Unit-3: Client-Side Scripting (JavaScript) and DHTML

Overview of JavaScript

What is JavaScript?

JavaScript is a **client-side scripting language** that runs directly in the user's web browser without requiring server processing. It is an object-oriented, loosely-typed, multi-platform language designed to make web pages interactive and dynamic.

Key Characteristics of JavaScript:

- **Client-Side Execution**: Code runs on the client's browser, not on the server
- **Loosely Typed Language**: Variables can hold any data type without explicit declaration
- **Object-Oriented**: Supports objects, properties, and methods
- **Platform Independent**: Works across different browsers and operating systems
- **Event-Driven**: Responds to user actions like clicks, mouse movements, and keyboard input
- **Lightweight**: Simple syntax and relatively easy to learn

Benefits of Client-Side Scripting:

1. **Reduced Server Load**: Processing is done on the client machine instead of the server
2. **Faster Execution**: Scripts are downloaded and executed directly on the user's computer
3. **Immediate User Feedback**: Response to user actions is instantaneous without server communication
4. **Input Validation**: Form data can be validated before submission to the server
5. **Enhanced User Experience**: Enables animations, interactive effects, and dynamic content changes
6. **Offline Functionality**: Basic operations can work without server connection
7. **Reduced Bandwidth**: Minimizes unnecessary data transmission to the server

Disadvantages of Client-Side Scripting:

1. **Browser Dependency**: Functionality depends on browser type, version, and JavaScript support

2. **Security Limitations**: Cannot access protected server resources or perform sensitive operations
3. **Code Visibility**: Source code is visible to all users (security risk)
4. **No Guarantee of Execution**: Users can disable JavaScript in their browsers
5. **Limited File System Access**: Cannot directly access user's file system for security reasons
6. **Debugging Complexity**: Harder to debug compared to server-side code

Use Cases of Client-Side Scripting:

- Building interactive user interface components (dropdown menus, buttons, navigation)
- Animations and graphical effects
- Form validation before server submission
- Real-time content updates without page refresh
- Dynamic HTML manipulation
- Cookie and local storage management
- Mathematical calculations and data processing

Simple JavaScript

How to Implement JavaScript:

JavaScript can be implemented in HTML documents in three ways:

1. Inline JavaScript (In HTML Elements)

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

2. **Embedded JavaScript (In

3. **External JavaScript (In Separate File)**

HTML File:
```html
<script src="script.js"></script>
```

script.js:

```
console.log("This is external JavaScript");
```

Simple JavaScript Examples:

Example 1: Display Content

```
document.write("Hello, JavaScript World!");
document.getElementById("myDiv").innerHTML = "Content updated!";
```

Example 2: Basic Calculation

```
var a = 10;
var b = 20;
var sum = a + b;
console.log("Sum: " + sum);  // Output: Sum: 30
```

Example 3: Simple Function Call

```
function greet() {
  alert("Welcome to JavaScript!");
}
greet();  // Calls the function
```

Variables in JavaScript

What are Variables?

Variables are containers for storing data values. In JavaScript, you can declare variables using `var`, `let`, or `const` keywords.

Variable Declaration Methods:

1. Using `var` (Function-Scoped)

```
var name = "John";
var age = 25;
var price = 99.99;
```

2. Using `let` (Block-Scoped)

```
let country = "India";
let city = "Bangalore";
```

3. Using `const` (Block-Scoped, Immutable)

```
const PI = 3.14159;
const MAX_USERS = 100;
```

Variable Naming Rules:

- Must begin with a letter, underscore (_), or dollar sign ($)
- Can contain letters, digits, underscores, and dollar signs
- Are case-sensitive (name, Name, NAME are different variables)

- Cannot contain spaces
- Avoid using JavaScript reserved words

Data Types in JavaScript:

JavaScript supports **7 primitive data types** and **1 non-primitive type**:

Primitive Data Types:

Type	Description	Example
String	Text enclosed in quotes	`"Hello"`, `'World'`
Number	Integer or decimal number	`42`, `3.14`
Boolean	True or false value	`true`, `false`
Undefined	Variable declared but not assigned	`var x;`
Null	Intentional absence of value	`var x = null;`
Symbol	Unique identifier (ES6)	`Symbol("id")`
BigInt	Large integer beyond Number limit	`1234567890123456789012n`

Non-Primitive Data Type:

Type	Description	Example
Object	Collection of key-value pairs	`{name: "John", age: 25}`

Type Checking:

```
typeof("Hello");      // "string"
typeof(42);           // "number"
typeof(true);         // "boolean"
typeof(undefined);    // "undefined"
typeof({});           // "object"
typeof(null);         // "object" (exception)
```

Variable Scope:

Global Scope: Variables declared outside all functions are globally accessible.

