Unit IV: Client Side Scripting with JavaScript

4.1 Structure of a JavaScript Program

1️⃣ What is JavaScript?

JavaScript (JS) is a **client-side scripting language** primarily used for making web pages interactive. It is executed in the browser and can modify HTML and CSS dynamically.

2️⃣ Basic Structure of a JavaScript Program

A JavaScript program consists of the following main components:

1. **Statements** – Individual lines of code that perform actions.
2. **Variables** – Containers for storing data.
3. **Functions** – Blocks of reusable code.
4. **Operators** – Used for performing calculations or logical operations.
5. **Control Structures** – Loops and conditional statements for controlling program flow.
6. **Objects** – Used to store multiple values in a structured way.

3️⃣ Where to Write JavaScript?

JavaScript can be placed in three different locations in an HTML document:

1. Inline JavaScript (Inside an HTML Element)

JavaScript can be directly written inside an HTML element using the `onclick`, `onmouseover`, or other event attributes.

✅ **Example:**

```
<button onclick="alert('Hello, JavaScript!')">Click Me</button>
```

2. Internal JavaScript (Inside a `<script>` Tag)

JavaScript can be written inside a `<script>` tag within an HTML file.

✅ **Example:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Internal JS</title>
  <script>
    function showMessage() {
      alert("Welcome to JavaScript!");
    }
  </script>
</head>
<body>
  <button onclick="showMessage()">Click Me</button>
</body>
</html>
```

3. External JavaScript (Separate `.js` File)

JavaScript can be written in an external file and linked to an HTML file using the `<script>` tag.

✅ **Example:**

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>External JS</title>
  <script src="script.js"></script>
</head>
<body>
  <button onclick="showMessage()">Click Me</button>
</body>
</html>
```

script.js

```javascript
function showMessage() {
  alert("Hello from an external JavaScript file!");
}
```

✔️ **Why use External JavaScript?**

- Separates logic from HTML
- Improves readability and maintainability
- Can be reused across multiple pages

- Improves performance by allowing caching

4️⃣ JavaScript Syntax Rules

1. JavaScript is **case-sensitive** (`variable` and `Variable` are different).
2. Each statement ends with a **semicolon (;)** (optional but recommended).
3. **Comments** are written using:
   - `//` for single-line comments
   - `/* */` for multi-line comments

✅ **Example:**

```
// This is a single-line comment

/* This is a
   multi-line comment */
```

5️⃣ Simple JavaScript Program

✅ **Example:**

```
// Declare a variable
let name = "Alice";

// Function to display a message
function greetUser() {
    alert("Hello, " + name + "!");
}

// Call the function
greetUser();
```

📝 Recap Table

Concept	Description	Example
Inline JS	Written inside an HTML element	`<button onclick="alert('Hello!')">Click Me</button>`
Internal JS	Written inside a `<script>` tag in the HTML file	`<script> function greet() { alert('Hi!'); } </script>`
External JS	Written in a separate `.js` file and linked in HTML	`<script src="script.js"></script>`
Variables	Used to store values	`let x = 5;`
Functions	Reusable blocks of code	`function greet() { alert("Hello!"); }`
Comments	Used to describe code	`// This is a comment`

💡 Practice Problems

1️⃣ Create an HTML page with an **internal JavaScript script** that displays an alert box saying `"Hello, World!"` when the page loads.

2️⃣ Write an **external JavaScript file** that defines a function `showDate()`, which displays the current date in an alert box.

3️⃣ Modify an existing HTML page to add a **button** that calls a JavaScript function named `changeColor()` to change the background color of the page.

4.2 Variables and Data Types in JavaScript

1️⃣ What are Variables?

A **variable** is a container for storing data. It allows you to store, modify, and retrieve values in a JavaScript program.

In JavaScript, you can declare variables using:

- `var` (old way, has function scope)
- `let` (modern, has block scope)
- `const` (for values that don't change)

✅ **Example:**

```
var oldVar = "I am using var"; // Function-scoped
let newVar = "I am using let"; // Block-scoped
const constantVar = "I never change"; // Constant
```

2️⃣ JavaScript Data Types

JavaScript has two main types of data:

1. **Primitive Data Types** (store single values)
2. **Non-Primitive Data Types** (store collections or complex data)

1️⃣ Primitive Data Types

Data Type	Description	Example
String	Text inside quotes	`"Hello, World!"`
Number	Any number (integer or float)	`42`, `3.14`
Boolean	True or false values	`true`, `false`

Undefined	Variable declared but not assigned	`let x;` (value is `undefined`)
Null	Intentional empty value	`let x = null;`
Symbol	Unique identifiers (ES6 feature)	`Symbol('id')`
BigInt	Large integers beyond `Number.MAX_SAFE_INTEGER`	`BigInt(9007199254740991)`

✅ **Example:**

```
let name = "Alice";    // String
let age = 25;          // Number
let isStudent = true;  // Boolean
let score;             // Undefined
let empty = null;      // Null
let id = Symbol("id"); // Symbol
let bigNum = BigInt(9999999999999); // BigInt
```

2️⃣ Non-Primitive Data Types

Data Type	Description	Example
Object	A collection of key-value pairs	`{name: "Alice", age: 25}`
Array	Ordered collection of values	`[1, 2, 3, 4, 5]`
Function	A block of reusable code	`function greet() { alert("Hi!"); }`

✅ **Example:**

```
let person = { name: "Alice", age: 25 }; // Object
let numbers = [10, 20, 30, 40];          // Array
function greet() {                       // Function
    alert("Hello!");
}
```

3️⃣ Variable Declaration Methods

Keyword	Scope	Reassignable?	Hoisted?
`var`	Function scope	✅ Yes	✅ Yes (with `undefined`)
`let`	Block scope	✅ Yes	❌ No
`const`	Block scope	❌ No	❌ No

✅ **Example (Scope and Hoisting)**

```
function testVar() {
    if (true) {
        var x = 10; // Function scoped
    }
```

```
    console.log(x); // ✅ Accessible
}

function testLet() {
    if (true) {
        let y = 20; // Block scoped
    }
    console.log(y); // ❌ Error: y is not defined
}

testVar();
testLet();
```

📝 Recap Table

Concept	Description	Example
Primitive Types	Basic types (String, Number, Boolean, etc.)	`let age = 30;`
Non-Primitive Types	Complex types (Objects, Arrays, Functions)	`let obj = { name: "John" };`
var	Function-scoped variable	`var x = 10;`
let	Block-scoped variable	`let y = 20;`
const	Constant variable (cannot be changed)	`const z = 30;`

💡 Practice Problems

1️⃣ Declare a variable `studentName` and store `"John Doe"` in it. Print it to the console.

2️⃣ Create a variable `isWeekend` and set it to `true`. Display it using `alert()`.

3️⃣ Create an object called `car` with properties `brand`, `model`, and `year`.

4️⃣ Write a function that takes a number and returns whether it's even or odd.

4.3 JavaScript Statements: Expressions, Keywords, Blocks, and Operators

1️⃣ What are JavaScript Statements?

A **statement** is an instruction that JavaScript can execute. JavaScript programs consist of multiple statements, which are executed in sequence.

✅ **Example of JavaScript Statements:**

```
let x = 5;   // Statement 1: Variable declaration
let y = 10;  // Statement 2: Variable declaration
let sum = x + y;  // Statement 3: Expression
console.log(sum); // Statement 4: Function call
```

2️⃣ JavaScript Expressions

An **expression** is a combination of values, variables, and operators that evaluates to a value.

Expression Type	Description	Example
Arithmetic Expression	Uses mathematical operations	`5 + 3` , `x * 10`
String Expression	Concatenates strings	`"Hello" + "World"`
Logical Expression	Uses logical operators (`&&` , `	
Assignment Expression	Assigns values	`let sum = a + b;`

✅ **Example of Expressions:**

```
let result = (10 + 5) * 2;   // Arithmetic Expression
let greeting = "Hello, " + "World!";  // String Expression
let isAdult = (age >= 18) && (age < 60);  // Logical Expression
```

3️⃣ JavaScript Keywords

JavaScript has **reserved words** that have predefined meanings and cannot be used as variable names.

Keyword	Purpose	Example
`var` , `let` , `const`	Declare variables	`let age = 25;`
`if` , `else` , `switch`	Control flow	`if (x > 0) { ... }`
`for` , `while` , `do`	Loops	`for (let i=0; i<10; i++) { ... }`
`function` , `return`	Define functions	`function sum() { return x + y; }`
`try` , `catch` , `finally`	Error handling	`try { ... } catch (err) { ... }`

✅ **Example:**

```
if (x > 10) {
    console.log("X is greater than 10");
} else {
    console.log("X is 10 or less");
}
```

4️⃣ JavaScript Blocks

A **block** is a group of statements enclosed in `{}` and is used in functions, loops, and conditionals.

✅ **Example of Blocks in Functions and Loops:**

```
function greet() {
    let name = "Alice";
    console.log("Hello, " + name);
}  // Block for function

for (let i = 0; i < 5; i++) {
    console.log("Number: " + i);
}  // Block for loop
```

5️⃣ JavaScript Operators

Operators perform operations on values and variables.

Types of JavaScript Operators

Operator Type	Description	Example
Arithmetic	Perform basic math operations	`+ - * / % **`
Assignment	Assign values to variables	`= += -= *= /= %=`
Comparison	Compare values	`== === != !== > < >= <=`
Logical	Combine boolean expressions	`&&
Bitwise	Operate on binary numbers	`&
Ternary	Shorter if-else statement	`condition ? true : false`

✅ **Examples of Operators:**

```
let sum = 10 + 5;   // Arithmetic
let isEqual = (10 === 10);   // Comparison
let isTrue = (5 > 3) && (10 < 20);  // Logical
let status = (age >= 18) ? "Adult" : "Minor";  // Ternary
```

📝 Recap Table

Concept	Description	Example
Statements	Instructions JavaScript executes	`let x = 10;`
Expressions	Return a value	`x + y`
Keywords	Reserved words in JavaScript	`if`, `else`, `function`
Blocks	Code inside `{}`	`{ let x = 10; }`
Operators	Perform operations	`+, -, *, /, ==, &&,

💡 Practice Problems

1️⃣ Write a JavaScript expression that checks if a number is positive.

2️⃣ Use a ternary operator to check if a person can vote (age ≥ 18).

3️⃣ Create a function that returns `"Even"` if a number is even and `"Odd"` otherwise.

4.3 Flow Controls and Loops in JavaScript

1️⃣ What is Flow Control?

Flow control determines how statements in a program are executed based on conditions. JavaScript provides **conditional statements** (to make decisions) and **loops** (to execute code repeatedly).

2️⃣ Conditional Statements

Conditional statements help execute specific code only when certain conditions are met.

1. if Statement

Executes a block of code if the condition is `true`.

✅ **Example:**

```
let age = 18;
if (age >= 18) {
    console.log("You are eligible to vote.");
}
```

---

2. if-else Statement

Executes one block if the condition is `true`, and another if it's `false`.

✅ **Example:**

```
let number = 10;
if (number % 2 === 0) {
    console.log("Even Number");
} else {
    console.log("Odd Number");
}
```

---

3. if-else if-else Statement

Used when there are multiple conditions.

✅ **Example:**

```
let marks = 85;
if (marks >= 90) {
   console.log("Grade: A");
} else if (marks >= 75) {
   console.log("Grade: B");
} else {
   console.log("Grade: C");
}
```

---

4. Switch Statement

Used when there are multiple possible values for a variable.

✅ **Example:**

```
let day = "Monday";
switch (day) {
   case "Monday":
      console.log("Start of the week!");
      break;
   case "Friday":
      console.log("Weekend is near!");
      break;
   default:
      console.log("A normal day.");
}
```

✔️ **Why use `switch` instead of `if-else`?**

- Cleaner when dealing with multiple conditions
- Faster execution in some cases

3️⃣ Loops in JavaScript

Loops allow executing the same block of code multiple times.

1. for Loop

Used when the number of iterations is known.

✅ **Example:**

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

✔️ **Explanation:**

- `let i = 1;` → Initialization
- `i <= 5;` → Condition
- `i++` → Increment

---

2. while Loop

Used when the number of iterations is unknown and based on a condition.

✅ **Example:**

```
let i = 1;
while (i <= 5) {
    console.log("Iteration: " + i);
    i++;
}
```

---

3. do-while Loop

Similar to `while`, but executes at least once before checking the condition.

