Thursday, July 3, 2025 11:13 PM

What is JavaScript?

JavaScript (JS) is the **programming language of the web**. It allows us to build dynamic, interactive experiences by manipulating content, responding to user input, fetching data from servers, and much more.

Core Role: HTML is structure, CSS is style, and JavaScript is behavior.

Why JavaScript is critical for React?

React is built with and on JavaScript — **JSX** is JavaScript with HTML-like syntax. To master React, you must master JavaScript fundamentals.

Variables, Data Types, and Operators

Declaring Variables

```
var name = "John";
let age = 25;
const country = "India";
```

- var is **function-scoped**.
- let and const are **block-scoped** (preferred for modern JavaScript).
- const cannot be reassigned (it's **constant**).

Primitive Data Types

```
let firstName = "Alice";      // String
let age = 30;                 // Number
let isOnline = true;          // Boolean
let score = null;             // Null (intentional absence of any object value)
let user;                     // Undefined (variable declared but not assigned a value)
let symbol = Symbol("id");    // Symbol (advanced, unique identifier)
let bigInt = 1234567890123456789012345678901234567890n; // BigInt (for very large integers)
```

Operators

- **Arithmetic**: +, -, *, /, % (remainder), ** (exponentiation)
- **Assignment**: =, +=, -=, *=, /=, %=, **=
- **Comparison**: == (loose equality), === (strict equality), != (loose inequality), !== (strict inequality), <, >, <=, >=
- **Logical**: && (AND), || (OR), ! (NOT)

Demo: Age Checker

```
Let age = 17;
if(age >= 18) {
```

```
  console.log("Adult");
} else{
  console.log("Minor");
}
```

Strings, Numbers, Arrays, and Objects

Strings

```
let greeting = "Hello";
console.log(greeting.length);       // Length of the string
console.log(greeting.toUpperCase()); // Convert to uppercase
console.log(greeting.substring(0, 3)); // Extract a part of the string
```

Numbers

```
let a = 10.5;
let b = parseInt("20");  // Converts string to integer
let c = parseFloat("20.75"); // Converts string to floating-point number
console.log(a + b);
```

Arrays

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.push("Orange");      // Adds to the end
console.log(fruits[1]);     // Access by index
```

Useful methods: push, pop, shift, unshift, slice, splice, includes, indexOf, join, forEach, map, filter, reduce (brief mention, detailed later).

Objects

```
let student = {
  name: "Farhan",
  marks: 90,
  pass: true
};
console.log(student.name);       // Dot notation
console.log(student["marks"]);   // Bracket notation (useful for dynamic keys)
```

Demo: User Profile

```
let user = {
  name: "Sara",
  age: 24,
  greet: function() {
    return `Hi, I'm ${this.name}`; // `this` refers to the current object
  }
};
console.log(user.greet());
```

Control Flow and Loops

if / else if / else

```
let score = 72;
if (score >= 90) {
  console.log("Excellent");
} else if (score >= 60) {
  console.log("Good");
} else {
  console.log("Fail");
}
```

switch Statement

```
let role = "admin";
switch (role) {
  case "admin":
    console.log("Full access");
    break; // Important to exit the switch block
  case "guest":
    console.log("Read only");
    break;
  default:
    console.log("No access");
}
```

Loops

- **for loop**: For a known number of iterations.

```
for(let i = 1; i <= 5; i++) {
  console.log(i);
}
```

- **while loop**: Continues as long as a condition is true.
  JavaScript

```
let i = 0;
while(i < 3) {
  console.log(i);
  i++;
}
```

- **for...of loop**: Iterating over iterable objects like arrays, strings, etc.
  JavaScript

```
const arr = ["A", "B", "C"];
for(let item of arr) {
  console.log(item);
}
```

- **for...in loop**: Iterating over object properties (less common for arrays).
  JavaScript

```
const myObject = { a: 1, b: 2};
for(let key in myObject) {
  console.log(`${key}: ${myObject[key]}`);
}
```

Demo: Sum of Numbers

```
let sum = 0;
for (let i = 1; i <= 10; i++) {
 sum += i;
}
console.log("Total:", sum);
```

Functions & Scope

Function Declarations & Expressions

- **Function Declaration**: Hoisted, can be called before defined.

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

- **Function Expression**: Not hoisted, treated like a variable assignment.
JavaScript

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

Arrow Functions (ES6+)

- Shorter syntax, especially for simple functions.
- Lexical this binding (important in React components).