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

function test() {
  console.log(globalVar);  // Can access global variable
}
```

Local Scope (Function Scope with var): Variables declared with `var` inside a function are only accessible within that function.

```
function test() {
  var localVar = "I am local";
  console.log(localVar);  // Works
}
console.log(localVar);    // Error: undefined
```

Block Scope (with let and const): Variables declared with `let` and `const` are scoped to the nearest enclosing block (if, for, while, etc.).

```
if (true) {
  let blockVar = "I am block-scoped";
  console.log(blockVar);  // Works
}
console.log(blockVar);    // Error: undefined
```

Hoisting:

Hoisting is JavaScript's default behavior of moving variable and function declarations to the top of their scope during compilation.

```
console.log(x);  // Output: undefined (not an error)
var x = 5;
console.log(x);  // Output: 5
```

Note: With `let` and `const`, hoisting works differently—they are in a "temporal dead zone" and cannot be accessed before declaration.

Functions in JavaScript

What are Functions?

Functions are reusable blocks of code that perform specific tasks. They help organize code, reduce repetition, and make programs more maintainable.

Function Declaration:

Function Declaration (Named Function)

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

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

Function Expression (Anonymous Function)

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

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

Arrow Function (ES6)

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

const square = x => x * x;  // Single parameter, shorter syntax

console.log(subtract(10, 3));  // Output: 7
console.log(square(4));        // Output: 16
```

Function Parameters and Arguments:

```
function greetUser(name, age) {
  console.log("Hello, " + name + "! You are " + age + " years old.");
}

greetUser("Alice", 25);  // Arguments: "Alice", 25
```

Return Statement:

```
function calculateAge(birthYear) {
  var currentYear = new Date().getFullYear();
  return currentYear - birthYear;
}

var age = calculateAge(1995);
console.log(age);  // Output: 29 (or current year - 1995)
```

Default Parameters:

```
function welcome(name = "Guest", message = "Welcome!") {
  console.log(message + ", " + name);
}

welcome();                    // Output: Welcome!, Guest
welcome("John");              // Output: Welcome!, John
welcome("Jane", "Hello");     // Output: Hello, Jane
```

Variable Number of Arguments:

```
function sum() {
  let total = 0;
  for (let i = 0; i < arguments.length; i++) {
    total += arguments[i];
  }
  return total;
}

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

Conditions (Conditional Statements)

Conditional statements control the flow of execution based on specific conditions.

1. if Statement

Executes code only if a condition is true.

```
var age = 18;

if (age >= 18) {
  console.log("You are an adult");
}
```

2. if...else Statement

Executes one block if condition is true, another if false.

```
var marks = 75;

if (marks >= 60) {
  console.log("Pass");
} else {
  console.log("Fail");
}
```

3. if...else if...else Statement

Tests multiple conditions.

```
var score = 85;

if (score >= 90) {
  console.log("Grade: A");
```

```
} else if (score >= 80) {
  console.log("Grade: B");
} else if (score >= 70) {
  console.log("Grade: C");
} else {
  console.log("Grade: F");
}
```

4. switch Statement

Selects one of many code blocks to execute.

```
var day = 3;

switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  default:
    console.log("Invalid day");
}
// Output: Wednesday
```

5. Ternary Operator

Shorthand for if…else.

```
var age = 20;
var status = age >= 18 ? "Adult" : "Minor";
console.log(status);  // Output: Adult
```

Logical Operators:

Operator	Description	Example
`&&` (AND)	True if both conditions are true	`x > 5 && x < 10`
`||` (OR)	True if at least one condition is true	`x === 1 || x === 2`
`!` (NOT)	Reverses the boolean value	`!(x > 5)`

Loops and Repetitions

Loops execute a block of code repeatedly as long as a specified condition is true.

1. for Loop

Used when you know the exact number of iterations.

