

Hands-on Lab: Working with Variables and Their Scope



Estimated time needed: 15 minutes

What you will learn

In this lab, you will develop an understanding of JavaScript variables, including `var`, `let`, and `const`. JavaScript variables act as containers for storing data, allowing dynamic programming by facilitating the storage, manipulation, and retrieval of values across various sections of code within applications. You will learn how to use variables to enhance code readability and maintainability, enabling efficient management and reuse of variable values throughout your entire programs.

Learning objectives

After completing this lab, you will be able to:

- **Understand where variables work:** Learn how `'var,'` `'let,'` and `'const'` variables operate in different parts of your code, where you can use them, and potential issues they might cause.
- **Know how variables behave:** Discover the distinct behaviors of `'var,'` `'let,'` and `'const.'` Understand how they can change and when they remain constant. Learn why declaring variables with `'var,'` `'let,'` and `'const'` is essential for maintaining safe and reliable data.
- **Learn the right way to use them:** Determine when to use `'let'` for changing data, `'const'` for constants, and why using `'var'` might not be the best choice anymore. It's important to selecting the right variable for the code.
- **Make your code stronger:** Through a deep understanding of `'var,'` `'let,'` and `'const,'` you'll write more resilient code less prone to breakage. Avoid bugs and create code that is not only comprehensible to you but also to others, ensuring it operates smoothly.

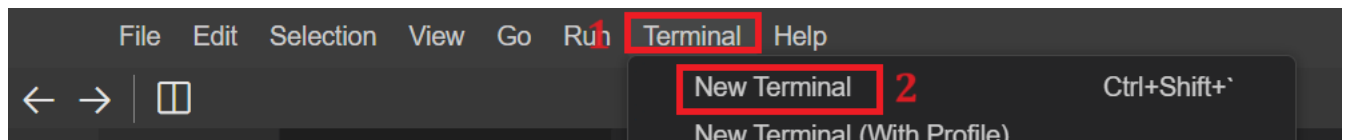
Prerequisites

- Basic Knowledge of HTML and Git commands.
- Basic understanding of JavaScript variables and their scope.
- Web browser with a console (Chrome DevTools, Firefox Console, and so on).

Step 1: Setting up the environment

1. First, you need to clone your main repository in the **Skills Network Environment**. Follow the given steps:

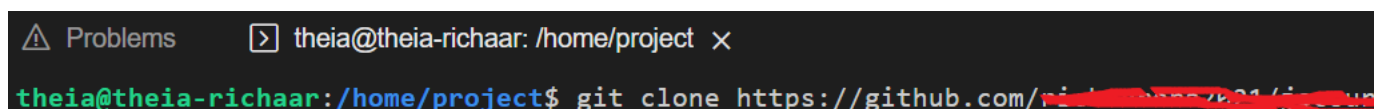
- Click on the terminal in the top-right window pane and then select **New Terminal**.



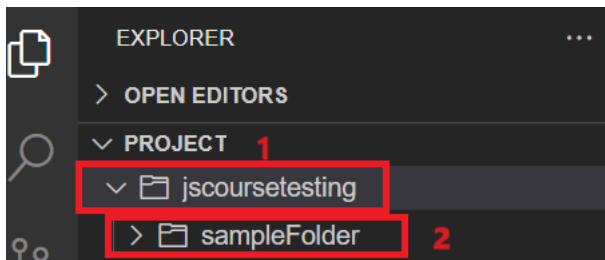
- Perform `git clone` command by writing given command in the terminal.

```
git clone <github-repository-url>
```

Note: Put your own GitHub repository link instead of `<github-repository-url>` which should look like below:



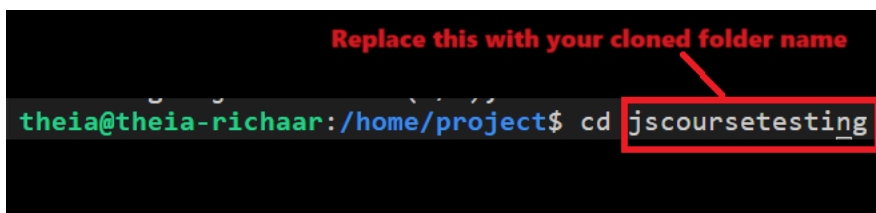
- The above step will create a cloned folder for your GitHub repository under the project folder in the explorer, as indicated by number 1 (this folder name is just for reference). You will also see a `SampleFolder` inside the cloned folder, as this folder was pushed in the first lab, as shown at number 2.



