VETRI TECHNOLOGY SOLUTIONS

100 % IT TRAINING WITH PLACEMENTS

JAVASCRIPT LEARNING MODULES

Day:1

What is JavaScript?

JavaScript is a client-side scripting language that allows developers to create dynamic web applications. It can:

- ✅ Change HTML content dynamically
- ✅ Modify CSS styles in real-time
- ✅ Handle user events (click, input, hover)
- ✅ Fetch and process data from APIs

Basic Structure of JavaScript

JavaScript can be written in three ways:

1. Inside HTML using <script> tag
2. Inside an external JavaScript file (.js)
3. Inline JavaScript (inside an HTML element's attribute)

1. JavaScript Inside HTML (Using <script> Tag)

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <title>JavaScript Example</title>

</head>

<body
```

```
  <h1>Welcome to JavaScript!</h1>

  <p id="demo">JavaScript will change this text.</p>


  <script>

    document.getElementById("demo").innerText = "Hello, JavaScript is
Working!";

  </script>


</body>

</html>
```

Output:

The <p> tag’s text will change dynamically when the page loads.

2. External JavaScript File (.js)

Best practice is to write JavaScript in an external file for better maintainability.

Step 1: Write HTML File (index.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>External JavaScript</title>

  <script src="script.js" defer></script> <!-- Link JS File -->
```

```
</head>
<body>


   <h1>Click the Button</h1>
   <p id="message">This text will change.</p>
   <button onclick="changeText()">Click Me</button>


</body>
</html>
```

Step 2: Create JavaScript File (script.js)

```
function changeText() {
   document.getElementById("message").innerText = "You clicked the button!
JavaScript works!";
}
```

Why use an external file?

✅ Separation of Concerns (HTML, CSS, and JS are separate)

✅ Reusable Code (Can be linked to multiple pages)

3. Inline JavaScript (Not Recommended)

You can add JavaScript **directly inside an HTML tag** but it's not a good practice.

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

📌 Why avoid inline JavaScript?

❌ Hard to maintain & update

❌ Difficult to debug

Why Learn JavaScript?

✅ Used in websites, web apps, games, chat apps, eCommerce
✅ Can be used for frontend & backend (Node.js)
✅ Essential for Full-Stack Development

JavaScript Keywords (Tabular Format)

JavaScript has reserved keywords that have special meanings and cannot be used as variable names.

JavaScript Keywords Table :

Category	Keywords
Variable Declaration	var, let, const
Data Types	string, number, boolean, null, undefined, symbol, bigint, object, function
Control Flow	if, else, switch, case, default
Loops	for, while, do, break, continue
Functions	function, return, yield, async, await
Class & Object	class, constructor, extends, super, this, new
Exception Handling	try, catch, finally, throw
Boolean & Comparison	true, false, instanceof, typeof, in
Scope & Modules	import, export, default
Memory Management	delete, void

Asynchronous JavaScript	async, await, Promise
Error Handling	try, catch, throw, finally
Special Statements	debugger, with, void
Strict Mode	use strict

Explanation of Important Keywords :

Keyword	Description	Example
var	Declares a variable (old method)	var name = "Alice";
let	Declares a block-scoped variable	let age = 25;
const	Declares a constant variable	const country = "India";
if	Executes code if condition is true	if (age > 18) { ... }
else	Executes alternative code if if condition is false	else { ... }
switch	Allows multiple condition checks	switch(value) { case 1: ... }
for	Loop that runs a fixed number of times	for (let i=0; i<10; i++) { ... }
while	Loop that runs while condition is true	while (condition) { ... }
function	Declares a function	function greet() { return "Hello"; }
return	Returns a value from a function	return x + y;
try	Defines a block to test for errors	try { ... } catch (e) { ... }
catch	Catches errors in try block	catch(error) { console.log(error); }
throw	Throws a custom error	throw "Invalid input";

class	Declares a class (ES6)	class Car { constructor() { } }
this	Refers to the current object	this.name = "Alice";
new	Creates an instance of an object	let obj = new Object();
import	Imports modules (ES6)	import myModule from './file.js';
export	Exports functions or variables (ES6)	export function myFunc() {}

JavaScript provides different methods to display output in the console and on the webpage. Below, we will use:

✅ console.log() → Prints data in the console
✅ console.table() → Displays data in a table format
✅ console.error() → Shows an error message in red
✅ console.warn() → Shows a warning message in yellow
✅ document.write() → Writes directly to the HTML page

Code Example

HTML File (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>JavaScript Console and Document Write</title>
</head>
<body>

   <h1>Console and Document Write Example</h1>

   <script>
```

```
        // Using console.log()
        console.log("vts");
        console.log(3456);
        console.log(12.09);
        console.log(true);
        console.log([1,6,4,77,6]);
        console.log({institute: "vts", location: "Surandai"});

        // Display data in tabular format in the console
        console.table({institute: "vts", location: "Surandai"});

        // Error and Warning messages in the console
        console.error("This is a sample error message!");
        console.warn("This is a warning message!");

        // Using document.write() to display content on the webpage
        document.write("<h2>Document Write Output</h2>");
        document.write("<p>Institute: vts</p>");
        document.write("<p>Location: Surandai</p>");
        document.write("<p>Number: " + 3456 + "</p>");
        document.write("<p>Decimal Number: " + 12.09 + "</p>");
        document.write("<p>Boolean Value: " + true + "</p>");
        document.write("<p>Array: " + [1,6,4,77,6] + "</p>");
    </script>

</body>
</html>
```

Output:

Console Output (Press F12 → Go to Console Tab)

```
vts
3456
12.09
true
(5) [1, 6, 4, 77, 6]
{institute: "vts", location: "Surandai"}

(Displayed as Table)
| (index)   | Value   |
|-----------|--------|
| institute | "vts"   |
| location  | "Surandai" |

❌ This is a sample error message!
⚠ This is a warning message!
```

Web Page Output (Rendered by document.write())

```
Institute: vts
Location: Surandai
Number: 3456
Decimal Number: 12.09
Boolean Value: true
Array: 1,6,4,77,6
```

1. Variables in JavaScript (var, let, const)

Variables **store data** and allow it to be used later.
JavaScript provides **three ways** to declare variables:

Keyword	Scope	Re-declaration	Reassignment
var	Function scope	✓ Allowed	✓ Allowed
let	Block scope	✗ Not Allowed	✓ Allowed
const	Block scope	✗ Not Allowed	✗ Not Allowed

Example: **var**, **let**, **const**

```
var a = 10;      // Function scoped, can be re-declared
let b = 20;      // Block scoped, can be reassigned
const c = 30;    // Block scoped, cannot be changed

console.log(a, b, c); // Output: 10 20 30

// Reassignment
b = 50;
console.log(b); // Output: 50

// const cannot be reassigned
// c = 60; ✗ Error: Assignment to constant variable
```

✓ **Use let** for variables that change.
✓ **Use const** for constants (values that don't change).
✓ **Avoid var** (it has function scope & can lead to bugs).

2. Primitive Data Types

Primitive data types store **simple values** and are **immutable**.

Data Type	**Example**	**Description**
String	"Hello"	Text data
Number	25, 3.14	Integer & Float values
Boolean	true, false	Logical values
Null	null	Represents an **empty value**
Undefined	undefined	Variable is declared but not assigned
Symbol	Symbol('id')	Unique identifier (used in Objects)
BigInt	BigInt(12345678901234567890)	Large numbers

Example: Primitive Data Types

```
let name = "Alice";      // String
let age = 25;            // Number
let isStudent = true;    // Boolean
let balance = null;      // Null (empty value)
let city;                // Undefined (not assigned)
let uniqueId = Symbol("id");  // Symbol
let bigNumber = BigInt(12345678901234567890); // BigInt

console.log(typeof name);       // "string"
console.log(typeof age);        // "number"
console.log(typeof isStudent);  // "boolean"
console.log(typeof balance);    // "object" (special case for null)
console.log(typeof city);       // "undefined"
console.log(typeof uniqueId);   // "symbol"
console.log(typeof bigNumber);  // "bigint"
```

3. Reference Data Types (Objects, Arrays, Functions)

Reference data types store **complex values** and are **mutable**.

Data Type	Example	Description
Array	["Apple", "Banana", "Cherry"]	Ordered list of values
Object	{ name: "Alice", age: 25 }	Key-value pairs
Function	function sayHello() {}	Block of reusable code

Example: Reference Data Types

```
// Array (Stores multiple values)
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits[1]); // "Banana"

// Object (Key-Value Pairs)
let user = {
   name: "Alice",
   age: 25,
   city: "New York"
};
console.log(user.name); // "Alice"

// Function (Reusable Code Block)
function greet(name) {
   return "Hello, " + name;
}
console.log(greet("Alice")); // "Hello, Alice"
```

Key Differences:

- ✅ Primitive data types store values directly.
- ✅ Reference data types store memory addresses (mutable).

4. Type Conversion (Converting Between Types)

JavaScript allows **automatic and manual type conversion**.

Conversion	Method	Example	Output
String to Number	parseInt(), parseFloat(), Number()	parseInt("100")	100
Number to String	.toString(), String()	(50).toString()	"50"
Boolean to String	String(true)	String(false)	"true"
String to Boolean	Boolean("Hello")	Boolean("")	true, false

Example: Type Conversion

```
// String to Number
console.log(parseInt("100"));  // 100
console.log(parseFloat("3.14")); // 3.14

// Number to String
console.log((50).toString());  // "50"

// Boolean to String
console.log(String(true));  // "true"

// String to Boolean
```

```
console.log(Boolean("Hello")); // true
console.log(Boolean(""));      // false
```

5. Checking Data Types (typeof, instanceof**)**

✅ **typeof** → Checks primitive data types.
✅ **instanceof** → Checks reference data types.

Example: **typeof** & **instanceof**

```
// Checking primitive data types
console.log(typeof "Hello");  // "string"
console.log(typeof 42);       // "number"
console.log(typeof true);     // "boolean"
console.log(typeof null);     // "object" (special case)
console.log(typeof undefined); // "undefined"

// Checking reference data types
let numbers = [1, 2, 3];
console.log(numbers instanceof Array); // true

let person = { name: "John", age: 30 };
console.log(person instanceof Object); // true
```

Sample Mini Project 1:

1. **E-Commerce Shopping Cart**

```
// Variable declarations using let and const
const storeName = "TechShop"; // String
let productName = "Wireless Mouse"; // String
let price = 29.99; // Number (float)
let inStock = true; // Boolean
let discount = null; // Null (No discount available)
let category; // Undefined (Not assigned yet)

// Display Product Details
console.log("Store:", storeName);
console.log("Product:", productName);
console.log("Price: $" + price);
console.log("Available:", inStock);
console.log("Discount:", discount);
console.log("Category:", category); // undefined
```

Real-World Usage: Used in **Amazon, Flipkart, Shopify** to store and display product details.

Sample Mini project 2: Online Food Ordering System

Requirements :

1. **Store and Manage Order Details:**

- The system should store restaurant name, order number, list of items, total price, and delivery status using appropriate data types (String, Number, Array, Boolean).

2. **Display Order Summary:**

- The program should print all order details to the console, ensuring a structured and readable format for users.

3. **Update Order Status:**

- The isDelivered status should be modifiable to reflect real-time order tracking (e.g., change false to true when the order is delivered).

4. **Support Multiple Items:**

- The items array should allow adding and removing items, ensuring flexibility for different types of food orders.

DAY 1 Task

JS1. Write a JavaScript program to display "Hello, JavaScript!" using console.log().
JS2. Create a variable using var, let, and const and print their values.
JS3. Demonstrate the difference between var, let, and const in terms of scope.
JS4. Check the type of a variable using typeof for different data types.
JS5. Create a user profile object containing name, age, and is Student properties.
JS6.Store and display a list of favorite colors using an array.
JS7. Perform type conversion: Convert a string "100" to a number and a number 100 to a string.
JS8. Write a function that returns a Boolean value based on a condition.
JS9. Demonstrate null vs undefined by creating two different variables and checking their types.
JS10. Use Symbol data type to create a unique ID and compare two symbols.
JS11. Use BigInt to store a large number and print its type using typeof.
JS12. Write a function that accepts a number as an argument and returns "Even" or "Odd".
JS13. Create a temperature conversion program that converts Celsius to Fahrenheit using a variable.

Mini project

1. Online Food Ordering System

Requirements:

1. **Store and Manage Order Details:**

- The system should store restaurant name, order number, food items, total price, and delivery status using appropriate data types (String, Number, Array, Boolean).

2. **Display Order Summary:**

- The program should print all order details in a structured format for users.

3. **Update Order Status:**

- The isDelivered status should be modifiable to track real-time delivery (false → true when delivered).

4. **Support Multiple Items:**

- The items array should allow adding and removing food items, ensuring flexibility for different types of food orders.

Real-World Usage: Zomato, Swiggy, Uber Eats.

2. Online Quiz System

Scenario: A quiz application records user answers and scores.

Requirements:

1. **Store Quiz Details:**

- The system should store **question, multiple-choice options, correct answer, and user-selected answer** using variables and arrays.

2. **Validate User Answer:**

- Compare userAnswer with correctAnswer and return a **Boolean result (true or false)**.

3. **Display Quiz Results:**

- Print the **question, options, user's answer, and correctness status** in a readable format.

4. **Extend for Multiple Questions:**

- The system should be able to **store multiple questions and track scores dynamically**.

Real-World Usage: Google Forms, Kahoot, Online Exams.

Day 2:

Operators in JavaScript

What is an Operator?

An operator is a symbol that performs operations on variables or values. Operators are used in mathematical calculations, logical conditions, comparisons, and data manipulation in real-world applications.

Real-Time Usage of Operators:
✓ Arithmetic Operators: Used in calculating cart totals in e-commerce sites.
✓ Comparison Operators: Used in validating user age in signup forms.
✓ Logical Operators: Used in user authentication systems.

Types of Operators

1. Arithmetic Operators

Used to perform **mathematical calculations**.

Operator	Description	Example
+	Addition	5 + 3 → 8
-	Subtraction	10 - 4 → 6
*****	Multiplication	6 * 2 → 12
/	Division	20 / 4 → 5
%	Modulus (Remainder)	10 % 3 → 1
******	Exponentiation	2 ** 3 → 8

Example Code:

```
let a = 10, b = 3;
console.log(a + b); // 13
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.33
console.log(a % b); // 1
console.log(2 ** 3); // 8
```

Real-World Use: Used in calculating order totals in shopping carts.

2. Assignment Operators

Used to **assign values** to variables.

Operator	Example	Equivalent To
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 2	x = x - 2
***=**	x *= 4	x = x * 4
/=	x /= 2	x = x / 2

%=	x %= 3	x = x % 3

Example Code:

```
let x = 10;
x += 5;  // x = x + 5
console.log(x); // 15
```

Real-World Use: Used in wallet balance updates in payment apps.

3. Comparison Operators

Used to **compare values** and return true or false.

Operator	Description	Example
==	Equal to	5 == "5" → true
===	Strict equal	5 === "5" → false
!=	Not equal	5 != 3 → true
!==	Strict not equal	5 !== "5" → true
>	Greater than	10 > 5 → true
<	Less than	3 < 7 → true
>=	Greater or equal	8 >= 8 → true
<=	Less or equal	5 <= 6 → true

Example Code:

```
console.log(5 == "5"); // true
console.log(5 === "5"); // false
console.log(10 > 3); // true
```

Real-World Use: Used in checking age restrictions in movie ticket booking.

4. Logical Operators

Used to perform **logical operations**.

Operator	Description	Example
&&	AND	(true && false) → false
`		`
!	NOT	!true → false

Example Code:

```
let age = 20;
let hasID = true;
console.log(age > 18 && hasID); // true
```

Real-World Use: Used in login authentication systems.

5. Increment & Decrement Operators

Operator	Example	Description
++	x++	Post-increment (use, then increase)
++	++x	Pre-increment (increase, then use)
--	x--	Post-decrement (use, then decrease)
--	--x	Pre-decrement (decrease, then use)

Example Code:

```
let num = 5;
console.log(num++); // 5 (then increases to 6)
console.log(++num); // 7 (increases before printing)
```

Real-World Use: Used in **stock decrement in inventory management**.

6. Ternary Operator

Short form of if-else.

Example Code:

```
let age = 18;
let canVote = age >= 18 ? "Eligible" : "Not Eligible";
console.log(canVote); // "Eligible"
```

Real-World Use: Used in **displaying discount messages** based on cart total.

Sample 1: Mini Project: Discount Calculator (E-Commerce Website)

```
let cartTotal = 120;  // Total price in shopping cart
let discount = cartTotal > 100 ? cartTotal * 0.10 : 0; // 10% discount if total > 100
let finalPrice = cartTotal - discount; // Final price after discount

console.log("Cart Total: $" + cartTotal);
console.log("Discount Applied: $" + discount);
console.log("Final Price to Pay: $" + finalPrice);
```

Sample 2: Mini Project: Age Verification System (Signup Page)

Scenario:
A website **checks if a user is 18 or older** before allowing them to sign up.

Requirements:

1. User's age should be stored in a variable (userAge).
2. The system should check if the user is 18 or older using the comparison operator (>=).
3. If the user is eligible, display "✅ Access Granted", otherwise "❌ Access

Denied".
4. The output should be displayed in the console.

Day 2 Tasks:

JS14. Write a program that calculates the total price of 3 products.
JS15. Create a script that checks if a number is even or odd using the modulus operator.
JS16. Write a program that increments a number from 1 to 10 using ++.
JS17. Check if a user is eligible to vote using comparison operators.
JS18. Write a program to compare two strings using == and ===.
JS19. Use && to check if a user has a valid email and password.
JS20. Write a program to update wallet balance using assignment operators.
JS21. Use a ternary operator to check if a product is available in stock.
JS22. Create a simple "greater than" condition to compare two numbers.
JS23. Write a JavaScript program to apply discounts if a cart total exceeds $50.
JS24. Use || to allow login using email or phone number.
JS25. Implement a "Buy 1 Get 1 Free" condition using if and &&.
JS26. Check if a year is a leap year using the modulus operator.

Mini Project:

1. **Voting Eligibility System**

Scenario:
A government website checks if a user is eligible to vote based on age and citizenship.

Requirements:

1. **User's country and age should be stored** in variables (citizen and age).
2. **The system should check both conditions**:

- The user **must be 18 or older** (age >= 18).
- The user **must be a citizen of the country** (citizen === "India").

  3. If both conditions are met, print "✅ Eligible to Vote", else "❌ Not Eligible to Vote".

  4. **The output should be displayed** in the console.

2. Temperature Converter (Weather App)

Scenario:
A weather app converts temperature from Celsius to Fahrenheit.

Requirements:

1. **Store the temperature in Celsius** in a variable (celsius).
2. **Use the conversion formula**:

- fahrenheit = (celsius * 9/5) + 32

  3. **The converted Fahrenheit value should be stored** in a variable (fahrenheit).

  4. **The result should be displayed** in the console.

Day 3:

Conditional Statements in JavaScript

What is a Conditional Statement?

A conditional statement in JavaScript is used to perform different actions based on different conditions. These conditions allow a program to make decisions and execute specific blocks of code based on whether a condition is true or false.

Types of Conditional Statements in JavaScript

1. if Statement
2. if-else Statement
3. if-else if-else Statement
4. Nested if Statement
5. Switch Statement
6. Ternary Operator (? :)
7. Logical Operators in Conditions (&&, ||, !)

1. if Statement

The if statement executes a block of code only if the condition is true.

Real-World Example:
If the temperature is **above 30°C**, display **"It's a hot day!"**.

```
let temperature = 35;
if (temperature > 30) {
   console.log("It's a hot day!");
}
```

2. if-else Statement

The if-else statement allows executing one block of code if the condition is true and another block if the condition is false.

Real-World Example:
Checking if a person is **eligible to vote** (age >= 18).

```
let age = 17;
if (age >= 18) {
   console.log("You are eligible to vote.");
} else {
    console.log("You are not eligible to vote.");
}
```

3. if-else if-else Statement

This is used when there are multiple conditions to check.

Real-World Example:
Determining the grade of a student based on their marks.

```
let marks = 85;
if (marks >= 90) {
    console.log("Grade: A");
} else if (marks >= 75) {
    console.log("Grade: B");
} else if (marks >= 50) {
    console.log("Grade: C");
} else {
    console.log("Grade: F");
}
```

4. Nested if Statement

An if statement inside another if statement.

Real-World Example:
Checking if a **user is logged in and has admin privileges**.

```
let isLoggedIn = true;
let isAdmin = false;
if (isLoggedIn) {
  if (isAdmin) {
    console.log("Welcome, Admin!");
  } else {
    console.log("Welcome, User!");
  }
} else {
  console.log("Please log in.");
}
```

5. Switch Statement

The switch statement is used to perform different actions based on different cases.

Real-World Example:
Checking the day of the week.

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

7. Logical Operators in Conditions

1. && (AND) – Both conditions must be true.
2. || (OR) – At least one condition must be true.
3. ! (NOT) – Reverses the boolean value.

Real-World Example:
Checking if a user **is logged in and is an admin**.

```
let isLoggedIn = true;
let isAdmin = true;

if (isLoggedIn && isAdmin) {
   console.log("Access granted to Admin Panel.");
} else {
   console.log("Access denied.");
}
```

Sample 1. Number Guessing Game

Randomly generate a number and let the user guess.

```
let secretNumber = Math.floor(Math.random() * 10) + 1;
let guess = parseInt(prompt("Guess a number between 1 and 10:"));

if (guess === secretNumber) {
   console.log("You guessed it right!");
} else {
   console.log("Wrong guess. Try again!");
}
```

Sample 2. Age Eligibility Checker

Requirements:

Input: User enters their **age** through a prompt.
Conditions to check:

- If age >= 18 → Eligible to vote.
- If age >= 21 → Can drink alcohol.
- If age >= 25 → Can rent a car.
- If age < 18 → Display "You are not eligible for these activities."
  Output:
  Display eligibility messages in the console.

Day 3 Tasks

JS27. Write an if condition to check if a number is positive.

JS28. Use if-else to check if a person is an adult.

JS29. Check if a number is even or odd using if-else.

JS30. Use if-else if-else to check grade categories.

JS31. Check if a person is eligible for a driver's license.

JS32. Create a nested if condition to check both login and admin status.

JS33. Use a switch statement to print messages for different days.

JS34. Use a ternary operator to check if a person is eligible for a senior citizen discount.

JS35. Check if a user is logged in using &&.

JS36. Use || to check if a username or email is provided.

JS37. Use ! to negate a boolean variable.

JS38. Check if the current year is a leap year.

JS39. Validate a password length using an if statement.

Mini Project

1. Simple Login System

Requirements:

1. User enters username and password via prompt.
   ✅ Conditions to check:
2. If username and password match stored values → Login successful.
3. If either username or password is incorrect → Invalid credentials!
   ✅ Output: Display a login success or failure message in the console.

2. Traffic Light System

Requirements:

- User enters a traffic light color (red, yellow, green) via prompt.
  ✅ Conditions to check:
- "red" → Display "Stop!"
- "yellow" → Display "Slow down!"
- "green" → Display "Go!"
- If the input is not one of the three colors → Display "Invalid color!"
  ✅ Output: Display the traffic rule based on the entered color.

Day 4:

Looping in JavaScript

Loops are used to execute a block of code multiple times until a specific condition is met. JavaScript provides different types of loops to handle various scenarios efficiently.

Types of Loops in JavaScript

1. **for Loop**
2. **while Loop**
3. **do-while Loop**
4. **for...in Loop (Used for objects)**
5. **for...of Loop (Used for arrays, strings, etc.)**
6. **break and continue Statements (Loop control)**

1. for Loop

The for loop is used when we know how many times we want to execute a block of code.

Real-World Example:
Printing numbers from 1 to 5 (Loop runs 5 times).

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

2. while Loop

The while loop runs as long as the condition is true.

Real-World Example:
A countdown timer (Loop runs until it reaches 0).

```
let count = 5;
while (count > 0) {
  console.log("Countdown:", count);
  count--;
}
```

3. do-while Loop

The do-while loop executes the code at least once, even if the condition is false.

Example:
Prompt user until they enter a valid number.

```
let num;
do {
  num = prompt("Enter a number greater than 10:");
} while (num <= 10);
console.log("Valid number entered:", num);
```

4. for...in Loop (Objects)

The for...in loop is used to iterate over object properties.

Example:
Iterating over a student object and printing details.

```
let student = { name: "John", age: 20, course: "JavaScript" };

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

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

The for...of loop is used to iterate over arrays, strings, and other iterable objects.

Example:
Iterating over an array of fruits.

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

for (let fruit of fruits) {
   console.log(fruit);
}
```

6. break and continue Statements

- break stops the loop immediately.
- continue skips the current iteration and moves to the next.

Example:
Find the first even number in an array (using break).

```
let numbers = [3, 7, 8, 5, 2];

for (let num of numbers) {
  if (num % 2 === 0) {
    console.log("First even number:", num);
    break;
  }
}
```

Skip printing negative numbers (using continue).

```
let nums = [5, -2, 10, -7, 15];

for (let num of nums) {
  if (num < 0) continue;
  console.log("Positive number:", num);
}
```

Sample Mini Project 1 : ATM Cash Withdrawal System.

```
let balance = 5000;

while (true) {
  let amount = prompt(`Your balance: $${balance}\nEnter withdrawal amount or
type "exit" to stop:`);

  if (amount.toLowerCase() === "exit") {
    console.log("Thank you for using the ATM. Have a great day!");
    break;
  }

  amount = parseFloat(amount);

  if (isNaN(amount) || amount <= 0) {
        console.log("Invalid amount! Please enter a valid number.");
        continue;
    }

    if (amount > balance) {
        console.log("Insufficient funds! Enter a lower amount.");
        continue;
    }

    balance -= amount;
    console.log(`Withdrawal successful! Remaining balance: $${balance}`);
}
```

Example Output:

```
Your balance: $5000
Enter withdrawal amount or type "exit" to stop: 1500
Withdrawal successful! Remaining balance: $3500

Your balance: $3500
Enter withdrawal amount or type "exit" to stop: 4000
Insufficient funds! Enter a lower amount.

Your balance: $3500
Enter withdrawal amount or type "exit" to stop: 500
Withdrawal successful! Remaining balance: $3000

Your balance: $3000
Enter withdrawal amount or type "exit" to stop: exit
Thank you for using the ATM. Have a great day!
```

Sample 2 :Shopping Cart System

Allows users to add/remove items and view the total cost in a shopping cart.

Requirements:

- User can **add items** to the cart.
- Each item has a **name and price**.
- User can **view cart items and total price**.
- User can **remove an item** from the cart.
- User can type "exit" to stop shopping.

Day 4 Tasks:

JS40. Print numbers from 1 to 10 using a for loop.

JS41. Print even numbers from 2 to 20 using a for loop.

JS42. Print numbers from 10 to 1 using a while loop.

JS43. Print "Hello World!" 5 times using a do-while loop.

JS44. Iterate over an array and print each element using a for...of loop.

JS45. Iterate over an object and print its properties using a for...in loop.

JS46. Sum numbers from 1 to 10 using a for loop.

JS47. Find the first number greater than 50 in an array using break.

JS48. Skip printing numbers divisible by 3 using continue.

JS49. Create a multiplication table for 5 using a for loop.

JS50. Reverse a string using a loop.

JS51. Count the number of vowels in a string using a loop.

JS52. Find the largest number in an array using a loop.

Mini project:

1. Multiplication Table Generator

1. User Input: The program should prompt the user to enter a number.
2. Loop Execution: Use a for loop to iterate from 1 to 10.
3. Multiplication Calculation: Multiply the input number with the loop counter.
4. Output Display: Print the multiplication result in a formatted way (num x i = result).

2. Sum of Digits of a Number

1. User Input: The program should prompt the user to enter a positive number.
2. Digit Extraction: Use the modulus operator % to extract the last digit.
3. Summation Process: Use a while loop to repeatedly sum the digits until the number becomes 0.
4. Output Display: Print the sum of the digits after the loop ends.

Day 5:

Functions in JavaScript

What is a Function?

A **function** is a block of reusable code designed to perform a specific task. It makes the code more modular, readable, and easier to maintain.

✅ **Key Features of Functions**:

- Functions **can take input** (parameters).
- Functions **can return** a value.
- Functions **can be called multiple times**.

Function Components

1. **Function Declaration**

A function is declared using the function keyword.

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

2. **Function Parameters and Arguments**

- **Parameters** → Variables defined in the function definition.
- **Arguments** → Actual values passed to the function when calling it.

🔷 **Real-World Example:**
✅ Function to calculate the area of a rectangle.

```
function calculateArea(length, width) { // Parameters
   let area = length * width;
   console.log("Area of Rectangle:", area);
}
calculateArea(10, 5); // Arguments
```

3. **Function with Return Value**

A function can return a value using the return statement.

◈ **Real-World Example:**

✓ Function to return the square of a number.