```
for (var i = 1; i <= 5; i++) {
  console.log("Number: " + i);
}
// Output: Number: 1, 2, 3, 4, 5
```

Syntax Breakdown: - **Initialization** (`i = 1`): Sets starting value - **Condition** (`i <= 5`): Loop continues while true - **Increment** (`i++`): Updates variable after each iteration

2. while Loop

Executes as long as condition is true (pre-test loop).

```
var i = 1;

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

3. do...while Loop

Executes at least once, then repeats while condition is true (post-test loop).

```
var i = 1;

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

Note: Even if condition is false initially, the code block executes once.

4. for...in Loop

Loops through properties of an object.

```
var person = {name: "John", age: 30, city: "NYC"};

for (var key in person) {
  console.log(key + ": " + person[key]);
```

```
}
// Output: name: John, age: 30, city: NYC
```

5. for...of Loop

Loops through values of an iterable object (array, string).

```
var fruits = ["Apple", "Banana", "Orange"];

for (var fruit of fruits) {
  console.log(fruit);
}
// Output: Apple, Banana, Orange
```

Loop Control Statements:

break Statement Exits the loop immediately.

```
for (var i = 1; i <= 10; i++) {
  if (i === 5) break;
  console.log(i);  // Output: 1, 2, 3, 4
}
```

continue Statement Skips the current iteration and continues with next.

```
for (var i = 1; i <= 5; i++) {
  if (i === 3) continue;
  console.log(i);  // Output: 1, 2, 4, 5
}
```

JavaScript Objects

What are Objects?

Objects are collections of related properties and methods. They represent real-world entities with characteristics and behaviors.

Creating Objects:

1. Object Literal Notation

```
var car = {
  brand: "BMW",
  model: "X5",
  year: 2023,
  color: "black",
```

```
  // Method
  displayInfo: function() {
    console.log(this.brand + " " + this.model);
  }
};

console.log(car.brand);       // Output: BMW
console.log(car["model"]);    // Output: X5
car.displayInfo();             // Output: BMW X5
```

2. Constructor Function

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

  this.getAge = function() {
    return new Date().getFullYear() - this.year;
  };
}

var myCar = new Car("Honda", "Civic", 2020);
console.log(myCar.brand);      // Output: Honda
console.log(myCar.getAge());  // Output: 4 (or current year - 2020)
```

3. Object.create() Method

```
var carProto = {
  getInfo: function() {
    return this.brand + " " + this.model;
  }
};

var newCar = Object.create(carProto);
newCar.brand = "Tesla";
newCar.model = "Model S";
console.log(newCar.getInfo());  // Output: Tesla Model S
```

Accessing Object Properties:

```
var student = {name: "John", roll: 101, gpa: 3.8};

// Dot notation
console.log(student.name);       // Output: John

// Bracket notation
```

```
console.log(student["roll"]);   // Output: 101

// Dynamic property access
var prop = "gpa";
console.log(student[prop]);     // Output: 3.8
```

Adding and Deleting Properties:

```
var book = {title: "JavaScript", author: "John"};

// Add new property
book.year = 2023;
book["pages"] = 450;

console.log(book);  // {title: "JavaScript", author: "John", year: 2023, pages: 450}

// Delete property
delete book.year;
console.log(book);  // {title: "JavaScript", author: "John", pages: 450}
```

Object Methods:

Methods are functions that belong to an object.

```
var calculator = {
  add: function(a, b) {
    return a + b;
  },

  subtract: function(a, b) {
    return a - b;
  },

  multiply: function(a, b) {
    return a * b;
  }
};

console.log(calculator.add(10, 5));       // Output: 15
console.log(calculator.subtract(10, 5));  // Output: 5
console.log(calculator.multiply(10, 5));  // Output: 50
```

The `this` Keyword:

The `this` keyword refers to the current object.

```
var person = {
  firstName: "John",
```

```
  lastName: "Doe",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

console.log(person.fullName());  // Output: John Doe
```

Object Properties and Methods Listing:

```
var obj = {name: "John", age: 30, city: "NYC"};

// Get all keys
console.log(Object.keys(obj));        // ["name", "age", "city"]

// Get all values
console.log(Object.values(obj));      // ["John", 30, "NYC"]

// Get key-value pairs
console.log(Object.entries(obj));     // [["name", "John"], ["age", 30], ["city", "NYC"]]
```