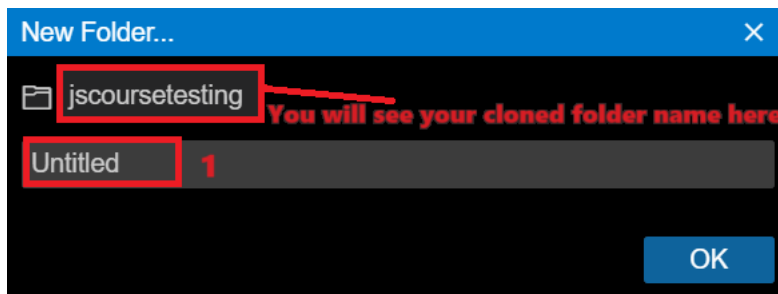
- Now you need to navigate inside the cloned folder. For this write given command in the terminal and press **Enter**.

```
cd <repository-folder-name>
```

Note: Write your cloned folder name instead of <repository-folder-name>.



- Now, select the **cloned Folder Name** folder, right-click on it, and choose **New Folder**. A pop up box will appear where you will see default name shown at number 1.



- Instead of default name enter the folder name as **Scope**. This will create the folder for you. Then, select the **Scope** folder, right-click, and choose **New File**. Again a pop box will appear with default name, replace default name with the file name as **scope_lab.html** and click OK. This will create your HTML file.
- Now select the **Scope** folder again, right click and select **New File**. Enter the file name as **scope_lab.js** and click OK. This action will generate your JavaScript file.
 - Create the basic template structure for **scope_lab.html** file by adding the below code.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Scoped Variables</title>
</head>
<body>
  <h1>Scope Lab for Var, Let and Const</h1>
  </h1>
  <script src="./scope_lab.js"></script>
</body>
</html>
```

Note: After pasting the code, save your file.

- The above HTML code has one <h1> tag in file and one <script> tag to include js file in **scope_lab.html** file using the **src** attribute.

Step 2: Defining variables with let, var, and const

1. Declare and initialize variables using let, var, and const. Then assign values to these variables for different scopes such as:

- Global scope
- Block scope

2. For the different scopes, include the given code in **scope_lab.js**. It is declared to show the scope of variables let, const, and var at a global and block level. Then, save the file.

