Internship Report – Frontend Dev

Week 4: JavaScript Basics

Name: Zainab	
Father Name: Assad Qayyum	
Date: 17th July, 2025	
Internship Domain: Front-end Intern	
Task: JS - Functions (Regular, Arrow), Scope	

Task Overview: (Day4)

Today's focus was to understand how **functions** and **scope** work in JavaScript. These are foundational concepts in JavaScript that help in organizing, reusing, and controlling the accessibility of code.

Content Covered:

- JavaScript Functions: (Regular, Arrow)
- Scope in JavaScript
    - Global scope
    - Function scope
    - Block scope

1. Functions in JavaScript:

A function is a block of code designed to perform a **particular task**. It runs only when it is called. Functions help you:

- Avoid code repetition
- Organize code into logical blocks
- Make your code modular and reusable

Parts of a Function:

1) **Function Name:** Identifier to call the function
2) **Parameters:** Placeholders for values the function will use
3) **Function Body:** Block of code that runs when the function is called
4) **Return Statement:** Sends a value back to where the function was called

How to Create a Function:

A. Function Declaration (Regular Function)

```
function greet(name) {
    return "Hello, " + name + "!";
  }
```

- **function** is a keyword
- **greet** is the name of the function
- **name** is a parameter
- The function returns a string using **return**

B. How to call it:

```
console.log(greet("Zainab"));        // Output: Hello, Zainab!
```

Types of functions:

Two main types:

a) Regular function:

A normal way to write a function that can be used anywhere and **has its own this** keyword.

Syntax:

```
function greet(name) {
  return "Hello, " + name;
}
```

- Defined using the function keyword
- Supports hoisting (can be called before declared)
- Has its own **this** context

b) Arrow function :

A shorter way to write functions that takes this from where it was written (**doesn't have its own this**).

Syntax:

```
const greet = (name) => {
  return "Hi, " + name;
};
```

- Introduced in ES6
- More concise syntax
- Inherits this from outer scope (does not have its own this)
- Cannot be used as constructors and not hoisted.

2. Scope in JavaScript:

Scope defines **where** in your code **variables are accessible** (visible). Understanding scope is crucial for avoiding errors and bugs in JavaScript.

Types of Scope in JavaScript:

a) Global Scope:

Variables **declared outside** all functions or blocks. And can be **accessed anywhere** in the code.

```
let message = "Hello World";
function sayHi() {
  console.log(message); // Accessible
}
console.log(message); // Accessible
```

b) Function scope:

Variables declared **inside a function** are accessible only within that function.