JavaScript Built-In Objects

1. String Object

Methods for string manipulation.

```
var str = "JavaScript";

console.log(str.length);           // 10
console.log(str.toUpperCase());    // JAVASCRIPT
console.log(str.toLowerCase());    // javascript
console.log(str.charAt(0));        // J
console.log(str.indexOf("Script")); // 4
console.log(str.substring(0, 4));  // Java
console.log(str.slice(0, 4));      // Java
console.log(str.split(""));        // Array of characters
console.log(str.replace("Java", "Type")); // TypeScript
console.log(str.includes("Script")); // true
```

2. Array Object

Methods for array manipulation.

```
var arr = [1, 2, 3, 4, 5];
```

```
console.log(arr.length);           // 5
console.log(arr.push(6));          // [1,2,3,4,5,6]
console.log(arr.pop());            // Returns 6, array becomes [1,2,3,4,5]
console.log(arr.shift());          // Returns 1, array becomes [2,3,4,5]
console.log(arr.unshift(0));       // [0,2,3,4,5]
console.log(arr.slice(1, 3));      // [2,3]
console.log(arr.join("-"));        // "0-2-3-4-5"
console.log(arr.reverse());        // [5,4,3,2,0]
console.log(arr.sort());           // [0,2,3,4,5]
console.log(arr.indexOf(3));       // 2
console.log(arr.includes(4));      // true
```

3. Date Object

Methods for date and time operations.

```
// Create Date object
var today = new Date();
var birthday = new Date("1995-12-25");
var specificDate = new Date(2023, 11, 25, 10, 30, 0);

// Get date components
console.log(today.getFullYear());  // Current year
console.log(today.getMonth());     // 0-11 (0 = January)
console.log(today.getDate());      // 1-31
console.log(today.getDay());       // 0-6 (0 = Sunday)
console.log(today.getHours());     // 0-23
console.log(today.getMinutes());   // 0-59
console.log(today.getSeconds());   // 0-59

// Set date components
today.setFullYear(2024);
today.setMonth(5);                 // June
today.setDate(15);

// Convert to string
console.log(today.toString());     // Full date-time string
console.log(today.toDateString()); // Date only
console.log(today.toTimeString()); // Time only
```

4. Math Object

Mathematical operations and constants.

```
console.log(Math.PI);              // 3.14159...
console.log(Math.E);               // 2.71828...
```

```
// Rounding methods
console.log(Math.round(4.6));      // 5
console.log(Math.floor(4.9));      // 4
console.log(Math.ceil(4.1));       // 5
console.log(Math.abs(-10));        // 10
console.log(Math.sqrt(16));        // 4
console.log(Math.pow(2, 3));       // 8
console.log(Math.max(5, 10, 3));   // 10
console.log(Math.min(5, 10, 3));   // 3
console.log(Math.random());        // Random number 0-1

// Generate random integer between 1 and 10
var random = Math.floor(Math.random() * 10) + 1;
```

5. Number Object

Methods for number operations.

```
var num = 42.5;

console.log(num.toFixed(1));       // "42.5"
console.log(num.toString());       // "42.5"
console.log(Number.isInteger(42)); // true
console.log(Number.isNaN(NaN));    // true
console.log(Number.MAX_VALUE);     // Largest number
console.log(Number.MIN_VALUE);     // Smallest positive number
```

DOM (Document Object Model)

What is the DOM?

The **Document Object Model (DOM)** is a programming interface that represents HTML documents as a tree structure. It allows JavaScript to access, manipulate, and modify HTML elements dynamically.

DOM Hierarchy:

```
window
  document
     html
        head
           title
           meta
           link
        body
            h1
```

```
p
div
...
```

Key DOM Objects:

1. window Object The global object in the browser, contains all other objects.

```
console.log(window.innerWidth);     // Browser window width
console.log(window.innerHeight);    // Browser window height
console.log(window.location.href); // Current URL
console.log(window.document);       // Document object

// Methods
window.alert("Alert message");
window.confirm("Continue?");
window.prompt("Enter name:", "");
window.open("https://example.com");
window.close();
```

2. document Object Represents the HTML page loaded in the browser.

```
console.log(document.title);        // Page title
console.log(document.url);          // Page URL
console.log(document.body);         // Body element
console.log(document.documentElement); // Root html element
```

3. location Object Contains the URL of the current page.

```
console.log(location.href);         // Full URL
console.log(location.hostname);     // Domain name
console.log(location.pathname);     // Path
console.log(location.protocol);     // Protocol (http/https)

location.href = "https://example.com";  // Navigate to URL
location.reload();                      // Reload page
```

