Hands-on Lab: Debug a JavaScript Application in Chrome Dev Tools

Skills Network

Estimated time needed: 15 minutes

What you will learn

In this lab, you will gain an understanding of how to debug code in JavaScript. You will learn about try and catch blocks and gain insight into how and where to use these blocks to make sure that your code is running smoothly.

Learning objectives

After completing this lab, you will be able to:

- **User input:** Provides two input fields for users to enter numerical values.

- **Operation execution:** Multiplies the two input values when the "Perform Operation" button is clicked.

- **Error handling:** Checks if the entered values are valid numbers before performing the multiplication operation. If the values are not valid numbers, it displays an error message instead of the result.

- **Debugging:** Includes a debugger statement within the multiply() function to pause execution and allow developers to inspect the code, variables, and execution flow using browser Developer Tools.

Prerequisites

- Basic knowledge of HTML.

- Basic understanding of console and debug.

- Web browser with a console (Chrome DevTools, Firefox Console, and so on).

Step 1: Setting up the environment

1. Firstly, you need to clone your main repository in the **Skills Network Environmemnt** which you have created in the first lab and where you have pushed all of your previous labs files and folders. Follow given steps:
   - Click on the terminal in the top-right window pane and then select **New Terminal**.
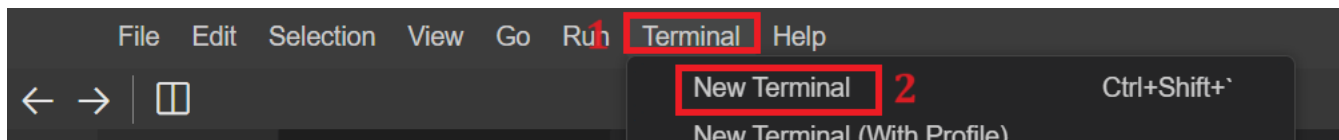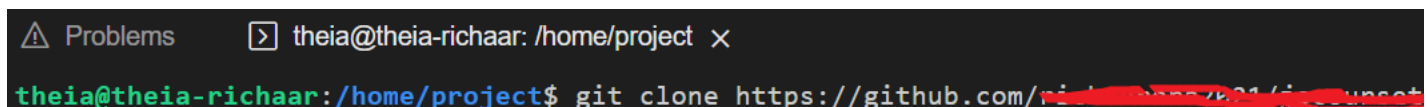

   - Perform `git clone` command by writing given command in the terminal.

   ```
   1. git clone <github-repository-url>
   ```
   Copied!
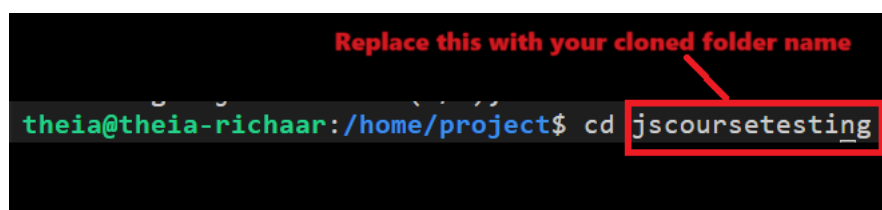
      Note: Put your own GitHub repository link instead of `<github-repository-url>`.

   - Above step will clone folder for your GitHub repository under project folder in explorer. You will also see multiple folders inside the cloned folder.

   - Now, you need to navigate inside the cloned folder. For this, write the given command in the terminal:

   ```
   1. cd <repository-folder-name>
   ```
   Copied!

Note: Write your cloned folder name instead of `<repository-folder-name>`. Perform `git clone` if you have logged out of **Skills Network Environment** and you cannot see any files or folder after you logged in.

2. Now select **cloned Folder Name** folder, right-click on it and click on **New Folder**. Enter folder name as **DebugCode**. It will create the folder for you. Then select **DebugCode** folder, right-click and select **New File**. Enter file named as **debug_code.html** and click OK. It will create your HTML file.

