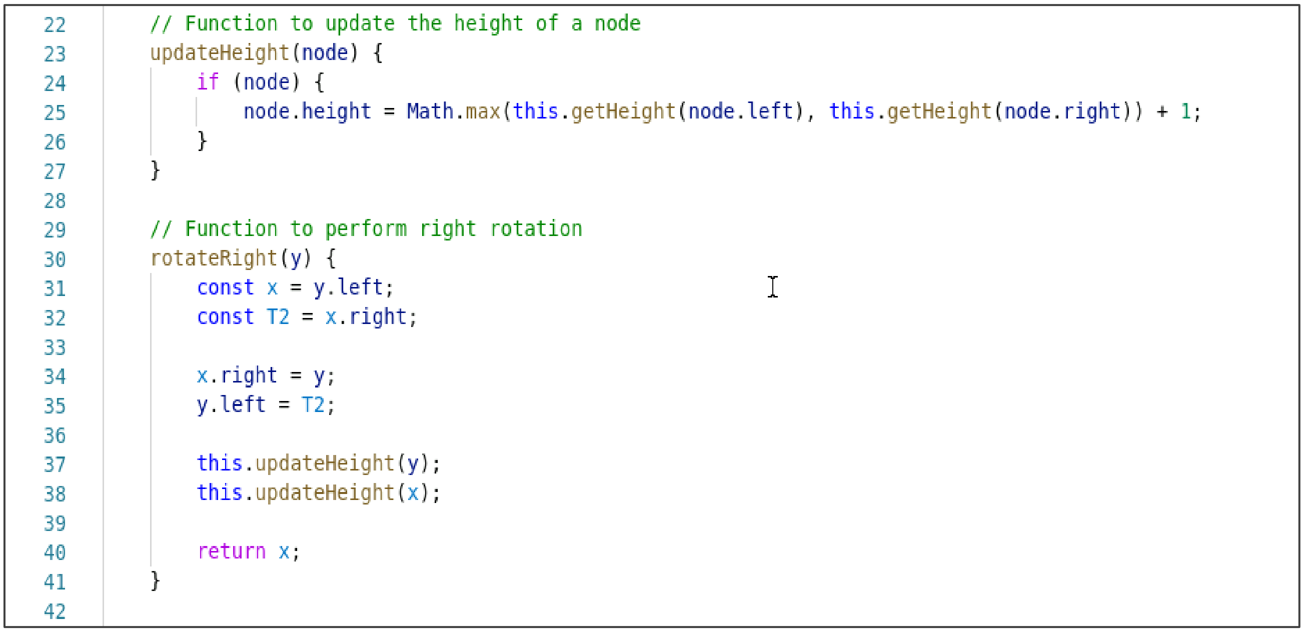
```
// Function to perform left rotation
rotateLeft(x) {
  const y = x.right;
  const T2 = y.left;

  y.left = x;
  x.right = T2;

  this.updateHeight(x);
  this.updateHeight(y);

  return y;
}

// Function to get the balance factor of a node
getBalanceFactor(node) {
  return node ? this.getHeight(node.left) - this.getHeight(node.right) : 0;
}

// Function to insert a node into the AVL tree
insert(data) {
  this.root = this._insert(this.root, data);
}

_insert(node, data) {
  // Perform standard BST insert
  if (!node) {
    return new AVLNode(data);
  }

  if (data < node.data) {
    node.left = this._insert(node.left, data);
  } else if (data > node.data) {
    node.right = this._insert(node.right, data);
  } else {
    return node; // Duplicate nodes are not allowed
  }

  // Update the height of the current node
  this.updateHeight(node);

  // Get the balance factor to check if the node became unbalanced
  const balance = this.getBalanceFactor(node);
```

```
    // Left Left Case
    if (balance > 1 && data < node.left.data) {
      return this.rotateRight(node);
    }

    // Right Right Case
    if (balance < -1 && data > node.right.data) {
      return this.rotateLeft(node);
    }

    // Left Right Case
    if (balance > 1 && data > node.left.data) {
      node.left = this.rotateLeft(node.left);
      return this.rotateRight(node);
    }

    // Right Left Case
    if (balance < -1 && data < node.right.data) {
      node.right = this.rotateRight(node.right);
      return this.rotateLeft(node);
    }

    return node;
  }
}

// Example usage
const avlTree = new AVLTree();
avlTree.insert(10);
avlTree.insert(5);
avlTree.insert(15);
avlTree.insert(3);
avlTree.insert(8);

console.log('AVL Tree root:', avlTree.root.data);
```