```
function square(num) {
   return num * num; // Returning the square
}

let result = square(4);
console.log("Square:", result);
```

4. **Function Expression**

Functions can be stored in a variable.

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

console.log(multiply(3, 4)); // Calling the function
```

5. **Arrow Function (ES6)**

A shorter syntax for function expressions.

```
const add = (a, b) => a + b;
console.log(add(5, 7)); // Output: 12
```

6. **Function with Default Parameters**

If a parameter is not passed, the default value is used.

```
function greetUser(name = "Guest") {
   console.log("Hello, " + name + "!");
}

greetUser(); // Default: "Guest"
greetUser("Alice"); // "Alice"
```

7. Immediately Invoked Function Expression (IIFE)

A function that runs **immediately** after being defined.

```
 (function() {
   console.log("This function runs immediately!");
})();
```

8. Callback Functions

A function passed as an argument to another function.

🔹 **Real-World Example:**
✅ **Processing an array using a callback function**.

```
function processNumbers(arr, callback) {
   for (let num of arr) {
      callback(num);
   }
}

processNumbers([1, 2, 3], (num) => {
   console.log(num * 2);
});
```

Sample Mini Project 1: Bill Splitting Calculator

Divides the total bill amount among a group of friends, including tip percentage.

```
function splitBill(totalAmount, people, tipPercentage = 0) {
   let tipAmount = (totalAmount * tipPercentage) / 100;
   let finalAmount = totalAmount + tipAmount;
   let perPerson = finalAmount / people;

   console.log(`Total Bill (with Tip): $${finalAmount.toFixed(2)}`);
   console.log(`Each person should pay: $${perPerson.toFixed(2)}`);
}

// Example Usage:
splitBill(100, 4, 10);  // Total: $110, Each: $27.50
splitBill(200, 5);      // No tip, Each: $40.00
```

Sample Mini Project 2: Shopping Cart System

Calculates the total price of items in a shopping cart.

```
let cart = [
   { item: "Laptop", price: 1000 },
   { item: "Mouse", price: 50 },
   { item: "Keyboard", price: 70 }
];

function calculateTotal(cartItems) {
   let total = 0;
   for (let item of cartItems) {
      total += item.price;
   }
```

```
    return total;
}
console.log("Total Price:", calculateTotal(cart));
```

Day 5 Tasks

JS53. Create a function that prints "Hello, World!".

JS54. Write a function that takes a number and returns its cube.

JS55. Write a function to check if a number is even or odd.

JS56. Write a function to find the factorial of a number.

JS57. Create a function to return the sum of two numbers.

JS58. Write a function that returns the largest of three numbers.

JS59. Write a function that checks if a string is a palindrome.

JS60. Create a function to convert Celsius to Fahrenheit.

JS61. Write a function to calculate the simple interest.

JS62. Create a function that reverses an array.

JS63. Write a function that counts the number of vowels in a string.

JS64. Write an arrow function that doubles each number in an array.

JS65. Create a function that takes a callback and calls it.

Mini Project

1. Student Grade Calculator

Calculates a student's grade based on their marks.

Requirements:

1. **User Input:** The user enters marks for different subjects.
2. **Total & Average Calculation:** The function calculates the total and average marks.
3. **Grade Assignment:**
   - 90+ → A
   - 75-89 → B
   - 50-74 → C
   - Below 50 → F
4. **Display Result:** Shows total marks, average, and grade.

2. Grocery Shopping List

✅ **Allows users to add, remove, and view grocery items dynamically.**

Requirements:

1. **User Input:** The user can add items to a grocery list.
2. **List Storage:** The system maintains an array to store items.
3. **View Items:** Users can view the current grocery list.
4. **Remove Items:** Users can remove an item by providing its name.

Day 6:

Arrays & Objects

What is an Array?

An **array** is a data structure that stores multiple values in a single variable. Each value is stored at a specific index starting from 0.

◆ Real-World Use Case of Arrays

✓ **Example:** A shopping cart stores multiple product names.

```
let shoppingCart = ["Laptop", "Mouse", "Keyboard", "Headset"];
console.log(shoppingCart); // Output: ['Laptop', 'Mouse', 'Keyboard', 'Headset']
```

Array Methods with Real-World Examples

1. push() - Add Element to End

✓ **Adding new items to a shopping cart**

```
let cart = ["Laptop", "Mouse"];
cart.push("Keyboard");
console.log(cart); // ['Laptop', 'Mouse', 'Keyboard']
```

2. pop() - Remove Last Element

✓ **Removing the last added product**

```
cart.pop();
console.log(cart); // ['Laptop', 'Mouse']
```

3. **shift() - Remove First Element**

✓ **Removing the first person from a queue**

```
let queue = ["John", "Mike", "Anna"];
queue.shift();
console.log(queue); // ['Mike', 'Anna']
```

4. **unshift() - Add Element to Beginning**

✓ **Adding a new person at the beginning of the queue**

```
queue.unshift("David");
console.log(queue); // ['David', 'Mike', 'Anna']
```

5. **forEach() - Loop through Elements**

✓ **Displaying all products in a store**

```
let products = ["Shoes", "Shirt", "Jeans"];
products.forEach(product => console.log(product));
```

6. **map() - Modify Each Element**

✓ **Applying a discount to product prices**

```
let prices = [100, 200, 300];
let discountedPrices = prices.map(price => price * 0.9);
console.log(discountedPrices); // [90, 180, 270]
```

7. **filter() - Filter Based on Condition**

✅ **Filtering out expensive products**

```
let expensiveProducts = prices.filter(price => price > 150);
console.log(expensiveProducts); // [200, 300]
```

8. find() - Find First Matching Element

✅ **Finding a product that costs exactly 200**

```
let product = prices.find(price => price === 200);
console.log(product); // 200
```

9. includes() - Check If Exists

✅ **Checking if an item is in stock**

```
console.log(products.includes("Shirt")); // true
```

Object :

What is an Object?

An **object** in JavaScript is a collection of properties (key-value pairs).

◆ **Real-World Use Case of Objects**

✅ **Example:** Storing user information in a system.

```
let user = {
   name: "Alice",
   age: 25,
   email: "alice@example.com"
};
console.log(user.name); // Output: Alice
```

Object Methods

1. Object.keys() - Get Object Keys

✅ **Getting all the property names of a user**

```
console.log(Object.keys(user)); // ['name', 'age', 'email']
```

2. Object.values() - Get Object Values

✅ **Getting all the values of a user**

```
console.log(Object.values(user)); // ['Alice', 25, 'alice@example.com']
```

3. Object.entries() - Get Key-Value Pairs

✅ **Converting an object into an array of key-value pairs**

```
console.log(Object.entries(user));
// [['name', 'Alice'], ['age', 25], ['email', 'alice@example.com']]
```

4. Object.assign() - Merge Objects

✅ **Merging user profile with additional details**

```
let address = { city: "New York", country: "USA" };
let completeUser = Object.assign(user, address);
console.log(completeUser);
```

5. delete - Remove Property from Object

✅ **Deleting email from user details**

```
delete user.email;
console.log(user);
```

Sample Mini Project 1: Product Inventory System

✅ **Manages products in stock and allows adding or removing products.**

```
let inventory = [
   { product: "Laptop", quantity: 5 },
   { product: "Phone", quantity: 10 }
];

function addProduct(productName, qty) {
   inventory.push({ product: productName, quantity: qty });
}

function removeProduct(productName) {
   inventory = inventory.filter(item => item.product !== productName);
}

addProduct("Tablet", 3);
removeProduct("Phone");
console.log(inventory);
```

Sample Mini Project 2: Student Record Management System

✅ **Stores student details (name, age, marks) and calculates the average marks.**

Requirements:

1. **Student Data Storage:** Store multiple students' details (name, age, marks) in an array of objects.
2. **Looping Through Students:** Use a loop to iterate over the student records.
3. **Calculate Average Marks:** Compute the average marks of all students.
4. **Display Output:** Print each student's details and the class average.

Day 6 Tasks

JS66. Create an array of 5 student names.

JS67. Add a new student name to the array using push().

JS68. Remove the last student using pop().

JS69. Use map() to convert all names to uppercase.

JS70. Use filter() to get students whose names start with "A".

JS71. Use forEach() to print each student name.

JS72. Create an object for a book with properties (title, author, price).

JS73. Use Object.keys() to get all book properties.

JS74. Use Object.values() to get all book values.

JS75. Add a new property genre to the book object.

JS76. Use delete to remove price from the book.

JS77. Merge a publisher object into the book object.

JS78. Convert the book object into an array of key-value pairs using Object.entries().

Mini Projects :

1. Movie Booking System

Manages a list of available movies and allows users to book tickets.

Requirements:

1. **Movie List Storage:** Maintain a list of available movie names in an array.
2. **User Input:** Accept the movie name that the user wants to book.
3. **Check Availability:** Validate if the movie exists in the list.
4. **Booking Confirmation:** If available, display "Ticket booked", otherwise "Movie not available".

2. Product Inventory System

Manages products in stock and allows adding or removing products.

Requirements:

1. **Inventory Storage:** Store available products (name, quantity) in an array of objects.
2. **Add Product Function:** Allow users to add a new product with its quantity.
3. **Remove Product Function:** Remove a product from the inventory by name.
4. **Display Updated Inventory:** Show the list of products after any change.

Day 7

Strings :

What is a String?

A **string** is a sequence of characters used to store text in JavaScript. It can be defined using:

- **Single quotes (')**
- **Double quotes (")**
- **Backticks (`) – Template literals**

✅ **Example:**

```
let str1 = "Hello";
let str2 = 'World';
let str3 = `JavaScript`;
console.log(str1, str2, str3);
```

String Manipulation :

1. **Concatenation (Joining Strings)**

✅ **Combining two or more strings using + or concat().**

```
let firstName = "John";
let lastName = "Doe";
console.log(firstName + " " + lastName); // Using +
console.log(firstName.concat(" ", lastName)); // Using concat()
```

2. **Template Literals (ES6)**

✅ **Using backticks (`) for multi-line strings & variable interpolation.**

```
let name = "Alice";
let age = 25;
console.log(`My name is ${name} and I am ${age} years old.`);
```

3. Changing Case (Upper & Lower)

✅ **Convert strings to uppercase or lowercase.**

```
let text = "Hello JavaScript!";
console.log(text.toUpperCase()); // HELLO JAVASCRIPT!
console.log(text.toLowerCase()); // hello javascript!
```

4. Extracting Parts of a String

✅ **Getting a portion of a string using slice(), substring(), or substr().**

```
let str = "JavaScript";
console.log(str.slice(0, 4)); // Java
console.log(str.substring(0, 4)); // Java
console.log(str.substr(0, 4)); // Java
```

5. Finding Substrings

✅ **Check if a string contains a certain word using includes(), indexOf(), or search().**

```
let sentence = "I love JavaScript!";
console.log(sentence.includes("JavaScript")); // true
console.log(sentence.indexOf("love")); // 2
console.log(sentence.search("love")); // 2
```

6. Replacing Text

✓ **Replace part of a string using replace() and replaceAll().**

```
let text1 = "I like JavaScript!";
console.log(text1.replace("like", "love")); // I love JavaScript!
console.log(text1.replaceAll("JavaScript", "Python")); // I like Python!
```

7. Splitting Strings into Arrays

✓ **Convert a string into an array using split().**

```
let words = "apple,banana,orange";
console.log(words.split(",")); // ['apple', 'banana', 'orange']
```

8. Removing Extra Spaces

✓ **Use trim(), trimStart(), and trimEnd() to remove spaces.**

```
let spacedText = "   Hello JavaScript!   ";
console.log(spacedText.trim()); // "Hello JavaScript!"
console.log(spacedText.trimStart()); // "Hello JavaScript!   "
console.log(spacedText.trimEnd()); // "   Hello JavaScript!"
```

9. Repeating a String

✓ **Repeat a string multiple times using repeat().**

```
let repeatText = "Hello ";
console.log(repeatText.repeat(3)); // "Hello Hello Hello "
```

10. Checking String Starts/Ends with a Value

✅ **Use startsWith() and endsWith() to check prefixes and suffixes.**

```
let str2 = "Welcome to JavaScript!";
console.log(str2.startsWith("Welcome")); // true
console.log(str2.endsWith("JavaScript!")); // true
```

String Methods

Method	Description	Example
concat()	Combines two strings	"Hello".concat(" World")
toUpperCase()	Converts to uppercase	"hello".toUpperCase() → "HELLO"
toLowerCase()	Converts to lowercase	"HELLO".toLowerCase() → "hello"
slice()	Extracts part of a string	"JavaScript".slice(0, 4) → "Java"
substring()	Similar to slice() but doesn't accept negative indexes	"JavaScript".substring(0, 4)
includes()	Checks if a string contains another	"Hello".includes("lo") → true
indexOf()	Finds the index of a word	"Hello".indexOf("l") → 2
replace()	Replaces a word	"I like JS".replace("like", "love")
split()	Converts a string into an array	"a,b,c".split(",")
trim()	Removes extra spaces	" Hello ".trim()
repeat()	Repeats a string	"Hi".repeat(3)

Method	Description	Example
startsWith()	Checks if a string starts with a value	"Hello".startsWith("H")
endsWith()	Checks if a string ends with a value	"Hello".endsWith("o")

Example:

◆ **Username Validation**

Check if a username meets length requirements.

```
function validateUsername(username) {
   if (username.length < 5) {
      return "Username must be at least 5 characters long.";
   } else {
      return "Valid username.";
   }
}

console.log(validateUsername("Sam")); // Invalid
console.log(validateUsername("Samuel")); // Valid
```

◆ **Email Validation**

Check if an email contains @ and .com.

```
function validateEmail(email) {
   return email.includes("@") && email.endsWith(".com");
}

console.log(validateEmail("test@gmail.com")); // true
console.log(validateEmail("invalid-email")); // false
```

Sample Mini project 1:Text Reverser

✓ **Reverses a given string.**

```
function reverseString(str) {
   return str.split("").reverse().join("");
}
console.log(reverseString("JavaScript")); // "tpircSavaJ"
console.log(reverseString("hello")); // "olleh"
```

Sample mini project 2:Word Counter

✓ **Counts the number of words in a user-inputted sentence.**

Requirements

1. The user enters a sentence.
2. The function should count the number of words.
3. Words are separated by spaces.
4. Display the word count.

Day 7 Tasks

JS79. Convert "javascript" to uppercase.

JS80. Convert "HELLO WORLD" to lowercase.

JS81. Extract "Script" from "JavaScript" using slice().

JS82. Replace "bad" with "good" in "This is a bad day".

JS83. Check if "Coding" is present in "I love Coding!".

JS84. Find the position of "world" in "Hello world!".

JS85. Remove spaces from " Trim me ".

JS86. Count the number of characters in "Programming".

JS87. Check if "apple" starts with "a".

JS88. Split "banana,apple,orange" into an array.

JS89. Repeat "JS " three times.

JS90. Concatenate "Web" and " Development".

JS91. Validate if "user123" has at least 5 characters.

Mini project

1. Username Generator

Generates a unique username based on user input.