Selecting DOM Elements:

By ID

```
var elem = document.getElementById("myId");
```

By Class Name

```
var elems = document.getElementsByClassName("myClass");
```

By Tag Name

```
var elems = document.getElementsByTagName("p");
```

By CSS Selector (Single)

```
var elem = document.querySelector(".myClass");
var elem = document.querySelector("#myId");
```

By CSS Selector (Multiple)

```
var elems = document.querySelectorAll("p.myClass");
```

Manipulating DOM Content:

```
var div = document.getElementById("myDiv");

// Get/Set inner HTML
div.innerHTML = "<h1>Hello</h1>";
console.log(div.innerHTML);

// Get/Set inner text
div.innerText = "Just text";
console.log(div.innerText);

// Get/Set text content
div.textContent = "Text content";
```

Manipulating DOM Attributes:

```
var link = document.getElementById("myLink");

// Get attribute
console.log(link.getAttribute("href"));

// Set attribute
link.setAttribute("href", "https://example.com");
link.setAttribute("target", "_blank");

// Remove attribute
link.removeAttribute("title");

// Check if attribute exists
console.log(link.hasAttribute("href"));
```

Manipulating DOM Styles:

```
var box = document.getElementById("myBox");

// Modify inline styles
box.style.backgroundColor = "blue";
box.style.color = "white";
box.style.padding = "20px";
box.style.fontSize = "18px";
box.style.border = "2px solid black";
box.style.display = "none";      // Hide element
box.style.display = "block";     // Show element
```

Adding/Removing DOM Elements:

```
// Create new element
var newPara = document.createElement("p");
newPara.textContent = "This is new paragraph";

// Add to parent
document.body.appendChild(newPara);

// Insert before element
var firstPara = document.querySelector("p");
document.body.insertBefore(newPara, firstPara);

// Remove element
newPara.remove();
// Or: parent.removeChild(newPara);

// Replace element
var oldElem = document.getElementById("old");
var newElem = document.createElement("div");
oldElem.parentNode.replaceChild(newElem, oldElem);
```

DOM Navigation:

```
var elem = document.querySelector("p");

console.log(elem.parentNode);      // Parent element
console.log(elem.parentElement);   // Parent element (same)
console.log(elem.childNodes);      // All child nodes
console.log(elem.children);        // All child elements
console.log(elem.firstChild);      // First child node
console.log(elem.firstElementChild); // First child element
console.log(elem.lastChild);       // Last child node
console.log(elem.lastElementChild); // Last child element
```

```
console.log(elem.nextSibling);     // Next sibling node
console.log(elem.nextElementSibling); // Next sibling element
console.log(elem.previousSibling); // Previous sibling node
console.log(elem.previousElementSibling); // Previous sibling element
```

Forms and Validation

HTML Form Overview:

```
<form id="myForm">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name" required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

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

  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>
```

JavaScript Form Validation:

Basic Validation with Conditional Logic

```
function validateForm() {
  // Get form values
  var name = document.getElementById("name").value;
  var email = document.getElementById("email").value;
  var password = document.getElementById("password").value;

  // Clear previous error messages
  document.getElementById("nameError").textContent = "";
  document.getElementById("emailError").textContent = "";
  document.getElementById("passwordError").textContent = "";

  let isValid = true;

  // Validate name
  if (name === "" || /\d/.test(name)) {
    document.getElementById("nameError").textContent = "Please enter a valid name";
    isValid = false;
  }
```

```
  // Validate email
  if (email === "" || !email.includes("@") || !email.includes(".")) {
    document.getElementById("emailError").textContent = "Please enter a valid email";
    isValid = false;
  }

  // Validate password
  if (password === "" || password.length < 6) {
    document.getElementById("passwordError").textContent = "Password must be at least 6 char
    isValid = false;
  }

  return isValid;
}

// Attach validation to form submit
document.getElementById("myForm").addEventListener("submit", function(e) {
  if (!validateForm()) {
    e.preventDefault();  // Prevent form submission
  }
});
```