3. Create a basic template structure of an HTML file by adding content provided below.

```
1.  1
2.  2
3.  3
4.  4
5.  5
6.  6
7.  7
8.  8
9.  9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
```

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.
4.  <head>
5.      <meta charset="UTF-8">
6.      <title>Debugging Example</title>
7.  </head>
8.
9.  <body>
10.     <h1>Debugging Example</h1>
11.     <label for="input1">Enter Number 1:</label>
12.     <input type="number" id="input1"><br><br>
13.     <label for="input2">Enter Number 2:</label>
14.     <input type="number" id="input2"><br><br>
15.     <button onclick="performOperation()">Perform Operation</button>
16.     <p id="result"></p>
17. </body>
18.
19. </html>
```
Copied!

Note: After pasting the code, save your file.

4. Now select the **DebugCode** folder again, right-click, and select **New File**. Enter the file named **debug_code.js** and click OK. This action will generate your JavaScript file.

5. To include js file in **debug_code.html**, use the script tag in the HTML file above the `</body>` tag. You can use the given code to include and save the script file.

```
1.  1
```

```
1.  <script src="./debug_code.js"></script>
```
Copied!

Step 2: Defining variables

1. Include given code in **debug_code.js** file.

```
1.  1
2.  2
3.  3
4.  4
5.  5
6.  6
7.  7
8.  8
9.  9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
```