✅ **Example:**

```
let i = 1;
do {
    console.log("Iteration: " + i);
    i++;
} while (i <= 5);
```

---

4. for...in Loop (for Objects)

Loops through properties of an **object**.

✅ **Example:**

```
let person = { name: "Alice", age: 25, city: "New York" };
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

---

5. for...of Loop (for Arrays)

Loops through **values** in an array.

✅ **Example:**

```
let colors = ["Red", "Green", "Blue"];
for (let color of colors) {
    console.log(color);
}
```

📝 Recap Table

Concept	Description	Example
if	Executes if the condition is `true`	`if (x > 10) { ... }`
if-else	Executes one block if `true`, another if `false`	`if (x > 10) { ... } else { ... }`
if-else if-else	Handles multiple conditions	`if (x > 90) { ... } else if (x > 75) { ... }`
switch	Checks multiple values of a variable	`switch(day) { case "Monday": ... }`
for	Loops a known number of times	`for (let i = 0; i < 5; i++) { ... }`
while	Loops while a condition is `true`	`while (i < 5) { ... }`
do-while	Runs at least once, then checks condition	`do { ... } while (i < 5);`
for...in	Loops through object properties	`for (let key in obj) { ... }`
for...of	Loops through array values	`for (let value of arr) { ... }`

💡 Practice Problems

1️⃣ Write a program that checks if a number is **positive, negative, or zero** using `if-else`.

2️⃣ Use a `for` loop to print numbers from **1 to 10**.

3️⃣ Create an array of five names and use a `for...of` loop to display them.

4️⃣ Use a `while` loop to print even numbers from **2 to 10**.

5️⃣ Write a `switch` statement to print **"Weekend"** if the day is **Saturday** or **Sunday** and **"Weekday"** otherwise.

4.4 Functions in JavaScript

1️⃣ What is a Function?

A **function** is a reusable block of code that performs a specific task. Functions help in:

✅ Reducing code duplication

✅ Making code **modular** and **readable**

✅ Allowing **code reuse**

2️⃣ Declaring Functions in JavaScript

JavaScript provides multiple ways to define functions:

1. Function Declaration (Named Function)

A function is declared using the `function` keyword and a name.

✅ **Example:**

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

---

2. Function Expression (Anonymous Function)

Functions can be stored in variables.

✅ **Example:**

```
let greet = function() {
    console.log("Hello, World!");
};
greet(); // Calling the function
```

✔️ **Difference Between Function Declaration and Function Expression:**

Feature	Function Declaration	Function Expression
Hoisted?	✅ Yes	❌ No
Can be called before definition?	✅ Yes	❌ No

---

3. Arrow Function (ES6)

A shorter way to write functions using `⇒`.

✅ **Example:**

```
let greet = () ⇒ console.log("Hello, World!");
greet();
```

✔️ **Why use arrow functions?**

- **Shorter** syntax

- No `this` binding issues

3️⃣ Parameters and Return Values

Functions can **accept inputs (parameters)** and **return values**.

1. Function with Parameters

✅ **Example:**

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

---

2. Function with Return Value

✅ **Example:**

```
function add(a, b) {
    return a + b;
}
let sum = add(5, 10);
console.log(sum); // Output: 15
```

✔️ **Why return a value?**

- Allows the function to **send data back**
- The result can be **used elsewhere** in the program

4️⃣ Default and Rest Parameters

1. Default Parameters

If a parameter is not passed, JavaScript uses the **default value**.

✅ **Example:**

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

---

2. Rest Parameters (`...`)

Allows a function to accept **any number of arguments**.

✅ **Example:**

```
function sum(...numbers) {
    let total = 0;
    for (let num of numbers) {
        total += num;
    }
    return total;
}
console.log(sum(1, 2, 3, 4)); // Output: 10
```

5️⃣ Function Scope and Hoisting

1. Scope in JavaScript

Scope defines where variables are accessible.

Type	Description	Example
Global Scope	Accessible anywhere	`let x = 10;`
Function Scope	Accessible only inside the function	`function test() { let y = 20; }`
Block Scope (ES6)	Accessible only inside `{}`	`if (true) { let z = 30; }`

✅ **Example:**

```
let x = 10;  // Global scope
function myFunction() {
    let y = 20;  // Function scope
    console.log(x); // Accessible
    console.log(y); // Accessible
}
console.log(y); // ❌ Error: y is not defined
```

---

2. Hoisting in JavaScript

Hoisting moves **function declarations** to the top of the code before execution.

✅ **Example (Hoisting Works with Function Declaration):**

```
greet(); // ✅ Works fine (Hoisted)
function greet() {
    console.log("Hello!");
}
```

✅ **Example (Hoisting Does Not Work with Function Expressions):**

```
greet(); // ❌ Error (Not Hoisted)
let greet = function() {
    console.log("Hello!");
};
```

✔️ **Why does hoisting matter?**

- Function declarations can be used **before they are defined**
- Function expressions **cannot** be used before declaration

📝 Recap Table

Concept	Description	Example
Function Declaration	Named function, hoisted	`function greet() {}`
Function Expression	Anonymous function, not hoisted	`let greet = function() {};`
Arrow Function	Shorter syntax, no `this` binding	`let greet = () => {};`
Parameters	Input values for functions	`function sum(a, b) {}`
Return Value	Sends a value back	`return a + b;`
Default Parameter	Assigns default values	`function greet(name="Guest") {}`
Rest Parameter	Handles multiple arguments	`function sum(...numbers) {}`
Scope	Variable accessibility	Global, Function, Block
Hoisting	Moves declarations to top	Functions are hoisted, expressions are not

---

💡 Practice Problems

1️⃣ Write a function `multiply` that takes two numbers and **returns** their product.

2️⃣ Create an **arrow function** named `square` that returns the square of a number.

3️⃣ Write a function `greetUser` that takes a name and prints `"Hello, [name]!"` with a **default name "Guest"** if no name is given.

4️⃣ Write a function `sumAll` that takes any number of arguments and returns their sum using **rest parameters**.

5️⃣ Test if function hoisting works by calling a function **before** its declaration.

4.4 Popup Boxes in JavaScript: Alert, Confirm, and Prompt

1️⃣ What are Popup Boxes?

Popup boxes are built-in JavaScript dialog boxes that display messages and interact with users.

✔️ **Why use popups?**

- To **display messages**
- To **get user input**
- To **confirm user actions**

2️⃣ Types of Popup Boxes

1. Alert Box (`alert()`)

Displays a message and an "OK" button. It is used to **inform** the user.

✅ **Example:**

```
alert("Welcome to JavaScript!");
```

📌 **When to use?**

- Displaying warnings
- Notifying the user of important information

---

2. Confirm Box (`confirm()`)

Displays a message with "OK" and "Cancel" buttons. Returns `true` if "OK" is clicked, and `false` if "Cancel" is clicked.

✅ **Example:**

```
let result = confirm("Are you sure you want to delete this?");
if (result) {
   console.log("Item deleted.");
} else {
   console.log("Action canceled.");
}
```

📌 **When to use?**

- Asking for confirmation before deleting something
- Preventing accidental actions

---

3. Prompt Box (`prompt()`)

Displays a text box where the user can enter a value. Returns the input as a string, or `null` if canceled.

✅ **Example:**

```
let name = prompt("Enter your name:");
if (name) {
```

```
    console.log("Hello, " + name + "!");
} else {
    console.log("You did not enter a name.");
}
```

📌 **When to use?**

- Asking the user for input
- Getting values for calculations

📝 Recap Table

Popup Box	Purpose	Buttons	Returns
Alert (`alert()` **)**	Displays a message	"OK"	`undefined`
Confirm (`confirm()` **)**	Asks for confirmation	"OK" / "Cancel"	`true` (OK) / `false` (Cancel)
Prompt (`prompt()` **)**	Gets user input	"OK" / "Cancel"	String (input) / `null` (Cancel)

💡 Practice Problems

1️⃣ Create an **alert box** that says **"JavaScript is awesome!"**

2️⃣ Use a **confirm box** to ask the user if they want to continue, and log the response.

3️⃣ Use a **prompt box** to ask for the user's **age** and display it in the console.

4.5.1 Objects and Properties in JavaScript

1️⃣ What is an Object?

An **object** is a collection of key-value pairs. Each key (or property) has a corresponding value. Objects allow us to **store multiple related values** in a structured way.

✅ **Example of an Object:**

```
let person = {
    name: "Alice",
    age: 25,
    city: "New York"
};
console.log(person.name); // Output: Alice
```

✔️ **Why use objects?**

- Store multiple related values
- Easily organize and retrieve data
- Used extensively in **real-world applications**

2️⃣ Creating Objects

There are **three** main ways to create an object:

1. Using Object Literals (Recommended)

```
let car = {
    brand: "Toyota",
    model: "Corolla",
    year: 2022
};
```

---

2. Using the `new Object()` Constructor

```
let car = new Object();
car.brand = "Toyota";
car.model = "Corolla";
car.year = 2022;
```

---

3. Using a Constructor Function

```
function Car(brand, model, year) {
    this.brand = brand;
    this.model = model;
    this.year = year;
}
let myCar = new Car("Toyota", "Corolla", 2022);
console.log(myCar.brand); // Output: Toyota
```

✔️ **Which method is best?**

- **Object literals** are the **simplest and most common**
- **Constructor functions** are useful for **creating multiple objects**

3️⃣ Accessing Object Properties

1. Dot Notation (`.`)

```
console.log(person.name); // Output: Alice
```

2. Bracket Notation (`[]`)

```
console.log(person["age"]); // Output: 25
```

✔️ **When to use bracket notation?**

- When property names **contain spaces**
- When accessing properties dynamically

✅ **Example:**

```
let property = "city";
console.log(person[property]); // Output: New York
```

4️⃣ Adding, Modifying, and Deleting Properties

1. Adding a New Property

```
person.country = "USA";
console.log(person);
```

2. Modifying an Existing Property

```
person.age = 30;
```

3. Deleting a Property

```
delete person.city;
console.log(person);
```

5️⃣ Object Methods (Functions Inside Objects)

Functions inside objects are called **methods**.

✅ **Example:**

```
let student = {
    name: "Bob",
    age: 21,
    greet: function() {
```

```
        return "Hello, my name is " + this.name;
    }
};
console.log(student.greet()); // Output: Hello, my name is Bob
```

✔️ **Why use methods?**

- Keep related actions inside the object
- Improve code organization

6️⃣ Looping Through Objects

Use `for...in` to loop through **object properties**.

✅ **Example:**

```
for (let key in student) {
    console.log(key + ": " + student[key]);
}
```

📝 Recap Table

Concept	Description	Example
Object	Collection of key-value pairs	`{ name: "Alice", age: 25 }`
Dot Notation	Access properties using `.`	`obj.name`
Bracket Notation	Access properties using `[]`	`obj["name"]`
Adding Property	Add new properties to an object	`obj.country = "USA";`
Modifying Property	Change an existing property	`obj.age = 30;`
Deleting Property	Remove a property	`delete obj.age;`
Object Method	Function inside an object	`obj.greet = function() {}`
Looping Objects	Looping through keys in an object	`for (let key in obj) {}`

💡 Practice Problems

1️⃣ Create an object `book` with properties **title, author, and pages**.

2️⃣ Write a method inside `book` that returns a summary of the book.

3️⃣ Add a new property **publishedYear** to the `book` object.

4️⃣ Use a `for...in` loop to print all properties of `book`.

4.5.2 Constructors in JavaScript

1️⃣ What is a Constructor?

A **constructor** is a special function used to create multiple objects with the same properties and methods. It acts as a **blueprint** for objects.

✔️ **Why use constructors?**

- Creates multiple objects **efficiently**
- Helps **organize code**
- Used in **Object-Oriented Programming (OOP)**

✅ **Example (Without Constructor):**

```
let car1 = { brand: "Toyota", model: "Corolla", year: 2022 };
let car2 = { brand: "Honda", model: "Civic", year: 2023 };
console.log(car1, car2);
```

🚨 **Problem:** We are manually creating multiple objects. This is inefficient.

2️⃣ Creating a Constructor Function

A **constructor function** starts with a **capital letter** and uses `this` to define properties.

✅ **Example:**

```
function Car(brand, model, year) {
    this.brand = brand;
    this.model = model;
    this.year = year;
}

