

Lesson 05 Demo 01

Working with IIFEs Callbacks and Closures

Objective: To demonstrate the process of implementing Immediately Invoked Function Expressions (IIFEs), callbacks, and closures in JavaScript for enhancing modularity, encapsulation, and asynchronous execution

Tools required: Visual Studio Code

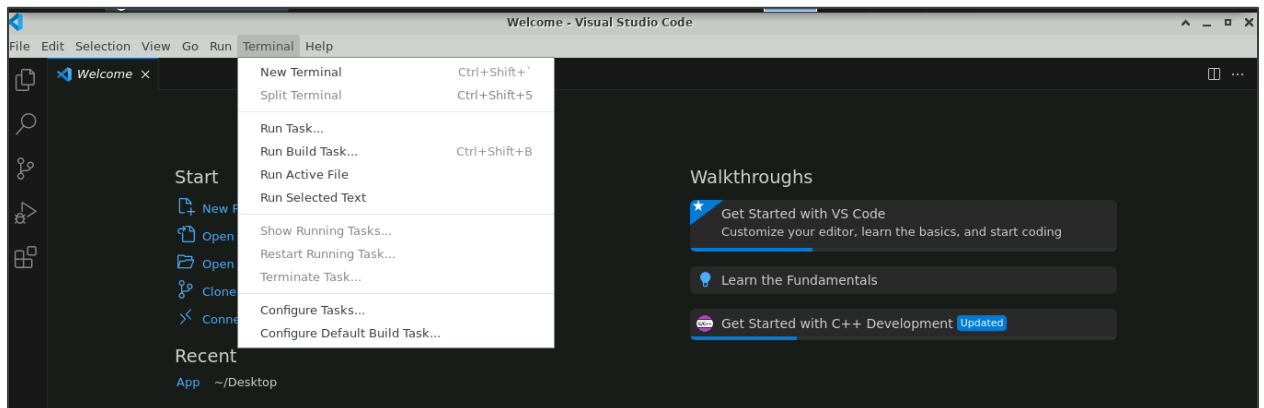
Prerequisites: None

Steps to be followed:

1. Write a JavaScript program with IIFEs, callbacks, and closures
2. Test and verify the IIFEs, callbacks, and closures in action

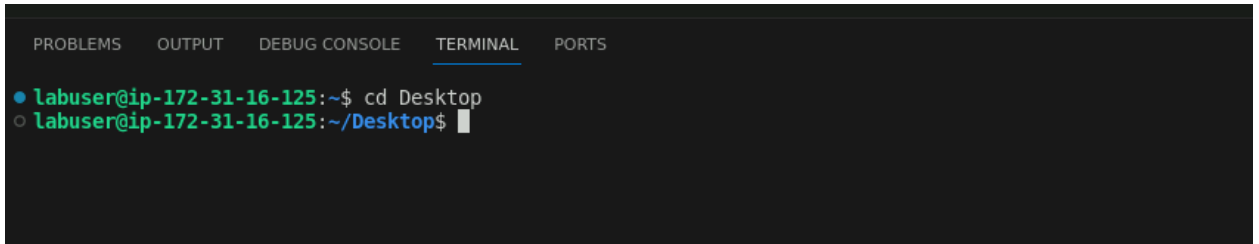
Step 1: Write a JavaScript program with IIFEs, callbacks, and closures