```
30. 30
31. 31

 1.
 2.      function performOperation() {
 3.      // Get user input from input fields
 4.      let num1 = parseInt(document.getElementById('input1').value);
 5.      let num2 = parseInt(document.getElementById('input2').value);
 6.      // Check if inputs are valid numbers
 7.      if (!isNaN(num1) && !isNaN(num2)) {
 8.      // Perform the operation
 9.                  let result = multiply(num1, num2);
10.
11.                  // Display the result
12.                  displayResult(result);
13.              } else {
14.                  displayResult('Please enter valid numbers');
15.              }
16.          }
17.
18.          function multiply(a, b) {
19.              // Introduce a debugger statement to pause execution
20.              debugger;
21.
22.              // Multiply the numbers
23.              return a * b;
24.          }
25.
26.          function displayResult(result) {
27.              // Display the result in the paragraph element
28.              const resultElement = document.getElementById('result');
29.              resultElement.textContent = `The result is: ${result}`;
30.          }
31.
```

Copied!

2. Above code has following parts:
   - performOperation() Function:
     - Retrieves numerical values entered by the user from HTML input fields (input1 and input2).
     - Validates that the entered values are valid numbers.
     - If both values are valid numbers, it calls the multiply() function passing these values, otherwise, it displays an error message.
   - multiply() Function:
     - Includes a debugger statement to pause code execution at this point for debugging purposes.
     - Multiplies two input numbers (a and b) and returns the result.
   - displayResult() Function:
     - Displays the result of the multiplication or an error message in a designated paragraph element (resultElement) on the webpage.

Step 3: Debug the code to see the flow of the code

1. To view how your HTML page, right-click the **debug_code.html** file after selecting this file, then select "Open with Live Server".

2. The server should start on port 5500, indicated by a notification on the bottom right side.

ⓘ Server is Started at port : 5500   ✕

                                        Don't show again
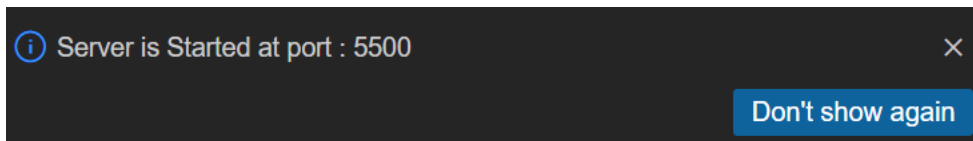
3. Click on the Skills Network button on the left (refer to number 1), it will open the "Skills Network Toolbox". Then click on the Launch Application (refer to number 2). From there you enter the port no. as 5500 at number 3 and click on this button ⬈ .
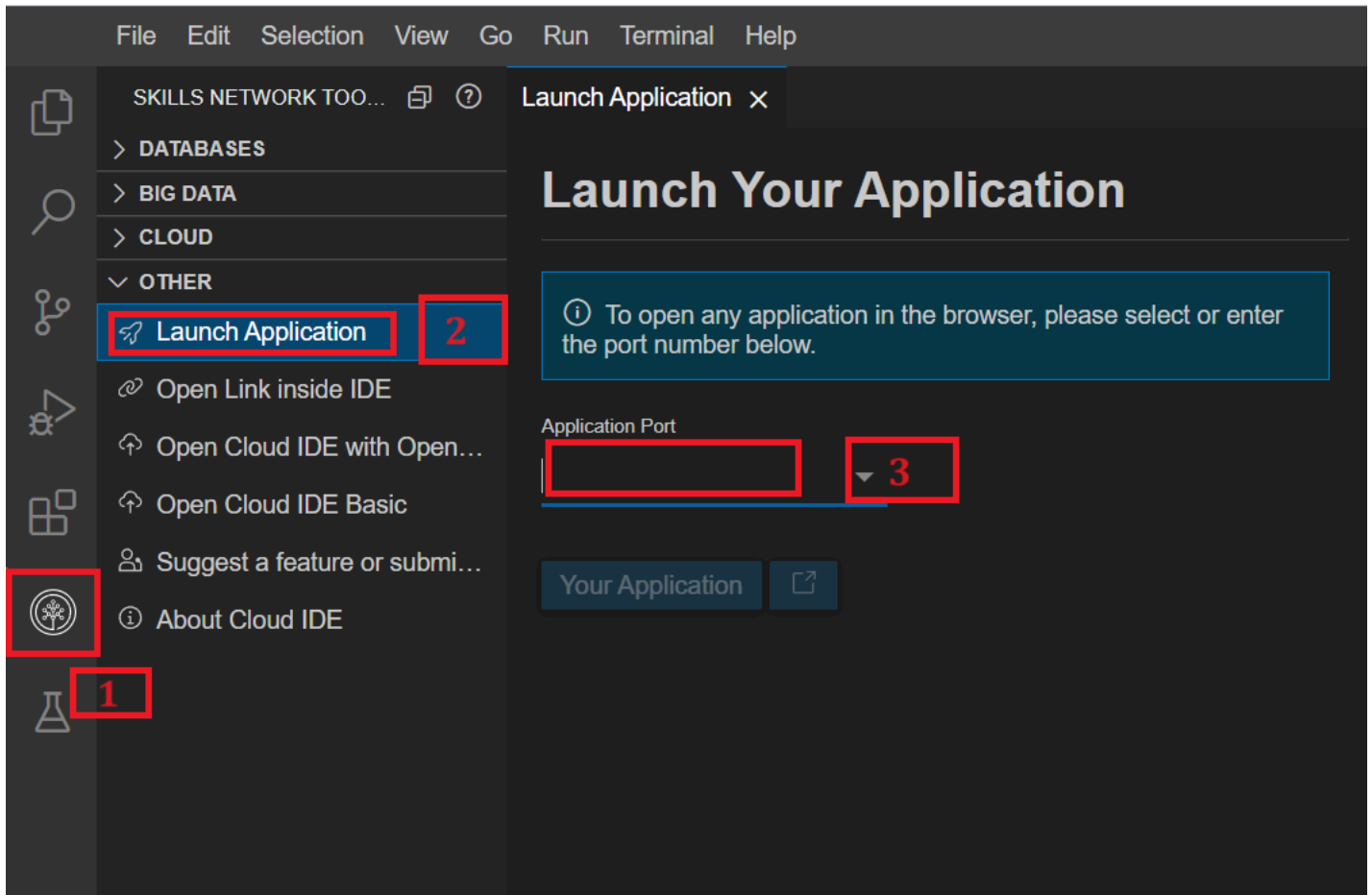
4. It will open your default browser where you will see **cloned-folder-name** folder name. Click on that **cloned-folder-name** folder name. After clicking you will see multiple folders name, among those folders name click on **DebugCode** folder. You will see files related to this folder where again you will click on **debug_code.html** file as shown below.
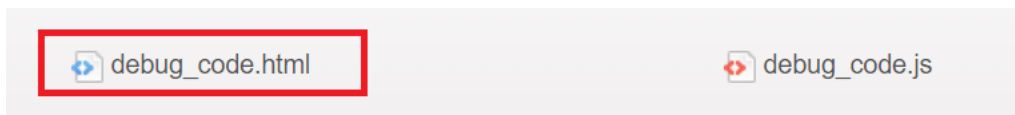
Note: Save your file after pasting the code. If you edit your code, refresh your browser on port 5500. No need to relaunch the application.

5. You will see the output as below.

Debugging Example

Enter Number 1: [_____]

Enter Number 2: [_____]

[Perform Operation]

6. Now enter the numbers in input boxes available and click on "Perform Operation" button.

7. Now to see the flow of this code, right-click in the same window of your browser where output is being displayed and then select "Inspect."

8. As soon as you click on the button, you will see the screen like as if it has paused, just like in the given screenshot. Also, value number 1 highlighted in red box in the screenshot shows that two numbers has been recived in variables a and b. Along with that, number 2 indicate where the pause has happened at debugger point, which is the current pointer of your code.

Paused in debu... ▐▶ ⌒

Debugging
Example

Enter Number 1:

`12`

Enter Number 2:

`13`

[Perform Operation]

⌂ Welcome   </> Elements   ▣ Console   🐛

▼ 🗔 top
  ▼ ☁ 127.0.0.1
    📄 more.