Requirements

1. The user enters their **first name** and **last name**.
2. The function should generate a username by:
   - Taking the first **3 letters** of the first name.
   - Taking the first **3 letters** of the last name.
   - Adding a **random number (100-999)** at the end.
3. Display the generated username.

2. Palindrome Checker (Real-World Mini Project)

Checks whether a given word or phrase is a palindrome (reads the same forward and backward).

Requirements

1. The user enters a **word or sentence**.
2. The function should **remove spaces and special characters**.
3. Convert the string to **lowercase** to ensure accurate comparison.
4. Check if the word is the same **forward and backward**.
5. Display "Palindrome" or "Not a Palindrome".

Day 8:

Document Object Model (DOM)

What is the DOM?

The **Document Object Model (DOM)** is a programming interface that allows JavaScript to interact with and manipulate the structure, content, and style of an HTML document.

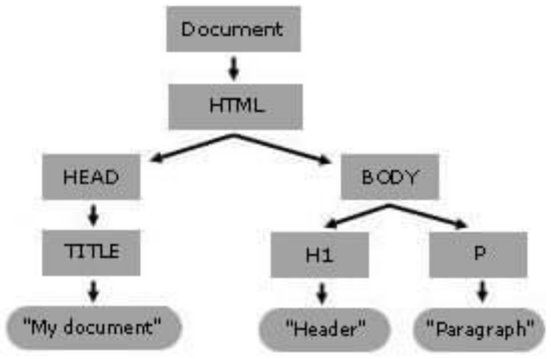
✅ **The DOM represents an HTML document as a tree of objects.**
✅ **Each HTML element is a node in this tree.**
✅ **JavaScript can be used to dynamically change, add, or remove elements in the DOM.**

DOM Structure

Consider the following HTML:

```
<!DOCTYPE html>
<html>
<head>
   <title>DOM Example</title>
</head>
<body>
   <h1 id="title">Hello, DOM!</h1>
   <p class="description">This is a DOM manipulation example.</p>
</body>
</html>
```

The **DOM Tree** for this structure looks like:



DOM Manipulation Methods

1. Selecting Elements

JavaScript can select HTML elements using different methods:

```
// Selecting by ID
let title = document.getElementById("title");

// Selecting by Class
let desc = document.getElementsByClassName("description");

// Selecting by Tag Name
let paragraphs = document.getElementsByTagName("p");

// Selecting using Query Selector
let firstPara = document.querySelector(".description"); // Returns first match
let allParas = document.querySelectorAll("p"); // Returns all matche
```

2. Modifying Content

Change text inside an element:

```
document.getElementById("title").innerText = "Welcome to the DOM!";
```

✅ **Change HTML inside an element:**

```
document.getElementById("title").innerHTML = "<em>Welcome to the
DOM!</em>";
```

✅ **Change the style of an element:**

```
document.getElementById("title").style.color = "blue";
```

3. Creating & Appending Elements

✅ **Create a new element and add it to the page:**

```
let newPara = document.createElement("p");
newPara.innerText = "This is a new paragraph!";
document.body.appendChild(newPara);
```

4. Removing Elements

✅ **Remove an element from the DOM:**

```
let title = document.getElementById("title");
title.remove(); // Removes the title element
```

5. Toggle Elements

✅ **Toggle visibility when a button is clicked:**

```
document.getElementById("toggleBtn").addEventListener("click", function() {
   let para = document.getElementById("para");
   para.style.display = para.style.display === "none" ? "block" : "none";
});
```

Example: DOM Manipulation (Interactive Page)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>DOM Example</title>
   <style>
      #box {
         width: 200px;
         height: 200px;
         background-color: red;
         margin-top: 20px;
      }
   </style>
</head>
```

```
<body>

  <h1 id="title">Welcome to DOM Manipulation</h1>
  <button onclick="changeText()">Change Text</button>
  <button onclick="changeColor()">Change Color</button>
  <button onclick="addElement()">Add Element</button>
  <button onclick="removeElement()">Remove Element</button>

  <div id="box"></div>

  <script>
    function changeText() {
      document.getElementById("title").innerText = "Text Changed!";
    }

    function changeColor() {
      document.getElementById("box").style.backgroundColor = "blue";
    }

    function addElement() {
      let newPara = document.createElement("p");
      newPara.innerText = "New paragraph added!";
      document.body.appendChild(newPara);
    }

    function removeElement() {
      let box = document.getElementById("box");
      if (box) {
        box.remove();
      }
    }
  </script>

</body>
```

</html>

Difference Between querySelector & querySelectorAll

Method	Returns	Example
querySelector()	First matching element	document.querySelector(".className")
querySelectorAll()	**All** matching elements (NodeList)	document.querySelectorAll("p")

JavaScript Alert Message

✅ **Displays an alert pop-up:**

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

✅ **Prompt for User Input:**

```
let name = prompt("Enter your name:");
alert("Hello, " + name + "!");
```

setInterval() in JavaScript

The setInterval() method repeatedly executes a function **at a specified time interval** (in milliseconds).

✅ **Syntax:**

```
setInterval(function, time);
```

- function: The function to execute.
- time: Interval in milliseconds (1000ms = 1 second).

Example: Digital Clock (Live Time Update)

✅ **Displays real-time updates every second.**

```
<h1 id="clock"></h1>

<script>
  function updateClock() {
    let now = new Date();
    document.getElementById("clock").innerText = now.toLocaleTimeString();
  }

  setInterval(updateClock, 1000); // Updates every second
</script>
```

🔹 A real-time **digital clock** on websites.

Example: Automatic Image Slider

✅ **Automatically changes images at an interval.**

```
<img id="slider" src="image1.jpg" width="300px">

<script>
  let images = ["image1.jpg", "image2.jpg", "image3.jpg"];
  let index = 0;

  function changeImage() {
    index = (index + 1) % images.length; // Loop back to first image
    document.getElementById("slider").src = images[index];
  }
  setInterval(changeImage, 3000); // Change image every 3 seconds
</script>
```

❖ **Image carousels** on e-commerce or portfolio websites.

Example: Auto Refresh Notification Counter

✅ **Simulates a live notification counter updating every 2 seconds.**

```
<p>New Notifications: <span id="counter">0</span></p>

<script>
  let count = 0;

  function updateCounter() {
    count++;
    document.getElementById("counter").innerText = count;
  }

  setInterval(updateCounter, 2000); // Updates every 2 seconds
</script>
```

❖**Live notifications** on social media platforms.

Example: Countdown Timer

✅ **Counts down from a given number and stops at zero.**

```
<p>Time Left: <span id="timer">10</span> seconds</p>

<script>
  let timeLeft = 10;

  let countdown = setInterval(() => {
    document.getElementById("timer").innerText = timeLeft;
```

```
        timeLeft--;

        if (timeLeft < 0) {
            clearInterval(countdown); // Stop countdown
            alert("Time's up!");
        }
    }, 1000);
</script>
```

🔹 **Quiz countdowns**, **flash sales**, and **form auto-submission timers**.

Example: Auto-Saving Feature

✅ **Automatically saves user input every 5 seconds.**

```
<textarea id="textInput" placeholder="Start typing..."></textarea>
<p id="saveStatus"></p>

<script>
    function autoSave() {
        let text = document.getElementById("textInput").value;
        if (text) {
            document.getElementById("saveStatus").innerText = "Saved!";
        }
    }

    setInterval(autoSave, 5000); // Auto-saves every 5 seconds
</script>
```

🔹 **Google Docs-style auto-save feature.**

Stopping setInterval() using clearInterval()

If you need to **stop an interval**, use clearInterval(intervalID).

✅ **Example: Stop Timer After 5 Counts**

```
let count = 0;
let intervalID = setInterval(() => {
   console.log("Count:", count);
   count++;

   if (count === 5) {
      clearInterval(intervalID); // Stops after 5 iterations
   }
}, 1000);
```

Sample Mini project 1:To-Do List App

✅ **Allows users to add, remove, and mark tasks as completed.**

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>To-Do List</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         text-align: center;
      }
      #todo-container {
         width: 300px;
         margin: auto;
         text-align: left;
      }
      ul {
         list-style-type: none;
         padding: 0;
      }
      li {
         background: #f3f3f3;
         padding: 10px;
         margin: 5px 0;
         display: flex;
         justify-content: space-between;
         align-items: center;
      }
      .completed {
         text-decoration: line-through;
         color: gray;
      }
      button {
         cursor: pointer;
      }
   </style>
</head>
<body>

   <h2>To-Do List</h2>
   <div id="todo-container">
      <input type="text" id="taskInput" placeholder="Enter a task">
      <button onclick="addTask()">Add</button>
      <ul id="taskList"></ul>
   </div>

   <script>
```

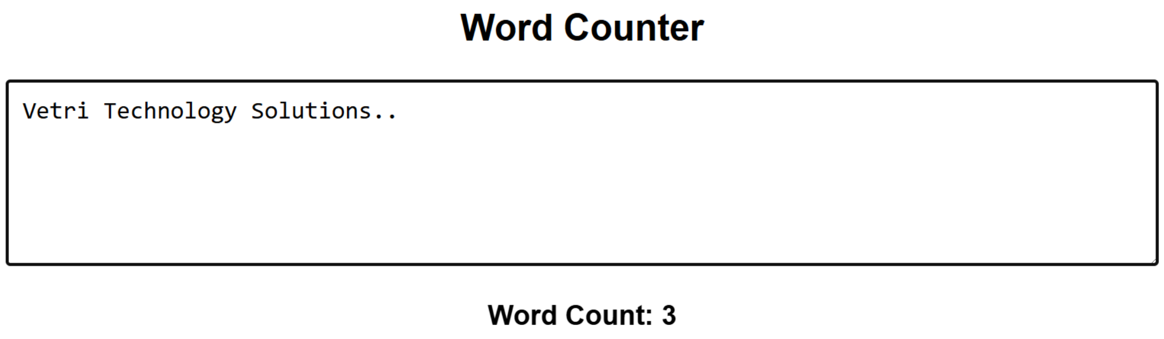
```
      function addTask() {
         let taskInput = document.getElementById("taskInput");
         let taskList = document.getElementById("taskList");

         if (taskInput.value.trim() !== "") {
            let li = document.createElement("li");
            li.innerText = taskInput.value;

            let removeBtn = document.createElement("button");
            removeBtn.innerText = "❌";
            removeBtn.onclick = function() {
               taskList.removeChild(li);
            };

            li.onclick = function() {
               li.classList.toggle("completed");
            };

            li.appendChild(removeBtn);
            taskList.appendChild(li);
            taskInput.value = "";
         }
      }
   </script>

</body>
</html>
```

Sample Mini Project 2: Word Counter

Counts the number of words in a textarea and updates dynamically.

Requirements:

1. **Textarea Input:** Users should be able to type text in a large input box.
2. **Real-Time Word Count:** The word count should update automatically as the user types.
3. **Whitespace Handling:** Consecutive spaces should not count as extra words.
4. **Empty Input:** If the textarea is empty, the word count should display 0.



Day 8 Tasks :

JS92. Change the background color of the page using a button click.

JS93. Change the text of a paragraph using a button.

JS94. Hide and show an element using a toggle button.

JS95. Create a new list item dynamically.

JS96. Remove a specific item from a list.

JS97. Change the font size of a paragraph when a button is clicked.

JS98. Change the text color of all paragraphs using querySelectorAll().

JS99. Add a border to an image when hovered.

JS100. Display an alert message when a button is clicked.

JS101. Toggle between two different text contents.

JS102. Add a new heading to the page dynamically.

JS103. Remove all elements inside a div.

JS104. Change the text input value dynamically.

Mini Project 1:

1. **Light/Dark Mode Toggle**

✓ **Allows users to switch between light and dark themes dynamically.**

Requirements:

1. **Default Light Mode:** The page should load in light mode by default.
2. **Toggle Button:** Clicking the button should switch between light and dark mode.
3. **Dark Mode Styling:** The background should become dark, and the text should turn white.

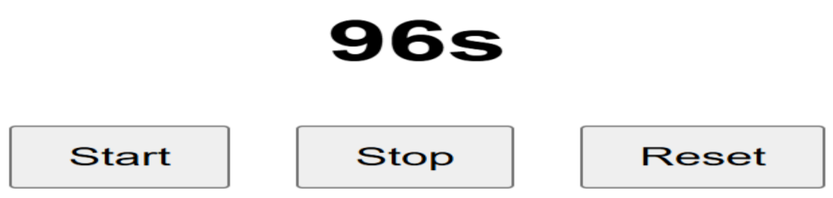
2. **Stopwatch Timer (Real-World Mini Project)**

✓ **A simple stopwatch that starts, stops, and resets using buttons.**

Requirements

1. **Start Timer:** Clicking the "Start" button should begin counting seconds.
2. **Stop Timer:** Clicking the "Stop" button should pause the timer.
3. **Reset Timer:** Clicking the "Reset" button should set the timer back to 0.

4. **Real-Time Update:** The displayed time should update every second while running.



Day 9

Event Handling

What is an Event in JavaScript?

An **event** in JavaScript is an action or occurrence that happens in the browser, such as:
✓ Clicking a button
✓ Pressing a key
✓ Hovering over an element
✓ Submitting a form

JavaScript allows us to detect and respond to these events using **Event Handling**.

Types of Events in JavaScript

1. **Mouse Events**

- click → When a user clicks an element
- dblclick → When a user double-clicks
- mouseover → When a user hovers over an element

- mouseout → When a user moves the mouse away
- mousedown / mouseup → When a user presses/releases a mouse button

2. Keyboard Events

- keydown → When a key is pressed
- keyup → When a key is released
- keypress → (Deprecated)

3. Form Events

- submit → When a form is submitted
- change → When an input value is changed
- focus → When an input field is selected
- blur → When an input field loses focus

4. Window Events

- load → When the page finishes loading
- resize → When the window is resized
- scroll → When the user scrolls
- unload → When the user leaves the page

What is an Event Handler?

An **Event Handler** is a function that executes when an event occurs.

✅ **Example: Inline Event Handling (Bad Practice)**

```
<button onclick="sayHello()">Click Me</button>

<script>
  function sayHello() {
    alert("Hello! You clicked the button.");
  }
</script>
```

What is an Event Listener?

An **Event Listener** is a method (addEventListener()) that listens for an event and executes a function when the event occurs.