// Creating objects using the constructor
let car1 = new Car("Toyota", "Corolla", 2022);
let car2 = new Car("Honda", "Civic", 2023);

console.log(car1.brand); // Output: Toyota
console.log(car2.model); // Output: Civic
```

✔️ **Why use `new`?**

- It **creates a new object**
- `this` inside the constructor refers to **the new object**

3️⃣ Adding Methods to a Constructor

Methods can be added inside a constructor to define object behavior.

✅ **Example:**

```
function Car(brand, model, year) {
    this.brand = brand;
    this.model = model;
    this.year = year;

    this.getDetails = function() {
        return this.brand + " " + this.model + " (" + this.year + ")";
    };
}

let myCar = new Car("Ford", "Mustang", 2021);
console.log(myCar.getDetails()); // Output: Ford Mustang (2021)
```

4️⃣ Using `prototype` to Add Methods

Instead of adding methods inside the constructor, we can use **prototype** to improve memory efficiency.

✅ **Example:**

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

// Adding method using prototype
Person.prototype.greet = function() {
    return "Hello, my name is " + this.name;
};

let person1 = new Person("Alice", 25);
console.log(person1.greet()); // Output: Hello, my name is Alice
```

✔️ **Why use prototype?**

- Saves memory by **sharing methods** among all objects
- Keeps the constructor **clean and optimized**

5️⃣ Difference Between Constructor and Object Literal

Feature	Constructor Function	Object Literal

Usage	Used to create multiple objects	Used for single objects
Syntax	Uses `function` and `this`	Uses `{ key: value }`
Scalability	Efficient for multiple objects	Not ideal for many objects

📝 Recap Table

Concept	Description	Example
Constructor Function	Creates multiple objects	`function Car() {}`
this Keyword	Refers to the new object	`this.brand = brand;`
new Keyword	Creates an instance of an object	`let car1 = new Car("Toyota", "Corolla");`
Method in Constructor	Function inside constructor	`this.getDetails = function() {}`
Prototype	Adds methods efficiently	`Person.prototype.greet = function() {}`

💡 Practice Problems

1️⃣ Create a constructor function `Book` with properties **title, author, and pages**.

2️⃣ Add a method `getSummary()` inside `Book` to return a short description.

3️⃣ Use `prototype` to add a method `isLongBook()` that returns `true` if pages > 300.

4️⃣ Create **two book objects** and test their methods.

4.6 Arrays in JavaScript

1️⃣ What is an Array?

An **array** is a special type of object that can store multiple values in a **single variable**.

✔️ **Why use arrays?**

- Store multiple values **efficiently**
- Access elements using an **index**
- Useful for **looping** and **manipulating data**

✅ **Example of an Array:**

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits[0]); // Output: Apple
```

2️⃣ Creating an Array

There are **two** ways to create an array:

1. Using Square Brackets (Recommended)

```
let numbers = [10, 20, 30, 40, 50];
```

---

2. Using the `new Array()` Constructor

```
let numbers = new Array(10, 20, 30, 40, 50);
```

🚨 **Not recommended** because it can cause unexpected behavior.

3️⃣ Accessing and Modifying Array Elements

1. Accessing Elements (Using Index)

```
let colors = ["Red", "Green", "Blue"];
console.log(colors[1]); // Output: Green
```

2. Modifying Elements

```
colors[2] = "Yellow";
console.log(colors); // Output: ["Red", "Green", "Yellow"]
```

3. Getting Array Length

```
console.log(colors.length); // Output: 3
```

4️⃣ Array Methods

JavaScript provides many **built-in methods** for working with arrays.

1. Adding and Removing Elements

Method	Action	Example
push()	Adds an element at the **end**	`arr.push("Mango")`
pop()	Removes the **last** element	`arr.pop()`
unshift()	Adds an element at the **start**	`arr.unshift("Orange")`
shift()	Removes the **first** element	`arr.shift()`

✅ **Example:**

```
let fruits = ["Apple", "Banana"];
fruits.push("Cherry"); // ["Apple", "Banana", "Cherry"]
```

```
fruits.pop(); // ["Apple", "Banana"]
```

2. Combining and Slicing Arrays

Method	Action	Example
concat()	Merges two arrays	`arr1.concat(arr2)`
slice()	Extracts a part of an array	`arr.slice(1, 3)`
splice()	Adds/removes elements	`arr.splice(1, 2, "Mango")`

✅ **Example:**

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];
let combined = arr1.concat(arr2); // [1, 2, 3, 4, 5, 6]
```

3. Searching in an Array

Method	Action	Example
indexOf()	Finds the index of an element	`arr.indexOf("Banana")`
includes()	Checks if an element exists	`arr.includes("Apple")`

✅ **Example:**

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits.indexOf("Banana")); // Output: 1
console.log(fruits.includes("Grapes")); // Output: false
```

4. Iterating Through Arrays

Method	Action	Example
forEach()	Loops through each element	`arr.forEach(func)`
map()	Creates a new array by applying a function	`arr.map(func)`
filter()	Filters elements based on a condition	`arr.filter(func)`

✅ **Example (forEach):**

```
let numbers = [1, 2, 3, 4];
numbers.forEach(num => console.log(num * 2)); // Output: 2, 4, 6, 8
```

✅ **Example (map):**

```
let numbers = [1, 2, 3, 4];
let squared = numbers.map(num => num * num);
```

```
console.log(squared); // Output: [1, 4, 9, 16]
```

✅ **Example (filter):**

```
let numbers = [10, 20, 30, 40, 50];
let bigNumbers = numbers.filter(num => num > 25);
console.log(bigNumbers); // Output: [30, 40, 50]
```

5️⃣ Multi-Dimensional Arrays

A **multi-dimensional array** is an array inside another array.

✅ **Example:**

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
console.log(matrix[1][2]); // Output: 6
```

✔️ **Why use multi-dimensional arrays?**

- Represent grids, tables, and matrices
- Useful for game development and complex data structures

📝 Recap Table

Concept	Description	Example
Array Declaration	Creates an array	`let arr = [1, 2, 3]`
Access Elements	Uses an index	`arr[0]`
Modify Elements	Changes values	`arr[1] = "New"`
push()	Adds an element to the end	`arr.push(10)`
pop()	Removes the last element	`arr.pop()`
concat()	Merges two arrays	`arr1.concat(arr2)`
slice()	Extracts a section	`arr.slice(1, 3)`
indexOf()	Finds an index	`arr.indexOf("Apple")`
includes()	Checks if exists	`arr.includes("Mango")`
forEach()	Loops through elements	`arr.forEach(func)`
map()	Creates a new array	`arr.map(func)`
filter()	Filters elements	`arr.filter(func)`

💡 Practice Problems

1️⃣ Create an array `numbers` with values **10, 20, 30, 40**.

2️⃣ Add `"50"` at the end and `"5"` at the start using array methods.

3️⃣ Remove the last element from the array.

4️⃣ Use `map()` to create a new array with each number **squared**.

5️⃣ Use `filter()` to return only numbers greater than `25`.

🌟 Topic 4.7: Built-in Objects in JavaScript

JavaScript provides a wide set of *built-in objects* that make it easier to work with common data types and browser features. These objects come ready-made with properties and methods.

🔍 Overview: What Are Built-in Objects?

Built-in objects are predefined JavaScript objects that help you handle:

- Browser windows and interactions (`window`)
- Text manipulation (`String`)
- Numbers and math (`Number`, `Math`)
- Dates and time (`Date`)
- Boolean logic (`Boolean`)
- Patterns and validation (`RegExp`)
- Forms and form elements (`Form`)
- HTML content and structure (`DOM`)

We'll break this topic down object-by-object. Let's begin with the **Window object**.

🪟 1. The `window` Object

📘 Explanation:

The `window` object represents the browser's window. In a browser environment, **everything is part of the window object** — including `alert`, `prompt`, `document`, `console`, etc.

You can omit `window.` and JavaScript will still understand what you mean.

📋 Common Properties and Methods of `window`:

Method/Property	Description

`alert()`	Displays an alert box with a message.
`confirm()`	Displays a confirmation dialog box. Returns `true` or `false`.
`prompt()`	Displays a prompt box to get input from the user. Returns string or `null`.
`open()`	Opens a new browser window.
`close()`	Closes the current browser window.
`setTimeout()`	Calls a function after a specified delay (milliseconds).
`setInterval()`	Calls a function repeatedly with specified delay.
`location.href`	Gets or sets the URL of the current page.

🧪 Example:

```html
<!DOCTYPE html>
<html>
<head>
  <title>Window Object Example</title>
</head>
<body>

<button onclick="greet()">Click Me</button>

<script>
function greet() {
  let name = window.prompt("Enter your name:");
  if (name) {
    window.alert("Hello, " + name + "!");
  }
}
</script>

</body>
</html>
```

💡 `prompt` gets input, `alert` displays it.

🧠 Practice Problems

1. Write a script that asks the user to enter a number and then displays double the number using `alert()`.
2. Use `confirm()` to ask the user if they want to continue. Show a message based on the user's choice.
3. Write a script using `setTimeout()` to display a message after 5 seconds.

🧾 Recap Table — `window` Object

Feature	Description	Example
`window.alert()`	Shows an alert box with a message	`alert("Hi!")`
`window.prompt()`	Gets user input via dialog box	`prompt("Your name?")`
`window.confirm()`	Confirms user choice (OK/Cancel)	`confirm("Are you sure?")`
`setTimeout()`	Executes function once after delay (ms)	`setTimeout(fn, 1000)`
`setInterval()`	Repeats execution every interval	`setInterval(fn, 2000)`
`window.open()`	Opens a new window	`window.open("https://...")`
`window.location.href`	Gets or sets the current URL	`location.href = "..."`

🔤 2. The `String` Object

📘 Explanation:

In JavaScript, a **string** is a sequence of characters. String values can be created using quotes (`''`, `""`, or backticks ```` for template literals).

You can use **string methods** to manipulate and examine strings.

✨ Common `String` Methods and Properties:

Method / Property	Description
`length`	Returns the length of the string
`charAt(index)`	Returns the character at the specified index
`toUpperCase()`	Converts string to uppercase
`toLowerCase()`	Converts string to lowercase
`substring(start, end)`	Extracts characters from `start` to `end-1`
`slice(start, end)`	Similar to `substring` but supports negative indices
`indexOf(substring)`	Returns index of first occurrence or -1 if not found
`lastIndexOf(substring)`	Returns last occurrence of substring
`includes(substring)`	Returns `true` if substring exists in string
`replace(search, new)`	Replaces matched substring
`trim()`	Removes whitespace from both ends
`split(separator)`	Splits string into array based on separator
`concat()`	Joins strings
`startsWith()`	Checks if string starts with given substring
`endsWith()`	Checks if string ends with given substring

🧪 Example:

```html
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
let message = "  Hello, Web Technology Student!  ";
let trimmed = message.trim();
let upper = trimmed.toUpperCase();
let firstWord = trimmed.split(" ")[0];

document.getElementById("demo").innerHTML =
  "Original: '" + message + "'<br>" +
  "Trimmed: '" + trimmed + "'<br>" +
  "Uppercase: " + upper + "<br>" +
  "First Word: " + firstWord;
</script>

</body>
</html>
```

🔍 Difference between `substring()` vs `slice()`:

Feature	`substring(start, end)`	`slice(start, end)`
Indexing	Cannot handle negative values	Can handle negative values
Use case	Extract substring from index to index	Extract part of string, with more control
Example	`"hello".substring(1, 4)` → `"ell"`	`"hello".slice(-3)` → `"llo"`

🧠 Practice Problems

1. Write a script to take a string from the user and convert it to all uppercase.
2. Ask the user for a sentence. Replace all spaces with .
3. Extract the domain name from the email `someone@example.com`.
4. Check if a string starts with `"CSIT"` and ends with `"2025"`.

🧾 Recap Table — `String` Object

Feature	Use	Example
`length`	Get string length	`"Hello".length → 5`
`charAt()`	Get character at index	`"Hello".charAt(1) → 'e'`

`toUpperCase()`	Convert to upper case	`"hi".toUpperCase() → 'HI'`
`substring()`	Extract part of string	`"abcde".substring(1,3) → 'bc'`
`slice()`	Like substring, with negative index	`"abcde".slice(-2) → 'de'`
`indexOf()`	Find position of substring	`"abc".indexOf('b') → 1`
`includes()`	Check if string contains substring	`"abc".includes('a') → true`
`split()`	Convert string to array	`"a,b,c".split(',') → ['a','b','c']`
`trim()`	Remove surrounding spaces	`" hi ".trim() → "hi"`

🔢 3. The `Number` Object

📘 Explanation:

JavaScript has one main numeric type: `Number`. It can represent both integers and floating-point numbers.

