📘 Internship Report ProSensia

📅 **Week:** 4
📆 **Day:** 5
🧑‍💻 **Name:** Hamza Rafique
🏢 **Organization:** ProSensia
🧠 **Topics Covered:**

- JavaScript Functions (Regular & Arrow)
- Function Scope

✅ 1. JavaScript Functions: An Introduction

Functions are the core building blocks of any JavaScript program. They allow us to organize code into reusable blocks that perform specific tasks. In JavaScript, there are two main types of functions:

🔷 Regular (Traditional) Functions

- Declared using the function keyword.
- Can be named or anonymous.
- Hoisted — meaning they can be called before they are defined.
- Have their own this context, which behaves differently depending on how the function is invoked (e.g., as a method or standalone function).

Example:

javascript

CopyEdit

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

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

🔷 Arrow Functions

- Introduced in ES6.
- Have a more concise syntax using =>.
- Do **not** have their own this; they inherit it from the enclosing scope (lexical this).
- Cannot be hoisted — must be defined before calling.
- Best suited for short and simple operations.

Example:

javascript

CopyEdit

```
const greet = (name) => `Hello, ${name}`;

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

✅ 2. Differences Between Regular and Arrow Functions

Feature	**Regular Function**	**Arrow Function**
Syntax	Longer	Shorter (=>)
this context	Own context	Lexical (from outer scope)
Hoisting	Yes	No
Use in Objects	Recommended	Use with caution

✅ 3. JavaScript Scope (Function Scope)

Scope refers to the visibility or accessibility of variables in different parts of the program.

🔷 Function Scope

- Variables declared inside a function using let, const, or var are **local** to that function.
- Cannot be accessed outside the function.

Example:

javascript

CopyEdit

```
function showScope() {
  let message = "I am inside the function";
  console.log(message);
}

showScope(); // Works
console.log(message); // Error: message is not defined
```

🔷 Lexical Scope

- In JavaScript, scope is determined at the time of writing the code (not at runtime).
- A function can access variables defined in its outer (enclosing) scope.

Example:

javascript

CopyEdit

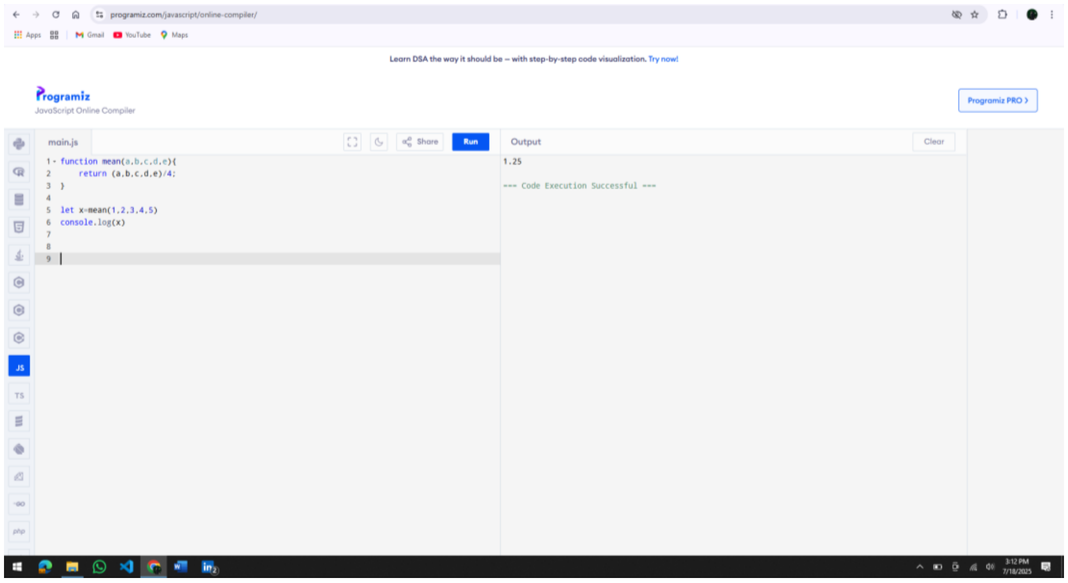
```
let outerVar = "Accessible";

function outer() {
  function inner() {
    console.log(outerVar); // Output: Accessible
  }
  inner();
}
outer();
```

💡 Key Takeaways

- Use **regular functions** when you need your own this or when defining object methods.
- Use **arrow functions** for short, anonymous functions or when preserving the this context is important.
- Understand the **scope** to avoid variable access errors and to write more secure and modular code.

📷 Screenshots:

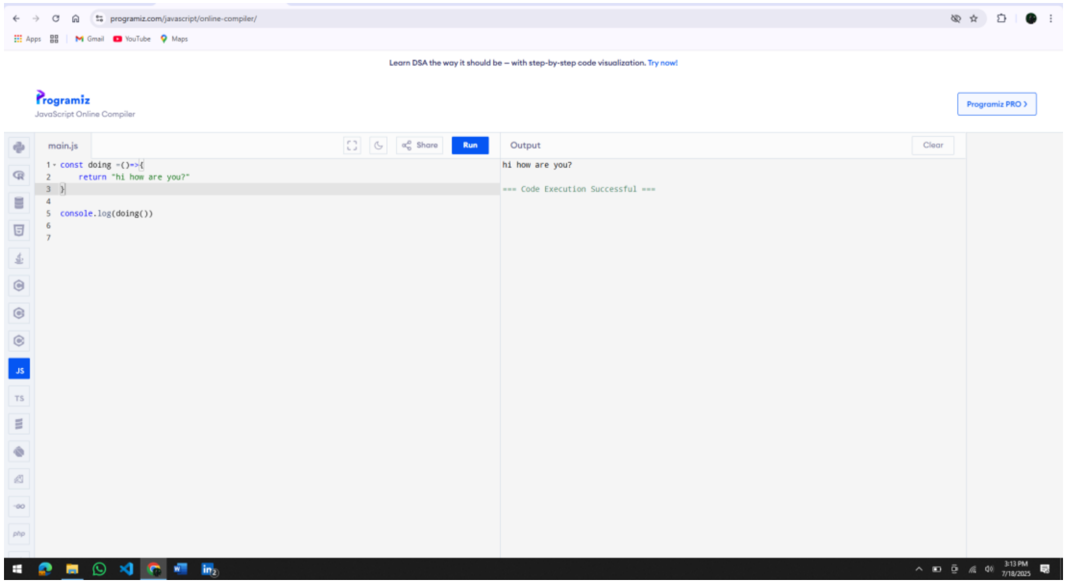

programiz.com/javascript/online-compiler/
Apps
Gmail
YouTube
Maps
Learn DSA the way it should be – with step-by-step code visualization. Try now!
Programiz
JavaScript Online Compiler
Programiz PRO
main.js
Share
Run
Output
Clear
3:12 PM
7/18/2025

```javascript
function mean(a,b,c,d,e){
    return (a,b,c,d,e)/4;
}

let x=mean(1,2,3,4,5)
console.log(x)
```

```
1.25

=== Code Execution Successful ===
```


programiz.com/javascript/online-compiler/
Apps
Gmail
YouTube
Maps
Learn DSA the way it should be – with step-by-step code visualization. Try now!
Programiz
JavaScript Online Compiler
Programiz PRO
main.js
Share
Run
Output
Clear
3:13 PM
7/18/2025

```javascript
const doing =()=>{
    return "hi how are you?"
}

console.log(doing())
```

```
hi how are you?

=== Code Execution Successful ===
```