✓ **Example: Using addEventListener() (Best Practice)**

```
<button id="myButton">Click Me</button>

<script>
   document.getElementById("myButton").addEventListener("click", function() {
      alert("Hello! You clicked the button.");
   });
</script>
```

Difference Between Event Handler and Event Listener

Feature	Event Handler (onclick)	Event Listener (addEventListener())
Multiple Functions	✗ Not possible	✓ Possible
Removal	✗ Cannot be removed	✓ Can be removed
Flexibility	✗ Limited	✓ More control

✓ **Example: Event Listener Can Be Removed**

```
function greet() {
   alert("Hello!");
}
document.getElementById("myButton").addEventListener("click", greet);

// Remove event listener after 5 seconds
setTimeout(() => {
   document.getElementById("myButton").removeEventListener("click", greet);
}, 5000);
```

Event Handling - Real-World Examples

1. **Button Click Event**

✅ **Changes the text when a button is clicked.**

```
<button id="changeText">Click Me</button>
<p id="message">Original Text</p>

<script>
   document.getElementById("changeText").addEventListener("click", function() {
      document.getElementById("message").innerText = "Text Changed!";
   });
</script>
```

2. **Keyboard Event (Key Press Detection)**

✅ **Detects which key is pressed in an input field.**

```
<input type="text" id="inputBox" placeholder="Type something...">
<p id="keyInfo"></p>

<script>
   document.getElementById("inputBox").addEventListener("keyup",
function(event) {
      document.getElementById("keyInfo").innerText = "Key Pressed: " +
event.key;
   });
</script>
```

3. **Mouse Hover Event**

✅ **Changes background color when hovering over an element.**

```
<div id="hoverBox" style="width: 200px; height: 100px; background:
lightgray;">Hover over me</div>

<script>
  let box = document.getElementById("hoverBox");

  box.addEventListener("mouseover", function() {
    box.style.backgroundColor = "yellow";
  });

  box.addEventListener("mouseout", function() {
    box.style.backgroundColor = "lightgray";
  });
</script>
```

4. **Form Submit Event**

✅ **Prevents default form submission and displays input.**

```
<form id="myForm">
  <input type="text" id="nameInput" placeholder="Enter Name">
  <button type="submit">Submit</button>
</form>
<p id="result"></p>

<script>
  document.getElementById("myForm").addEventListener("submit",
function(event) {
    event.preventDefault();
    let name = document.getElementById("nameInput").value;
    document.getElementById("result").innerText = "Hello, " + name + "!";
  });
</script>
```

event.preventDefault() :

- The event.preventDefault() method prevents the default behavior of an event from occurring.
- It is commonly used in form submissions, anchor tags (<a>), and other default browser actions.

Why Use event.preventDefault()?

When working with **Event Listeners**, sometimes we want to override the browser's default behavior.

✅ **Examples where event.preventDefault() is used:**

1. Preventing form submission from refreshing the page
2. Preventing a link (<a>) from navigating to a new page
3. Disabling right-click context menu
4. Preventing drag-and-drop from opening a file

Example 1: Prevent Form Submission (Avoid Page Refresh)

By default, when you submit a form, the page **reloads**. Using event.preventDefault(), we can **prevent the refresh** and handle the data manually.

✅ **Without preventDefault() (Page refreshes)**

```
<form id="myForm">
  <input type="text" placeholder="Enter Name">
  <button type="submit">Submit</button>
</form>
```

```
<script>
   document.getElementById("myForm").addEventListener("submit",
function(event) {
      console.log("Form Submitted!"); // This runs, but the page refreshes
   });
</script>
```

Fix: Prevent the Form from Refreshing

```
<form id="myForm">
   <input type="text" id="nameInput" placeholder="Enter Name">
   <button type="submit">Submit</button>
</form>
<p id="output"></p>

<script>
   document.getElementById("myForm").addEventListener("submit",
function(event) {
      event.preventDefault(); // Prevents page refresh
      let name = document.getElementById("nameInput").value;
      document.getElementById("output").innerText = "Hello, " + name + "!";
   });
</script>
```

✅ **Now the form submission doesn't refresh the page!**

Examples :

1. **Toggle Light/Dark Mode**

Allows users to switch between light and dark themes dynamically.

```
<button id="themeToggle">Toggle Dark Mode</button>

<script>
   document.getElementById("themeToggle").addEventListener("click", function()
{
      document.body.classList.toggle("dark-mode");
   });
</script>

<style>
   .dark-mode {
      background-color: black;
      color: white;
   }
</style>
```

2. Real-Time Character Counter

✅ **Counts and updates the number of characters in a textarea.**

```
<textarea id="textInput" placeholder="Start typing..."></textarea>
<p>Character Count: <span id="charCount">0</span></p>

<script>
   document.getElementById("textInput").addEventListener("input", function() {
      document.getElementById("charCount").innerText = this.value.length;
   });
</script>
```

3. Image Carousel (Next & Previous Buttons)

✅ **Changes images using "Next" and "Previous" buttons.**

```
<img id="carousel" src="image1.jpg" width="300px">
<button id="prev">Previous</button>
<button id="next">Next</button>

<script>
  let images = ["image1.jpg", "image2.jpg", "image3.jpg"];
  let index = 0;

  document.getElementById("next").addEventListener("click", function() {
    index = (index + 1) % images.length;
    document.getElementById("carousel").src = images[index];
  });
  document.getElementById("prev").addEventListener("click", function() {
    index = (index - 1 + images.length) % images.length;
    document.getElementById("carousel").src = images[index];
  });
</script>
```

4. **Countdown Timer**

✓ **Starts a countdown when a button is clicked.**

```
<button id="startTimer">Start Countdown</button>
<p id="countdown"></p>
<script>
  document.getElementById("startTimer").addEventListener("click", function() {
    let time = 10;
    let interval = setInterval(() => {
      document.getElementById("countdown").innerText = time + " seconds";
      time--;
      if (time < 0) {
        clearInterval(interval);
        document.getElementById("countdown").innerText = "Time's up!";
```

```
            }
        }, 1000);
    });
</script>
```

Sample Mini project1:Drag and Drop Box

Allows users to drag and drop an element within a designated area.

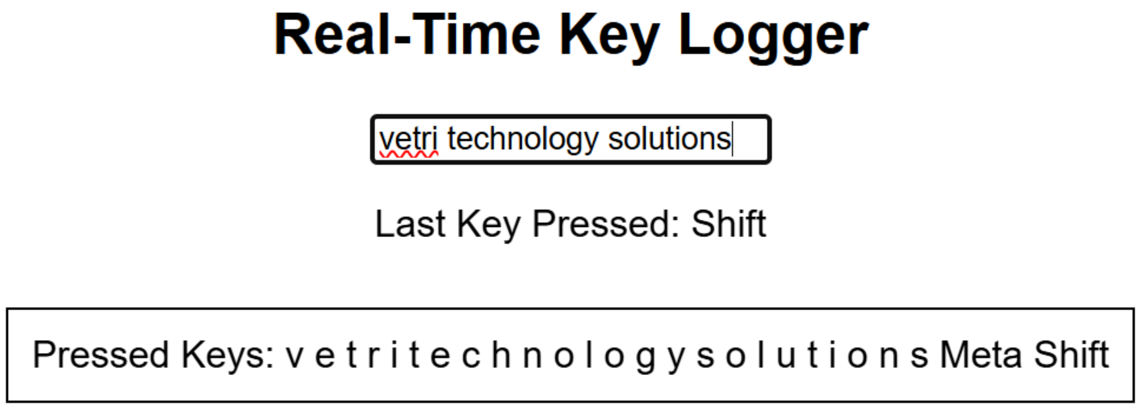
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Drag and Drop</title>
    <style>
        #dragBox {
            width: 100px;
            height: 100px;
            background-color: blue;
            color: white;
            display: flex;
            align-items: center;
            justify-content: center;
            cursor: grab;
            position: absolute;
            top: 50px;
            left: 50px;
        }
        #dropZone {
            width: 200px;
            height: 200px;
            border: 2px dashed black;
            display: flex;
            align-items: center;
            justify-content: center;
            margin-top: 150px;
            text-align: center;
        }
    </style>
</head>
<body>
    <h2>Drag the Box into the Drop Zone</h2>
    <div id="dragBox" draggable="true">Drag Me</div>
    <div id="dropZone">Drop Here</div>
    <script>
        let dragBox = document.getElementById("dragBox");
        let dropZone = document.getElementById("dropZone");

        dragBox.addEventListener("dragstart", function(event) {
            event.dataTransfer.setData("text", event.target.id);
        });

        dropZone.addEventListener("dragover", function(event) {
            event.preventDefault(); // Allows the drop
        });

        dropZone.addEventListener("drop", function(event) {
            event.preventDefault();
            let draggedElement =
document.getElementById(event.dataTransfer.getData("text"));
            dropZone.appendChild(draggedElement);
            dragBox.style.position = "static";
            dropZone.innerHTML = "<b>Item Dropped Successfully!</b>";
        });
    </script>
</body>
</html>
```

Sample Mini project 2 :Keyboard Event: Real-Time Key Logger

✅ **Displays keys pressed by the user in real-time.**

📌 **Requirements:**

1. Detect **every keypress** in an input field.
2. Show the **last key pressed** dynamically.
3. Display **all pressed keys** in a list.



Real-Time Key Logger

vetri technology solutions

Last Key Pressed: Shift

Pressed Keys: v e t r i t e c h n o l o g y s o l u t i o n s Meta Shift

Day 9 Tasks :

JS105. Change the text of a button when clicked.

JS106. Show an alert when the mouse enters a specific area.

JS107. Detect when the user presses the "Enter" key.

JS108. Change the background color of a page when a button is clicked.

JS109. Toggle a password field between text and password.

JS110. Display the coordinates of the mouse pointer when moved.

JS111. Prevent the default behavior of a form submission.

JS112. Count the number of times a button is clicked.

JS113. Create a button that disappears when clicked.

JS114. Toggle an element's visibility when another element is clicked.

JS115. Play a sound when a button is clicked.

JS116. Display a loading animation when the page is loading.

JS117. Change an image when a button is clicked.

Mini project

1. Click Counter App

Counts the number of times a button is clicked.

Requirements:

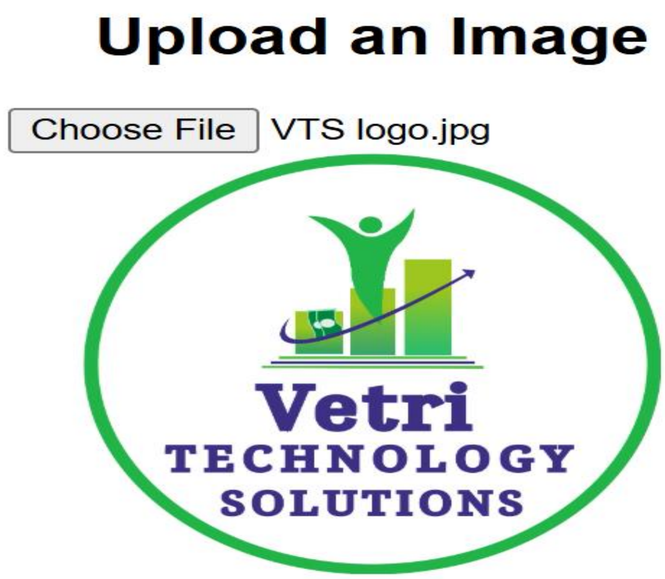
- Clicking the button should **increase the counter**.
- Add a **reset button** to clear the count.

2. Image Preview Before Uploading

Allows users to preview an image before uploading.

Requirements:

1. Select an image from the file input.
2. Display the **selected image** in a preview area.



DAY 10

What is Web Storage?

Web Storage API provides mechanisms to store **key-value** data in a user's browser.

Types of Web Storage

Storage Type	Persistence	When to Use
LocalStorage	Data is stored **permanently** (until manually cleared)	Storing user preferences, theme settings, saved cart items
SessionStorage	Data is stored **until the tab is closed**	Temporary form data, session authentication, real-time activities

LocalStorage in JavaScript

- Stores data **permanently** (until manually removed).
- Stores **strings only** (other data types must be converted to strings using JSON.stringify()).
- **Accessible across multiple tabs** in the same browser.

LocalStorage Methods

Method	Description
localStorage.setItem(key, value)	Stores data in LocalStorage
localStorage.getItem(key)	Retrieves stored data
localStorage.removeItem(key)	Deletes a specific key from storage
localStorage.clear()	Clears all stored data

Example: Storing and Retrieving Data in LocalStorage

```
// Save data
localStorage.setItem("username", "JohnDoe");

// Retrieve data
let user = localStorage.getItem("username");
console.log(user); // Output: JohnDoe

// Remove data
localStorage.removeItem("username");

// Clear all LocalStorage
localStorage.clear();
```

✅ **Real-World Use Cases**: ✔ Storing **user p**references (theme settings).
✔ Saving **shopping cart items** on an e-commerce site.

SessionStorage in JavaScript

- Stores data **only for the session** (disappears when the tab is closed).
- Works **similar to LocalStorage**, but doesn't persist after tab close.

SessionStorage Methods

Method	Description
sessionStorage.setItem(key, value)	Stores data for the session
sessionStorage.getItem(key)	Retrieves stored session data
sessionStorage.removeItem(key)	Deletes a specific session key
sessionStorage.clear()	Clears all session storage data

Example: Storing and Retrieving Data in SessionStorage

```
// Save session data
sessionStorage.setItem("sessionUser", "Alice");

// Retrieve session data
let sessionUser = sessionStorage.getItem("sessionUser");
console.log(sessionUser); // Output: Alice

// Remove session data
sessionStorage.removeItem("sessionUser");

// Clear all session storage
sessionStorage.clear();
```

Real-World Use Cases: ✓ Storing **temporary form inputs**.
Keeping **login session active** within a single tab.

What is JSON.stringify()?

JSON.stringify() is used to convert a JavaScript object or array into a JSON-formatted string.
This is useful when storing objects in localStorage, sessionStorage, or sending data to an API.

Example: Convert an Object to a JSON String

```
let user = {
   name: "John",
   age: 25,
   city: "New York"
};

let jsonString = JSON.stringify(user);
console.log(jsonString);
```

Output:

```
'{"name":"John","age":25,"city":"New York"}'  // JSON string format
```

✅ **Real-World Use Case: ✓ Storing objects in LocalStorage**

```
localStorage.setItem("user", JSON.stringify(user));
```

What is JSON.parse()?

JSON.parse() is used to **convert a JSON string back into a JavaScript object**.
This is useful when retrieving data from an API, localStorage, or sessionStorage.

Example: Convert a JSON String Back to an Object

```
let jsonString = '{"name":"John","age":25,"city":"New York"}';

let userObject = JSON.parse(jsonString);
console.log(userObject);
```

Output:

```
{ name: "John", age: 25, city: "New York" }  // JavaScript object
```

Real-World Use Case: ✔ Retrieving objects from LocalStorage

Summary

Method	Description	Example
JSON.stringify(object)	Converts an object/array into a JSON string	JSON.stringify({name: "Alice"}) → {"name":"Alice"}
JSON.parse(jsonString)	Converts a JSON string back into an object/array	JSON.parse('{"name":"Alice"}') → {name: "Alice"}

Example:

1. Dark Mode Toggle (Save Theme in LocalStorage)

Persists theme settings across page reloads.