```
// Global scope
var globalVar = "I'm a global variable";
let globalLet = "I'm also global, but scoped with let";
const globalConst = "I'm a global constant";
{
  // Block scope
  var blockVar = "I'm a block-scoped var";
  let blockLet = "I'm a block-scoped let";
  const blockConst = "I'm a block-scoped const";
}
```

Note: After pasting the code, save your file.

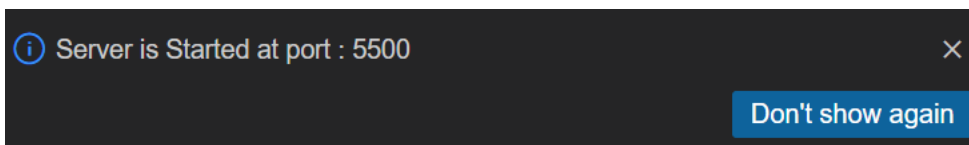
Step 3: Scope of variables at a global level


1. Now, to observe the scope at a global level, include the given code after the previous code (after line number 12) and save the file.

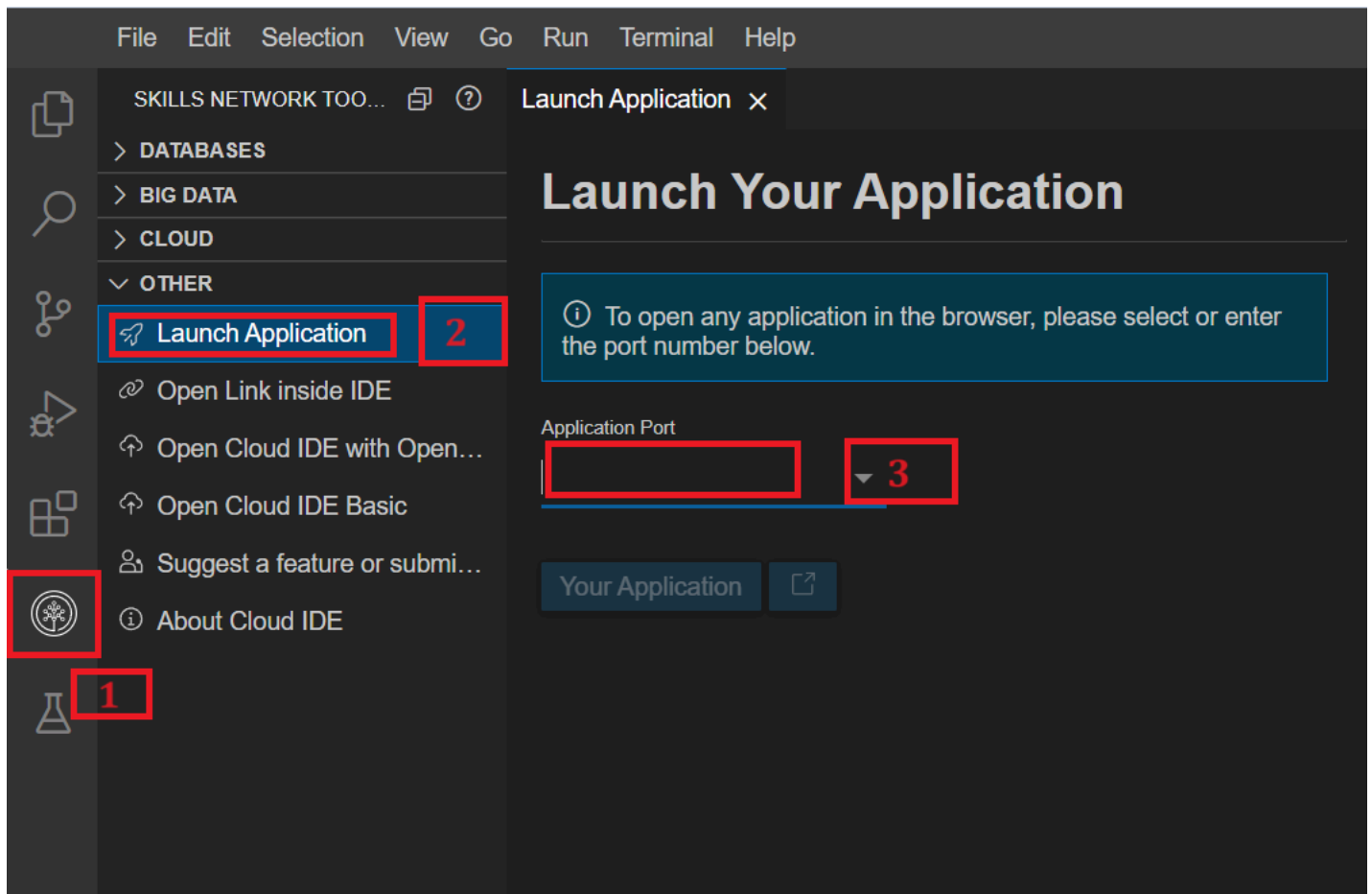
```
// Global scope
console.log(globalVar); // Output: "I'm a global variable"
console.log(globalLet); // Output: "I'm also global, but scoped with let"
console.log(globalConst); // Output: "I'm a global constant"
```

2. To view how your HTML page is displayed in the browser, use the built-in Live Server extension with the following instructions:

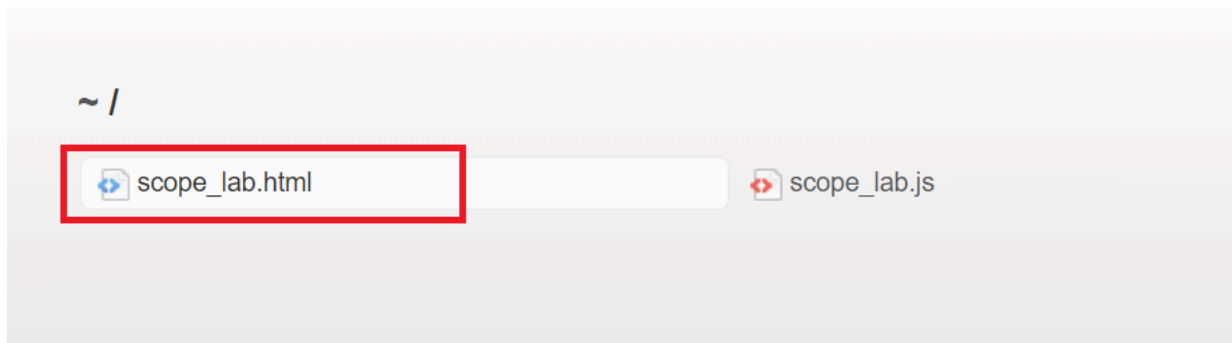
- Select **scope_lab.html**, right click on this file and then choose 'Open with Live Server'.
- A notification will appear at the bottom right, indicating that the server has started on port 5500.