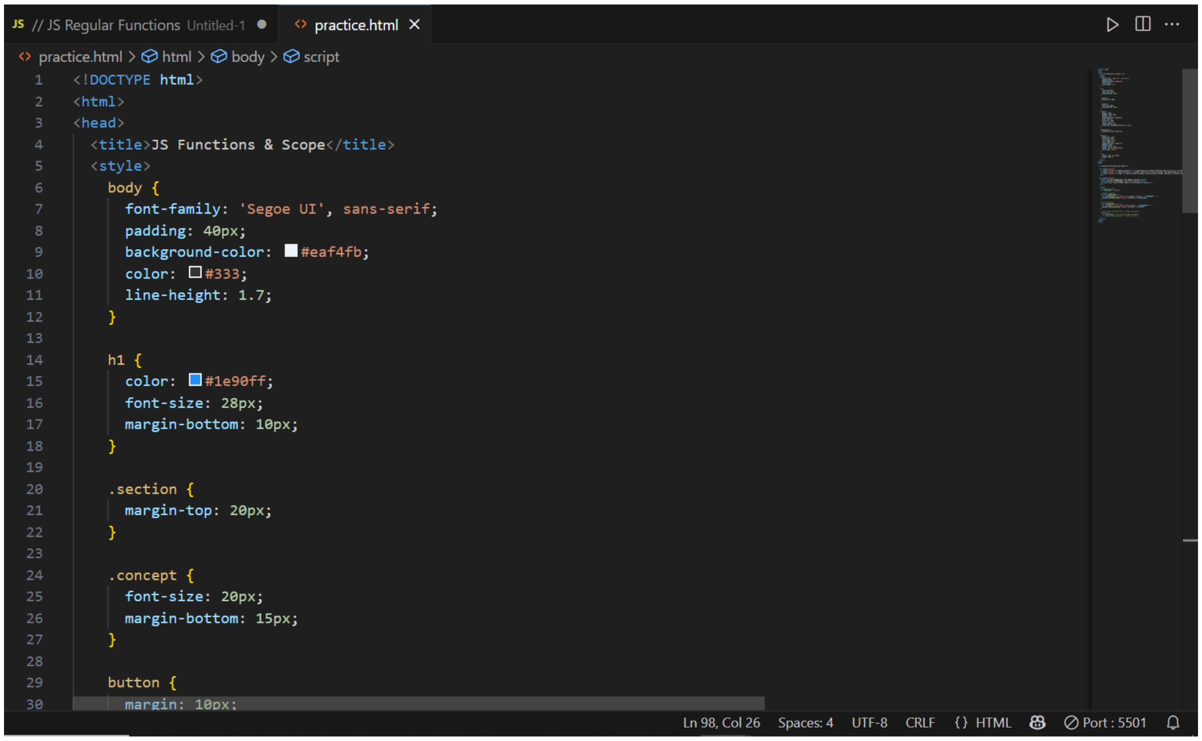
```
function showName() {
  let name = "Ali";
  console.log(name); // works
}
// console.log(name);   error
```

c) Block scope:

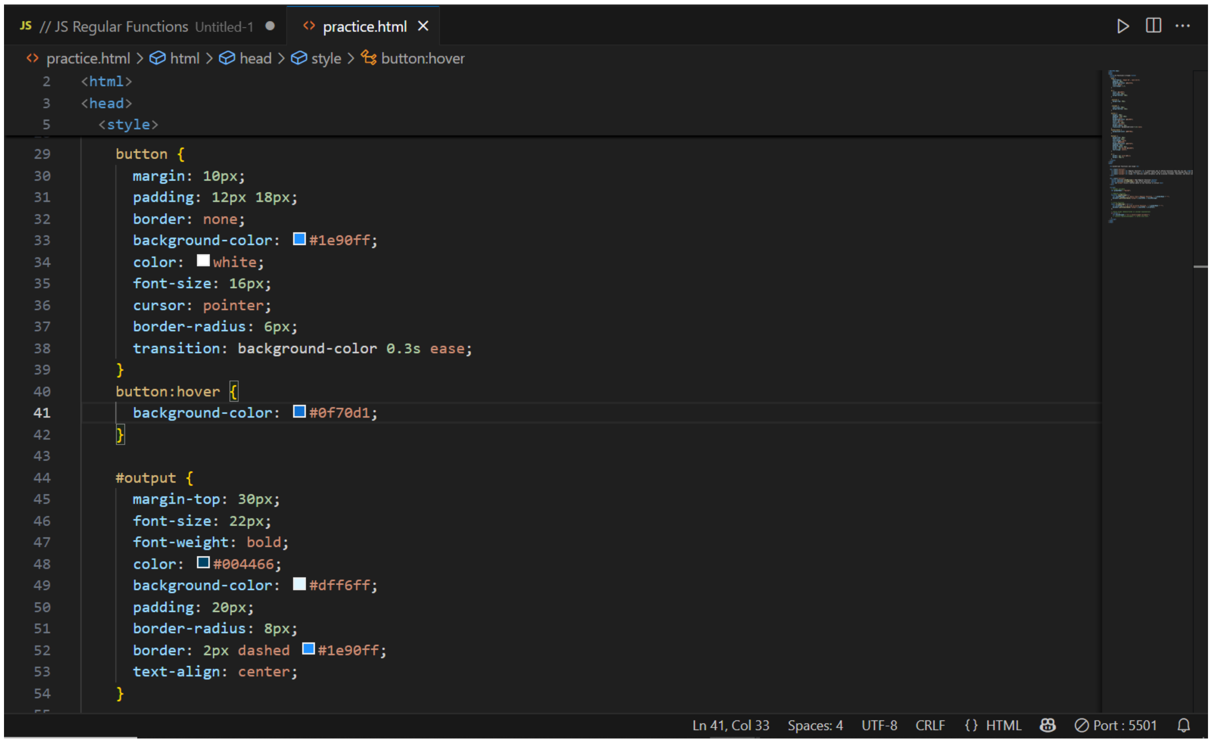
- Block = code inside { }, like if, for, etc.
- Variables declared with let or const inside a block are **not accessible outside**

```
if (true) {
  let color = "blue";
}
// console.log(color);  error
```

