1. Introduction to JavaScript

JavaScript is a **lightweight**, **interpreted** language used to make web pages **interactive**.

- It is **client-side**, meaning it runs in the browser.
- It can **modify** HTML, CSS, and handle user interactions.
- It does **not** need compilation (unlike Java, C++).

---

2. Writing JavaScript Code

JavaScript code can be written in:

1. **Inside HTML <script> tag**<script>
       console.log("Hello, JavaScript!");
   </script>

2. **External file (.js)**<script src="script.js"></script>

---

3. Output in JavaScript

JavaScript provides multiple ways to show output:

- console.log("Hello") → Prints in the browser console
- document.write("Hello") → Writes directly to the webpage
- alert("Hello") → Displays a pop-up alert
- prompt("Enter Name") → Takes input from the user

---

4. JavaScript Variables

Variables store data. Three ways to declare variables:

- var (old, function-scoped)
- let (modern, block-scoped)
- const (constant, cannot be reassigned)

Example:
let name = "Alice";
const PI = 3.14;

---

5. JavaScript Operators

A. Arithmetic Operators

Used for basic math operations.

```
let sum = 5 + 3;   // 8
let product = 5 * 3; // 15
```

B. Comparison Operators

Used for **checking conditions**.

```
console.log(5 > 3); // true
console.log(5 == "5"); // true (loose equality)
console.log(5 === "5"); // false (strict equality)
```

C. Logical Operators

Used for combining conditions.

```
console.log(true && false); // false
console.log(true || false); // true
console.log(!true); // false
```

---

6. Conditional Statements

JavaScript uses if, if-else, switch for decision-making.

A. if-else Statement

```
let marks = 85;
if (marks > 90) {
   console.log("Grade A+");
} else if (marks > 80) {
   console.log("Grade A");
} else {
   console.log("Grade B");
}
```

B. switch Statement

```
let day = 3;
switch (day) {
   case 1: console.log("Monday"); break;
   case 2: console.log("Tuesday"); break;
   default: console.log("Other day");
}
```

---

7. JavaScript Arrays

Arrays store multiple values in a **single** variable.

Creating Arrays

```
let fruits = ["Apple", "Banana", "Cherry"];
```

Accessing Elements

```
console.log(fruits[0]); // Apple
```

Array Methods

```
fruits.push("Mango");  // Add at end
fruits.pop();          // Remove last element
fruits.sort();         // Sort alphabetically
```

---

8. JavaScript Loops

Loops help execute code **multiple times**.

A. for Loop

```
for (let i = 0; i < 5; i++) {
   console.log(i);
}
```

B. while Loop

```
let i = 0;
while (i < 5) {
   console.log(i);
   i++;
}
```

---

9. JavaScript Strings

Strings store text data.

```
let message = "Hello, World!";
```

String Methods

```
console.log(message.length); // 13
console.log(message.toUpperCase()); // "HELLO, WORLD!"
console.log(message.includes("Hello")); // true
console.log(message.replace("World", "JavaScript")); // "Hello, JavaScript!"
```

---

10. Functions in JavaScript

Functions **reusable blocks** of code.

A. Function Declaration

```
function greet(name) {
    return `Hello, ${name}!`;
}
console.log(greet("Alice"));
```

B. Arrow Functions (ES6)

```
const add = (a, b) => a + b;
console.log(add(3, 5)); // 8
```

---

11. JavaScript Objects

Objects store **key-value pairs**.

A. Creating Objects

```
let person = {
    name: "Alice",
    age: 25,
    greet: function() {
        console.log("Hello!");
    }
};
```

B. Accessing Object Properties

```
console.log(person.name); // Alice
person.greet(); // Hello!
```

---

12. JavaScript Classes (ES6)

Classes help create **multiple objects** with similar properties.

```
class Employee {
    constructor(name, salary) {
        this.name = name;
        this.salary = salary;
    }
    getSalary() {
        return this.salary;
    }
}
let emp1 = new Employee("Alice", 50000);
console.log(emp1.getSalary()); // 50000
```

---

13. JavaScript Popups

A. Alert

```
alert("This is an alert box!");
```

B. Confirm

```
let result = confirm("Do you agree?");
console.log(result); // true or false
```

C. Prompt

```
let name = prompt("Enter your name:");
console.log(name);
```

HTML DOM (Document Object Model)