1.1 Open Visual Studio Code and navigate to the terminal to create an **src** folder



1.2 Execute the following command to navigate to the desktop directory:

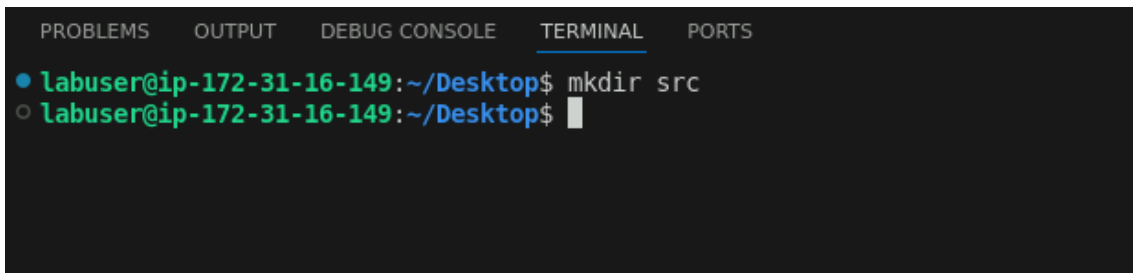
cd Desktop

A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active. It shows a command prompt for 'labuser@ip-172-31-16-125:~\$' followed by the command 'cd Desktop'. The next line shows the prompt 'labuser@ip-172-31-16-125:~/Desktop\$' with a cursor, indicating the command was successful.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● labuser@ip-172-31-16-125:~$ cd Desktop
○ labuser@ip-172-31-16-125:~/Desktop$
```

1.3 Run the following command to create a folder named **src**:

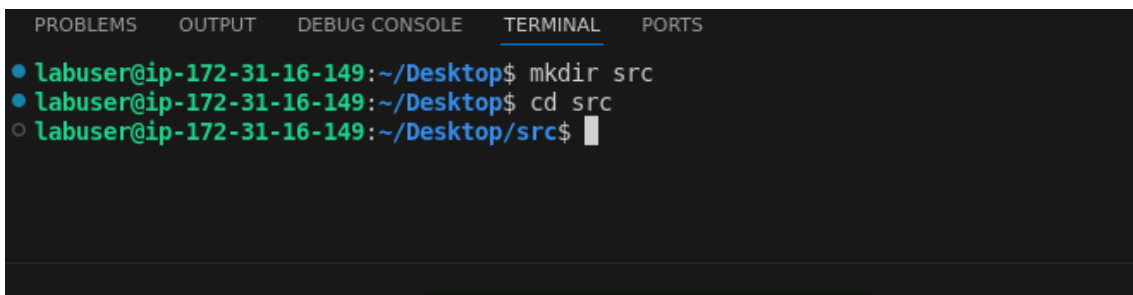
mkdir src

A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active. It shows a command prompt for 'labuser@ip-172-31-16-149:~/Desktop\$' followed by the command 'mkdir src'. The next line shows the prompt 'labuser@ip-172-31-16-149:~/Desktop\$' with a cursor, indicating the command was successful.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● labuser@ip-172-31-16-149:~/Desktop$ mkdir src
○ labuser@ip-172-31-16-149:~/Desktop$
```

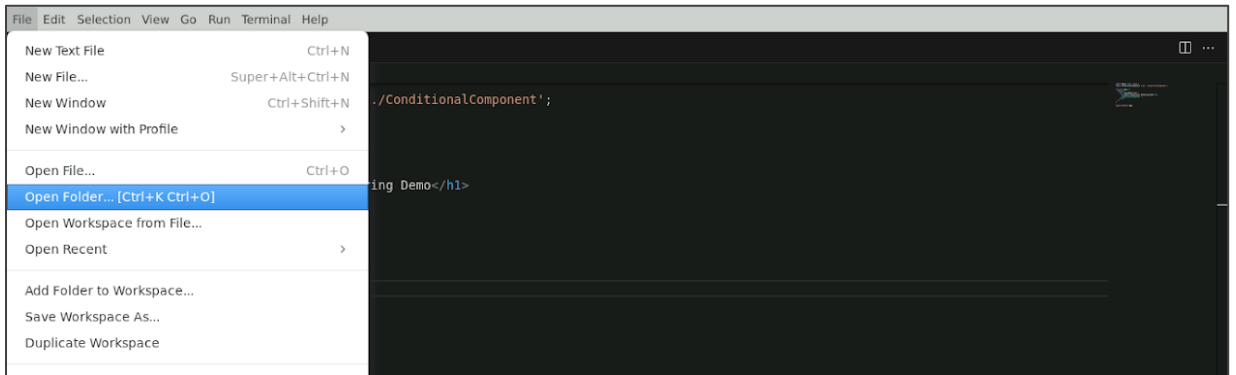
1.4 Execute the following command to navigate to the **src** directory:

cd src

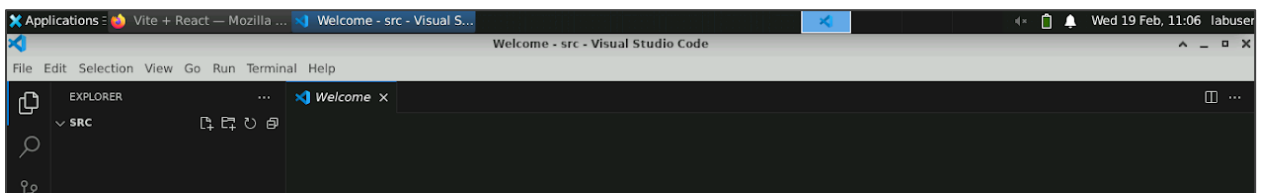
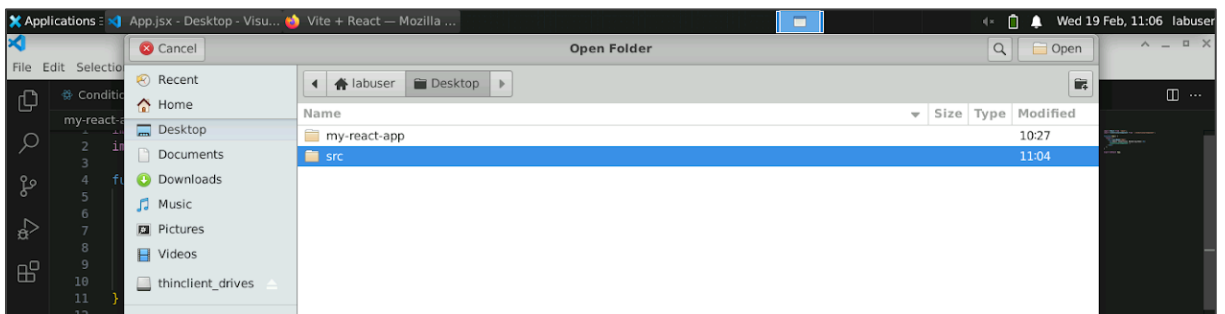
A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active. It shows a sequence of commands: first 'mkdir src' and then 'cd src'. The prompt changes from 'labuser@ip-172-31-16-149:~/Desktop\$' to 'labuser@ip-172-31-16-149:~/Desktop/src\$' after the second command, indicating successful navigation.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● labuser@ip-172-31-16-149:~/Desktop$ mkdir src
● labuser@ip-172-31-16-149:~/Desktop$ cd src
○ labuser@ip-172-31-16-149:~/Desktop/src$
```

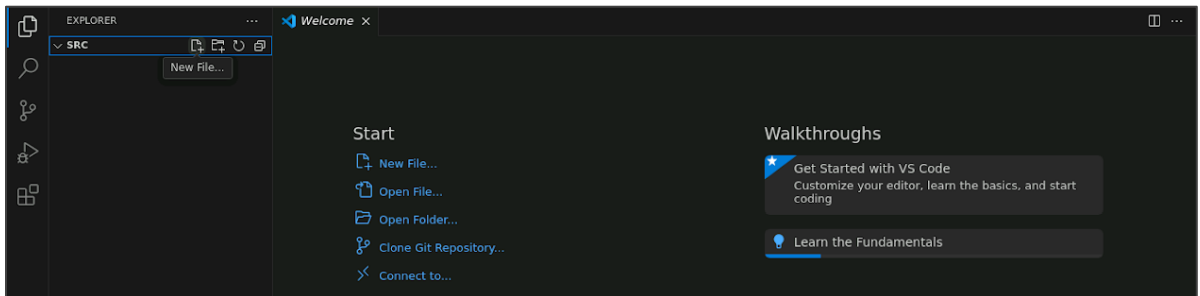
1.5 Click on **File** and open the folder



1.6 Open the **src** folder

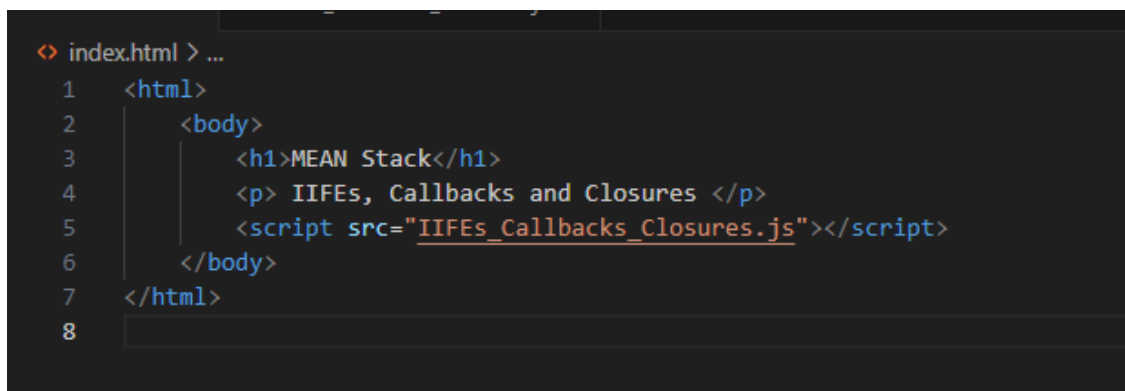


1.7 Click on the **src** folder of the project, select the **New File...** option, and enter the filename as **index.html**



1.8 Write the given code in **index.html**:

```
<html>
  <body>
    <h1>MEAN Stack</h1>
    <p> IIFEs, Callbacks and Closures </p>
    <script src="IIFEs_Callbacks_Closures.js"></script>
  </body>
</html>
```

A screenshot of a code editor with a dark background. The file name 'index.html' is visible in the top left. The code is written in a light blue font and is as follows:

```
<html>
  <body>
    <h1>MEAN Stack</h1>
    <p> IIFEs, Callbacks and Closures </p>
    <script src="IIFEs_Callbacks_Closures.js"></script>
  </body>
</html>
```

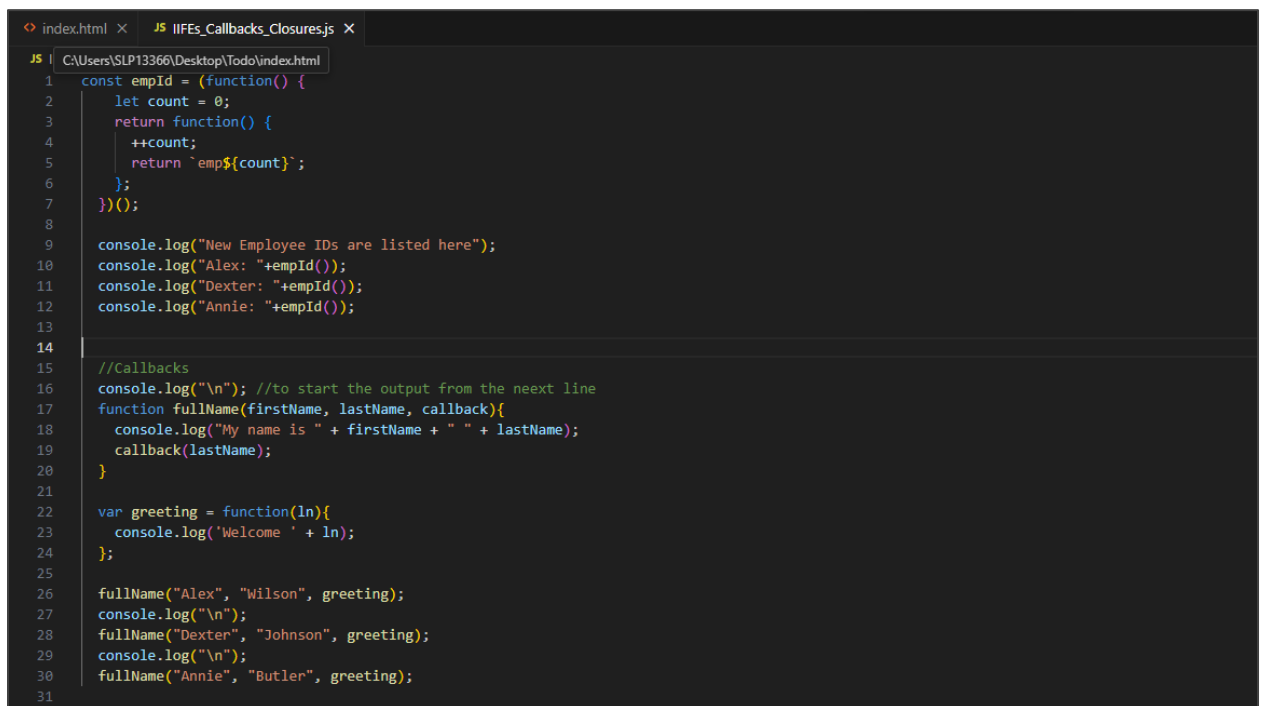
1.9 Click on the **src** folder of the project, select the **New File...** option, enter the filename as **IIFEs_Callbacks_Closures.js**, and write the code shown below:

```
const empId = (function() {
  let count = 0;
  return function() {
    ++count;
    return `emp${count}`;
  };
})();
console.log("New Employee IDs are listed here");
console.log("Alex: "+empId());
console.log("Dexter: "+empId());
console.log("Annie: "+empId());
//Callbacks
console.log("\n"); //to start the output from the next line
function fullName(firstName, lastName, callback){
  console.log("My name is " + firstName + " " + lastName);
```

```

    callback(lastName);
}
var greeting = function(ln){
    console.log('Welcome ' + ln);
};
fullName("Alex", "Wilson", greeting);
console.log("\n");
fullName("Dexter", "Johnson", greeting);
console.log("\n");
fullName("Annie", "Butler", greeting);

```



The screenshot shows a code editor with two tabs: 'index.html' and 'JS IIFEs_Closures_Closures.js'. The code in the JS file is as follows:

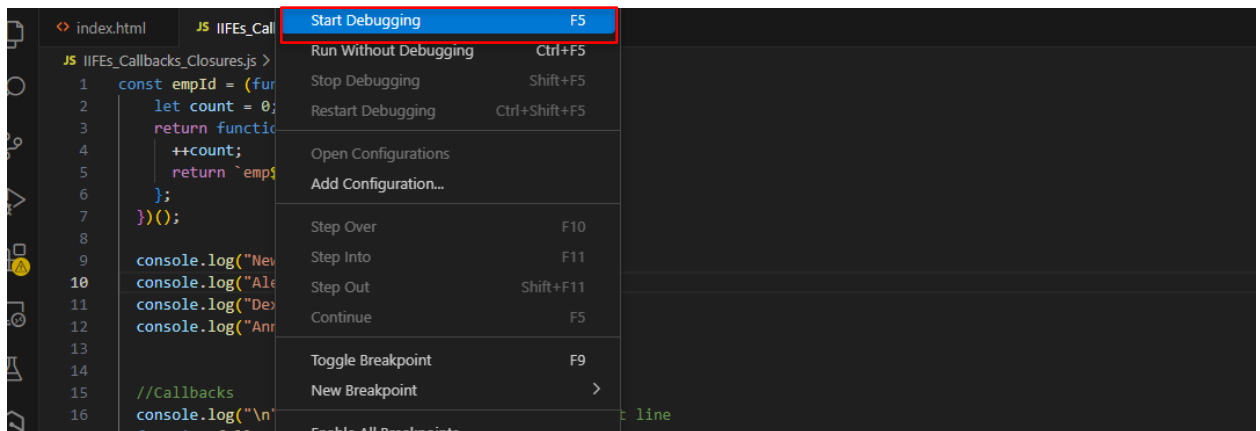
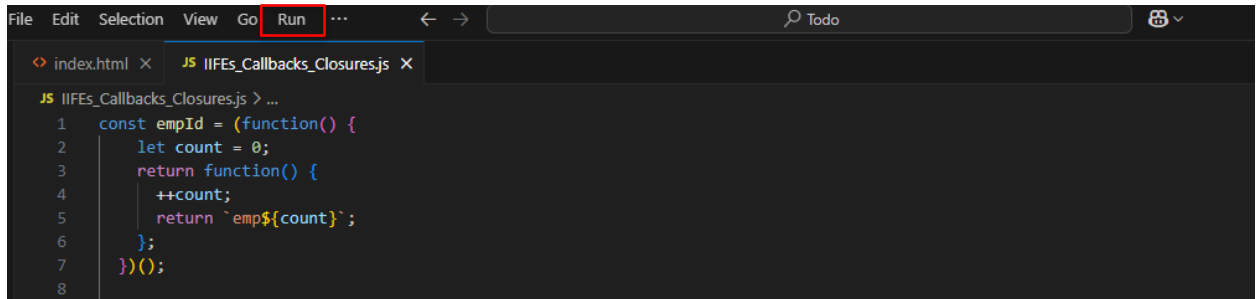
```