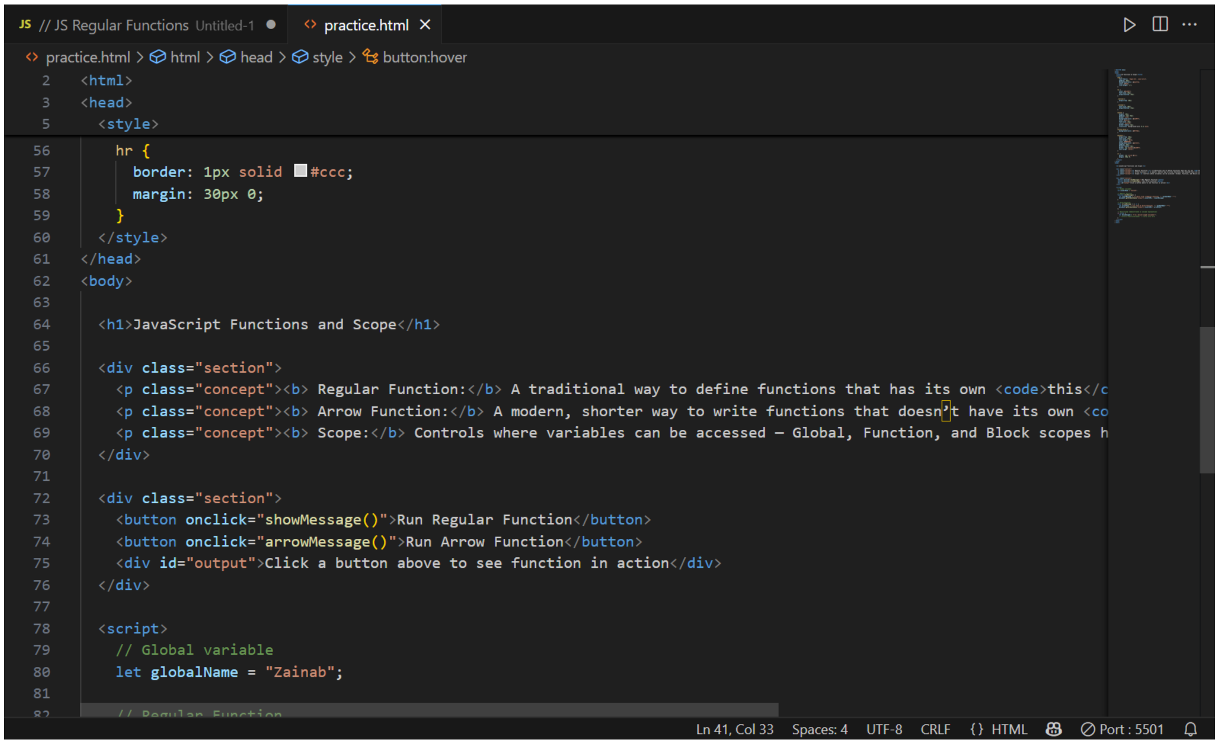
Practice Code:


// JS Regular Functions Untitled-1
practice.html
practice.html > html > body > script
Ln 98, Col 26
Spaces: 4
UTF-8
CRLF
{} HTML
Port : 5501

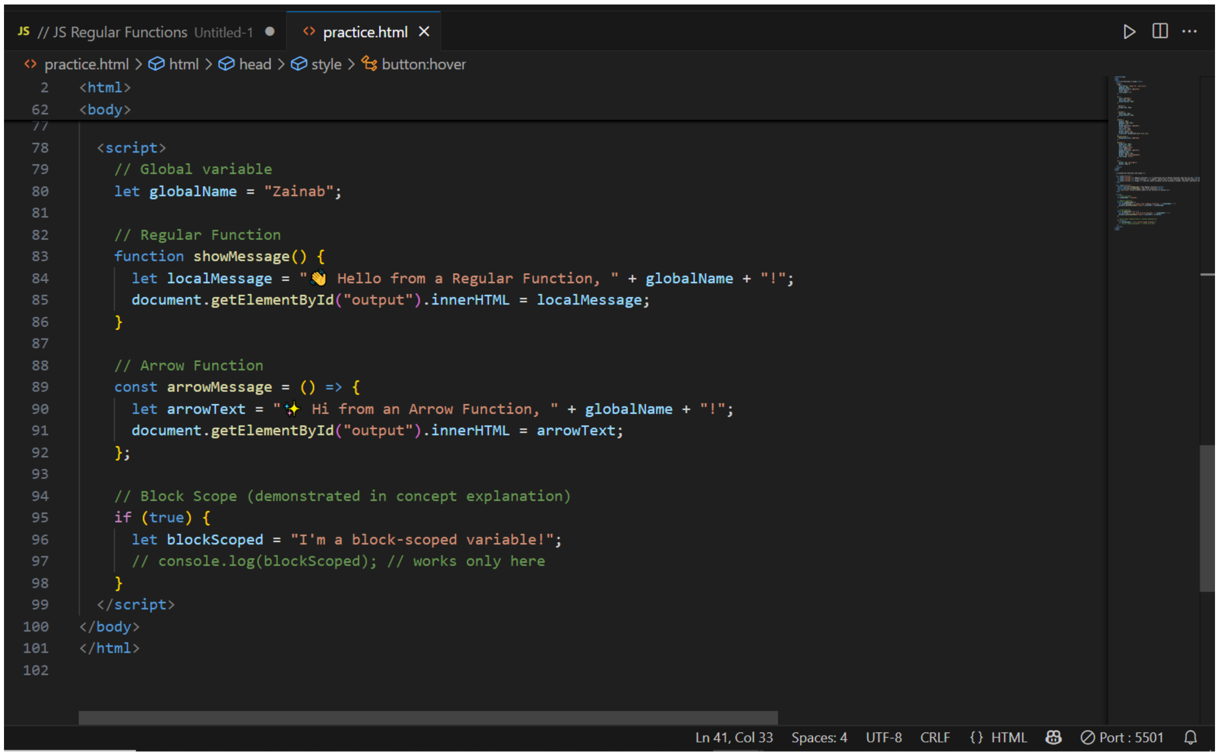
```
<!DOCTYPE html>
<html>
<head>
  <title>JS Functions & Scope</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      padding: 40px;
      background-color: #eaf4fb;
      color: #333;
      line-height: 1.7;
    }

    h1 {
      color: #1e90ff;
      font-size: 28px;
      margin-bottom: 10px;
    }

    .section {
      margin-top: 20px;
    }

    .concept {
      font-size: 20px;
      margin-bottom: 15px;
    }

    button {
      margin: 10px;
```


// JS Regular Functions Untitled-1
practice.html
practice.html > html > head > style > button:hover
Ln 41, Col 33 Spaces: 4 UTF-8 CRLF {} HTML Port : 5501