The **DOM** represents the structure of a webpage as a **tree of objects**.
It allows JavaScript to **interact with** and **modify** elements on a webpage.
🔹 **Example DOM Structure:**

```
<html>
  <body>
    <p id="para">Hello World</p>
  </body>
</html>
```

🔹 **JavaScript to Access DOM Elements:**

```
let para = document.getElementById("para"); // Selects the paragraph
console.log(para.innerText); // Prints: Hello World
```

---

2. JavaScript Event Handling

Events **detect user actions** like clicks, key presses, etc.
🔹 **Types of Events:**

- **Mouse Events**: click, mouseover, dblclick
- **Keyboard Events**: keydown, keyup, keypress
- **Form Events**: submit, change, focus
- **Window Events**: load, resize, scroll

A. Handling Events - 3 Ways

1️⃣ **Inline Event Handling (Not Recommended)**

```
<button onclick="alert('Button Clicked!')">Click Me</button>
```

2️⃣ **Using DOM Property**

```
document.getElementById("btn").onclick = function () {
    alert("Button Clicked!");
};
```

3️⃣ **Using addEventListener() (Best Practice)**

```
document.getElementById("btn").addEventListener("click", function () {
    alert("Button Clicked!");
});
```

---

3. JavaScript innerText vs innerHTML

🔹 **innerText** → Sets/gets only **text content**
🔹 **innerHTML** → Sets/gets **HTML content**

```
document.getElementById("para").innerText = "New Text"; // Changes text
document.getElementById("para").innerHTML = "<b>Bold Text</b>"; // Inserts HTML
```

---

4. Changing HTML Attributes Using JavaScript

JavaScript can modify **attributes like src, href, alt**, etc.

```
<img id="image" src="old.jpg" width="100">
<button onclick="changeImage()">Change Image</button>
<script>
  function changeImage() {
    let img = document.getElementById("image");
    img.src = "new.jpg";  // Changes image
    img.setAttribute("width", "200");  // Changes width
  }
</script>
```

---

5. JavaScript eval() Function

eval() **executes a string as JavaScript code**.

```
let a = 10, b = 20;
let sum = eval("a + b"); // Evaluates the expression "10 + 20"
console.log(sum); // 30
```

🚨 **Caution**: eval() is a security risk and should be avoided.

---

6. Profit & Loss Formula

💰 **Profit / Loss = Selling Price (SP) - Cost Price (CP)**
💹 **Percentage Formula:**

- **Profit %** = (Profit / CP) × 100
- **Loss %** = (Loss / CP) × 100

🔹 **JavaScript Code to Calculate Profit & Loss**

```
function calculateProfitLoss(cp, sp) {
    let result;
    if (sp > cp) {
        let profit = sp - cp;
        let profitPercent = (profit / cp) * 100;
        result = `Profit: ₹${profit} (${profitPercent.toFixed(2)}%)`;
    } else if (sp < cp) {
        let loss = cp - sp;
        let lossPercent = (loss / cp) * 100;
        result = `Loss: ₹${loss} (${lossPercent.toFixed(2)}%)`;
    } else {
        result = "No Profit, No Loss";
    }
    return result;
}
console.log(calculateProfitLoss(1000, 1200)); // Output: Profit: ₹200 (20%)
```

1. Form Validation in JavaScript

Form validation ensures **correct and valid data** before submission.
It helps in:
✅ Preventing incorrect user inputs
✅ Reducing errors before data reaches the server
✅ Enhancing user experience

Types of Form Validation

🔹 **Client-Side Validation** (Using JavaScript) - Fast, but can be bypassed
🔹 **Server-Side Validation** (Using Backend Code) - More secure

---

2. addEventListener() Method

Used to attach an event (like click, submit) to an element.
🔹 **Syntax:**
javascript
CopyEdit
```
element.addEventListener(event, function, useCapture);
```

- **event**: The event type (e.g., "click", "submit")
- **function**: Function to execute when the event occurs
- **useCapture**: (Optional) true for capturing phase, false for bubbling

🔹 **Example: Click Event**
javascript
CopyEdit
```
document.getElementById("btn").addEventListener("click", function() {
    alert("Button clicked!");
});
```