When you use a number like `let a = 5`, it's a primitive. But JavaScript wraps it in a `Number` object so you can use helpful **methods and properties**.

💡 Creating Numbers:

```
let x = 42;              // number (primitive)
let y = new Number(42);     // Number object (not recommended)
```

👉 Use the primitive style (`let x = 42`) for best performance and behavior.

✨ Common `Number` Methods and Properties:

Method / Property	Description
`toFixed(n)`	Rounds to `n` decimal places and returns a string
`toString()`	Converts number to string
`parseInt(str)`	Converts string to integer
`parseFloat(str)`	Converts string to floating-point number
`isNaN(value)`	Checks if the value is `NaN` (Not a Number)
`isFinite(value)`	Checks if value is a finite number
`Number.MAX_VALUE`	Largest representable number
`Number.MIN_VALUE`	Smallest representable number (closest to 0)
`Number.POSITIVE_INFINITY`	Infinity constant
`Number.NEGATIVE_INFINITY`	Negative infinity constant

🧪 Example:

```html
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
let num = 7.56789;

let rounded = num.toFixed(2); // 7.57
let str = num.toString();     // "7.56789"
let isNumber = isNaN(num);    // false
let parsed = parseInt("42px"); // 42

document.getElementById("demo").innerHTML =
  "Rounded: " + rounded + "<br>" +
  "To String: " + str + "<br>" +
  "Is NaN?: " + isNumber + "<br>" +
  "Parsed Int: " + parsed;
</script>

</body>
</html>
```

🔍 `parseInt` vs `parseFloat` vs `Number()`

Function	Description	Example
`parseInt()`	Converts string to integer	`parseInt("12.34") → 12`
`parseFloat()`	Converts string to float	`parseFloat("12.34") → 12.34`
`Number()`	Converts string to number (if valid)	`Number("12.34") → 12.34`

🧠 Practice Problems

1. Ask the user to input a number with decimals and round it to 2 decimal places.
2. Write a function that converts a string like `"123abc"` into a number and ignores non-digit characters.
3. Create a script to check whether the input is a valid number using `isNaN()`.

🧾 Recap Table — `Number` Object

Feature	Use	Example
`toFixed(n)`	Round to `n` decimals	`(3.1415).toFixed(2) → "3.14"`
`toString()`	Convert to string	`(123).toString() → "123"`

`parseInt()`	Convert to integer	`parseInt("50px") → 50`
`parseFloat()`	Convert to float	`parseFloat("5.67") → 5.67`
`isNaN()`	Check if not a number	`isNaN("abc") → true`
`Number()`	Converts valid string to number	`Number("42") → 42`

✅ 4. The `Boolean` Object

📘 Explanation:

In JavaScript, a **Boolean** represents one of two values:

- `true`
- `false`

Boolean values are often used in **conditional statements**, loops, comparisons, and logic-based programming.

Just like with other types, you can create a Boolean as a **primitive** or using the **Boolean object**, but primitives are preferred.

✨ Creating Boolean Values

```
let isOn = true;              // primitive boolean
let isOff = new Boolean(false); // Boolean object (not recommended)
```

❗ Difference Between Boolean Object and Primitive

Feature	Primitive Boolean	Boolean Object (`new Boolean()`)
Type	`boolean`	`object`
Truthy/Falsy	`false` is falsy	`new Boolean(false)` is truthy!

```
if (new Boolean(false)) {
  console.log("This runs!"); // Yes, because object is always truthy
}
```

✅ **Recommendation:** Always use `true` / `false` as primitive values.

💡 Boolean Conversion Rules

JavaScript can convert other types into Boolean using:

- **Automatic coercion** (in conditions)
- **Explicit conversion** (`Boolean(value)`)

🧾 Values Converted to `false` (falsy):

Value	Type
`false`	Boolean
`0`, `-0`	Number
`""`	String
`null`	Null
`undefined`	Undefined
`NaN`	Number

All **other values** are treated as `true` (truthy).

🧪 Example:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
let x = 0;
let y = "Hello";
let z = "";

document.getElementById("demo").innerHTML =
  "Boolean(0): " + Boolean(x) + "<br>" +
  "Boolean('Hello'): " + Boolean(y) + "<br>" +
  "Boolean(''): " + Boolean(z);
</script>

</body>
</html>
```

📌 Output:

```
Boolean(0): false
Boolean('Hello'): true
Boolean(''): false
```

🧠 Practice Problems

1. Take a string input from the user. Convert it to Boolean and display whether it's truthy or falsy.
2. Write a function that checks if a user input is empty and returns `false` if it is.

3. Test Boolean conversion for: `undefined`, `null`, `NaN`, `0`, `"0"`, and `"false"`.

🧾 Recap Table — `Boolean` Object

Concept	Description	Example
Primitive Boolean	Preferred way to use	`let flag = true;`
Boolean object (avoid)	Treated as truthy even if `false`	`new Boolean(false)`
Falsy values	Convert to false in Boolean context	`0`, `""`, `null`, `undefined`
Truthy values	All other values	`"hello"`, `1`, `{}`, `[]`
Convert to Boolean	Use `Boolean(value)`	`Boolean("") → false`

📅 5. The `Date` Object

📘 Explanation:

The `Date` **object** in JavaScript is used to work with **dates and times** — you can create, manipulate, and format date and time values.

It automatically represents the current date and time unless specified otherwise.

✨ Creating Date Objects

Syntax	Description
`new Date()`	Current date and time
`new Date(milliseconds)`	Time since Jan 1, 1970
`new Date(dateString)`	Parses a date string
`new Date(year, month, ...)`	Specific date (note: month is 0-based!)

```
let now = new Date();                    // current date/time
let specific = new Date(2024, 3, 18);    // April 18, 2024 (months: 0-11)
let parsed = new Date("2025-12-25T10:00:00");   // December 25, 2025, 10 AM
```

🕐 Common `Date` Methods

Method	Description	Example Output
`getFullYear()`	Gets 4-digit year	`2025`
`getMonth()`	Gets month (0-11)	`3` (April)
`getDate()`	Gets day of the month (1-31)	`18`
`getDay()`	Gets day of week (0-Sun, 6-Sat)	`5` (Friday)
`getHours()`	Gets hours (0–23)	`14`
`getMinutes()`	Gets minutes (0–59)	`35`

`getSeconds()`	Gets seconds (0–59)	`10`
`getTime()`	Gets milliseconds since Jan 1, 1970	`1725434000000`
`toDateString()`	Converts to readable date string	`"Fri Apr 18 2025"`
`toTimeString()`	Converts to readable time string	`"14:35:10 GMT+0545 (Nepal Time)"`

🧪 Example:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
let now = new Date();
let message = "Today is: " + now.toDateString() +
          "<br>Time: " + now.toTimeString() +
          "<br>Year: " + now.getFullYear() +
          "<br>Month: " + (now.getMonth() + 1) +
          "<br>Day: " + now.getDate();

document.getElementById("demo").innerHTML = message;
</script>

</body>
</html>
```

🧠 Practice Problems

1. Create a date object for your birthday and print the day of the week you were born on.
2. Write a function that calculates the number of days between today and a future exam date.
3. Display the current time in `HH:MM:SS` format and update it every second (hint: `setInterval()`).

🧾 Recap Table — `Date` Object

Feature	Description	Example
`new Date()`	Current date and time	`new Date()`
`getFullYear()`	Get year (e.g. 2025)	`date.getFullYear()`
`getMonth()`	Get month (0-11)	`date.getMonth()`
`getDate()`	Day of month (1-31)	`date.getDate()`
`getDay()`	Day of week (0-6, Sun-Sat)	`date.getDay()`

`toDateString()`	Returns readable date	`"Fri Apr 18 2025"`
`getTime()`	Milliseconds since 1970	`date.getTime()`

➗ 6. The `Math` Object

📘 Explanation:

The `Math` **object** provides a library of **mathematical constants and functions**. Unlike `Date` or `String`, you **don't need to create** a new `Math` object — just use its methods directly:

```
let result = Math.sqrt(25); // 5
```

It's built-in and always available!

✨ Common `Math` Properties

Property	Description	Example
`Math.PI`	π ≈ 3.141592653589793	`Math.PI`
`Math.E`	Euler's number ≈ 2.718	`Math.E`
`Math.SQRT2`	Square root of 2	`Math.SQRT2`

✨ Common `Math` Methods

Method	Description	Example
`Math.abs(x)`	Absolute value	`Math.abs(-5) → 5`
`Math.ceil(x)`	Rounds up to nearest integer	`Math.ceil(4.2) → 5`
`Math.floor(x)`	Rounds down to nearest integer	`Math.floor(4.9) → 4`
`Math.round(x)`	Rounds to nearest integer	`Math.round(4.6) → 5`
`Math.max(a, b, ...)`	Returns the highest value	`Math.max(3, 5, 1) → 5`
`Math.min(a, b, ...)`	Returns the lowest value	`Math.min(3, 5, 1) → 1`
`Math.pow(x, y)`	Returns x raised to the power y	`Math.pow(2, 3) → 8`
`Math.sqrt(x)`	Square root	`Math.sqrt(16) → 4`
`Math.random()`	Returns random number between 0 (inclusive) and 1 (exclusive)	`0.0 ≤ x < 1.0`
`Math.trunc(x)`	Removes decimal part	`Math.trunc(4.7) → 4`

🧪 Example:

```
<!DOCTYPE html>
<html>
<body>
```

```
<p id="demo"></p>

<script>
let a = Math.sqrt(49);          // 7
let b = Math.round(3.75);        // 4
let c = Math.random();          // random between 0 and 1
let d = Math.pow(2, 5);         // 32

document.getElementById("demo").innerHTML =
  "√49 = " + a + "<br>" +
  "Round(3.75) = " + b + "<br>" +
  "Random number = " + c.toFixed(2) + "<br>" +
  "2^5 = " + d;
</script>

</body>
</html>
```

🔢 Generate Random Integer (e.g., 1 to 10)

```
let rand = Math.floor(Math.random() * 10) + 1;
```

🧠 Practice Problems

1. Generate a random number between 5 and 15.
2. Write a script to find the maximum and minimum of 3 numbers input by the user.
3. Create a calculator that computes `a^b` for any two input numbers.
4. Use `Math.trunc()` to remove decimal points from user input.

🧾 Recap Table — `Math` Object

Feature	Use	Example
`Math.PI`	Constant π	`Math.PI → 3.14...`
`Math.sqrt(x)`	Square root	`Math.sqrt(9) → 3`
`Math.pow(x,y)`	Power	`Math.pow(2,3) → 8`
`Math.abs(x)`	Absolute value	`Math.abs(-5) → 5`
`Math.floor()`	Round down	`Math.floor(2.9) → 2`
`Math.ceil()`	Round up	`Math.ceil(2.1) → 3`
`Math.round()`	Nearest integer	`Math.round(2.5) → 3`
`Math.random()`	Random (0 ≤ x < 1)	`Math.random() → 0.35...`

`Math.max()`	Highest among args	`Math.max(1,2,3) → 3`

🔍 7. The `RegExp` Object (Regular Expressions)

📘 Explanation:

A **Regular Expression (RegExp)** is a pattern used to **match character combinations in strings**. It's extremely useful for:

- Validating form inputs
- Searching within text
- Replacing parts of strings

JavaScript supports RegExp using the **RegExp object** and **literal syntax**.