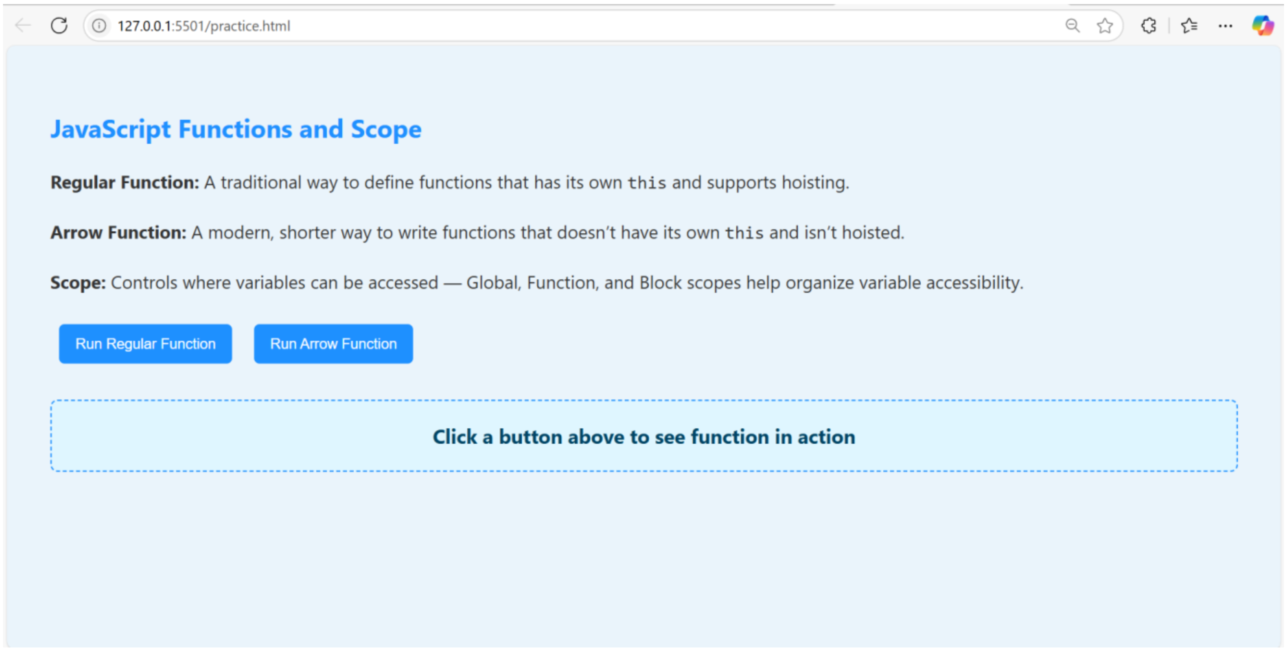
```
<html>
<head>
  <style>
    button {
      margin: 10px;
      padding: 12px 18px;
      border: none;
      background-color: #1e90ff;
      color: white;
      font-size: 16px;
      cursor: pointer;
      border-radius: 6px;
      transition: background-color 0.3s ease;
    }
    button:hover {
      background-color: #0f70d1;
    }

    #output {
      margin-top: 30px;
      font-size: 22px;
      font-weight: bold;
      color: #004466;
      background-color: #dff6ff;
      padding: 20px;
      border-radius: 8px;
      border: 2px dashed #1e90ff;
      text-align: center;
    }
```


// JS Regular Functions Untitled-1
practice.html
practice.html > html > head > style > button:hover
Ln 41, Col 33 Spaces: 4 UTF-8 CRLF {} HTML Port : 5501

```
<html>
<head>
  <style>
    hr {
      border: 1px solid #ccc;
      margin: 30px 0;
    }
  </style>
</head>
<body>

  <h1>JavaScript Functions and Scope</h1>

  <div class="section">
    <p class="concept"><b> Regular Function:</b> A traditional way to define functions that has its own <code>this</c
    <p class="concept"><b> Arrow Function:</b> A modern, shorter way to write functions that doesn't have its own <co
    <p class="concept"><b> Scope:</b> Controls where variables can be accessed — Global, Function, and Block scopes h
  </div>

  <div class="section">
    <button onclick="showMessage()">Run Regular Function</button>
    <button onclick="arrowMessage()">Run Arrow Function</button>
    <div id="output">Click a button above to see function in action</div>
  </div>

  <script>
    // Global variable
    let globalName = "Zainab";

    // Regular Function
```