```
<button id="toggleTheme">Toggle Dark Mode</button>

<script>
  function loadTheme() {
    let theme = localStorage.getItem("theme");
    if (theme === "dark") {
      document.body.classList.add("dark-mode");
      }
   }

   function toggleTheme() {
      document.body.classList.toggle("dark-mode");
      let theme = document.body.classList.contains("dark-mode") ? "dark" : "light";
      localStorage.setItem("theme", theme);
   }

   document.getElementById("toggleTheme").addEventListener("click",
toggleTheme);
   loadTheme();
</script>

<style>
   .dark-mode { background-color: black; color: white; }
</style>
```

Real-World Use Case:
Used in websites with dark mode settings.

2. Simple Login System (SessionStorage)

User session expires when the tab is closed.

```
<button id="login">Login</button>
<button id="logout">Logout</button>
<p id="status"></p>
```

```
<script>
   function checkSession() {
      let user = sessionStorage.getItem("user");
      document.getElementById("status").innerText = user ? "Logged In" : "Logged
Out";
   }

   document.getElementById("login").addEventListener("click", function() {
      sessionStorage.setItem("user", "Logged In");
      checkSession();
   });

   document.getElementById("logout").addEventListener("click", function() {
      sessionStorage.removeItem("user");
      checkSession();
   });

   checkSession();
</script>
```

Real-World Use Case:
Used in **web apps where sessions expire on tab close**.

Sample Mini project 1: Add to Cart with Search Functionality Using LocalStorage

Users can add products to the cart, search for products, and persist data after page reload.

Features:

✓ **Search Functionality:** Users can search for products dynamically.
✓ **Add to Cart Feature:** Items are stored in LocalStorage and persist even after a page refresh.

✓ **Cart Display:** Shows cart items dynamically with the option to remove them.
✓ **Cart Total Calculation:** Calculates and updates the total price dynamically.
✓ **Clear Cart Option:** Allows users to reset the cart.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Shopping Cart with Total</title>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin: 20px; }
    input, button { padding: 10px; margin: 5px; }
    .product-list, .cart-list { list-style: none; padding: 0; }
    li { padding: 10px; margin: 5px; border: 1px solid #ccc; display: flex; justify-
content: space-between; }
  </style>
</head>
<body>

  <h2>Product List</h2>
  <input type="text" id="search" placeholder="Search for a product"
onkeyup="filterProducts()">
  <ul id="productList"></ul>

  <h2>Shopping Cart</h2>
  <ul id="cartList"></ul>
  <p><strong>Total Price: $<span id="cartTotal">0.00</span></strong></p>
  <button onclick="clearCart()">Clear Cart</button>

  <script>
    let products = [
      { name: "Laptop", price: 1200 },
      { name: "Smartphone", price: 800 },
      { name: "Tablet", price: 500 },
      { name: "Headphones", price: 150 },
      { name: "Smartwatch", price: 250 },
      { name: "Camera", price: 600 },
      { name: "Keyboard", price: 100 },
      { name: "Mouse", price: 50 }
    ];

    function displayProducts() {
      let productList = document.getElementById("productList");
      productList.innerHTML = "";
      products.forEach(product => {
          let li = document.createElement("li");
          li.innerHTML = `${product.name} - $${product.price} <button
onclick="addToCart('${product.name}', ${product.price})">Add to Cart</button>`;
          productList.appendChild(li);
      });
    }

    function filterProducts() {
      let query = document.getElementById("search").value.toLowerCase();
      let filtered = products.filter(product =>
product.name.toLowerCase().includes(query));
      let productList = document.getElementById("productList");
      productList.innerHTML = "";
      filtered.forEach(product => {
          let li = document.createElement("li");
          li.innerHTML = `${product.name} - $${product.price} <button
onclick="addToCart('${product.name}', ${product.price})">Add to Cart</button>`;
          productList.appendChild(li);
      });
    }

    function loadCart() {
      let cart = JSON.parse(localStorage.getItem("cart")) || [];
      let cartList = document.getElementById("cartList");
      cartList.innerHTML = "";
      let total = 0;

      cart.forEach((item, index) => {
          total += item.price;
          let li = document.createElement("li");
          li.innerHTML = `${item.name} - $${item.price} <button
onclick="removeFromCart(${index})">❌</button>`;
          cartList.appendChild(li);
      });

      document.getElementById("cartTotal").innerText = total.toFixed(2);
    }

    function addToCart(name, price) {
      let cart = JSON.parse(localStorage.getItem("cart")) || [];
      cart.push({ name, price });
      localStorage.setItem("cart", JSON.stringify(cart));
      loadCart();
    }

    function removeFromCart(index) {
      let cart = JSON.parse(localStorage.getItem("cart"));
      cart.splice(index, 1);
      localStorage.setItem("cart", JSON.stringify(cart));
      loadCart();
    }

    function clearCart() {
      localStorage.removeItem("cart");
      loadCart();
    }
```

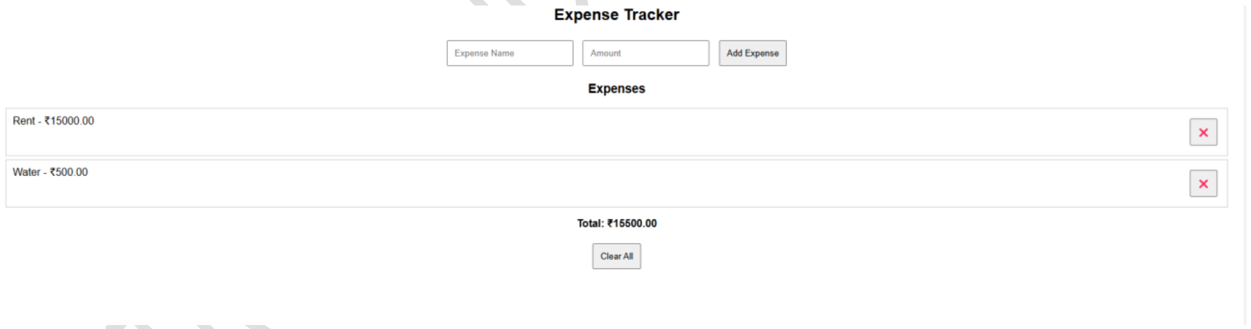
```
        displayProducts();
        loadCart();
    </script>

</body>
</html>
```

Sample Mini Project 2 : Expense Tracker with LocalStorage

A simple expense tracker that allows users to log expenses, save data in LocalStorage, and display a summary.

Features:

✓ **Add Expenses:** Users can enter an expense name and amount.
✓ **Persist Data:** Expenses are saved in **LocalStorage** and persist after page reload.
✓ **Remove Expenses:** Users can remove individual expenses.
✓ **Calculate Total Expenses:** Automatically updates the total amount.



Rent - ₹15000.00
Water - ₹500.00
Total: ₹15500.00

Day 10 Task

JS118. Store and retrieve a **username** using LocalStorage.
JS119. Save and load **dark/light mode settings** using LocalStorage.
JS120. Implement a **shopping cart** that remembers selected items using LocalStorage.
JS121. Store and retrieve **session-based login status** using SessionStorage.

JS122. Save **form input values** temporarily in SessionStorage and restore on page reload.
JS123. Store **last visited page** in LocalStorage and display it on the next visit.
JS124. Implement a **simple click counter** using LocalStorage.
JS125. Save and retrieve a **to-do list** using LocalStorage with JSON.
JS126. Convert a **JavaScript object to a JSON string** using JSON.stringify() and store it.
JS127. Convert a **JSON string back to a JavaScript object** using JSON.parse() and use it.
JS128. Create a **survey form** where responses are saved in SessionStorage.
JS129. Store the number of **page visits** in LocalStorage.
JS130. Create a **quiz system** where answers persist using SessionStorage and JSON.

Mini Project 1: Stopwatch with SessionStorage

A simple stopwatch that continues running even after page refresh.

Features:

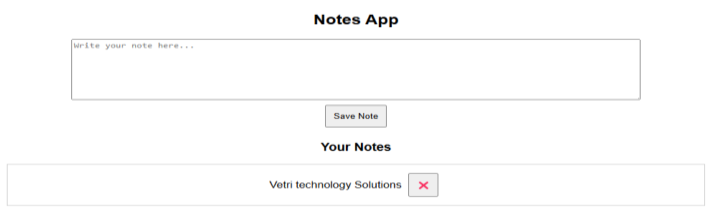
✓ Start, Stop, and Reset the timer.
✓ SessionStorage keeps the timer running even after page refresh.

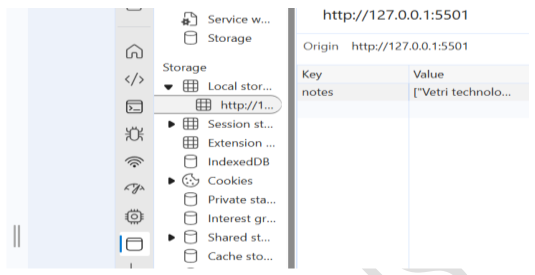
Mini Project 2: Notes App with LocalStorage

A simple note-taking app that allows users to add, delete, and save notes persistently.

Features:

✓ **Add Notes:** Users can write and save notes.
✓ **Persist Data:** Notes are stored in **LocalStorage** and remain after page reload.
✓ **Delete Notes:** Users can remove notes individually.





Key	Value
notes	["Vetri technolo...

Day 11

What is an Array of Objects?

An array of objects is a data structure where each element is an object containing key-value pairs. It is useful for storing structured data like user details, product lists, or API responses.

Why Use Arrays of Objects?

✓ Used in real-world applications like e-commerce, databases, and API responses.
✓ Helps in managing multiple records efficiently.
✓ Supports sorting, filtering, and mapping operations.

Example: Array of Objects

```
let students = [
  { name: "Alice", age: 22, grade: "A" },
  { name: "Bob", age: 24, grade: "B" },
  { name: "Charlie", age: 23, grade: "A" }
];

console.log(students);
```

Output:

```
[
  { name: 'Alice', age: 22, grade: 'A' },
  { name: 'Bob', age: 24, grade: 'B' },
  { name: 'Charlie', age: 23, grade: 'A' }
]
```

Accessing Elements in an Array of Objects

You can access, modify, and iterate over objects in an array.

Accessing a Specific Object

```
console.log(students[0]);  // First object
console.log(students[1].name);  // Output: "Bob"
```

Updating an Object

```
students[0].grade = "A+";
console.log(students[0]);  // Updated object
```

Looping Through the Array (Using forEach)

```
students.forEach(student => {
  console.log(`${student.name} has grade ${student.grade}`);
});
```

Output:

```
Alice has grade A+
Bob has grade B
Charlie has grade A
```

Array Methods with Objects

1. map(): Transform Array Elements

```
let studentNames = students.map(student => student.name);
console.log(studentNames); // ["Alice", "Bob", "Charlie"]
```

2. filter(): Get Specific Objects

```
let topStudents = students.filter(student => student.grade === "A");
console.log(topStudents);
```

3. find(): Find a Specific Object

```
let charlie = students.find(student => student.name === "Charlie");
console.log(charlie);
```

4. sort(): Sorting Objects

```
students.sort((a, b) => a.age - b.age);
console.log(students);
```

Sorted by age:

```
 [
   { name: "Alice", age: 22, grade: "A+" },
   { name: "Charlie", age: 23, grade: "A" },
   { name: "Bob", age: 24, grade: "B" }
]
```

Example 1: E-Commerce Product List

✅ **Use Case:** An online store has multiple products, and we store product details using an array of objects.

```
let products = [
   { id: 1, name: "Laptop", price: 75000, category: "Electronics", stock: 10 },
   { id: 2, name: "Headphones", price: 2500, category: "Accessories", stock: 50 },
   { id: 3, name: "Smartphone", price: 45000, category: "Electronics", stock: 20 },
   { id: 4, name: "Shoes", price: 3000, category: "Fashion", stock: 100 }
];

// Display all product names
products.forEach(product => console.log(product.name));

// Find a product by ID
let product = products.find(p => p.id === 2);
console.log("Product Found:", product);

// Filter products by category
let electronics = products.filter(p => p.category === "Electronics");
console.log("Electronics Products:", electronics);
```

✅ **Operations Performed:**
✔ **Retrieve** product details.
✔ **Find** a specific product by ID.
✔ **Filter** products by category.

Sample mini project 1:

Real-World Example 3: Student Marks Database

✅ **Use Case:** A school stores student records and calculates average marks.

```
let students = [
   { name: "John", marks: [80, 75, 90] },
   { name: "Emma", marks: [85, 95, 92] },
   { name: "David", marks: [70, 60, 65] }
];
```

```
// Calculate and display average marks
students.forEach(student => {
  let avgMarks = student.marks.reduce((sum, mark) => sum + mark, 0) /
student.marks.length;
  console.log(`${student.name} - Average Marks: ${avgMarks}`);
});
```

Sample mini project 2: Requirements for Customer Orders Management

1**. Store Order Details:** Maintain order ID, customer name, ordered items, and total amount in an array of objects.

2**. Retrieve Specific Order:** Allow searching for an order using the **order ID** or customer name.

3**. Calculate Total Revenue:** Sum up all order totals to get the **total revenue** from sales.

4**. Filter Orders by Customer:** Display all orders placed by a specific customer.

Day11. Tasks

JS131. Create an **array of employees** with name, age, and position.
JS132. Fetch and display a list of **products from an API**.
JS133. Sort an array of objects **by age in ascending order**.
JS134. Filter an array to show **only students with grade 'A'**.
JS135. Find a **specific product in an array** by name.
JS136. Convert an array of objects to **only an array of names**.
JS137. Modify an object **inside an array** and update its value.
JS138. Count how many objects meet **a specific condition**.
JS139. Implement a **search function** to find objects dynamically.
JS140. Group objects **based on a property value**.
JS141. Remove a specific object **from an array** by property.

JS142. Add **new objects dynamically** to an array.
JS143. Display the **highest-priced product** in an array.

Mini project:

1. Inventory Management System Requirements

- Store Product Details: Maintain product ID, name, category, quantity, and price in an array of objects.
- 2. Check Stock Availability: Allow searching for a product by ID or name to check stock status.
- 3. Update Inventory: Enable adding or removing stock when new shipments arrive or products are sold.
- 4. Calculate Total Inventory Value: Compute the total worth of all products in stock.

2. Library Book Management System Requirements

- 1. Store Book Records: Maintain book ID, title, author, genre, availability status, and borrower details in an array of objects.
- 2. Search Books: Allow searching for books by title, author, or genre.
- 3. Track Borrowed Books: Update book status when issued and return it upon submission.
- 4 List Available Books: Display only books that are currently available for borrowing.

Day 12

What is JSON (JavaScript Object Notation)?

JSON (JavaScript Object Notation) is a lightweight format for storing and transferring data. It is commonly used in APIs and web applications to exchange data between a server and a client.