✨ Creating a RegExp

Syntax	Example	Description
Literal	`/pattern/flags`	Preferred
Constructor (object)	`new RegExp("abc")`	Less common

```
let pattern1 = /hello/i;          // literal with case-insensitive flag
let pattern2 = new RegExp("hello"); // using constructor
```

✨ Common RegExp Flags

Flag	Meaning
`g`	Global (match all)
`i`	Case-insensitive
`m`	Multi-line matching

✨ Useful Metacharacters & Syntax

Pattern	Description	Example Match
`.`	Any character except newline	`/h.t/` matches `hat`
`^`	Start of string	`/^hi/` matches `hi`
`$`	End of string	`/end$/` matches `the end`
`\d`	Digit (0–9)	`/\d/` matches `3`
`\w`	Word character (a-z, A-Z, 0–9, _)	`/\w/`
`\s`	Whitespace	`/\s/`
`+`	One or more	`/a+/` matches `aaa`
`*`	Zero or more	`/a*/` matches `""`, `a`, `aa`

`?`	Zero or one	`/a?/`
`[abc]`	Any one of `a`, `b`, or `c`	
`[^abc]`	Not `a`, `b`, or `c`	
`(x	y)`	Either `x` or `y`

✨ Common RegExp Methods

Method	Use Case	Example Result
`test(str)`	Returns `true` if match is found	`/cat/.test("catalog")` → `true`
`exec(str)`	Returns matched result or `null`	`/\d+/.exec("Item 32")` → `32`
`match()` (string method)	Returns array of matches	`"abc".match(/a/)`
`replace()`	Replaces matched substring	`"hi 123".replace(/\d+/, "**")`

🧪 Example — Validate Email

```
<!DOCTYPE html>
<html>
<body>

<input type="text" id="email" placeholder="Enter email">
<button onclick="validateEmail()">Check</button>
<p id="result"></p>

<script>
function validateEmail() {
  let email = document.getElementById("email").value;
  let regex = /^[a-zA-Z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$/i;
  let result = regex.test(email);
  document.getElementById("result").innerHTML = result ? "Valid" : "Invalid";
}
</script>

</body>
</html>
```

🧠 Practice Problems

1. Write a RegExp to validate a **phone number** (e.g., `9841234567`).
2. Create a pattern that only matches strings that **start with "CSIT"** and **end with digits**.
3. Replace all numbers in a string with `#` using RegExp.
4. Match all words starting with capital letters in a sentence.

🧾 Recap Table — `RegExp` Object

Feature	Use	Example
`/pattern/flags`	Create a regular expression	`/abc/i`
`.test(str)`	Check if string matches pattern	`/abc/.test("abc") → true`
`.exec(str)`	Return matched string or null	`/\d+/.exec("Item 42") → "42"`
`replace()`	Replace matched parts	`"123abc".replace(/\d/, "#")`
`^`, `$`	Start, end of string	`/^hi$/` matches `"hi"` only
`\d`, `\w`, `\s`	Digits, word chars, whitespace	
`+`, `*`, `?`	Quantifiers	
`[abc]`, `[^abc]`	Character sets	
`(x	y)`	Alternation (either-or)

📝 8. The `Form` Object

📘 Explanation:

The **Form object** in JavaScript refers to the **HTML form element** and its **associated input controls**. It allows you to:

- Access form fields
- Validate input values
- Handle form submissions dynamically

JavaScript accesses forms through the `document.forms` collection or by referencing them with `id` or `name`.

🛠️ Accessing a Form and Its Elements

Assuming we have:

```
<form id="loginForm" name="login">
  <input type="text" name="username">
  <input type="password" name="password">
</form>
```

You can access it via:

```
let form = document.forms["login"];
let username = form["username"].value;
```

Or:

```
let username = document.getElementById("loginForm").elements["username"].value;
```

🔍 Common Properties of the Form Object

Property / Method	Description
`form.name`	Gets the name of the form
`form.elements`	Returns all form elements as a collection
`form.elements[i]`	Access individual input elements
`form.submit()`	Submits the form programmatically
`form.reset()`	Resets the form fields
`element.value`	Gets/sets the value of an input field
`element.checked`	Checks if checkbox/radio is selected

🧪 Example — Access Form Input

```html
<!DOCTYPE html>
<html>
<body>

<form id="infoForm">
  Name: <input type="text" name="username">><br>
  Age: <input type="number" name="age">><br>
  <button type="button" onclick="showData()">Submit</button>
</form>

<p id="output"></p>

<script>
function showData() {
  let form = document.getElementById("infoForm");
  let name = form.elements["username"].value;
  let age = form.elements["age"].value;
  document.getElementById("output").innerHTML =
    "Name: " + name + "<br>Age: " + age;
}
</script>

</body>
</html>
```

💡 Form vs Form Elements

Term	Refers To
`form`	Entire `<form>` element

`form.elements`	All input elements inside the form
`element.value`	The input entered by the user
`element.checked`	Used for checkbox or radio input

🧠 Practice Problems

1. Create a form with fields: Name, Email, and Checkbox for “I agree”. Use JavaScript to validate that all fields are filled before submitting.
2. Write a script that resets a form when a "Clear" button is clicked.
3. Access and display the selected value from a dropdown using the form object.

🧾 Recap Table — `Form` Object

Feature	Use	Example
`document.forms`	Access all forms	`document.forms[0]`
`form.elements`	Access inputs in form	`form.elements["username"]`
`element.value`	Get/set user input	`form.elements["email"].value`
`element.checked`	Check checkbox/radio status	`form.elements["agree"].checked`
`form.submit()`	Submit form using JavaScript	`form.submit()`
`form.reset()`	Reset form values	`form.reset()`

🌳 9. The `DOM` (Document Object Model)

📘 Explanation:

The **Document Object Model (DOM)** is a **tree-like representation** of the structure and content of an HTML or XML document.

Every element in a web page (like `<div>`, `<p>`, `<input>`, etc.) is represented as an **object in the DOM**, and JavaScript can interact with those objects to:

- Change content
- Modify styles
- Add/remove elements
- Handle events

🧠 DOM Structure Overview

Example HTML:

```
<!DOCTYPE html>
<html>
  <head><title>My Page</title></head>
```

```
  <body>
    <h1>Hello</h1>
    <p id="info">Welcome to the site.</p>
  </body>
</html>
```

This gets converted into a **tree of nodes**:

```
Document
 └── html
    ├── head
    │   └── title
    └── body
        ├── h1
        └── p#info
```

✨ Accessing DOM Elements

Method	Description	Returns
`getElementById("id")`	Selects a single element by its unique ID	Element
`getElementsByClassName("class")`	Selects elements by class name	HTMLCollection
`getElementsByTagName("tag")`	Selects elements by tag name	HTMLCollection
`getElementsByName("name")`	Selects form elements by `name` attribute	NodeList
`querySelector("selector")`	Selects the **first** matching CSS selector	Element
`querySelectorAll("selector")`	Selects **all** elements matching CSS selector	NodeList

🔧 Modifying DOM Elements

Action	Code Example
Change content	`element.innerHTML = "New Text"`
Change value (input)	`element.value = "New Value"`
Change style	`element.style.color = "red"`
Add class	`element.classList.add("highlight")`
Remove element	`element.remove()`
Create element	`document.createElement("div")`
Append child	`parent.appendChild(child)`

🔍 `getElementsByName()`

- Specifically useful in **forms**.

- Returns a **NodeList** of elements with the specified `name` attribute.
- Often used with radio buttons, checkboxes, and input fields that **share the same name**.

```
html
CopyEdit
<input type="radio" name="gender" value="Male">
<input type="radio" name="gender" value="Female">
```

```
javascript
CopyEdit
let genderInputs = document.getElementsByName("gender");
for (let g of genderInputs) {
  if (g.checked) {
    alert("Selected: " + g.value);
  }
}
```

🧪 Example — Change Text & Style

```
<!DOCTYPE html>
<html>
<body>

<h2 id="heading">Original Heading</h2>
<button onclick="changeText()">Change</button>

<script>
function changeText() {
  let heading = document.getElementById("heading");
  heading.innerHTML = "Updated Heading";
  heading.style.color = "blue";
}
</script>

</body>
</html>
```

✨ Traversing the DOM

Property	Description

`parentNode`	Gets the parent of an element
`childNodes`	Gets all child nodes
`firstChild`	Gets the first child
`lastChild`	Gets the last child
`nextSibling`	Gets the next sibling
`previousSibling`	Gets the previous sibling

🧠 Practice Problems

1. Change the background color of a `<div>` when a button is clicked.
2. Dynamically create a list (`<ul>`) and add 3 items using JavaScript.
3. Write a function that removes a paragraph from the page when clicked.
4. Use `querySelector` to target a class and update its font size.

🧾 Recap Table — `DOM` Object

Method	Description	Example
`getElementById("id")`	Unique element by ID	`document.getElementById("header")`
`getElementsByClassName()`	Elements with given class	`document.getElementsByClassName("box")`
`getElementsByTagName()`	Elements with tag name	`document.getElementsByTagName("p")`
`getElementsByName("name")`	Elements with `name` attribute	`document.getElementsByName("email")`
`querySelector()`	First match using CSS selector	`document.querySelector(".note")`
`querySelectorAll()`	All matches using CSS selector	`document.querySelectorAll("div.card")`

✅ Topic 4.8: User Defined Objects; Event Handling and Form Validation

---

📘 What are User Defined Objects?

In JavaScript, **user-defined objects** are custom structures that allow you to store and manage related data and functions. While JavaScript has built-in objects (like `Date`, `Math`, `Array`), sometimes you need to model real-world entities — like a `Student`, `Book`, or `Car`. This is where user-defined objects come in.

✅ Creating User Defined Objects – 3 Ways:

Method	Syntax	Example
1. Using Object Literals	`let obj = {key1: val1, key2: val2}`	`let student = { name: "Ram", age: 20 };`
2. Using Constructor Function	`function ObjName() { this.key = val; }`	`function Student(name, age) { this.name = name; this.age = age; }`

3. Using ES6 Classes	`class ObjName { constructor() {...} }`	`class Car { constructor(model) { this.model = model; } }`

🔍 Example 1: Object Literal

```
let student = {
  name: "Ram",
  age: 21,
  course: "CSIT",
  displayInfo: function () {
    console.log(`Name: ${this.name}, Age: ${this.age}, Course: ${this.course}`);
  }
};

student.displayInfo();
```

🔍 Example 2: Constructor Function

```
function Student(name, age, course) {
  this.name = name;
  this.age = age;
  this.course = course;
  this.displayInfo = function () {
    console.log(`Name: ${this.name}, Age: ${this.age}, Course: ${this.course}`);
  };
}

let s1 = new Student("Sita", 22, "CSIT");
s1.displayInfo();
```

🔍 Example 3: Using ES6 Class

```
class Book {
  constructor(title, author) {
    this.title = title;
    this.author = author;
  }

  display() {
    console.log(`Title: ${this.title}, Author: ${this.author}`);
  }
}
```

```
const book1 = new Book("Intro to JS", "John Doe");
book1.display();
```

✍️ Practice Problem 1:

Create a `Person` object using constructor function with properties: name, age, and address. Add a method to display full info.

✍️ Practice Problem 2:

Use ES6 Class to define a `Laptop` object with properties: brand, RAM, and processor. Add a method to show configuration.

📋 Recap Table – User Defined Objects

Concept	Syntax/Use	Key Notes
Object Literal	`let obj = { key: value }`	Easiest way to define object
Constructor Function	`function Obj() { this.key = value; }`	Allows creating multiple instances
ES6 Class	`class Obj { constructor(...) {} }`	More modern and readable
`this` keyword	Refers to current object	Used in methods to access properties
Method inside object	`methodName: function() {}`	Can also be written using ES6 shorthand

🔄 Event Handling in JavaScript

🔍 What is Event Handling?

Event handling in JavaScript is how we make web pages **interactive**. Events are actions or occurrences that happen in the browser, and we can write code (event handlers) to respond to them.

For example:

- A **click** on a button
- A **keypress** in a textbox
- A **mouse hover** on an image
- A **form submission**
- A **page load**

🎯 Why is Event Handling Important?

Without events, web pages would be static. Event handling lets users interact with the page in real time and makes dynamic web apps possible.