```
21      let num1 = parseInt(document.g
22      let num2 = parseInt(document.g
23
24          // Perform the operation
25          let result = multiply(num1
26
27          // Display the result
28          displayResult(result);
29      }
30
31      function multiply(a, b) {   a = 12,
32          // Introduce a debugger statem
33   2      debugger;
34
35          // Multiply the numbers
36          return a * b;
37      }
38
39      function displayResult(result) {
40          // Display the result in the p
41          const resultElement = document
42          resultElement.textContent = `T
43      }
44  </script>
```

{ } Line 33, Column 13   Coverage: n/a

9. Then you need to click the forward arrow highlighted with a red box in given screenshot.

10. As soon as you will click on that button, your current pointer will move to the `return a*b`, indicating that the flow of the code has now reached to this point.

Paused in debu... �II▶ ⌒

Debugging Example

Enter Number 1:

`12`

Enter Number 2:

`13`

Perform Operation

⌄ ⋮ ← | more.html ✕

⌄ ▢ top

⌄ ☁ 127.0.0.1

📄 more.

```
21        let num1 = parseInt(document.g
22        let num2 = parseInt(document.ge
23
24            // Perform the operation
25            let result = multiply(num1,
26
27            // Display the result
28            displayResult(result);
29        }
30
31        function multiply(a, b) {  a = 12,
32            // Introduce a debugger stateme
33            debugger;
34
35            // Multiply the numbers
36            return a * b;
37        }
38
39        function displayResult(result) {
40            // Display the result in the pa
41            const resultElement = document.
42            resultElement.textContent = `Th
43        }
44    </script>
```

11. Click the forward button again, and the pointer will move towards number 1 in the given screenshot, indicating that the **displayResult()** function is called. Also number 2 indicate num1 and num2 has 12 and 13 values respectivly.

```
result"></p>

>
ction performOperation() {
 // Get user input from input fields
 let num1 = parseInt(document.getElementById('input1').value);    num1 = 12
 let num2 = parseInt(document.getElementById('input2').value);    num2 = 13
                                                                                    2

     // Perform the operation
     let result = multiply(num1, num2);    result = 156, num1 = 12, num2 =

     // Display the result                                        3
     displayResult(result);        1


ction multiply(a, b) {
 // Introduce a debugger statement to pause execution
 debugger;

 // Multiply the numbers
 return a * b;


ction displayResult(result) {
```

12. Again click on forward button and the pointer will move to number 1 in given screenshot, indicating that it is now trying to access the element whose ID is result. Also **result** variable has the awnswer to the multiplication.

```
result"></p>

    function multiply(a, b) {
        // Introduce a debugger statement to pause execution
        debugger;

        // Multiply the numbers
        return a * b;
    }

    function displayResult(result) {    result = 156    2
        // Display the result in the paragraph element
   1     const resultElement = document.getElementById('result');
        resultElement.textContent =  The result is: ${result} ;
    }
 </script>
-- Code injected by live-server -->
script>
  // <![CDATA[   <-- For SVG support
  if ('WebSocket' in window) {
      (function () {
          function refreshCSS() {
              var sheets = [].slice.call(document.getElementsByTagName(
              var head = document.getElementsByTagName("head")[0];
```

13. Click the forward button again, and the pointer will shift towards number 1 in the screenshot, indicating that it has included the result in that element using `textContent`.