// JS Regular Functions Untitled-1
practice.html
practice.html > html > head > style > button:hover
Ln 41, Col 33 Spaces: 4 UTF-8 CRLF {} HTML Port : 5501

```
<html>
<body>

  <script>
    // Global variable
    let globalName = "Zainab";

    // Regular Function
    function showMessage() {
      let localMessage = "👋 Hello from a Regular Function, " + globalName + "!";
      document.getElementById("output").innerHTML = localMessage;
    }

    // Arrow Function
    const arrowMessage = () => {
      let arrowText = "✨ Hi from an Arrow Function, " + globalName + "!";
      document.getElementById("output").innerHTML = arrowText;
    };

    // Block Scope (demonstrated in concept explanation)
    if (true) {
      let blockScoped = "I'm a block-scoped variable!";
      // console.log(blockScoped); // works only here
    }
  </script>
</body>
</html>
```



JavaScript Functions and Scope

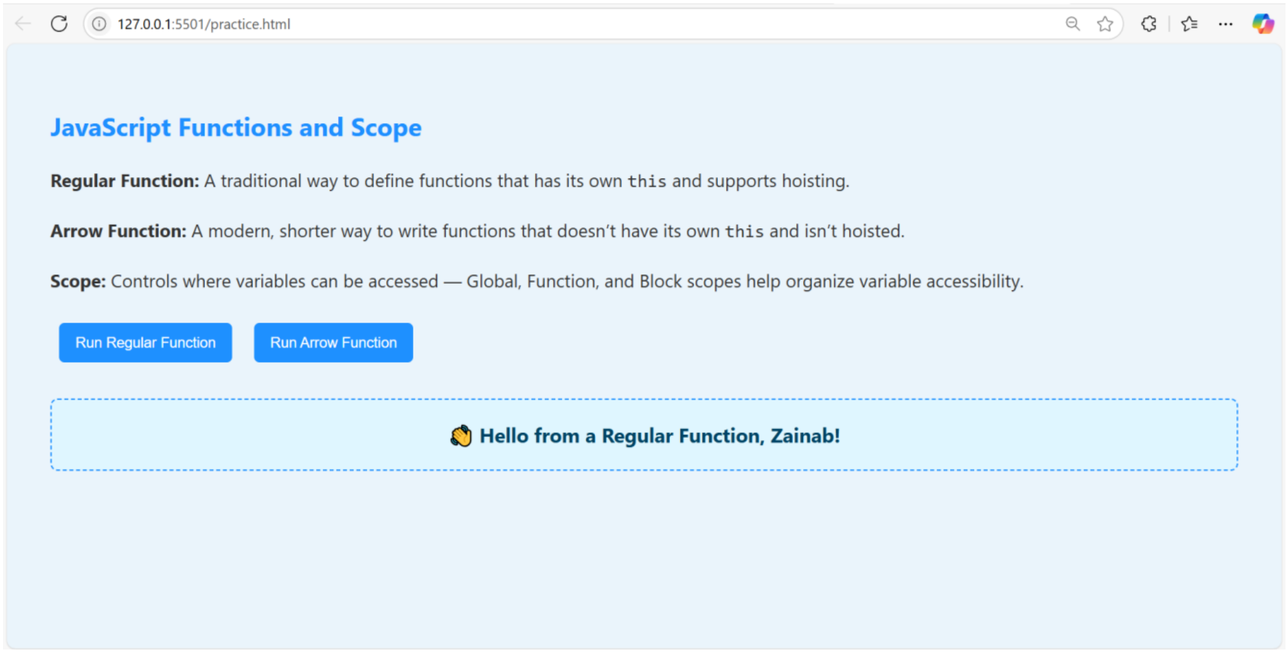
Regular Function: A traditional way to define functions that has its own `this` and supports hoisting.