📌 Event Types (Commonly Used)

Event Type	Triggered When	Used On
`onclick`	Element is clicked	Button, div, link, etc.
`onmouseover`	Mouse moves over an element	Image, div
`onmouseout`	Mouse leaves the element	Image, div
`onkeydown`	Key is pressed	Input, document
`onkeyup`	Key is released	Input
`onchange`	Input/Select value is changed	`<input>`, `<select>`
`onfocus`	Element gains focus	Input
`onblur`	Element loses focus	Input
`onsubmit`	Form is submitted	`<form>`
`onload`	Page is fully loaded	`<body>`, `<img>`

🧠 3 Ways to Handle Events

1. Inline Event Handling (in HTML)

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

2. Using HTML DOM Property in JavaScript

```
<button id="btn">Click</button>

<script>
  document.getElementById("btn").onclick = function() {
    alert("Button Clicked");
  };
</script>
```

3. Using `addEventListener()` – ✅ Best Practice

```
<button id="btn">Click</button>

<script>
  const button = document.getElementById("btn");
  button.addEventListener("click", function() {
    alert("Clicked with addEventListener");
  });
</script>
```

> ✅ Why addEventListener() is preferred?
>
> - Allows **multiple event handlers** on the same element
> - More **flexible and cleaner** code

🧪 Example: Input Validation on Blur

```
<input type="text" id="username" placeholder="Enter username" />

<script>
 document.getElementById("username").addEventListener("blur", function() {
  if (this.value.length < 4) {
   alert("Username must be at least 4 characters.");
  }
 });
</script>
```

💪 Practice Problems

1. Create a form where:
   - A message is shown when a user focuses on a textbox.
   - A warning is shown when a user tries to leave a required field empty (on blur).
   - The background color of a textbox changes on mouse over.
2. Use `addEventListener` to handle:
   - Button click
   - Keypress event
   - Page load event (e.g., `window.addEventListener("load", ...)`)

🧾 Recap Table: Event Handling

Concept	Explanation
Event	User/browser action (e.g., click, load, keypress)
Event Handler	Function that runs in response to an event
Inline Handling	Directly in HTML using `onclick`, etc.
DOM Property Handling	Using `element.onclick = function()` in JS
`addEventListener()`	Preferred method, allows multiple handlers
Form Event Example	`onsubmit`, `onchange`, `onblur`, `onfocus`

Keyboard/Mouse Events	`onkeydown`, `onkeyup`, `onmouseover`, `onmouseout`, etc.

🧾 What is Form Validation?

Form validation ensures that the user has entered valid data before the form is submitted. It can be done using:

1. **HTML (built-in validation)**
2. **JavaScript (custom validation)**

We'll explore both.

✅ 1. HTML Form Validation (Just for knowledge)

HTML provides built-in validation using attributes in `<input>` elements.

Example:

```
<form>
  <label>Email:</label>
  <input type="email" required>
  <br>

  <label>Age:</label>
  <input type="number" min="18" max="100" required>
  <br>

  <input type="submit">
</form>
```

Key attributes:

Attribute	Use
`required`	Field must be filled
`type="email"`	Input must be a valid email
`min`, `max`	Sets numeric limits
`pattern`	Allows regex for custom patterns
`maxlength`	Max number of characters

This is easy and works out-of-the-box, but has limited customization.

🧠 2. JavaScript Form Validation (Custom)

JS validation gives full control. You can check inputs before submitting the form.

Basic Example:

```
<form id="myForm">
  <label>Username:</label>
  <input type="text" id="username" required>
  <br>

  <label>Password:</label>
  <input type="password" id="password" required>
  <br>

  <input type="submit" value="Login">
</form>

<script>
  document.getElementById('myForm').addEventListener('submit', function(e) {
    const username = document.getElementById('username').value.trim();
    const password = document.getElementById('password').value;

    if (username === '' || password === '') {
      alert("Both fields are required!");
      e.preventDefault(); // Stop form submission
    } else if (password.length < 6) {
      alert("Password must be at least 6 characters long.");
      e.preventDefault();
    }
  });
</script>
```

🔍 Advanced Example with Email and Pattern:

```
<form id="registerForm">
  <label>Email:</label>
  <input type="email" id="email" required>
  <br>

  <label>Phone (10 digits):</label>
  <input type="text" id="phone" required>
  <br>

  <input type="submit" value="Register">
</form>
```

```
<script>
 document.getElementById('registerForm').addEventListener('submit', function(e) {
  const email = document.getElementById('email').value.trim();
  const phone = document.getElementById('phone').value.trim();

  const phonePattern = /^\d{10}$/;

  if (!email.includes('@')) {
   alert("Enter a valid email.");
   e.preventDefault();
  }

  if (!phonePattern.test(phone)) {
   alert("Phone number must be 10 digits.");
   e.preventDefault();
  }
 });
</script>
```

🛠️ Practice Exercise for You

Try building a form with these fields:

- Name (required)
- Email (must be valid)
- Password (min 8 chars)
- Confirm password (must match)
- Age (between 18 and 60)

📄 Example: A Complete form with all types of input fields and JavaScript validation to cover:

- Text (letters only)
- Numbers
- Email
- Password (length)
- Confirm Password (match)
- Phone (10 digits)
- Age (range)

- Dropdown (select option)
- Checkbox (agree to terms)

💡 Complete HTML + JS Form Validation Example

```html
<!DOCTYPE html>
<html>
<head>
  <title>Full Form Validation</title>
</head>
<body>
  <h2>Registration Form</h2>
  <form id="fullForm">
    <!-- Name: Letters only -->
    <label>Name:</label>
    <input type="text" id="name" required>
    <br><br>

    <!-- Age: 18 - 60 -->
    <label>Age:</label>
    <input type="number" id="age" min="18" max="60" required>
    <br><br>

    <!-- Email -->
    <label>Email:</label>
    <input type="email" id="email" required>
    <br><br>

    <!-- Phone: Exactly 10 digits -->
    <label>Phone:</label>
    <input type="text" id="phone" required>
    <br><br>

    <!-- Password -->
    <label>Password:</label>
    <input type="password" id="password" required>
    <br><br>

    <!-- Confirm Password -->
    <label>Confirm Password:</label>
    <input type="password" id="confirmPassword" required>
    <br><br>
```

```
  <!-- Gender: Dropdown -->
  <label>Gender:</label>
  <select id="gender" required>
    <option value="">--Select--</option>
    <option value="male">Male</option>
    <option value="female">Female</option>
  </select>
  <br><br>

  <!-- Terms: Checkbox -->
  <label>
    <input type="checkbox" id="terms" required>
    I agree to the terms and conditions
  </label>
  <br><br>

  <input type="submit" value="Register">
</form>

<script>
  document.getElementById('fullForm').addEventListener('submit', function(e) {
    const name = document.getElementById('name').value.trim();
    const age = parseInt(document.getElementById('age').value);
    const email = document.getElementById('email').value.trim();
    const phone = document.getElementById('phone').value.trim();
    const password = document.getElementById('password').value;
    const confirmPassword = document.getElementById('confirmPassword').value;
    const gender = document.getElementById('gender').value;
    const terms = document.getElementById('terms').checked;

    // Regex patterns
    const namePattern = /^[A-Za-z\s]+$/;
    const phonePattern = /^\d{10}$/;

    // Name validation
    if (!namePattern.test(name)) {
      alert("Name must contain only letters and spaces.");
      e.preventDefault();
      return;
    }

    // Age validation
    if (isNaN(age) || age < 18 || age > 60) {
      alert("Age must be between 18 and 60.");
      e.preventDefault();
```

```
    return;
  }

  // Email validation (basic check)
  if (!email.includes('@') || !email.includes('.')) {
    alert("Please enter a valid email address.");
    e.preventDefault();
    return;
  }

  // Phone number validation
  if (!phonePattern.test(phone)) {
    alert("Phone number must be exactly 10 digits.");
    e.preventDefault();
    return;
  }

  // Password validation
  if (password.length < 6) {
    alert("Password must be at least 6 characters long.");
    e.preventDefault();
    return;
  }

  // Confirm password match
  if (password !== confirmPassword) {
    alert("Passwords do not match.");
    e.preventDefault();
    return;
  }

  // Gender selection
  if (gender === "") {
    alert("Please select a gender.");
    e.preventDefault();
    return;
  }

  // Terms and conditions
  if (!terms) {
    alert("You must agree to the terms and conditions.");
    e.preventDefault();
    return;
  }
```

```
    // If all validations pass:
    alert("Form submitted successfully!");
  });
 </script>
</body>
</html>
```

✅ What This Covers

Field	Validation
Name	Letters and spaces only
Age	Must be between 18–60
Email	Contains "@" and "."
Phone	Exactly 10 digits
Password	At least 6 characters
Confirm Password	Must match password
Gender	Must choose from dropdown
Terms	Must be checked

`.test()` vs `.match()`

In JavaScript, when you're working with **regular expressions (RegExp)**, you'll often see `.test()` and `.match()` used. They both help check if a string matches a pattern, but they are used **differently**.

🔍 `test()` – Does It Match?

- Belongs to the **RegExp object**.
- Returns a **boolean** (`true` or `false`).
- Used to **check if the pattern exists** in a string.

✅ Example:

```
const pattern = /^[A-Za-z]+$/;
const name = "Alice";

console.log(pattern.test(name)); // true
```

📥 `match()` – What Matches?

- Belongs to the **String object**.

- Returns **matched parts** of the string **as an array**, or `null` if no match.
- Good when you want to **extract matched text**.

✅ Example:

```
const name = "Alice123";
const result = name.match(/[A-Za-z]+/g);

console.log(result); // ["Alice"]
```

🔄 Side-by-Side Comparison

Feature	`test()`	`match()`
Returns	`true` / `false`	Array of matches or `null`
Use case	Check if it matches	Extract what matched
Called on	`RegExp`	`String`
Example use	`pattern.test("Hello")`	`"Hello123".match(/[A-Za-z]+/g)`

🧪 Quick Practice:

```
const email = "user@example.com";
const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

console.log(emailPattern.test(email));  // true (valid email?)
console.log(email.match(emailPattern)); // ['user@example.com']
```

📋 Some common regex for HTML form validation

#	Field Type	Regex Pattern	Description
1	**Email**	`/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/`	Valid email like `name@example.com`
2	**Password (Strong)**	`/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/`	Min 8 chars, with upper, lower, digit, special char
3	**Phone Number**	`/^\+?[0-9]{10,15}$/`	Allows optional `+`, 10–15 digits
4	**Date (YYYY-MM-DD)**	`/^\d{4}-(0[1-9]	1[0-2])-(0[1-9]
5	**Username**	`/^[a-zA-Z0-9_-]{3,16}$/`	3–16 chars, letters, numbers, `_`, `-`
6	**ZIP Code (US)**	`/^\d{5}(-\d{4})?$/`	`12345` or `12345-6789`
7	**URL**	`/^(https?:\/\/)?([\w\d-]+\.)+\w{2,}(\/.+)?$/`	Valid http/https URL

8	**Credit Card**	`/^\d{13,16}$/`	13–16 digit numbers
9	**Only Numbers**	`/^\d+$/`	Digits only
10	**Only Letters**	`/^[A-Za-z]+$/`	Letters only, no spaces
11	**Letters & Spaces**	`/^[A-Za-z\s]+$/`	Letters and spaces — good for names

❗ 4.9 Error Handling (Try / Catch)

🔍 What is Error Handling?

In JavaScript, **error handling** allows you to catch and respond to **runtime errors** so that they don't crash your script or webpage.

This is useful when:

- A user gives unexpected input
- A script accesses undefined variables
- Network or logic issues occur
- Debugging code during development

🧠 The Try-Catch Syntax

JavaScript provides the `try...catch` statement to handle exceptions (errors).

```
try {
  // Code that may cause an error
} catch (error) {
  // Code to handle the error
}
```

⚙️ How It Works:

Block	Purpose
`try`	Holds the code you want to test. If there's an error, execution jumps to `catch`.
`catch`	Captures the error and lets you handle it (show message, log it, etc.)
`finally` *(optional)*	Executes code regardless of whether an error occurred or not.

✅ Basic Example

```
try {
  let x = y + 5; // y is not defined
```

```
} catch (error) {
  alert("An error occurred: " + error.message);
}
```

Output:

```
An error occurred: y is not defined
```

🔄 With `finally` Block

```
try {
  let result = 10 / 0;
  console.log(result);
} catch (e) {
  console.log("Error:", e.message);
} finally {
  console.log("This will run no matter what.");
}
```

🔬 Real World Example: Input Validation with Error Handling

```
<input type="text" id="numberInput" placeholder="Enter a number" />
<button onclick="validateNumber()">Submit</button>

<script>
  function validateNumber() {
    try {
      let input = document.getElementById("numberInput").value;
      if (isNaN(input)) {
        throw new Error("Input is not a number!");
      }
      alert("Valid number: " + input);
    } catch (e) {
      alert("Error: " + e.message);
    }
  }
</script>
```

💪 Practice Problems

1. Wrap the following code in a `try...catch` block and show a custom alert if an error occurs:

```
let user = getUser(); // Assume this function might not exist
```

1. Create a form that throws a custom error if:
   - Email input does not contain `@`
   - Password is less than 6 characters

🧾 Recap Table: Error Handling

Keyword	Use
`try`	Wrap code that might throw an error
`catch`	Handle the error gracefully
`error.message`	Access message string from the error object
`finally`	Runs regardless of error (cleanup, logs, etc.)
`throw`	Manually generate an error

🍪 4.10 Handling Cookies in JavaScript

🔍 What are Cookies?

