Report Day: 4.5

Submitted By: Ayesha Abid

Submitted To: Mr. Ali Hyder Hidayat

Group: Frontend Development @ProSensia

Dated: 18/07/2025

Learning Outcomes:

1. Introduction

Functions in JavaScript are blocks of reusable code designed to perform a particular task. They promote code reusability, modularity, and clarity. JavaScript supports different types of functions, including regular functions and arrow functions. Additionally, the concept of scope defines the accessibility of variables within functions and blocks.

2. Functions in JavaScript

Functions are used to group statements that perform a specific task. They can be **called/invoked** multiple times.

2.1 Regular Functions

Regular functions use the function keyword.

Syntax:

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
greet("Ayesha");  // Output: Hello, Ayesha!
```

2.2 Function with Return Value

A function can also return a value using the return keyword.

Syntax:

```
function add(a, b) {
  return a + b;}
let result = add(5, 3);  // result = 8
```

2.3 Arrow Functions (ES6)

Arrow functions are a shorter syntax for writing functions. They are **anonymous** by nature and do **not have their own this** context.

Syntax:

const functionName = (parameters) => {

  // code to execute

};

3. Scope in JavaScript

Scope refers to the current context of code, which determines the accessibility of variables.

3.1 Global Scope

Variables declared **outside** any function or block are in the **global scope**.

```
let name = "Ayesha";

function greet() {
  console.log("Hello " + name);  // Accessible
}
```

3.2 Function Scope

Variables declared **inside a function** are not accessible from outside.

```
function test() {
  let x = 10;
  console.log(x);  // Accessible
}
console.log(x);  // Error: x is not defined
```

3.3 Block Scope (let, const)

Variables declared with let or const inside {} are block-scoped.

```
if (true) {
  let a = 5;
  const b = 10;
}
// console.log(a);  // Error: a is not defined
```

3.4 Scope Chain

When accessing a variable, JavaScript looks in the current scope first, then moves up to the outer scope, and continues until it reaches the global scope.

```
let globalVar = "global";

function outer() {
  let outerVar = "outer";

  function inner() {
    console.log(globalVar); // accessible
    console.log(outerVar);  // accessible
  }

  inner();
}
```

5. Conclusion

Understanding **functions** both regular and arrow and **scope** is crucial for writing effective JavaScript code. Functions improve reusability and readability, while scope ensures variables are kept in the right context.