Day 3: Functions, Scope, and Hoisting in JavaScript

1. Functions in JavaScript

A **function** is a reusable block of code that performs a specific task. Functions help us avoid repeating code and make programs modular and easier to maintain.

Types of Functions

1. **Function Declaration**

```
function add(a, b) {
    return a + b;
}

console.log(add(5, 7)); // 12
```

- Declared using the `function` keyword.
- Can be called **before** or after its declaration (due to hoisting, which we'll see later).

- **Function Expression**

```
const multiply = function(a, b) {
    return a * b;
};

console.log(multiply(4, 5)); // 20
```

- Stored in a variable.
- **Cannot** be called before the line it is defined (not hoisted like declarations).

- **Arrow Functions** (ES6)

```
const subtract = (a, b) => a - b;

console.log(subtract(10, 3)); // 7
```

- Shorter syntax.
- `this` behaves differently (important in advanced JavaScript).

---

2. Parameters and Return Values

- **Parameters**: Values you pass into a function.
- **Return Value**: Value that a function gives back.

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

console.log(greet("Umesh")); // Hello Umesh
```

Tips:

- Functions can have **default parameters**:

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

---

3. Scope: Global vs Local

Scope determines where variables are accessible.

Global Scope

- Variable declared **outside** a function.
- Accessible anywhere in the code.

```
let globalVar = "I am global";

function showVar() {
    console.log(globalVar);
}

showVar(); // I am global
console.log(globalVar); // I am global
```

Local Scope

- Variable declared **inside** a function.
- Only accessible **inside** that function.

```
function localExample() {
    let localVar = "I am local";
    console.log(localVar); // I am local
}

localExample();
console.log(localVar); // Error: localVar is not defined
```

Block Scope

- Variables declared with `let` or `const` inside `{}` (like loops or conditionals) are block-scoped.

```
if(true){
    let blockVar = "Block";
    console.log(blockVar); // Block
}
console.log(blockVar); // Error
```

---

4. Hoisting Basics

Hoisting: JavaScript moves function declarations and variable declarations to the top of the scope before code execution.

Function Hoisting

```
console.log(sum(2, 3)); // 5

function sum(a, b){
    return a + b;
}
```

- Works because **function declarations are hoisted**.

Variable Hoisting

```
console.log(x); // undefined
var x = 10;

console.log(y); // Error
let y = 20;
```

- `var` is hoisted (but initialized as `undefined`), `let` and `const` are not hoisted.

5. Tasks / Practice

1. **Add Two Numbers Function**

```
function add(a, b){
    return a + b;
}
console.log(add(5, 7)); // 12
```

1. **Test Variable Access**

```
let globalVar = "I am global";

function testScope(){
    let localVar = "I am local";
    console.log(globalVar); // I am global
    console.log(localVar); // I am local
}

testScope();
console.log(globalVar); // I am global
// console.log(localVar); // Error
```

1. **Observe Hoisting**

```
console.log(hoistVar); // undefined
var hoistVar = "Hoisted";

// console.log(noHoist); // Error
// let noHoist = "Cannot access";
```

6. Real-World Applications of Functions

1. **Web Development**
2. Handling form inputs, validations, and user interactions.
3. Example: `function validateEmail(email){...}`
4. **AI / Machine Learning**
5. Functions are used in:
    - Preprocessing data
    - Performing mathematical operations
    - Calling APIs for AI models

```
function normalizeData(data){
    return data.map(x => x/255);
}
```

6. In AI pipelines (like TensorFlow.js), functions manage training, predictions, and evaluation steps.
7. **Games**
8. Functions handle player actions, score calculations, and rendering elements.
9. **Automation**
10. Scripts/functions automate repetitive tasks (e.g., batch image resizing, scraping websites, or sending notifications).

---

7. Mini Exercise (Advanced)

Write a function that calculates BMI

```
function calculateBMI(weight, height){
    return weight / (height * height);
}

console.log(calculateBMI(70, 1.75)); // 22.86
```

Use Case: In AI fitness apps, similar functions are used to normalize data before feeding it to models.

Key Takeaways

- **Functions** make your code modular and reusable.
- **Parameters & return values** allow functions to accept inputs and give outputs.
- **Scope** controls accessibility of variables.
- **Hoisting** allows functions and variables to behave differently depending on their declaration type.
- Mastering these concepts is essential for **JavaScript web apps, AI scripts, and data preprocessing pipelines**.