---

3. Event Object (event)

When an event occurs, JavaScript automatically passes an **event object** with useful details.
🔹 **Example: Accessing Event Properties**
javascript
CopyEdit
```
document.getElementById("btn").addEventListener("click", function(event) {
    console.log("Event Type:", event.type);  // Output: click
    console.log("Clicked Element:", event.target); // Output: <button> element
});
```

Key Properties of event:

- type → Type of event (click, keypress, etc.)
- target → The element that triggered the event
- key → Key pressed (for keyboard events)
- clientX / clientY → Mouse coordinates

---

4. Event Propagation (Bubbling vs Capturing)

When an event occurs in nested elements, it follows a **two-phase flow**:
1️⃣ **Capturing Phase (Trickling Down)** → Starts from <html> and moves down to the target
2️⃣ **Bubbling Phase (Bubbling Up)** → Starts from the target and moves up to <html>
🔹 **Example: Bubbling (Default Behavior)**
javascript
CopyEdit

```
document.getElementById("parent").addEventListener("click", function() {
  alert("Parent clicked!");
});

document.getElementById("child").addEventListener("click", function() {
  alert("Child clicked!");
});
```

👉 Clicking on "Child" will trigger both alerts (child first, then parent).
🔹 **Stopping Bubbling (event.stopPropagation())**
javascript
CopyEdit

```
document.getElementById("child").addEventListener("click", function(event) {
  event.stopPropagation(); // Prevents bubbling
  alert("Only child clicked!");
});
```

---

5. Prevent Default Behavior (event.preventDefault())

Some elements (e.g., links, forms) have default behaviors. We can prevent them.
🔹 **Example: Preventing Link Navigation**
javascript
CopyEdit

```
document.querySelector("a").addEventListener("click", function(event) {
  event.preventDefault(); // Stops navigation
  alert("Link clicked, but not opening the page!");
});
```

---

6. Regular Expressions (Regex) for Validation

Regular expressions (Regex) help check **patterns** in input values.

A. Name Validation (Only Letters & Spaces)

javascript
CopyEdit
```
let namePattern = /^[A-Za-z\s]{3,}$/;
console.log(namePattern.test("John Doe")); // ✅ true
console.log(namePattern.test("J0hn123")); // ❌ false
```

B. Mobile Number Validation (Starts with 6-9, Exactly 10 Digits)

javascript
CopyEdit
```
let mobilePattern = /^[6-9]\d{9}$/;
console.log(mobilePattern.test("9876543210")); // ✅ true
console.log(mobilePattern.test("5123456789")); // ❌ false
```

C. Email Validation (Contains @ and .)

javascript
CopyEdit
```
let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
console.log(emailPattern.test("test@example.com")); // ✅ true
console.log(emailPattern.test("test.example.com")); // ❌ false
```

---

7. Implementing Form Validation in JavaScript

1️⃣ Select Form Elements

javascript
CopyEdit
```
let form = document.getElementById("regForm");
let nameInput = document.getElementById("name");
let mobileInput = document.getElementById("mobile");
let emailInput = document.getElementById("email");
```

2️⃣ Add Event Listener for Form Submission

javascript
CopyEdit
```
form.addEventListener("submit", function(event) {
    event.preventDefault(); // Prevents form submission if there are errors
});
```

3️⃣ Validate Name Input

javascript
CopyEdit
```
let namePattern = /^[A-Za-z\s]{3,}$/;
if (!namePattern.test(nameInput.value)) {
    alert("Invalid Name! Must contain only letters and spaces.");
}
```

4️⃣ Validate Mobile Number

javascript
CopyEdit
```
let mobilePattern = /^[6-9]\d{9}$/;
if (!mobilePattern.test(mobileInput.value)) {
    alert("Invalid Mobile Number! Must be 10 digits and start with 6-9.");
}
```

5️⃣ Validate Email Address

javascript
CopyEdit
```
let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailPattern.test(emailInput.value)) {
    alert("Invalid Email Address!");
}
```

6️⃣ Show Success Message

javascript
CopyEdit
```
if (namePattern.test(nameInput.value) && mobilePattern.test(mobileInput.value) &&
emailPattern.test(emailInput.value)) {
    alert("Registration Successful!");
    form.reset(); // Clears the form
}
```

Destructuring in JavaScript

Destructuring is a feature in JavaScript that allows **extracting values** from arrays or objects into variables. It makes code more readable and reduces the need for **index-based** extractions.

