Chapter: Functions and Scope

This chapter introduces the concept of functions and explores how scope determines variable accessibility in JavaScript programs. These are foundational concepts essential for writing reusable and organized code.

---

1. Functions
What are Functions?
Functions are reusable blocks of code designed to perform specific tasks.
They help organize and simplify code by breaking it into smaller, manageable pieces.
Types of Functions

Function Declaration
A standard way to define a function.

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

Function Expression
Assigns a function to a variable.

```
const add = function (a, b)
{
   return a + b;
};
console.log(add(5, 3)); // Output: 8
```

Arrow Function (ES6)
A concise syntax for functions.

```
const multiply = (a, b) => a * b;
console.log(multiply(4, 2)); // Output: 8
```

Parts of a Function
Function Name: Identifies the function (optional in expressions).
Parameters: Input values that the function can use (optional).
Body: The block of code that runs when the function is called.
Return Statement: Sends the output of the function back to the caller.

Why Use Functions?

- Code reusability.
- Improves readability and maintainability.
- Allows abstraction, where details are hidden from the user.

Example:

```
function calculateArea(length, width)
{
   return length * width;
}
let area = calculateArea(5, 10);
console.log("Area:", area); // Output: Area: 50
```

---

2. Scope
What is Scope?
Scope determines where variables, functions, and objects are accessible in your code.

Types of Scope
Global Scope:
Variables declared outside any function or block are global.
They can be accessed from anywhere in the code.

```
let globalVar = "I am global";
function showGlobal()
{
   console.log(globalVar); // Accessible
}
showGlobal(); // Output: I am global
```

Local Scope:
Variables declared inside a function are local.
They are only accessible within that function.

```
function localExample()
{
   let localVar = "I am local";
   console.log(localVar);
}
localExample(); // Output: I am local
console.log(localVar); // Error: localVar is not defined
```

Block Scope (ES6)
Variables declared with let and const inside a block ({}) are block-scoped.
They are only accessible within that block.

```
 {
   let blockVar = "I am block-scoped";
   console.log(blockVar);
}
// console.log(blockVar); // Error: blockVar is not defined
```

Scope Chain
When a variable is accessed, JavaScript searches for it in the following order:
Local Scope
Parent Function's Scope
Global Scope

Example:
```
let globalVar = "Global";
function outerFunction() {
   let outerVar = "Outer";

   function innerFunction() {
      let innerVar = "Inner";
      console.log(globalVar); // Accessible
      console.log(outerVar);  // Accessible
      console.log(innerVar);  // Accessible
   }
```

```
    innerFunction();
}
outerFunction();
```

Common Mistakes with Scope

Accidentally Overwriting Global Variables

Avoid creating variables without let, const, or var inside a function, as they default to global scope.

```
function mistake() {
    globalVar = "Oops, global!";
}
mistake();
console.log(globalVar); // Unintentionally global
```

Shadowing

A local variable with the same name as a global variable hides the global variable within its scope.

```
let name = "Global Name";
function shadowExample()
 {
    let name = "Local Name";
    console.log(name); // Output: Local Name
}
shadowExample();
console.log(name); // Output: Global Name
```

Real-World Examples

Login System

A function to validate a username and password.

```
function validateLogin(username, password)
{
    const storedUsername = "admin";
    const storedPassword = "1234";

    if (username === storedUsername && password === storedPassword) {
        return "Login successful!";
    } else {
        return "Invalid credentials.";
    }
}
console.log(validateLogin("admin", "1234")); // Output: Login successful!
```

Cart System

A function to calculate the total price of items in a shopping cart.

```
javascript
Copy code
function calculateTotal(cartItems) {
    let total = 0;

    for (let item of cartItems) {
        total += item.price;
    }

    return total;
}
const cart = [{ price: 100 }, { price: 200 }, { price: 50 }];console.log("Total:", calculateTotal(cart)); //
Output: Total: 350
```

Classroom Exercises:
Create a Function to Calculate Square of a Number
Demonstrate Scope Behavior
Function to Find Even or Odd

Summary
Functions allow reusable and modular code.
Scope controls where variables are accessible, reducing bugs and conflicts.
By mastering these concepts, students can start writing structured, clean, and scalable code.
This structure ensures students understand both concepts and can apply them effectively in real-life scenarios.