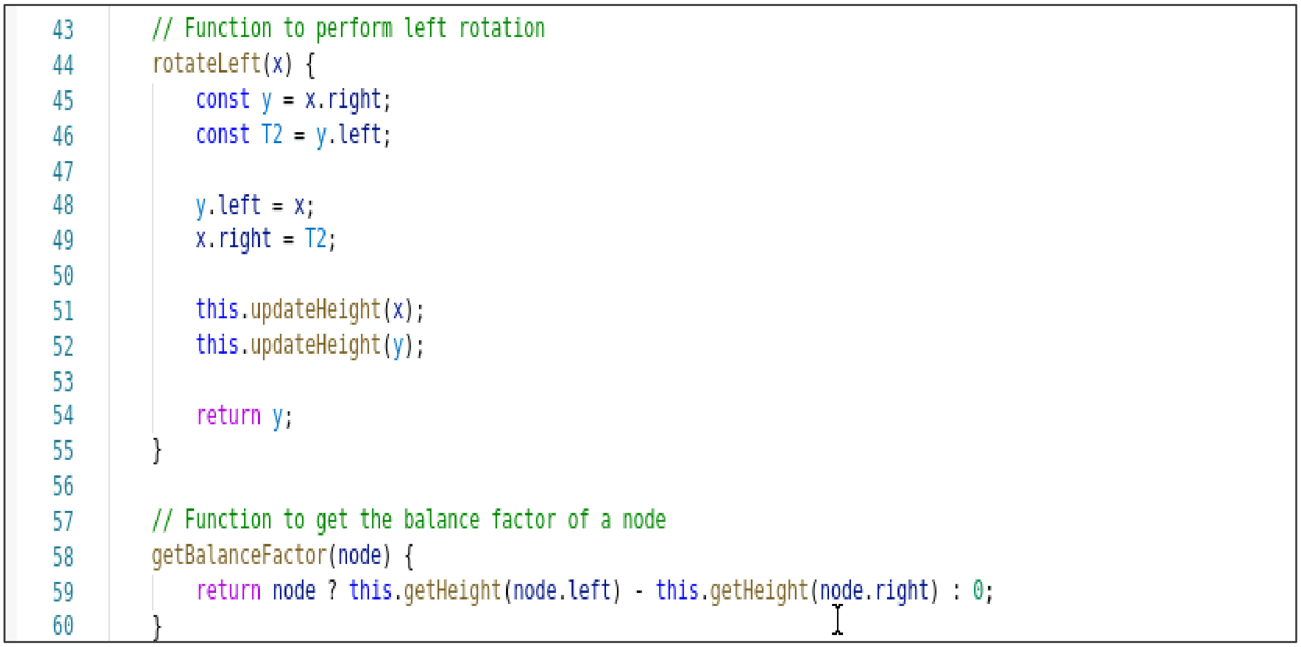


JS AVLTree.js > ...