```
function multiply(a, b) {
    // Introduce a debugger statement to pause execution
    debugger;

    // Multiply the numbers
    return a * b;
}

function displayResult(result) {   result = 156
    // Display the result in the paragraph element
    const resultElement = document.getElementById('result');   resu
    resultElement.textContent = `The result is: ${result}`;
}
</script>
-- Code injected by live-server -->
script>
    // <![CDATA[  <-- For SVG support
    if ('WebSocket' in window) {
        (function () {
            function refreshCSS() {
                var sheets = [].slice.call(document.getElementsByTagName("
                var head = document.getElementsByTagName("head")[0];
```

14. Then, click on the forward button again, and the pointer will go to number 1, indicating that the button has been clicked, and it will also show the result at number 2 in given screenshot.

```
Paused in deb...  ▶|  ⌢      ⍈    ⟱    ▢      🏠 Welcome    </> Elements      >_ Console        ⚙ Sour

Debugging                    ⌄    ⋮      ⬅    more.html ✕
Example                  ▼ 🗔 top        3
                                         4    <head>
Enter Number 1:          ▼ ☁ 127.0.0.1   5        <meta charset="UTF-8">
12                         📄 more.       6        <title>Debugging Example</title>
                                         7    </head>
Enter Number 2:                          8
13                                       9    <body>
                                        10        <h1>Debugging Example</h1>
Perform Operation                       11        <label for="input1">Enter Number 1:</label>
                                        12        <input type="text" id="input1"><br><br>
The result is: 156                      13        <label for="input2">Enter Number 2:</label>
                                        14        <input type="text" id="input2"><br><br>
        2                        1   15    <button onclick="performOperation()">Perfor
                                        16    <p id="result"></p>
                                        17
                                        18        <script>
                                        19            function performOperation() {
                                        20                // Get user input from input fields
                                        21                let num1 = parseInt(document.getEle
                                        22                let num2 = parseInt(document.getEle
                                        23
                                        24                // Perform the operation
                                        25                let result = multiply(num1, num
                                        26
```
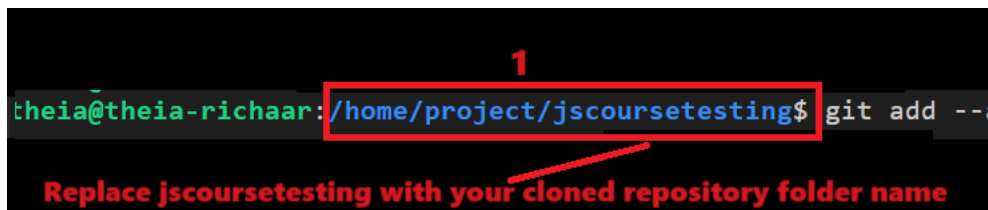
15. After this, it will come out of the debugging zone.

Step 4: Perform Git commands

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

    1. 1
    1. `git add --a`
    [Copied!]

   ○ Make sure the terminal has the path as follows:

   ```
                                    1
   theia@theia-richaar:/home/project/jscoursetesting$ git add --a
   ```
   Replace jscoursetesting with your cloned repository folder name

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

    1. 1
    1. `git config --global user.email "you@example.com"`
    [Copied!]

    1. 1
    1. `git config --global user.name "Your Name"`
    [Copied!]

   Then perform the commit command as given:

    1. 1
    1. `git commit -m "message"`
    [Copied!]

3. Next, perform `git push` by writing the given command in the terminal.

```
1. 1
1. git push origin
```
[Copied!]

- After the push command, the system will prompt you to enter your username and password. Enter the username for your GitHub account and the password that you created in the first lab. After entering the credentials, all of your latest folders and files will be pushed to your GitHub repository.

Practice task

1. You need to perform arithmetic operations such as addition, multiplication, and division simultaneously using the same function.

2. Additionally, you need to check the flow of the code, which will depend on the arithmetic operation you are performing first.

3. Also, try assigning one value in the form of characters and observe how this value is displayed using the debugger.

Summary

1. You have learned how to debug code to see the step by step flow of the program.

2. You have also learned that how using debugger keyword can also allow you to see the flow of input values which is being stored in variables in javaScript.