Key Features:

✓ **Human-readable format**, similar to JavaScript objects.
✓ Supports **strings, numbers, arrays, objects, booleans, and null**.
✓ Used in **APIs, databases (MongoDB), and local storage**.
✓ Works with **JavaScript, Python, PHP, and more**.

JSON Syntax Rules

Example of a JSON Object:

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

Example of JSON Array of Objects:

```
[
   { "id": 1, "name": "John", "role": "Developer" },
   { "id": 2, "name": "Sarah", "role": "Designer" },
   { "id": 3, "name": "Michael", "role": "Manager" }
]
```

Example of Nested JSON:

```
{
  "student": {
    "name": "Emma",
    "age": 22,
    "courses": ["Math", "Science", "History"]
  }
}
```

How JSON Works in Real-World Projects

How JSON is used:
✓ Frontend (React, Vue, Angular): Fetch data from APIs and display it dynamically.
✓ Backend (Node.js, Django, Flask): Store and send data in JSON format.
✓ Databases (MongoDB, Firebase): Store JSON data for fast retrieval.
✓ LocalStorage/SessionStorage: Save user preferences in JSON.

How to Store & Retrieve an Array of Objects in a JSON File

1. Storing an Array of Objects in a JSON File (data.json)

Example JSON File (data.json):

```
[
  { "id": 1, "name": "Alice", "email": "alice@example.com" },
  { "id": 2, "name": "Bob", "email": "bob@example.com" }
]
```

2. Fetch JSON Data in JavaScript

Example: Fetching and Displaying JSON Data

```
fetch("data.json")
   .then(response => response.json()) // Convert response to JSON
   .then(data => console.log(data))   // Display JSON data
   .catch(error => console.error("Error:", error));
```

Output:

```
 [
   { id: 1, name: "Alice", email: "alice@example.com" },
   { id: 2, name: "Bob", email: "bob@example.com" }
]
```

3. Display JSON Data in HTML

Example: Fetch Data and Display in a Table

```
<table border="1">
   <thead>
      <tr>
         <th>ID</th>
         <th>Name</th>
         <th>Email</th>
      </tr>
   </thead>
   <tbody id="userTable"></tbody>
</table>

<script>
   fetch("data.json")
      .then(response => response.json())
```

```
        .then(users => {
            let table = document.getElementById("userTable");
            users.forEach(user => {
                let row = `<tr>
                    <td>${user.id}</td>
                    <td>${user.name}</td>
                    <td>${user.email}</td>
                </tr>`;
                table.innerHTML += row;
            });
        })
        .catch(error => console.error("Error:", error));
</script>
```

Real-World Use Case:

Used in admin dashboards, user lists, and API-based applications.

What is Asynchronous JavaScript?

Asynchronous JavaScript allows code execution to continue without waiting for a task to complete.

Why is it needed?

✓ Web applications often fetch data from a server, which takes time.
✓ Without async programming, the page freezes until the data loads.
✓ Async functions allow the browser to stay responsive while waiting.

Synchronous vs Asynchronous JavaScript

Feature	Synchronous (Blocking)	Asynchronous (Non-Blocking)
Execution	Runs one task at a time	Runs multiple tasks simultaneously
Blocking	Blocks the next code execution	Doesn't block execution
Speed	Slower for long tasks	Faster for UI and network operations
Example	File read/write	API calls, setTimeout, setInterval

Example of Synchronous Code (Blocking)

```
console.log("Start");
for (let i = 0; i < 1000000000; i++) {} // Blocking loop
console.log("End");
```

Example of Asynchronous Code (Non-Blocking)

```
console.log("Start");
setTimeout(() => console.log("Async Task Completed"), 2000);
console.log("End");
```

Output:

```
Start
End
Async Task Completed (after 2 seconds)
```

Asynchronous JavaScript Methods

1. setTimeout()

Executes a function after a delay

```
setTimeout(() => {
   console.log("This message appears after 3 seconds.");
}, 3000);
```

2. setInterval()

Repeats a function at a fixed interval

```
setInterval(() => {
   console.log("This message repeats every 2 seconds.");
}, 2000);
```

3. Callbacks (Old Method)

A function passed as an argument to another function

```
function fetchData(callback) {
   setTimeout(() => {
      callback("Data loaded!");
   }, 2000);
}

fetchData(message => console.log(message));
```

4. Promises (Modern Method)

Handles asynchronous operations with .then() and .catch()

```
let myPromise = new Promise((resolve, reject) => {
   setTimeout(() => resolve("Promise Resolved!"), 2000);
});

myPromise.then(data => console.log(data));
```

5. Async/Await (Best Method)

Simplifies handling asynchronous code

```
async function fetchData() {
   let response = await fetch("https://jsonplaceholder.typicode.com/users");
   let data = await response.json();
   console.log(data);
}
fetchData();
```

Best for real-world applications like:
✓ Fetching API data
✓ User authentication
✓ Database operations

Sample mini project 1: Weather App

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Weather App</title>
<style>
body {
font-family: Arial, sans-serif;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
height: 100vh;
background-color: #f4f4f4;
}
.container {
```

```
width: 350px;
background: white;
padding: 20px;
border-radius: 8px;
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
text-align: center;
}
input {
width: 90%;
padding: 10px;
margin: 10px 0;
border: 1px solid #ccc;
border-radius: 5px;
}
button {
width: 100%;
padding: 10px;
background: #007bff;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
}
button:hover {
background: #0056b3;
}
#weather {
margin-top: 15px;
font-size: 18px;
font-weight: bold;
}
#icon {
margin-top: 10px;
width: 50px;
```

```
height: 50px;
}
</style>
</head>
<body>
<div class="container">
<h2>Weather App</h2>
<input type="text" id="city" placeholder="Enter city name">
<button onclick="getWeather()">Get Weather</button>
<p id="weather"></p>
<img id="icon" style="display: none;">
</div>
<script>
async function getWeather() {
let city = document.getElementById("city").value.trim();
let weatherField = document.getElementById("weather");
let iconField = document.getElementById("icon");
if (!city) {
weatherField.textContent = "Please enter a city name.";
return;
}
weatherField.textContent = "Fetching weather...";
iconField.style.display = "none";
const apiKey = "0e2ffc16a6efd774caf9dbdabc9125bb"; // Replace with your
OpenWeather API key
const apiUrl =
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}
&units=metric`;
try {
const response = await fetch(apiUrl);
const data = await response.json();
if (data.cod !== 200) {
weatherField.textContent = "City not found!";
return;
}
let temp = data.main.temp;
let description = data.weather[0].description;
let icon = `http://openweathermap.org/img/wn/${data.weather[0].icon}.png`;
weatherField.textContent = `🌡 ${temp}°C - ${description}`;
iconField.src = icon;
iconField.style.display = "block";
} catch (error) {
weatherField.textContent = "Error fetching weather data!";
console.error(error);
}
}
</script>
</body>
</html>
```

Sample Mini project 2:

1. Employee Management System

Description: A simple web app to fetch, display, add, and delete employees stored in a JSON file.

Requirements:

1. Fetch and display **employee details** from employees.json.
2. Allow users to **add a new employee** and update the JSON file.
3. Implement a **delete button** to remove an employee from the list.
4. Use fetch() to retrieve and update the employee list dynamically.

Day 12 Tasks :

JS144. Convert a JavaScript object to a JSON string using JSON.stringify().
JS145. Convert a JSON string back to a JavaScript object using JSON.parse().
JS146. Create a JSON file (data.json) and fetch it using JavaScript.
JS147. Store an array of objects in LocalStorage as JSON and retrieve it.
JS148. Fetch JSON data from an API and display it in a table.
JS149. Modify a JSON object inside an array and update it.
JS150. Filter JSON data to display only objects matching a specific condition.

JS151. Sort an array of objects stored in JSON by a specific property (e.g., price, age).
JS152. Find an object inside a JSON array based on a property value.
JS153. Create a function that adds new objects to an existing JSON file.
JS154. Remove a specific object from an array inside a JSON file.
JS155. Fetch data from a JSON file and display it in a dropdown list.

JS156. Fetch data from an API using fetch() and handle errors gracefully using .catch().

Mini project :

E-Commerce Product Catalog (JSON + Filter & Sort)

✅ **Description:** Displays a list of products fetched from a JSON file and allows filtering and sorting.

Requirements:
1. Fetch and display product data (name, price, category) from products.json.
2. Implement price filtering (e.g., show products below $500).
3. Add sorting functionality (e.g., sort by price low to high).
4. Display product details when a user clicks on a product.

Real-World Use Case:
✔ Used in **e-commerce platforms** for product listings.

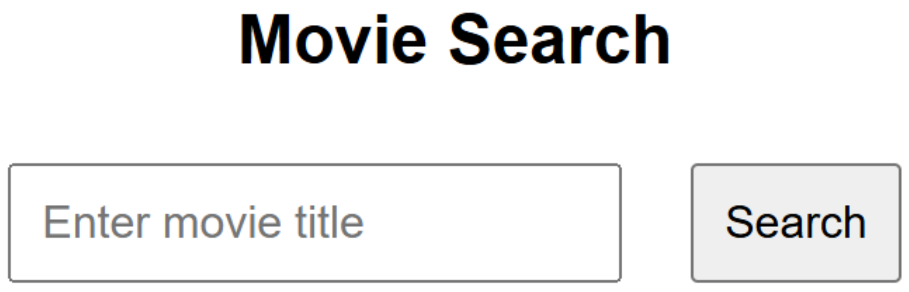
1. .

1. **Movie Search App (Fetch API + Async/Await)**

✅ **Fetches movie details from an API based on user input.**

Requirements:

2. User enters a movie title.
3. Fetch movie details (title, year, plot, poster) from an API.
4. Display the movie information dynamically.



Day 13

What is Fetch API?

The Fetch API is a modern way to make network requests in JavaScript. It replaces the older AJAX (XMLHttpRequest) method and is used to fetch resources, such as APIs, files, or remote data.

Key Features:

✓ Uses **Promises** (.then() & .catch()).
✓ **Simpler & cleaner** than AJAX.
✓ Supports **JSON, text, and other formats**.
✓ Works **asynchronously**, preventing UI blocking.

Fetch API Basic Syntax

```
fetch("https://jsonplaceholder.typicode.com/posts") // Fetch data from API
  .then(response => response.json()) // Convert response to JSON
  .then(data => console.log(data)) // Use the data
  .catch(error => console.error("Error:", error)); // Handle errors
```

Real-World Use Case:

✓ Used in web applications to fetch data from a server (e.g., fetching user profiles).

What is AJAX (Asynchronous JavaScript and XML)?

AJAX allows web pages to update data without reloading. It works by making HTTP requests using the XMLHttpRequest object.

Key Features:

✓ Works asynchronously to improve performance.
✓ Can send and receive data without refreshing the page.
✓ Older method, now replaced by Fetch API in modern JavaScript.

AJAX Basic Syntax Using XMLHttpRequest

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://jsonplaceholder.typicode.com/users", true);
xhr.onload = function() {
  if (xhr.status === 200) {
    console.log(JSON.parse(xhr.responseText));
  }
};
xhr.send();
```

Real-World Use Case:

✓ Used in live search suggestions, form submission without page reload, and dynamic content updates.

Fetch API vs. AJAX

Feature	Fetch API	AJAX (XMLHttpRequest)
Ease of Use	Simpler, uses Promises	More complex, uses callbacks
Data Handling	Supports JSON easily	Requires JSON.parse()
Error Handling	.catch() for errors	Requires manual onerror handling
Modern Support	Supported in **all modern browsers**	Older method, still supported but less used

✅ **Use Fetch API for modern web development!**

Sample mini project 1:Dictionary App (Fetch API)

✅ **Fetches word definitions using an online dictionary API.**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dictionary App</title>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin: 20px; }
    input, button { margin: 10px; padding: 10px; font-size: 16px; }
  </style>
</head>
<body>

  <h2>Dictionary App</h2>
  <input type="text" id="word" placeholder="Enter a word">
  <button id="getDefinition">Get Definition</button>
  <p id="definition"></p>

  <script>
    async function fetchDefinition() {
      let word = document.getElementById("word").value;
      let response = await
fetch(`https://api.dictionaryapi.dev/api/v2/entries/en/${word}`);
      let data = await response.json();

      if (data.title === "No Definitions Found") {
        document.getElementById("definition").innerText = "Word not found.";
        return;
      }

      document.getElementById("definition").innerText = `${data[0].word}:
${data[0].meanings[0].definitions[0].definition}`;
    }

    document.getElementById("getDefinition").addEventListener("click",
fetchDefinition);
  </script>