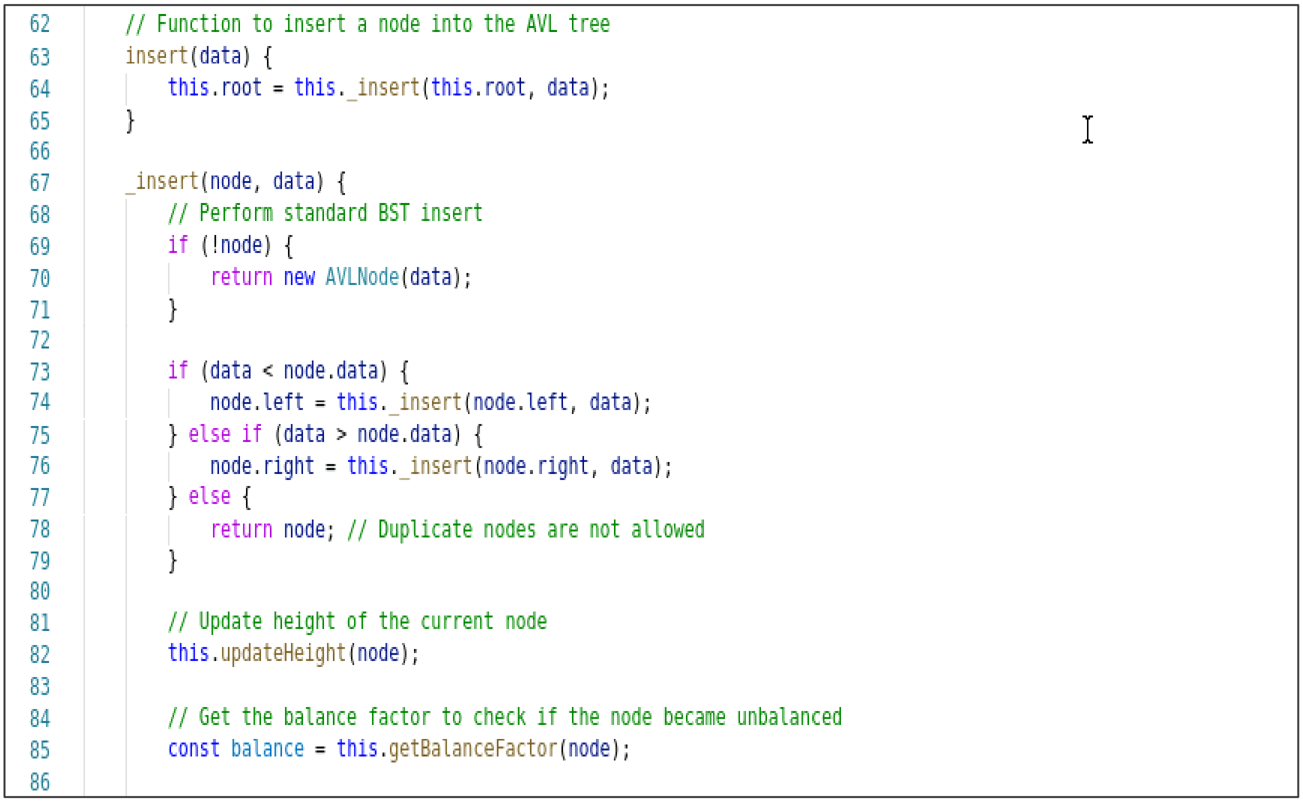
```
// AVL tree node definition
class AVLNode {
    constructor(data) {
                    data;
        this.left = null;
        this.right = null;
        this.height = 1;
    }
}

// AVL tree implementation
class AVLTree {
    constructor() {
        this.root = null;
    }

    // Function to get the height of a node
    getHeight(node) {
        return node ? node.height : 0;
    }