A **cookie** is a small piece of data stored in the browser by the website. It helps **remember information across sessions**, like:

- Logged-in user details
- Site preferences
- Cart items in e-commerce
- Session tracking

🧠 Key Characteristics of Cookies

- Stored as **key-value pairs**
- Saved in the user's browser
- Sent to the server with every request (in HTTP headers)
- Can be created, read, modified, and deleted using JavaScript

---

⚙️ Setting, Getting, and Deleting Cookies in JavaScript

✅ 1. Setting a Cookie

```
document.cookie = "username=csit_user";
```

👉 You can also set **expiry time**:

```
document.cookie = "username=csit_user; expires=Fri, 1 May 2025 12:00:00 UTC";
```

👉 Or add **path** (default is current path):

```
document.cookie = "username=csit_user; path=/";
```

🔍 2. Reading (Getting) a Cookie

```
console.log(document.cookie);
```

Returns all cookies in one string, like:

```
"username=csit_user; theme=dark"
```

You'll often have to **split and search** to get a specific cookie:

```
function getCookie(name) {
  let cookies = document.cookie.split("; ");
  for (let cookie of cookies) {
    let [key, value] = cookie.split("=");
    if (key === name) return value;
  }
  return null;
}
```

❌ 3. Deleting a Cookie

To delete, **set the expiry to a past date**:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

🧪 Example: Set and Get Username Cookie

```
<button onclick="setUser()">Set Username</button>
<button onclick="getUser()">Get Username</button>

<script>
  function setUser() {
```

```
  document.cookie = "username=csit_user; expires=Fri, 1 May 2025 12:00:00 UTC; path=/";
  alert("Cookie set!");
}

function getUser() {
  const name = getCookie("username");
  alert(name ? "Welcome back, " + name : "No user found.");
}

function getCookie(name) {
  let cookies = document.cookie.split("; ");
  for (let cookie of cookies) {
    let [key, value] = cookie.split("=");
    if (key === name) return value;
  }
  return null;
}
</script>
```

🧪 Practice Problems

1. Create a form with:
   - Username input
   - Submit button that stores the username in a cookie
2. On page load, if a cookie exists, display:
   - "Welcome, [username]" on the page

🧾 Recap Table: Handling Cookies

Operation	JavaScript Code
Set Cookie	`document.cookie = "key=value; expires=...; path=/";`
Get All Cookies	`document.cookie`
Get Specific Cookie	Use a function to search and return a cookie by name
Delete Cookie	Set it with an expiry date in the past (`1970`)
Useful For	Remembering login, theme, cart, preferences

4.11 jQuery Syntax; jQuery Selectors (Element, Id, Class), jQuery Events (Mouse, Keyboard, Form,

Document/Window) and jQuery Effects (Hide/Show, Fade, Slide, Animate, Stop, Callback, Chaining)

📚 4.11 (Part 1) — jQuery Syntax

✨ What is jQuery?

- **jQuery** is a **JavaScript library** designed to simplify HTML DOM tree traversal and manipulation, event handling, CSS animation, and Ajax.
- It makes **client-side scripting** much **easier and shorter** than vanilla JavaScript.

📜 Basic jQuery Syntax

The **basic syntax** of jQuery is:

```
$(selector).action()
```

Where:

Part	Meaning
`$`	It defines that we are using jQuery.
`selector`	It selects the HTML element(s) we want to work with.
`action()`	It is the jQuery action or method to be performed on the selected element(s).

🛠️ Example of jQuery Syntax

Let's say we want to **hide** a paragraph (`<p>`) when we click a button.

Here's the **HTML**:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("button").click(function(){
       $("p").hide();
     });
   });
  </script>
```

```
</head>
<body>

<h2>My First jQuery Example</h2>

<p>This is a paragraph.</p>
<button>Click me to hide the paragraph</button>

</body>
</html>
```

🔍 Explanation:

- `$(document).ready(function(){ ... });` ensures the jQuery code runs **only after the document is fully loaded**.
- `$("button").click(function(){ ... });` binds a **click event** to the `<button>`.
- When the button is clicked, all `<p>` elements will **hide**.

🧠 Important points about jQuery Syntax:

- Always include **jQuery library** before you use any jQuery code (either download it or use CDN link).
- You should wrap your jQuery code inside `$(document).ready()` to make sure the page is fully loaded before any script runs.
- You use **selectors** to pick HTML elements.
- Then you perform **actions/methods** on the selected elements.

✏️ Practice Problems (with Answers)

Problem 1:

Write a jQuery script to **change the text color of all** `<h1>` **elements to blue** when the page is loaded.

➡️ **Answer:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Change Color Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("h1").css("color", "blue");
   });
  </script>
</head>
```

```
<body>

<h1>Hello, World!</h1>

</body>
</html>
```

Problem 2:

Write a jQuery script to **alert** "Welcome!" when a user clicks on a paragraph.

➡️ **Answer:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Alert Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("p").click(function(){
       alert("Welcome!");
     });
   });
  </script>
</head>
<body>

<p>Click me!</p>

</body>
</html>
```

🧹 Recap Table

Feature	Description	Example
`$`	Symbol to access jQuery functions	`$(selector)`
`selector`	Selects HTML elements	`$("p")` selects all `<p>`
`action()`	jQuery method/action performed on elements	`.hide()`, `.show()`, `.css()`
`$(document).ready()`	Ensures the DOM is fully loaded before running code	`$(document).ready(function(){ ... });`
Including jQuery Library	Must include `<script>` tag linking to jQuery	`<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>`

📚 4.11 (Part 2) — jQuery Selectors (Element, Id, Class)

✨ What are jQuery Selectors?

- **Selectors** in jQuery are used to **find and select HTML elements** based on their **tag name**, **id**, **class**, **attributes**, **types**, and much more.
- Once an element is selected, you can perform actions on it using jQuery methods.

👉 Think of selectors like a **pointer** saying: "**Hey jQuery, work with these elements!**"

🛠️ Types of Selectors We Need to Cover Here:

Type of Selector	Syntax Example	What it Selects
Element Selector	`$("p")`	Selects all `<p>` elements
ID Selector	`$("#idname")`	Selects a single element with the given id
Class Selector	`$(".classname")`	Selects all elements with the given class

📄 1. Element Selector

➡️ **Syntax:**

```
$("elementname")
```

- It selects **all** elements of that type in the page.

➡️ **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Element Selector Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
      $("p").click(function(){
         $(this).hide();
      });
   });
  </script>
</head>
<body>

<p>Paragraph 1 - Click me!</p>
```

```
<p>Paragraph 2 - Click me too!</p>

</body>
</html>
```

🔍 **What Happens?**

- Clicking on **any paragraph** will **hide** that paragraph.

📄 2. ID Selector

➡️ **Syntax:**

```
$("#idname")
```

- ID is **unique** to each element.
- Always use **#** before the id name.

➡️ **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title>ID Selector Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("#special").css("color", "red");
   });
  </script>
</head>
<body>

<h2 id="special">This heading will turn red!</h2>
<h2>This heading will stay normal.</h2>

</body>
</html>
```

🔍 **What Happens?**

- Only the heading with **id="special"** will have **red** color.

📄 3. Class Selector

➡️ **Syntax:**

```
$(".classname")
```

- Class can be used by **multiple elements**.
- Always use **.** before the class name.

➡️ **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Class Selector Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
      $(".highlight").css("background-color", "yellow");
   });
  </script>
</head>
<body>

<p class="highlight">This paragraph will be highlighted!</p>
<p>This paragraph will not.</p>
<p class="highlight">This paragraph will also be highlighted!</p>

</body>
</html>
```

🔍 **What Happens?**

- All elements with the **class="highlight"** will have **yellow background**.

🧠 Quick Table: Difference Between Element, ID, and Class Selectors

Selector Type	Symbol Used	Targets	Can it select multiple elements?	Example
Element	None	All elements of the same tag	✅ Yes	`$("p")`
ID	`#`	One unique element by id	❌ No (id should be unique)	`$("#header")`
Class	`.`	All elements sharing same class	✅ Yes	`$(".menu")`

✏️ Practice Problems

Problem 1:

Select all `<h1>` elements and make their font size **50px**.

➡️ **Answer:**

```
$("h1").css("font-size", "50px");
```

Problem 2:

Select the element with id `banner` and set its background color to **green**.

➡️ **Answer:**

```
$("#banner").css("background-color", "green");
```

Problem 3:

Select all elements with class `note` and change their text to **italic**.

➡️ **Answer:**

```
$(".note").css("font-style", "italic");
```

🧹 Recap Table

Concept	Syntax	Example	Notes
Element Selector	`$("tagname")`	`$("h1")`	Selects all elements of that tag
ID Selector	`$("#idname")`	`$("#logo")`	Selects one unique element
Class Selector	`$(".classname")`	`$(".highlight")`	Selects all elements of that class

📚 4.11 (Part 3) — jQuery Events (Mouse, Keyboard, Form, Document/Window)

✨ What are jQuery Events?

- **Events** are actions that happen in the browser, like **clicking a button**, **moving the mouse**, **typing in a textbox**, **submitting a form**, etc.
- jQuery **makes it very easy** to capture these events and respond to them.

👉 Think of events as **"When this happens, do this"**.

📖 Types of Events We Need to Cover:

Event Type	Common Examples

Mouse Events	click, dblclick, mouseenter, mouseleave
Keyboard Events	keypress, keydown, keyup
Form Events	submit, change, focus, blur
Document/Window Events	load, resize, scroll

📄 1. Mouse Events

Mouse events are triggered when you interact with the mouse.

Event	Description
click	When an element is clicked
dblclick	When an element is double-clicked
mouseenter	When mouse pointer enters the element
mouseleave	When mouse pointer leaves the element

➡️ Example: Mouse Event - `click()`

```
<!DOCTYPE html>
<html>
<head>
  <title>Mouse Click Event</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
      $("#btn").click(function(){
         alert("Button Clicked!");
      });
   });
  </script>
</head>
<body>

<button id="btn">Click me</button>

</body>
</html>
```

🔍 What Happens?

When you click the button, an **alert** pops up.

📄 2. Keyboard Events

These happen when the user interacts with the keyboard.

Event	Description
keypress	When a key is pressed down
keydown	When a key is pressed down (fires earlier than keypress)
keyup	When a key is released

➡️ **Example: Keyboard Event -** `keydown()`

```
<!DOCTYPE html>
<html>
<head>
  <title>Keyboard KeyDown Event</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
      $("#inputbox").keydown(function(){
         $("#msg").text("You pressed a key!");
      });
   });
  </script>
</head>
<body>

<input type="text" id="inputbox" placeholder="Type here...">
<p id="msg"></p>

</body>
</html>
```

🔍 **What Happens?**

Every time you press a key inside the textbox, the message appears.

📄 3. Form Events

Form events occur when users interact with forms.

Event	Description
submit	When a form is submitted
change	When the value of an input changes
focus	When an input field gets focus
blur	When an input field loses focus

➡️ **Example: Form Event -** `submit()`

```html
<!DOCTYPE html>
<html>
<head>
  <title>Form Submit Event</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("#myForm").submit(function(event){
       event.preventDefault(); // prevent real submission
       alert("Form Submitted!");
     });
   });
  </script>
</head>
<body>

<form id="myForm">
  Name: <input type="text" name="name">
  <button type="submit">Submit</button>
</form>

</body>
</html>
```

🔍 **What Happens?**

When you submit the form, an alert is shown instead of actually submitting the form.

📄 4. Document/Window Events

These are triggered when the **page/document** or **browser window** itself changes.