A. Array Destructuring

Extract values from an array using square brackets [].

🔹 **Basic Example**

javascript
CopyEdit
```
let gadgets = ["Mobile", "Laptop", "Printer"];

let [first, second, third] = gadgets;

console.log(first);  // Output: Mobile
console.log(second); // Output: Laptop
console.log(third);  // Output: Printer
```

🔹 **Skipping Elements**

javascript
CopyEdit
```
let gadgets = ["Mobile", "Laptop", "Printer"];

let [first, , third] = gadgets; // Skipping "Laptop"

console.log(first);  // Mobile
console.log(third);  // Printer
```
🔹 **Extract Only Some Elements**

javascript
CopyEdit
```
let gadgets = ["Mobile", "Laptop", "Printer"];

let [first] = gadgets; // Gets only the first element

console.log(first);  // Mobile
```

B. Object Destructuring

Extract properties from objects using curly braces {}.

🔹 **Basic Example**

javascript
CopyEdit
```
let employee = {
   name: "Raman Verma",
   job: "Software Developer",
   salary: 80000
};

const { name, job, salary } = employee;

console.log(name);  // Output: Raman Verma
console.log(job);   // Output: Software Developer
console.log(salary);// Output: 80000
```
🔹 **Using Default Values**

javascript
CopyEdit
```
let employee = { name: "Raman Verma", job: "Software Developer" };

const { name, job, salary = 50000 } = employee;  // Default salary if not provided

console.log(salary); // Output: 50000
```

---

2. Spread Operator (...)

The **spread operator (...)** allows expanding elements of an **array or object** into individual elements.

A. Copying an Array

```
javascript
CopyEdit
let arr = [12, 34, 56];
let copiedArr = [...arr]; // Creates a new array copy

console.log(copiedArr); // Output: [12, 34, 56]
```

B. Merging Two Arrays

```
javascript
CopyEdit
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];

let mergedArray = [...arr1, ...arr2];

console.log(mergedArray); // Output: [1, 2, 3, 4, 5, 6]
```

C. Copying an Object

```
javascript
CopyEdit
let employee1 = { name: "Raman", job: "Developer" };

let employee2 = { ...employee1 }; // Creates a copy

console.log(employee2); // Output: { name: "Raman", job: "Developer" }
```

D. Merging Objects

```
javascript
CopyEdit
let obj1 = { name: "Raman" };
let obj2 = { job: "Software Developer" };
let obj3 = { salary: 80000 };

let mergedObject = { ...obj1, ...obj2, ...obj3 };

console.log(mergedObject);
// Output: { name: "Raman", job: "Software Developer", salary: 80000 }
```

E. Spread in Function Calls

```
javascript
CopyEdit
let numbers = [12, 34, 26];

let sum = (a, b, c) => a + b + c;
```

```
console.log(sum(...numbers)); // Output: 72
```

---

3. Rest Operator (...)

The **rest operator (...)** collects multiple values into an array. It is the **opposite of the spread operator**.

A. Rest in Array Destructuring

javascript
CopyEdit
```
let [first, ...rest] = ["Mobile", "Laptop", "Printer"];

console.log(first); // Output: Mobile
console.log(rest);  // Output: ["Laptop", "Printer"]
```

B. Rest in Object Destructuring

javascript
CopyEdit
```
let employee = { name: "Raman", job: "Developer", salary: 80000 };

let { name, ...empInfo } = employee;

console.log(name);    // Output: Raman
console.log(empInfo); // Output: { job: "Developer", salary: 80000 }
```

C. Rest in Function Parameters

The **rest operator** is useful when handling multiple function arguments.
javascript
CopyEdit
```
function addNumbers(...numbers) {
   return numbers.reduce((sum, num) => sum + num, 0);
}

console.log(addNumbers(1, 2, 3, 4, 5)); // Output: 15
```

Arrow Functions (=>) in JavaScript

Arrow functions are a **concise way** to write functions using the => (arrow) syntax.
🔹 Introduced in **ES6**
🔹 Shorter than regular functions
🔹 No need for {} and return in single-line expressions