```



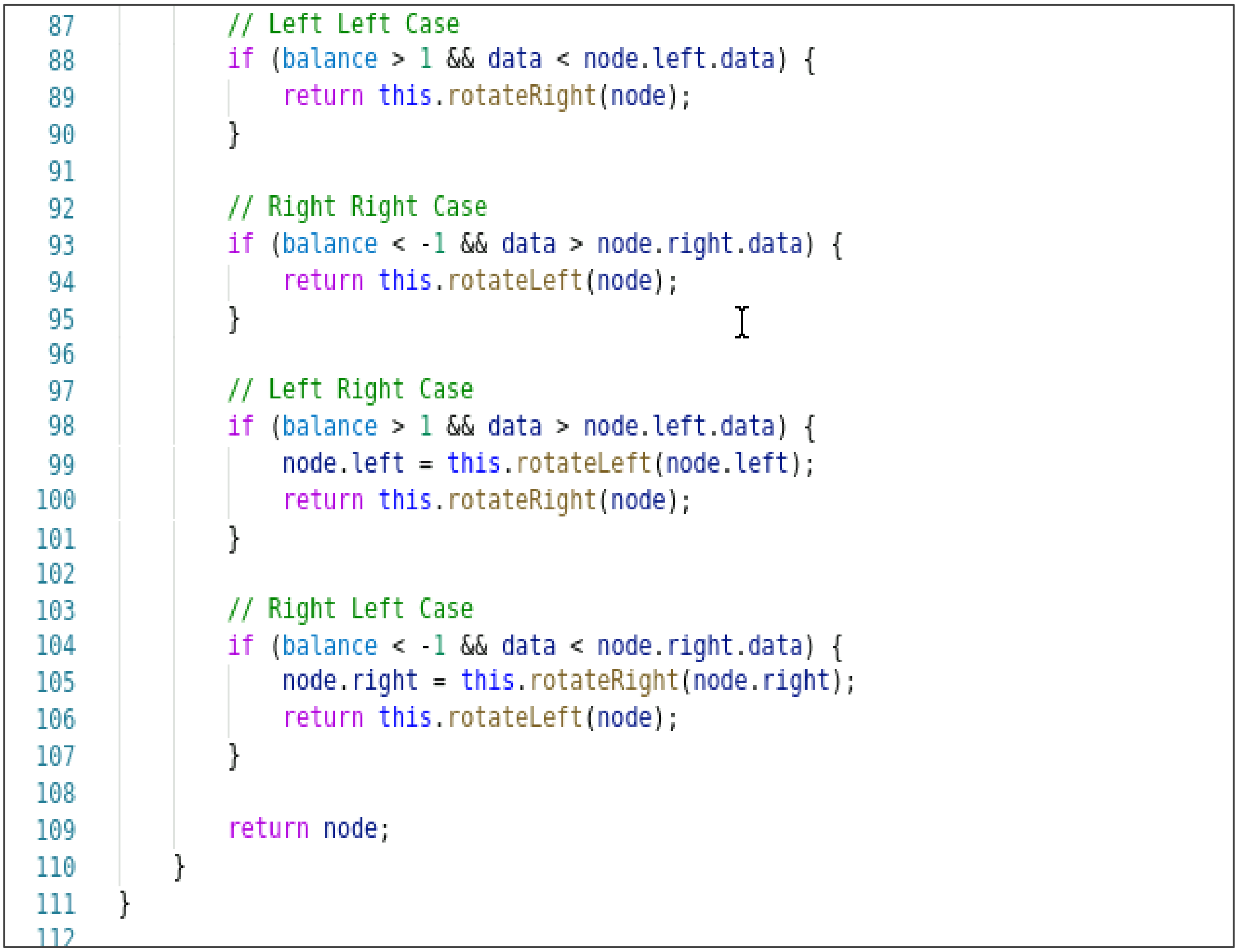
```
    // Function to update the height of a node
    updateHeight(node) {
        if (node) {
            node.height = Math.max(this.getHeight(node.left), this.getHeight(node.right)) + 1;
        }
    }

    // Function to perform right rotation
    rotateRight(y) {
        const x = y.left;
        const T2 = x.right;

        x.right = y;
        y.left = T2;

        this.updateHeight(y);
        this.updateHeight(x);

        return x;
    }

```



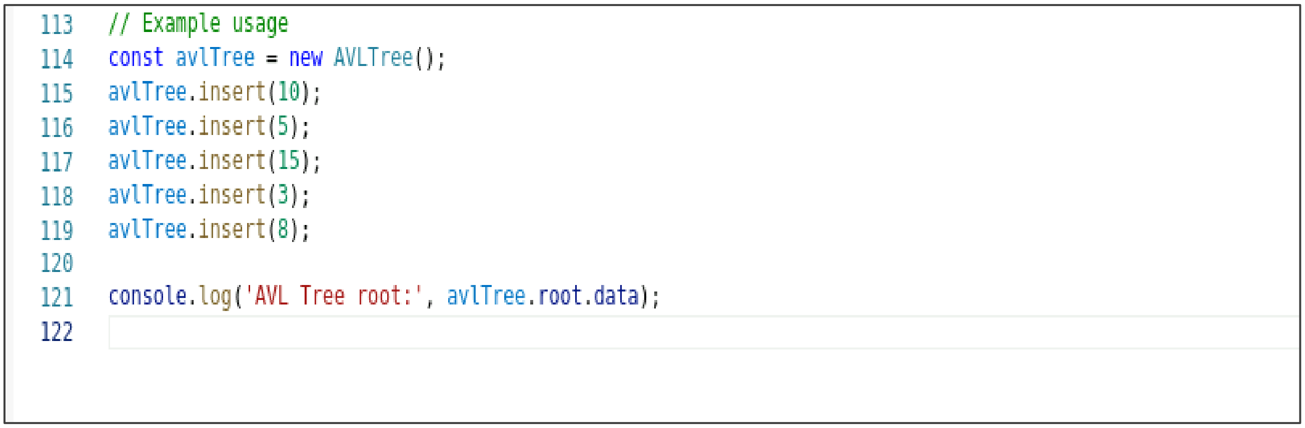
```
    // Function to perform left rotation
    rotateLeft(x) {
        const y = x.right;
        const T2 = y.left;

        y.left = x;
        x.right = T2;

        this.updateHeight(x);
        this.updateHeight(y);

        return y;
    }

    // Function to get the balance factor of a node
    getBalanceFactor(node) {
        return node ? this.getHeight(node.left) - this.getHeight(node.right) : 0;
    }
```



```
    // Function to insert a node into the AVL tree
    insert(data) {
        this.root = this._insert(this.root, data);
    }

    _insert(node, data) {
        // Perform standard BST insert
        if (!node) {
            return new AVLNode(data);
        }

        if (data < node.data) {
            node.left = this._insert(node.left, data);
        } else if (data > node.data) {
            node.right = this._insert(node.right, data);
        } else {
            return node; // Duplicate nodes are not allowed
        }

        // Update height of the current node
        this.updateHeight(node);

        // Get the balance factor to check if the node became unbalanced
        const balance = this.getBalanceFactor(node);