Event	Description
load	When the page is completely loaded
resize	When the browser window is resized
scroll	When the page is being scrolled

➡️ **Example: Window Event -** `resize()`

```html
<!DOCTYPE html>
<html>
<head>
  <title>Window Resize Event</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
```

```
    $(document).ready(function(){
        $(window).resize(function(){
            alert("Window resized!");
        });
    });
  </script>
</head>
<body>

<h2>Resize the window to see the effect!</h2>

</body>
</html>
```

🔍 **What Happens?**

If you resize your browser window, an alert pops up.

🧠 Quick Summary Table: jQuery Events

Event Type	Common Functions	Example
Mouse Events	click(), dblclick(), mouseenter(), mouseleave()	$("#btn").click()
Keyboard Events	keydown(), keypress(), keyup()	$("#input").keydown()
Form Events	submit(), change(), focus(), blur()	$("#form").submit()
Document/Window Events	load(), resize(), scroll()	$(window).resize()

✏️ Practice Problems

Problem 1:

Show an alert when a user hovers the mouse over a `<div>`.

➡️ **Answer:**

```
$("div").mouseenter(function(){
  alert("You hovered over the div!");
});
```

Problem 2:

Change the text color to green when a textbox loses focus.

➡️ **Answer:**

```
$("input").blur(function(){
  $(this).css("color", "green");
```

```
});
```

Problem 3:

Display a message "Form Submitted Successfully" instead of submitting a form.

➡️ **Answer:**

```
$("form").submit(function(event){
   event.preventDefault();
   alert("Form Submitted Successfully!");
});
```

📚 4.11 (Part 4) — jQuery Effects (Hide/Show, Fade, Slide, Animate, Stop, Callback, Chaining)

✨ What are jQuery Effects?

- **Effects** in jQuery are **pre-built visual actions** you can perform on HTML elements.
- They let you create **animations**, **transitions**, and **dynamic behavior** easily.
- Effects improve **user experience** by making the webpage feel more **interactive**.

📖 List of Common jQuery Effects to Learn:

Effect Type	Purpose
Hide/Show	Show or hide elements
Fade	Fade elements in/out (smooth opacity change)
Slide	Slide elements up/down
Animate	Create custom animations
Stop	Stop an ongoing animation
Callback	Run a function after an effect is finished
Chaining	Run multiple effects/methods together

📄 1. Hide and Show

➡️ **Syntax:**

```
$(selector).hide(speed,callback);
$(selector).show(speed,callback);
```

- `speed` → Optional (e.g., `"slow"`, `"fast"`, or milliseconds like `1000`).

- `callback` → Optional function to run after effect finishes.

➡️ **Example: Hide and Show**

```
<!DOCTYPE html>
<html>
<head>
  <title>Hide and Show Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("#hideBtn").click(function(){
       $("p").hide(1000);
     });
     $("#showBtn").click(function(){
       $("p").show(1000);
     });
   });
  </script>
</head>
<body>

<p>This is a paragraph to hide and show.</p>
<button id="hideBtn">Hide</button>
<button id="showBtn">Show</button>

</body>
</html>
```

📄 2. Fade

➡️ **Syntax:**

```
$(selector).fadeIn(speed,callback);
$(selector).fadeOut(speed,callback);
```

➡️ **Example: Fade In and Out**

```
<!DOCTYPE html>
<html>
<head>
  <title>Fade Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
```

```
    $(document).ready(function(){
      $("#fadeOutBtn").click(function(){
        $("div").fadeOut();
      });
      $("#fadeInBtn").click(function(){
        $("div").fadeIn();
      });
    });
  </script>
</head>
<body>

<div style="width:100px;height:100px;background-color:blue;"></div><br>
<button id="fadeOutBtn">Fade Out</button>
<button id="fadeInBtn">Fade In</button>

</body>
</html>
```

📄 3. Slide

➡️ **Syntax:**

```
$(selector).slideUp(speed,callback);
$(selector).slideDown(speed,callback);
```

➡️ **Example: Slide Up and Down**

```
<!DOCTYPE html>
<html>
<head>
  <title>Slide Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function(){
      $("#slideUpBtn").click(function(){
        $("div").slideUp();
      });
      $("#slideDownBtn").click(function(){
        $("div").slideDown();
      });
    });
  </script>
</head>
```

```
<body>

<div style="width:100px;height:100px;background-color:red;"></div><br>
<button id="slideUpBtn">Slide Up</button>
<button id="slideDownBtn">Slide Down</button>

</body>
</html>
```

📄 4. Animate

- `animate()` lets you **create custom animations** by changing CSS properties.

➡️ **Syntax:**

```
$(selector).animate({params},speed,callback);
```

➡️ **Example: Animate**

```
<!DOCTYPE html>
<html>
<head>
  <title>Animate Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("#animateBtn").click(function(){
       $("div").animate({
         left: '250px',
         opacity: '0.5',
         height: '150px',
         width: '150px'
       });
     });
   });
  </script>
  <style>
   div {
     width: 100px;
     height: 100px;
     background: green;
     position: relative;
   }
  </style>
```

```
</head>
<body>

<div></div><br>
<button id="animateBtn">Animate Div</button>

</body>
</html>
```

📄 5. Stop

- Use `stop()` to **immediately stop** a running animation or effect.

➡️ Example: Stop Animation

```
<!DOCTYPE html>
<html>
<head>
  <title>Stop Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("#startBtn").click(function(){
       $("div").slideDown(5000);
     });
     $("#stopBtn").click(function(){
       $("div").stop();
     });
   });
  </script>
</head>
<body>

<div style="width:100px;height:100px;background-color:purple;display:none;"></div><br>
<button id="startBtn">Start Slide</button>
<button id="stopBtn">Stop Slide</button>

</body>
</html>
```

📄 6. Callback

- A **callback function** is executed **after** the current effect is finished.

➡️ **Example: Hide with Callback**

```
<!DOCTYPE html>
<html>
<head>
  <title>Callback Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("button").click(function(){
       $("p").hide("slow", function(){
         alert("The paragraph is now hidden");
       });
     });
   });
  </script>
</head>
<body>

<p>This paragraph will hide slowly and then alert!</p>
<button>Hide and Alert</button>

</body>
</html>
```

📄 7. Chaining

- You can **chain multiple actions** together in one line!

➡️ **Example: Chaining**

```
<!DOCTYPE html>
<html>
<head>
  <title>Chaining Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
   $(document).ready(function(){
     $("button").click(function(){
       $("p").css("color", "red").slideUp(2000).slideDown(2000);
     });
   });
  </script>
</head>
```

```
<body>

<p>Watch me chain!</p>
<button>Start Chain</button>

</body>
</html>
```

🧠 Quick Summary Table: jQuery Effects

Effect	Purpose	Example
hide() / show()	Hide or show elements	`$("p").hide()`
fadeIn() / fadeOut()	Fade elements in/out	`$("div").fadeOut()`
slideUp() / slideDown()	Slide elements up or down	`$("div").slideUp()`
animate()	Custom animation by changing CSS	`$("div").animate({...})`
stop()	Stop an animation immediately	`$("div").stop()`
callback	Perform an action after another finishes	`"hide" then alert()`
chaining	Chain multiple methods	`$("p").slideUp().slideDown()`

✏️ Practice Problems

Problem 1:

Fade out a paragraph slowly when a button is clicked.

➡️ **Answer:**

```
$("button").click(function(){
  $("p").fadeOut("slow");
});
```

Problem 2:

Slide up a div over 3 seconds and after sliding up, display an alert.

➡️ **Answer:**

```
$("div").slideUp(3000, function(){
  alert("Div is slid up!");
});
```

Problem 3:

Change the background color of a div to yellow and then slide it down.

➡️ **Answer:**

```
$("div").css("background-color", "yellow").slideDown();
```

📚 4.12 — Introduction to JSON (Syntax, Data Types, Parsing JSON)

✨ What is JSON?

- **JSON** stands for **JavaScript Object Notation**.
- It is a **lightweight** data-interchange format that is **easy to read** and **easy for machines to parse and generate**.
- JSON is often used to **exchange data between a server and a web page**.

👉 Think of JSON like **a structured text** used to **represent objects or data**.

📖 Key Features of JSON

Feature	Description
Lightweight	Minimal syntax and easy to read
Text Format	Stored as plain text (like `.txt` or `.json`)
Language Independent	Can be used with almost any programming language
Structured	Data is organized into key-value pairs and arrays

🛠️ JSON Syntax Rules

✅ JSON is written in **key/value pairs**:

```
"key": "value"
```

✅ JSON is **always enclosed** inside **curly braces** `{}` when it represents an object.

✅ **Keys** must be **strings** enclosed in **double quotes** `" "`.

✅ **Values** can be:

- String
- Number
- Object (another JSON object)
- Array
- Boolean

- null

✅ JSON data is **separated by commas**.

📄 Example of a Simple JSON Object:

```
{
  "name": "Alice",
  "age": 25,
  "city": "New York"
}
```

- `name`, `age`, and `city` are **keys**.
- `"Alice"`, `25`, and `"New York"` are **values**.

📖 JSON Data Types

Data Type	Example
String	`"name": "Alice"`
Number	`"age": 25`
Object (Nested JSON)	`"address": {"city":"New York", "zip":"10001"}`
Array	`"hobbies": ["reading", "gaming", "traveling"]`
Boolean	`"isStudent": false`
null	`"middleName": null`

📄 Example with All Data Types

```
{
  "name": "Bob",
  "age": 30,
  "isStudent": false,
  "skills": ["HTML", "CSS", "JavaScript"],
  "address": {
    "city": "Los Angeles",
    "zipcode": "90001"
  },
  "middleName": null
}
```

🔥 JSON vs JavaScript Object

They **look similar**, but **in JSON**:

- Keys **must** be strings (with double quotes).
- No functions allowed (only pure data).

Feature	JSON	JavaScript Object
Quotes	Keys must have double quotes	Keys may not need quotes
Functions	Not allowed	Allowed
Comments	Not allowed	Allowed

🔍 Parsing JSON

- **Parsing** means **reading** JSON data and **converting it** into a JavaScript object.
- In JavaScript, use `JSON.parse()` to convert JSON string → JavaScript object.

➡️ Example: Parsing JSON

```
<!DOCTYPE html>
<html>
<head>
  <title>Parsing JSON Example</title>
  <script>
    var jsonString = '{"name":"Charlie","age":28,"city":"Boston"}';

    var obj = JSON.parse(jsonString);  // Parsing JSON

    console.log(obj.name);  // Output: Charlie
    console.log(obj.age);   // Output: 28
  </script>
</head>
<body>

<h2>Check console for output!</h2>

</body>
</html>
```

🔍 Stringify JavaScript Object to JSON

- You can also **convert** a JavaScript object **into a JSON string** using `JSON.stringify()`.

➡️ Example: Stringify JavaScript Object

```html
<!DOCTYPE html>
<html>
<head>
  <title>Stringify Example</title>
  <script>
    var obj = { name: "Dave", age: 35, city: "Chicago" };

    var jsonString = JSON.stringify(obj); // Converts object to JSON string

    console.log(jsonString);
    // Output: {"name":"Dave","age":35,"city":"Chicago"}
  </script>
</head>
<body>

<h2>Check console for output!</h2>

</body>
</html>
```

🧠 Quick Summary Table: JSON Essentials

Topic	Description	Example
JSON Definition	Lightweight text-based data format	`{ "name": "Alice" }`
JSON Data Types	String, Number, Object, Array, Boolean, null	`"skills": ["HTML", "CSS"]`
JSON Parsing	Convert JSON string to JS Object	`JSON.parse()`
JSON Stringify	Convert JS Object to JSON string	`JSON.stringify()`
Quotes	Keys **must** have double quotes	`"name": "value"`

✏️ Practice Problems

Problem 1:

Write a JSON object to store a person's name, age, and hobbies.

➡️ **Answer:**

```json
{
  "name": "Eve",
  "age": 22,
  "hobbies": ["painting", "cycling", "swimming"]
}
```

Problem 2:

Parse the following JSON string into an object and display the city:

```
var data = '{"name":"John","city":"London"}';
```

➡️ **Answer:**

```
var obj = JSON.parse(data);
console.log(obj.city); // Output: London
```

Problem 3:

Convert the following JavaScript object into JSON string:

```
var product = {name:"Laptop", price:800};
```

➡️ **Answer:**

```
var jsonString = JSON.stringify(product);
console.log(jsonString);
// Output: {"name":"Laptop","price":800}
```