- Then, click the Skills Network button on the left (refer to number 1). This action will open the "Skills Network Toolbox." Next, select "Launch Application" (refer to number 2). Once there, enter port number **5500** in "Application Port" (refer to number 3) and click this button .

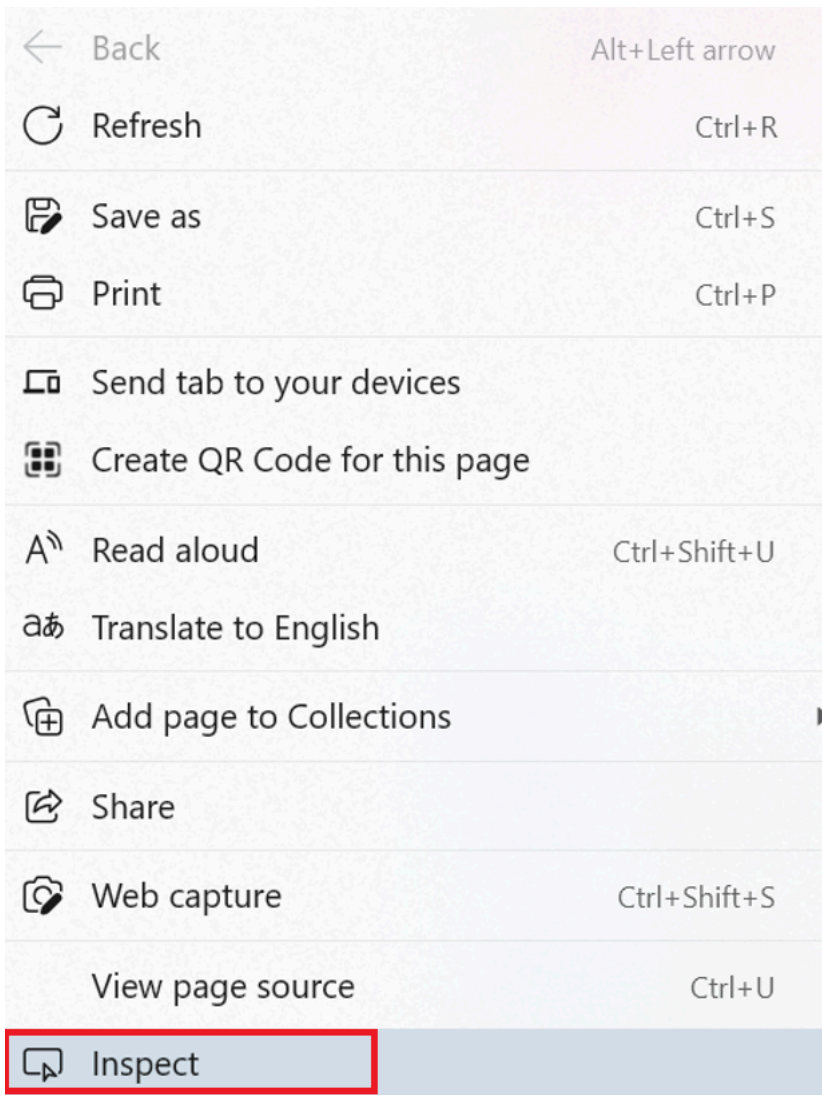


- It will open your default browser, where you will first see the name of your cloned folder. Click on that folder, and inside it, among other folders, you will find the Scope folder. Click on the Scope folder, and then select the HTML file, as shown below.

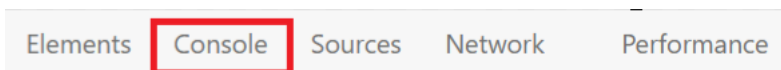


Note: Save your file after pasting the code. If you edit your code, simply refresh your browser running on port 5500. No need to relaunch the application.

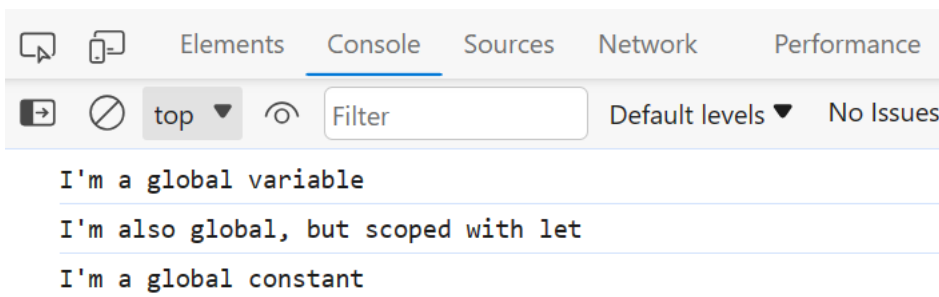
3. To view the output in the browser, right-click on the window that opens after clicking the "scope_lab.html" file, and then choose the "inspect" option.



4. Next, go to console tab.



5. You will see output as given below.



Step 4: Scope of variables for block scope

1. Now to see the scope at the block level, include the provided code for variables declared at the block level after the previous JavaScript code, and save the file.