```

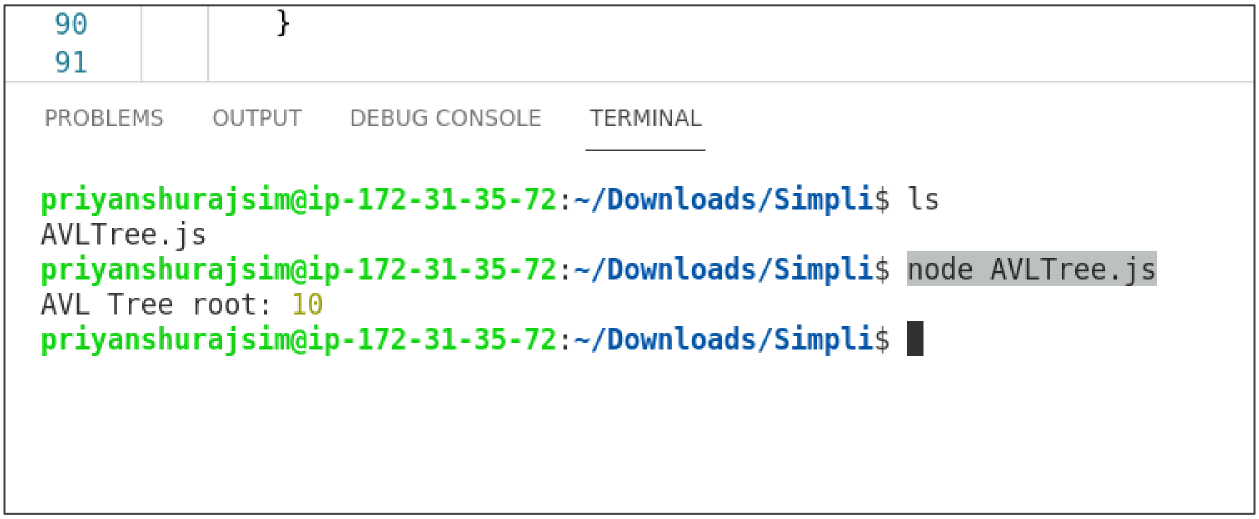


```javascript
        // Left Left Case
        if (balance > 1 && data < node.left.data) {
            return this.rotateRight(node);
        }

        // Right Right Case
        if (balance < -1 && data > node.right.data) {
            return this.rotateLeft(node);
        }

        // Left Right Case
        if (balance > 1 && data > node.left.data) {
            node.left = this.rotateLeft(node.left);
            return this.rotateRight(node);
        }

        // Right Left Case
        if (balance < -1 && data < node.right.data) {
            node.right = this.rotateRight(node.right);
            return this.rotateLeft(node);
        }

        return node;
    }
}
```



```javascript
// Example usage
const avlTree = new AVLTree();
avlTree.insert(10);
avlTree.insert(5);
avlTree.insert(15);
avlTree.insert(3);
avlTree.insert(8);

console.log('AVL Tree root:', avlTree.root.data);
```

1.3 Press **Ctrl** + **S** to save the file and execute it in the **TERMINAL** using the commands given below:
ls
node AVLTree.js



```
    }
```

```
priyanshurajsim@ip-172-31-35-72:~/Downloads/Simpli$ ls
AVLTree.js
priyanshurajsim@ip-172-31-35-72:~/Downloads/Simpli$ node AVLTree.js
AVL Tree root: 10
priyanshurajsim@ip-172-31-35-72:~/Downloads/Simpli$ 
```

By following these steps, you have successfully implemented an AVL tree in JavaScript to maintain height-balanced binary search tree properties.