**Introduction:**

Variables in JavaScript can be declared using var, let, or const. While var was traditionally used, let and const introduced in ES6 offer better scoping and immutability options, making your code safer and easier to debug. This lab focuses on converting a code snippet written with var to use let and const, while addressing potential errors and understanding the differences between these declarations.

**Goal:**

By the end of this lab, learners will:

1. Understand the differences between var, let, and const.
2. Identify and resolve issues that arise when converting var to let and const.
3. Write cleaner, modern JavaScript code by using appropriate variable declarations.

**Objectives:**

1. Replace var declarations with let and const based on the variable's usage:

* Use const for values that should not change.
* Use let for variables that can be reassigned.

2. Resolve issues caused by hoisting when converting var to let or const.

3. Handle cases where var is redeclared and modify the code to work correctly with let.

4. Test the updated code to ensure it runs without errors.

**Instructions:**

**Part 1: Understanding the Code with var**

**Step 1:** Open the Provided File

1. Locate the file named **var\_to\_let\_const.js** in the **PROJECT** folder.

2. Open the file to review the JavaScript code that is written using var.

3. Analyse the structure and identify the following key elements in the code:

* **Redeclaration**: Notice where variables are declared multiple times using var.
* **Hoisting**: Identify variables that are used before they are declared.
* **Scoping**: Observe how var allows variables to "leak" outside of their intended block scope.
* **Constants Misuse**: Look for variables declared as var but logically represent values that should not change.

**Initial Code (var\_to\_let\_const.js)**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

// This is the initial code using `var` with some issues to fix.

var x = 5; // Variable declaration and initialization

var x = 10; // Redeclaration, works with var but not allowed with let

console.log(a); // Hoisting example: variable is used before declaration

var a = 20;

if (true) {

    var z = 30; // Variable declared with var is accessible outside block

}

console.log(z); // Should cause a scoping issue when switched to let/const

var b = 40; // Variable that changes value

b = 50; // Should be let, as it changes value

var PI = 3.14; // Declared as a variable but should ideally be a constant

PI = 3.14159; // This should throw an error when changed to const

// Your task:

// 1. Replace `var` declarations with `let` or `const` as appropriate.

// 2. Resolve issues caused by hoisting, redeclaration, and scoping.

// 3. Ensure the program works correctly after refactoring.

**Part 2: Refactoring the Code to Use let and const**

**Step 1:** Refactor the Code

1. Replace **var** with **const** where appropriate:

* Identify variables that should not change during the program's execution and declare them with const.

1

2

3

4

5

6

7

// Original:

var PI = 3.14;

PI = 3.14159; // Error when using const

// Refactored:

const PI = 3.14; // Use const because the value should remain constant

2. Replace **var** with **let** where necessary

* For variables that are reassigned but not redeclared, use let.

1

2

3

4

5

6

7

8

// Original:

var b = 40;

b = 50;

// Refactored:

let b = 40;

b = 50; // Use let because the value changes

3. Fix **Hoisting** Issues

* Move variable declarations to ensure they are declared before use.

1

2

3

4

5

6

7

8

// Original:

console.log(a); // Undefined with var, error with let/const

var a = 20;

// Refactored:

let a = 20;

console.log(a); // Declare before use

4. Resolve **Scoping** Issues

* Replace var with let or const in block scopes, ensuring the variable is only accessible within the block.

1

2

3

4

5

6

7

8

9

10

11

12

// Original:

if (true) {

    var z = 30;

}

console.log(z); // Accessible with var, error with let/const

// Refactored:

if (true) {

    let z = 30; // Use let to make z block-scoped

}

console.log(z); // Error: z is not accessible outside the block

**Part 3:** Test Your Code

**Step 1**: Use console.log statements to verify the outputs of your refactored code.

**Step 2**: Final Refactored Code (Solution)

**Note:** Before looking the given solution, it is highly recommended to try writing the code yourself based on the hints that are provided and concepts you have learned.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

// Refactored Code

// Using const for a value that doesn’t change

const x = 10;

console.log(x);

// Using let for variables that can change but are not redeclared

let a = 20;

console.log(a);

if (true) {

    let z = 30; // Block-scoped variable

    console.log(z);

}

let b = 40; // Variable that can be reassigned

b = 50;