```
const square = n => n * n; // Implicit return for single expression
console.log(square(5));

const greet = (name) => { // Explicit return for multiple statements
 console.log("Hello,");
 return `Hi, ${name}`;
};
```

Scope & Hoisting

- **Scope**: Determines the accessibility of variables.
  - **Global Scope**: Accessible everywhere.
  - **Function Scope**: Variables declared with var are function-scoped.
  - **Block Scope**: Variables declared with let and const are block-scoped (within {}).

- **Hoisting**: JavaScript moves declarations to the top of their scope during compilation.

```
function outer() {
 let outerVar = "outside"; // outerVar is block-scoped to outer()
 function inner() {
  console.log(outerVar); // inner can access outerVar (closure)
 }
```

```
  inner();
}
outer();

// Example of hoisting difference
console.log(hoistedVar); // Undefined, but no error (var is hoisted)
var hoistedVar = "I am hoisted";

// console.log(notHoistedLet); // ReferenceError (let/const are not initialized until definition)
// let notHoistedLet = "I am not truly hoisted";
```

Arrays, Higher-Order Functions & Callbacks

map, filter, reduce (Higher-Order Array Methods)

These methods are fundamental for functional programming patterns in JavaScript and are heavily used in React.

```
let nums = [1, 2, 3, 4];

let doubled = nums.map(n => n * 2);      // Creates a new array with transformed elements
console.log(doubled); // [2, 4, 6, 8]

let even = nums.filter(n => n % 2 === 0); // Creates a new array with elements that pass a test
console.log(even);    // [2, 4]

let total = nums.reduce((acc, val) => acc + val, 0); // Reduces array to a single value
console.log(total);   // 10
```

Callback Functions

A function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```
function greetUser(name, callback) {
  callback(`Hello, ${name}`);
}

greetUser("Sam", message => console.log(message)); // message => console.log(message) is the callback
```

Demo: Student Marks Filter

```
let students = [
  { name: "Ali", marks: 45 },
  { name: "Reema", marks: 75 },
  { name: "Farhan", marks: 80 }
];

let passed = students.filter(s => s.marks > 50); // Filter out students who passed
console.log(passed);
```

Output:

```
[
  { name: 'Reema', marks: 75 },
  { name: 'Farhan', marks: 80 }
]
```

DOM Manipulation & Events

Accessing Elements

The **Document Object Model (DOM)** is a programming interface for web documents. It represents the page structure as a tree of objects.

```
<h1 id="title">Welcome</h1>
<button class="my-button">Click Me</button>
<script>
  function changeText() {
    document.getElementById("title").innerText = "Changed! again"; // Get element by ID and change its text
  }

  // Common methods:
  document.getElementById("title");        // Get element by its ID
  document.getElementsByClassName("my-button"); // Get elements by class name (returns HTMLCollection)
  document.getElementsByTagName("p");      // Get elements by tag name (returns HTMLCollection)
  document.querySelector("#title");        // Get first element matching CSS selector
  document.querySelectorAll(".my-button");  // Get all elements matching CSS selector (returns NodeList)
</script>
```

Event Listeners

Allow JavaScript to react to user actions (clicks, key presses, form submissions, etc.) or browser events.

```
const btn = document.querySelector(".my-button"); // Select the button
btn.addEventListener("click", () => { // Add an event listener for 'click'
  alert("Button Clicked!");
});

// Another example: Mouseover event
const titleElement = document.getElementById("title");
titleElement.addEventListener("mouseover", () => {
  titleElement.style.backgroundColor = "yellow";
});
titleElement.addEventListener("mouseout", () => {
  titleElement.style.backgroundColor = ""; // Reset
});
```