</body>
</html>
```

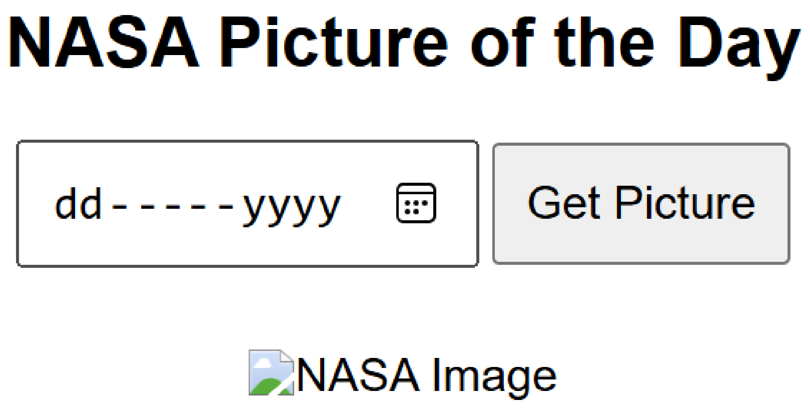
Sample mini project 2:

NASA Picture of the Day Viewer (Fetch API)

✅ **Fetches and displays NASA's Astronomy Picture of the Day (APOD).**

Features:

- Fetches NASA's daily space image using an API.
- Displays title, description, and image/video.
- Uses date selection to view previous pictures.


NASA Picture of the Day

Day 13 Tasks:

1. Fetch a **random joke** from an API and show it on the page.
2. Fetch **posts from an API** and display them in a list.
3. Use Fetch API to **submit a form to an API**.
4. Fetch **real-time weather data** using an API.
5. Fetch and display **GitHub user profiles**.
6. Create an **image gallery** using the Fetch API.
7. Implement **pagination** using Fetch API.

8. Use AJAX to fetch and display **comments dynamically**.
9. Fetch and display **currency exchange rates**.
10. Load data from an API **when a button is clicked**.
11. Handle **API errors gracefully**.
12. Implement **search functionality using Fetch API**.

Mini project

1. **Typing Speed Tester (Fetch API)**

✅ **Measures typing speed based on fetched random sentences.**

Features:

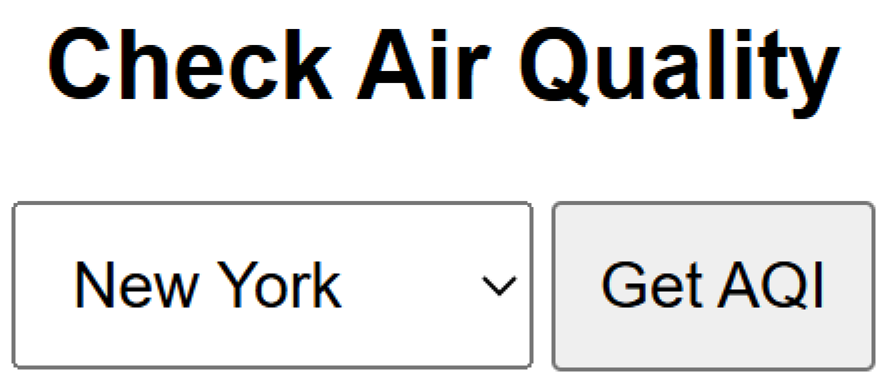
- Fetches **random sentences** using an API.
- Tracks **time taken to type the sentence**.
- Displays **typing speed (words per minute).**

2. **Real-Time Air Quality Checker (Fetch API)**

✅ **Fetches and displays live air quality index (AQI) of a selected city.**

📌 **Features:**

- Fetches **real-time air quality data** from an API.
- Displays **AQI, pollutants level, and health advice**.
- Uses **dropdown selection for multiple cities**.



Day 14

◆ **What is Error Handling in JavaScript?**

Error handling in JavaScript refers to **detecting, managing, and preventing errors** in a program to avoid unexpected crashes. JavaScript provides different mechanisms for handling errors effectively.

✅ **Why is error handling important?**

- Prevents **program crashes**.
- Improves **user experience**.
- Helps **debug and identify** issues quickly.
- Ensures **smooth execution** of the code.

Types of Errors in JavaScript

Error Type	Description	Example
Syntax Error	Occurs when there is an incorrect syntax.	console.log("Hello" (Missing closing))

Error Type	Description	Example
Reference Error	Occurs when accessing an undefined variable.	console.log(x); (x is not defined)
Type Error	Happens when performing an invalid operation on a variable.	null.toUpperCase(); (Cannot read properties of null)
Range Error	Thrown when a number is outside a valid range.	let arr = new Array(-1); (Array size cannot be negative)
URI Error	Happens when using an incorrect URL function.	decodeURIComponent("%"); (Invalid URI component)

How to Handle Errors in JavaScript?

Try...Catch for Handling Errors

✅ Used to catch and handle errors without stopping program execution.

```
try {
   let x = y + 10; // y is not defined
} catch (error) {
   console.log("An error occurred:", error.message);
}
```

Output:

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

🔹 **Without try...catch, the script would crash**.
🔹 **With try...catch, execution continues smoothly**.

Try...Catch with Finally

✅ The finally block always executes, whether an error occurs or not.

```
try {
   let x = 10 / 0;
} catch (error) {
   console.log("Error:", error.message);
} finally {
   console.log("Execution completed!");
}
```

Output:

```
Execution completed!
```

Throwing Custom Errors

✅ We can manually throw errors using throw.

```
function checkAge(age) {
   if (age < 18) {
      throw new Error("Age must be 18 or above!");
   }
   return "Access Granted";
}
try {
   console.log(checkAge(16));
} catch (error) {
   console.log("Error:", error.message);
}
```

Output:

```
Error: Age must be 18 or above!
```

Using console Methods for Debugging

✅ JavaScript provides console methods to debug errors.

```
console.log("This is a log message");  // Normal log
console.warn("This is a warning");     // Warning message
console.error("This is an error!");    // Error message
console.table([{ name: "John", age: 30 }, { name: "Alice", age: 25 }]); // Display
table
```

✅ **Useful in debugging real-time issues in applications.**

Debugging with debugger

✅ **The debugger statement pauses execution in the browser's Developer Tools.**

```
let x = 10;
debugger; // Code execution pauses here
console.log(x);
```

Example Code :

1. **Login Form Validation (Error Handling)**

✅ **Ensures proper user input and handles missing fields.**

📌 **Features:**

- Throws an error when **fields are empty**.
- Displays **custom error messages**.
- Uses try...catch to handle errors.

```
<form id="loginForm">
  <input type="text" id="username" placeholder="Enter username">
  <input type="password" id="password" placeholder="Enter password">
  <button type="submit">Login</button>
</form>
<p id="errorMessage"></p>

<script>
  document.getElementById("loginForm").addEventListener("submit",
function(event) {
      event.preventDefault();
      try {
        let username = document.getElementById("username").value;
        let password = document.getElementById("password").value;

        if (!username || !password) {
          throw new Error("All fields are required!");
        }

        alert("Login successful!");
      } catch (error) {
        document.getElementById("errorMessage").innerText = error.message;
      }
  });
</script>
```

✅ **Real-World Use Case:**

✔ Used in **web authentication systems** to validate form data.

2. API Request Error Handling

✅ **Handles errors when fetching data from an API.**

```
async function fetchData() {
   try {
      let response = await fetch("https://jsonplaceholder.typicode.com/users");
      if (!response.ok) {
         throw new Error("API Error: Unable to fetch data!");
      }
      let data = await response.json();
      console.log(data);
   } catch (error) {
      console.error("Error:", error.message);
   }
}

fetchData();
```

✅ **Real-World Use Case:**
✔ Used in **web apps where API failures must be managed gracefully**.

Sample Mini Project 1: Calculator with Error Handling

✅ **Handles input errors such as division by zero.**

```
<input type="number" id="num1">
<input type="number" id="num2">
<button onclick="calculate()">Divide</button>
<p id="result"></p>
```

```
<script>
  function calculate() {
    try {
      let num1 = parseFloat(document.getElementById("num1").value);
      let num2 = parseFloat(document.getElementById("num2").value);

      if (isNaN(num1) || isNaN(num2)) {
        throw new Error("Invalid number input!");
      }
      if (num2 === 0) {
        throw new Error("Cannot divide by zero!");
      }

      document.getElementById("result").innerText = `Result: ${num1 / num2}`;
    } catch (error) {
      document.getElementById("result").innerText = `Error: ${error.message}`;
    }
  }
</script>
```

Sample Mini Project 2: Debugging Dashboard

✅ **Displays error logs using console.error() & LocalStorage.**

```
function logError(message) {
   console.error("Error Logged:", message);
   localStorage.setItem("errorLog", message);
}

try {
   let x = undefined;
   console.log(x.toUpperCase());
} catch (error) {
   logError(error.message);
}
```

Day 13 Tasks:

1. Write a program that **handles a syntax error** using try...catch.
2. Catch and handle a **Reference Error** when trying to use an undefined variable.
3. Use try...catch to **handle division by zero** error.
4. Throw a **custom error** when a user inputs an invalid age.
5. Create a function that **throws a TypeError** if input is not a number.
6. Implement **nested try...catch blocks** to handle multiple errors.
7. Use console.table() to **display an array of objects in tabular form**.
8. Debug an application using console.warn() and console.error().
9. Implement a **debugger breakpoint** inside a loop
10. Handle an **API call failure** using try...catch.
11. Use finally to **execute cleanup code** even if an error occurs.
12. Prevent **infinite recursion** by handling a function call limit.
13. Implement a **real-world error logging system** using LocalStorage.

Mini project :

1. **User Registration Form with Validation & Error Handling**

✓ **Ensures all required fields are filled and checks for valid email format.**

Features:

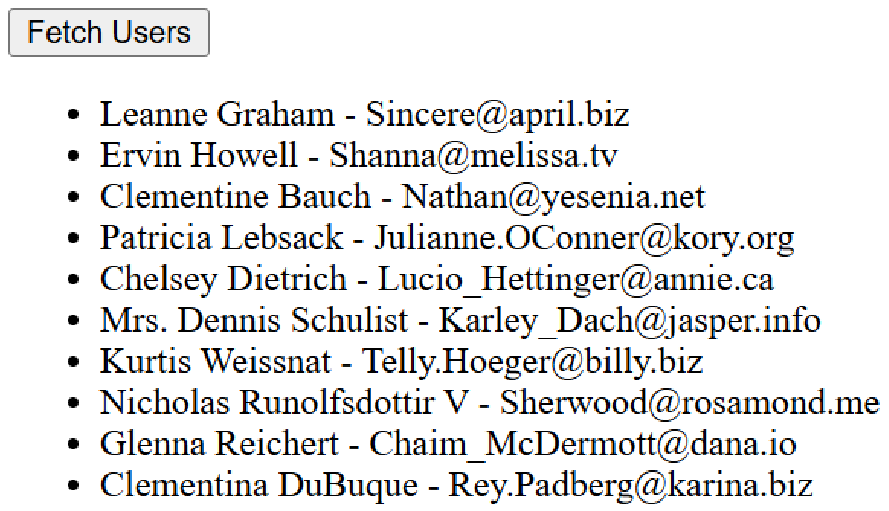
- Throws an error when **fields are empty** or **email is invalid**.
- Uses try...catch to **handle validation errors**.
- Displays **error messages below the input fields**.

2. **API Response Logger**

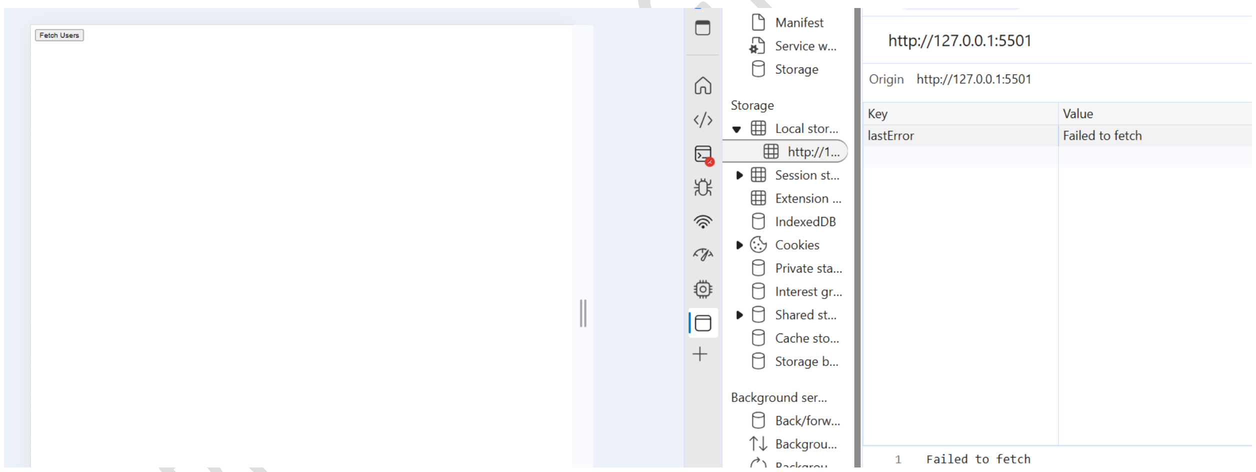
✓ **Logs API response errors and stores them in LocalStorage for debugging.**

Features:

- Logs **successful API responses**.
- Catches **API failures** and stores errors in LocalStorage.
- Uses try...catch for **error handling**.



- Leanne Graham - Sincere@april.biz
- Ervin Howell - Shanna@melissa.tv
- Clementine Bauch - Nathan@yesenia.net
- Patricia Lebsack - Julianne.OConner@kory.org
- Chelsey Dietrich - Lucio_Hettinger@annie.ca
- Mrs. Dennis Schulist - Karley_Dach@jasper.info
- Kurtis Weissnat - Telly.Hoeger@billy.biz
- Nicholas Runolfsdottir V - Sherwood@rosamond.me
- Glenna Reichert - Chaim_McDermott@dana.io
- Clementina DuBuque - Rey.Padberg@karina.biz



http://127.0.0.1:5501

Origin http://127.0.0.1:5501

Key	Value
lastError	Failed to fetch

```
Failed to fetch
```

Day 14 & 15

Main Projects

1. **Chat Application (Real-time Messaging)**

Scenario: A simple chat app to send and receive messages.

✓ **Requirements:**

- Input field to enter messages.
- Display sent & received messages.
- Store messages in **LocalStorage**.
- Add timestamps for messages.

2. **Drag and Drop File Uploader**

Scenario: Allows users to **drag & drop** files for upload.

✓ **Requirements:**

- A drop area for files.
- Preview uploaded file.
- Display file name & size.
- Simulate upload using **JavaScript events**.

3. **Music Player**

Scenario: A **custom music player** with play, pause, and skip controls.

✓ **Requirements:**

- Play, pause, and stop buttons.
- Volume control.
- Display **song title & duration**.

- Next & previous track buttons.

4. **Image Slider**

Scenario: A **carousel** to slide through images.

✓ **Requirements:**

- Auto and manual **slide navigation**.
- Next & previous buttons.
- Thumbnail navigation.

5. **Loan EMI Calculator**

Scenario: Calculate **monthly EMI payments** based on loan amount.

✓ **Requirements:**

- Input fields for loan amount, interest, and years.
- Calculate **monthly EMI**.
- Display **total payable amount**.

6. **Dark Mode Toggle**

Scenario: Enable **dark mode** with a toggle button.

✓ **Requirements:**

- Button to switch between **light and dark mode**.
- Save preference in **LocalStorage**.

7. **Student Management System (CRUD Operations)**

Scenario: A web application to add, update, delete, and display student records using an array of objects.

Features:

✓ **Add, edit, and delete students** from a dynamic table.

✓ **Store student data** (name, age, class, grade) as an **array of objects**.

✓ **Save data in LocalStorage** so that records persist after page reload.

✓ **Search functionality** to find students by name.

8. **Task Manager / To-Do List (Advanced)**

Description: A **task manager** that allows users to **add, edit, delete, and filter tasks** stored as an array of objects.

Features:

✓ **Add and delete tasks** dynamically.

✓ **Categorize tasks** (e.g., Work, Personal, Study).

✓ **Sort tasks by due date** (oldest to newest).

✓ **Mark tasks as completed** and filter them based on status.

9. **Online Bookstore Management System**

Description: A **book catalog** where users can **add, search, and filter books** stored as an array of objects.

Features:

✓ **Display a list of books** with details (title, author, price, category).
✓ **Filter books by category and price range**.
✓ **Search books by title or author** dynamically.
✓ **Add books to a shopping cart** (stored in LocalStorage).

10. Employee Attendance Management System

Description: A system where employers can **mark attendance, view records, and store employee data**.

Features:

✓ **Store employee records** (name, ID, department, attendance status).
✓ **Mark attendance** as Present or Absent.
✓ **Filter employees** by department or status.
✓ **Sort employees by name or attendance count**.

****