1  const empId = (function() {
2      let count = 0;
3      return function() {
4          ++count;
5          return `emp${count}`;
6      };
7  })();
8
9  console.log("New Employee IDs are listed here");
10 console.log("Alex: " + empId());
11 console.log("Dexter: " + empId());
12 console.log("Annie: " + empId());
13
14
15 //Callbacks
16 console.log("\n"); //to start the output from the next line
17 function fullName(firstName, lastName, callback){
18     console.log("My name is " + firstName + " " + lastName);
19     callback(lastName);
20 }
21
22 var greeting = function(ln){
23     console.log('Welcome ' + ln);
24 };
25
26 fullName("Alex", "Wilson", greeting);
27 console.log("\n");
28 fullName("Dexter", "Johnson", greeting);
29 console.log("\n");
30 fullName("Annie", "Butler", greeting);
31

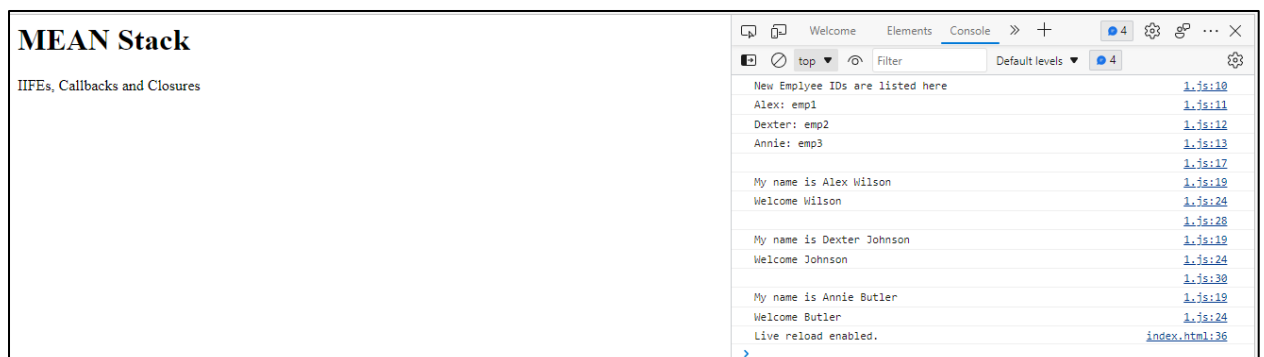
```

Step 2: Test and verify the IIFEs, callbacks, and closures in action

2.1 Click on **Run** and then on **Start Debugging** to execute the JavaScript file



2.2 When the server starts running, right-click and select the **Inspect Element** option and click on the **Console** tab



By following the above steps, you have successfully implemented IIFEs, callbacks, and closures in JavaScript, demonstrating how they help in creating modular, encapsulated,

and efficient code execution with controlled variable scope and asynchronous behavior.