Changing Styles

Directly manipulate the CSS properties of elements.

```
document.getElementById("title").style.color = "blue"; // Change text color
document.body.style.backgroundColor = "#f0f0f0"; // Change body background color
```

```
// Adding/removing CSS classes is often preferred for managing styles
document.getElementById("title").classList.add("highlight");
document.getElementById("title").classList.remove("highlight");
document.getElementById("title").classList.toggle("active"); // Toggles class
```

ES6+ Features, Template Literals, Spread, Destructuring

Destructuring Assignment (ES6+)

A convenient way to extract values from arrays or properties from objects into distinct variables.

```
// Object Destructuring
const user = { name: "Ali", age: 30, city: "Delhi" };
const { name, age } = user; // Extracts 'name' and 'age' properties
console.log(name); // Ali
console.log(age);  // 30

// Array Destructuring
const colors = ["red", "green", "blue"];
const [firstColor, secondColor] = colors;
console.log(firstColor);  // red
console.log(secondColor); // green
```

Spread & Rest Operators (ES6+)

- **Spread (...)**: Expands an iterable (like an array or string) into individual elements.

```
// Spreading arrays
let nums = [1, 2, 3];
let newNums = [...nums, 4, 5]; // Creates a new array by spreading 'nums'
console.log(newNums); // [1, 2, 3, 4, 5]

// Spreading objects (for merging or copying)
const person = { name: "John", age: 28 };
const updatedPerson = { ...person, age: 29, city: "New York" };
console.log(updatedPerson); // { name: 'John', age: 29, city: 'New York' }
```

- **Rest (...)**: Gathers an indefinite number of arguments into an array.

```
function sum(...args) { // 'args' becomes an array of all passed arguments
  return args.reduce((a, b) => a + b, 0);
}
console.log(sum(1, 2, 3, 4)); // 10
console.log(sum(5, 10));     // 15
```

Template Literals (ES6+)

Allow for embedded expressions and multi-line strings using backticks (`).

```
let product = "Laptop";
let price = 50000;
```

```
console.log(`The ${product} costs ₹${price}.`); // Embed variables directly
console.log(`
  This is a multi-line
  string example.
`);
```

Ternary Operator
A shorthand for an if...else statement.

```
let age = 20;
let result = age >= 18 ? "Adult" : "Minor"; // condition ? valueIfTrue : valueIfFalse
console.log(result); // Adult
```

Final Demo: Todo List (Basic)

This demo ties together DOM manipulation, event handling, and basic data handling.

```
<input type="text" id="task" placeholder="Add a new task...">
<button onclick="addTask()">Add Task</button>
<ul id="list"></ul>

<script>
function addTask() {
  let taskInput = document.getElementById("task");
  let taskText = taskInput.value.trim(); // Get value and remove leading/trailing whitespace

  if (taskText === "") { // Prevent adding empty tasks
    alert("Please enter a task!");
    return;
  }

  let li = document.createElement("li"); // Create a new list item
  li.innerText = taskText;              // Set its text

  // Optional: Add a delete button to each task
  let deleteBtn = document.createElement("button");
  deleteBtn.innerText = "Delete";
  deleteBtn.style.marginLeft = "10px";
  deleteBtn.onclick = function() {
    li.remove(); // Remove the parent list item
  };
  li.appendChild(deleteBtn);

  document.getElementById("list").appendChild(li); // Add the new list item to the UL
  taskInput.value = ""; // Clear the input field
}
</script>
```

Summary Checklist

- Variables, data types, and operators
- Control flow (if/else, switch) and loops (for, while, for...of, for...in)

- Strings, numbers, arrays, and objects
- Functions (declarations, expressions, arrow functions) & Scope (global, function, block, hoisting)
- DOM manipulation and events (accessing elements, event listeners, changing styles)
- Higher-order array functions (map, filter, reduce) and callback functions
- ES6+ features (template literals, destructuring, spread/rest operators, ternary operator)
- 3 working demos (Age Checker, Student Marks Filter, Basic Todo App)