```
//Block Scope
console.log(blockVar);
console.log(blockLet);
```

2. You will see the output in console tab as below.

I'm a block-scoped var [scope_lab.js:19](#)

✖ ▶ Uncaught ReferenceError: blockLet is [scope_lab.js:20](#) not defined
at [scope_lab.js:20:13](#)

3. You will see output for **blockVar**, because it can be accessed outside the curly brackets {}. But you will see **ReferenceError** for **blockLet** because it cannot be accessed outside the curly brackets due to which it is known as block scoped variables. Such variables can only be accessed within curly brackets. You will see the same error for **const** as well.

Note: A ReferenceError occurs when code attempts to use a variable that has not been declared or is outside of its scope, preventing proper access or execution.

Step 5: Scope of variables for function scope

1. In this step, you will explore how function-scoped variables work. For this, include the given code at the end of the JavaScript file and save the file.

```
function show(){
  var functionVar = "I'm a block-scoped var";
  let functionLet = "I'm a block-scoped let";
  const functionConst = "I'm a block-scoped const";
}
show();
console.log(functionVar); // Throws ReferenceError
console.log(functionLet); // Throws ReferenceError
console.log(functionConst); // Throws ReferenceError
```

2. For the above code, when you view the output in the console, you will again see a **ReferenceError** for var, let, and const. To see the output, you need to add the // before the codes as shown below.

```
console.log(blockVar);
console.log(blockLet);
```

After putting // in front of above code it will look like as shown below.

```
// console.log(blockVar);
// console.log(blockLet);
```

Note: You can use // to convert a statement into a comment.

3. You will see **ReferenceError** for **functionVar** because it cannot be accessed outside the function making it a function scoped variable. Similarly, you will not be able to access let and const because they are block scoped variables.

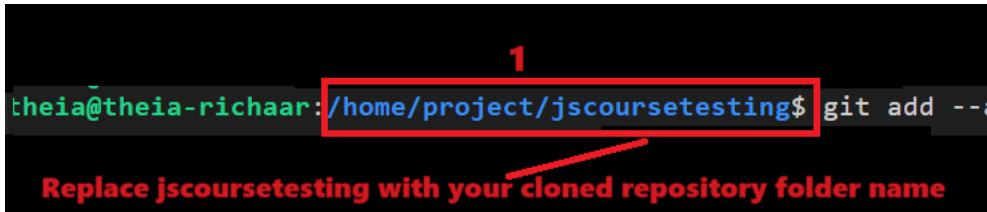
Note: You will only see an error for functionVar because it acts as a blockage, preventing the execution of the subsequent lines of code. Ensure that your application is live throughout this process and that you have saved the file.

Step 6: Perform Git commands

1. Perform git add to add the latest files and folder in the git environment.

```
git add --a
```

- Make sure the terminal has the path as follows:



2. Then, perform `git commit` in the terminal. While performing `git commit`, terminal can show a message to set up your `git config --global` for the `user.name` and `user.email`. Then, you need to perform `git config` command as well for the `user.name` and `user.email`.

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

```
git commit -m "message"
```

3. Then perform `git push` just by writing the given command in the terminal.

```
git push origin
```

- It will push all of your latest folders and files to your GitHub repository.

Practice task

1. Create one block using `{}` and declare variables using **let**, **const**, and **var**.
2. Then, try to reassign these variables within the curly braces `{}`. Check for any errors that may occur.
3. Next, try to reassign the same variables outside the block `{}` and check if assignment outside the scope of variables where the variables are assigned can be done or not.

Summary

1. Variable scope overview:

- Global scope: Variables declared outside any block or function have a global scope and are accessible throughout the entire script.
- Block scope: Variables declared within curly braces {} have block scope, accessible only within that block.
- Function scope: Variables declared within a function have function scope, limited to that function's block.

2. Variable declaration and initialization:

- Used var, let, and const to declare and initialize variables in different scopes: global, block, and function.
- Demonstrated the behavior of these variables in respective scopes by accessing them outside their defined scope.

3. Output and scope analysis:

- Global variables were accessible everywhere in the script.
- Block-scoped variables (inside {}) had limited accessibility, resulting in ReferenceErrors when accessed outside their blocks.
- Function-scoped variables (inside a function) also led to ReferenceErrors when accessed outside the function.

© IBM Corporation. All rights reserved.