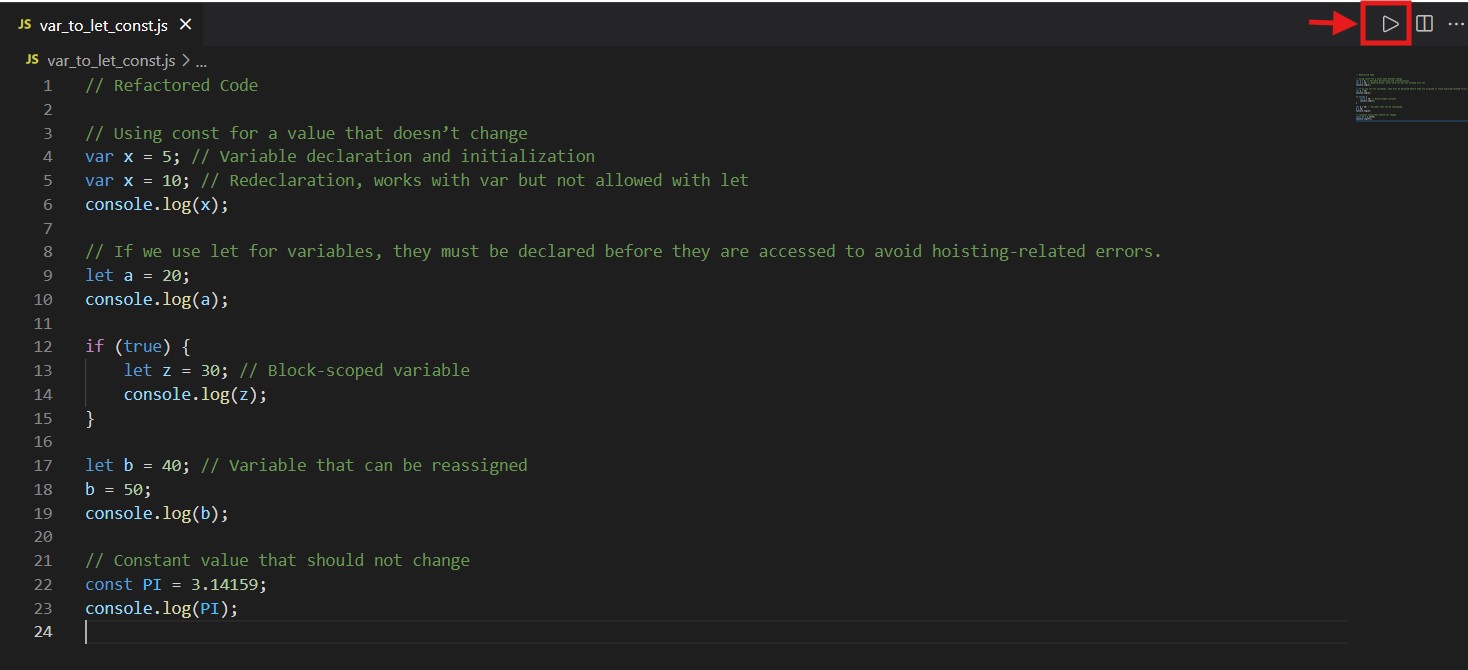
console.log(b);

// Constant value that should not change

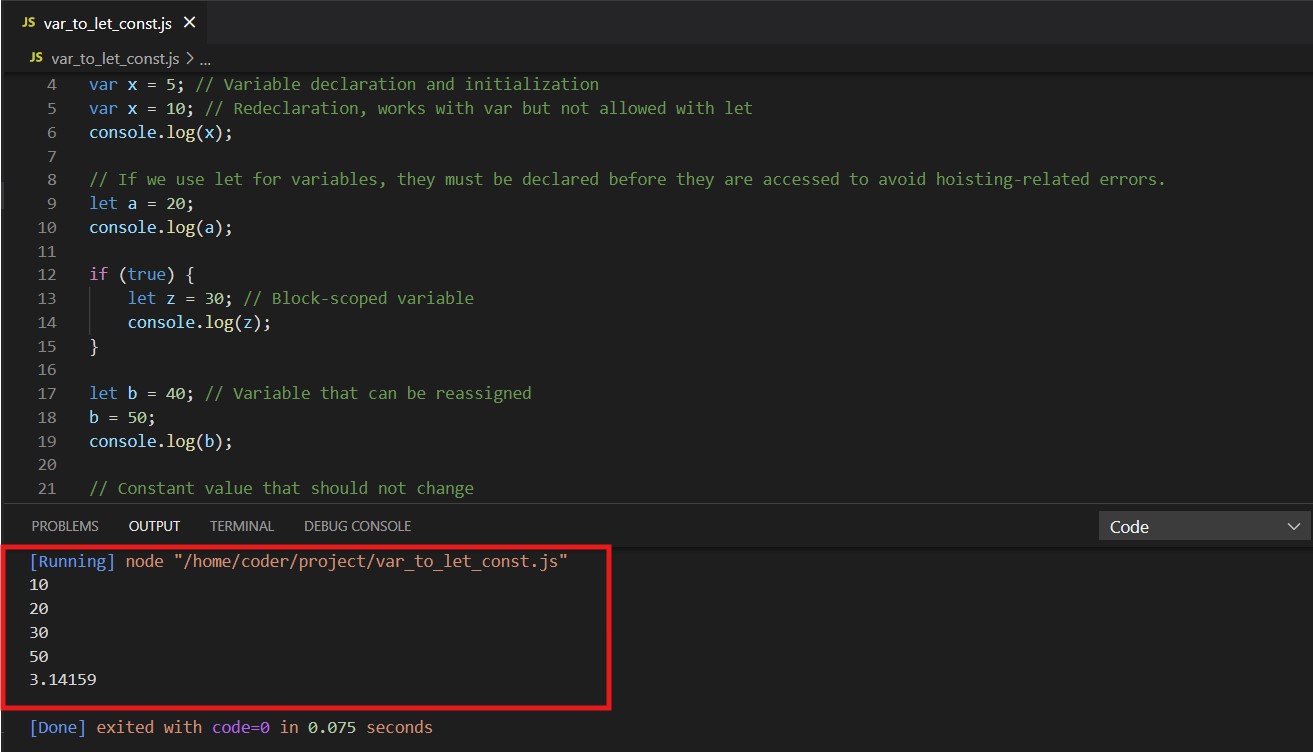
const PI = 3.14159;

console.log(PI);

**Step 3**: Execute the program by clicking the play button displayed below.



**Step 4:** After running the code, the output will be displayed as shown below, based on the provided console.log statements.



**Key Takeaways:**

1. **var vs let vs const:**

* var is function-scoped and allows redeclaration, leading to potential bugs.
* let is block-scoped and cannot be redeclared.
* const is block-scoped and used for values that should not change.

2. **Hoisting:**

* Variables declared with var are hoisted and initialized as undefined.
* Variables declared with let or const are not accessible before their declaration.

3. **Best Practices:**

* Use const by default, and use let only when reassignment is necessary.
* Avoid using var in modern JavaScript code.