Arrow Function: A modern, shorter way to write functions that doesn't have its own `this` and isn't hoisted.

Scope: Controls where variables can be accessed — Global, Function, and Block scopes help organize variable accessibility.

Click a button above to see function in action

When “Run Regular function” button is clicked:


127.0.0.1:5501/practice.html

JavaScript Functions and Scope

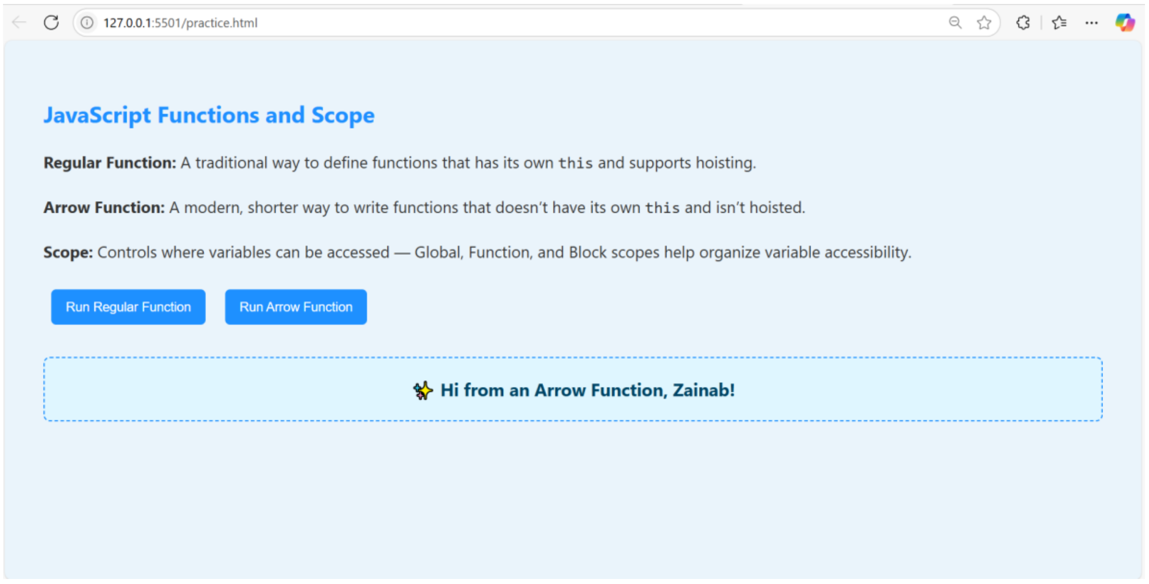
Regular Function: A traditional way to define functions that has its own `this` and supports hoisting.

Arrow Function: A modern, shorter way to write functions that doesn't have its own `this` and isn't hoisted.

Scope: Controls where variables can be accessed — Global, Function, and Block scopes help organize variable accessibility.

👋 Hello from a Regular Function, Zainab!

When “Run Arrow function” button is clicked:


127.0.0.1:5501/practice.html

JavaScript Functions and Scope

Regular Function: A traditional way to define functions that has its own `this` and supports hoisting.

Arrow Function: A modern, shorter way to write functions that doesn't have its own `this` and isn't hoisted.

Scope: Controls where variables can be accessed — Global, Function, and Block scopes help organize variable accessibility.

✨ Hi from an Arrow Function, Zainab!

Conclusion:

Understanding functions and scope is essential for writing clean, reusable, and error-free JavaScript code. Regular and arrow functions allow different styles of function creation, while scope determines how data is accessed and controlled. These concepts are the building blocks for interactive web applications.