A. Basic Arrow Function Syntax

javascript
CopyEdit

```
const add = (a, b) => a + b;

console.log(add(3, 5)); // Output: 8
```

- **No need for function keyword**
- **No need for {} and return if it's a single expression**

B. Arrow Function with One Parameter

If the function has **only one parameter**, parentheses () are **optional**.

javascript
CopyEdit

```
const square = num => num * num;

console.log(square(5)); // Output: 25
```

C. Arrow Function with Multiple Parameters

If there are **two or more parameters**, parentheses **must** be used.

javascript
CopyEdit

```
const sum = (x, y) => x + y;

console.log(sum(4, 6)); // Output: 10
```

D. Arrow Function with Multiple Statements

If there are **multiple statements**, use {} and return.

javascript
CopyEdit

```
const subtract = (x, y) => {
   let result = x - y;
   return result;
};

console.log(subtract(6, 3)); // Output: 3
```

E. Arrow Function with No Parameters

If there are **no parameters**, use empty parentheses ().

javascript
CopyEdit

```
const message = () => "Hello World!";

console.log(message()); // Output: Hello World!
```

F. Arrow Functions in Array Methods

Arrow functions are commonly used in **array methods** like map(), filter(), and reduce().
🔹 **Using map()** → Creates a new array by modifying each element
javascript
CopyEdit
```
let numbers = [1, 2, 3, 4, 5];
let cubes = numbers.map(num => num * num * num);

console.log(cubes); // Output: [1, 8, 27, 64, 125]
```
🔹 **Using filter()** → Returns a new array with elements that satisfy a condition
javascript
CopyEdit
```
let ages = [23, 78, 45, 88];
let seniors = ages.filter(age => age >= 70);

console.log(seniors); // Output: [78, 88]
```
🔹 **Using reduce()** → Accumulates array values into a single value
javascript
CopyEdit
```
let nums = [10, 20, 30, 40];
let total = nums.reduce((acc, num) => acc + num, 0);

console.log(total); // Output: 100
```

---

2. Template Literals (Template Strings) in JavaScript

Template literals (also called **template strings**) are **modern ways** to create strings.
🔹 Enclosed using **backticks** (`) instead of quotes (' or ")
🔹 Allows **multi-line strings**
🔹 Supports **string interpolation** (${})

A. Multi-Line Strings with Template Literals

javascript
CopyEdit
```
let message = `Good Morning
Have a Nice Day
Good Bye!!`;

console.log(message);
```
✅ No need for \n → **Preserves line breaks naturally**

B. String Interpolation (${})

🔹 **Insert variables inside a string**
javascript
CopyEdit

let name = "Raman Verma";
let job = "Software Developer";
let salary = 80000;

let empInfo = `Hi, My name is ${name} and I work as a ${job} with a salary of Rs. ${salary}`;

console.log(empInfo);
✅ **No need for + for string concatenation**

C. Embedding Expressions in Template Literals

You can **embed JavaScript expressions** inside template literals.
javascript
CopyEdit
let x = 10;
let y = 20;
let sum = `The sum of ${x} and ${y} is ${x + y}`;

console.log(sum);
✅ **Calculations inside ${}**

D. Using Template Literals in Functions

javascript
CopyEdit
```
let employee = {
   name: "Raman Verma",
   job: "Software Developer",
   salary: 80000
};

let employeeInfo = employee => {
   return `Hi, My name is ${employee.name} and I work as a ${employee.job} with a salary of
Rs. ${employee.salary}`;
};

console.log(employeeInfo(employee));
```

E. Creating HTML Templates with Template Literals

Template literals can **generate dynamic HTML templates**.
🔹 **Basic HTML Template Example**
javascript
CopyEdit
let greet = "Have a Nice Day";

let htmlTemplate = `

```
<div>
   <p>${greet}</p>
</div>`;

console.log(htmlTemplate);
```

🔹 **E-Commerce Example: Product Pricing**

javascript

CopyEdit

```
let rate = 1000;
let quantity = 20;

let htmlTemplate = `
<div>
   <span>The total price of the product is Rs. ${rate * quantity}</span>
</div>`;

console.log(htmlTemplate);
```

DNR NIGGER

@IOS