Validation with Regular Expressions

```
function validateWithRegex(firstName, email, password) {
  // Name: letters only
  const nameRegex = /^[a-zA-Z\s]{3,30}$/;
  if (!nameRegex.test(firstName)) {
    return "Name must be 3-30 letters";
  }

  // Email format
  const emailRegex = /^([a-zA-Z0-9._%+-]+)@([a-zA-Z0-9.-]+)\.([a-zA-Z]{2,})$/;
  if (!emailRegex.test(email)) {
    return "Invalid email format";
  }

  // Password: min 8 chars, 1 uppercase, 1 number, 1 special char
  const passRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%?&])[A-Za-z\d@$!%?&]{8,}$/;
  if (!passRegex.test(password)) {
    return "Password must have 8+ chars, uppercase, number, special character";
  }

  return "Valid";
}
```

```
console.log(validateWithRegex("John Doe", "john@example.com", "Pass@1234"));
```

Real-Time Validation (On Input)

```
var emailInput = document.getElementById("email");
var emailError = document.getElementById("emailError");

emailInput.addEventListener("input", function() {
  var email = this.value;
  var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  if (emailRegex.test(email)) {
    emailError.textContent = "";
    emailError.style.color = "green";
  } else {
    emailError.textContent = "Invalid email format";
    emailError.style.color = "red";
  }
});
```

Form Elements Access:

```
var form = document.getElementById("myForm");

// Access form elements
console.log(form.elements[0].value);         // First element value
console.log(form.elements["name"].value);    // By name
console.log(form.name);                      // Form name attribute
console.log(form.action);                    // Form action

// Reset form
form.reset();

// Submit form programmatically
form.submit();
```

DHTML (Dynamic HTML)

What is DHTML?

DHTML is not a single language but a combination of **HTML, CSS, JavaScript, and the DOM** used together to create dynamic, interactive, and animated web pages. It allows real-time changes to web content without reloading the page.

Components of DHTML:

1. **HTML**: Provides the structure and content
2. **CSS**: Defines styling and layout
3. **JavaScript**: Adds interactivity and dynamic behavior
4. **DOM**: Interface between JavaScript and HTML/CSS

How DHTML Works:

```
// Get HTML element
var element = document.getElementById("myElement");

// Modify CSS styles dynamically
element.style.backgroundColor = "red";
element.style.display = "block";
element.style.fontSize = "20px";

// Change HTML content
element.innerHTML = "New content";

// Modify HTML attributes
element.setAttribute("class", "active");
```

Combining HTML, CSS, and JavaScript:

HTML:

```
<div id="box" class="box">Click me to toggle</div>
<button id="toggleBtn">Toggle Box</button>
```

CSS:

```
.box {
  width: 100px;
  height: 100px;
  background-color: blue;
  display: block;
  transition: 0.3s;
}

.box.hidden {
  display: none;
}

.box.active {
  background-color: red;
  width: 150px;
```

```
  height: 150px;
}
```

JavaScript:

```
var box = document.getElementById("box");
var toggleBtn = document.getElementById("toggleBtn");

toggleBtn.addEventListener("click", function() {
  box.classList.toggle("active");
});

// Hide/Show box
document.getElementById("hideBtn").addEventListener("click", function() {
  box.classList.toggle("hidden");
});
```

Events and Event Handling

What are Events?

Events are actions or occurrences that happen in the browser, triggered by user interactions or browser operations.

Common Events:

Event	Trigger
`onclick`	Element is clicked
`ondblclick`	Element is double-clicked
`onmouseover`	Mouse pointer moves over element
`onmouseout`	Mouse pointer leaves element
`onmousemove`	Mouse moves over element
`onmousedown`	Mouse button is pressed down
`onmouseup`	Mouse button is released
`onload`	Page/image has finished loading
`onunload`	Page is being unloaded
`onchange`	Input field value changes
`onfocus`	Element gets focus
`onblur`	Element loses focus
`onkeydown`	Key is pressed down
`onkeyup`	Key is released
`onkeypress`	Key is pressed
`onsubmit`	Form is submitted
`onreset`	Form is reset

Event	Trigger
`onscroll`	Page/element is scrolled

Methods of Attaching Events:

1. Inline Event Handler (In HTML)

```
<button onclick="alert('Button clicked!')">Click me</button>
```

2. Assigning to Event Property (In JavaScript)

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

3. Using addEventListener() (Recommended)

```
var btn = document.getElementById("myBtn");
btn.addEventListener("click", function(event) {
  console.log("Button clicked!");
  console.log(event);
});
```

Advantages of addEventListener: - Multiple event handlers can be attached to same element - Event can be removed with removeEventListener - Better for separating JavaScript from HTML

Event Object:

When an event occurs, an event object is passed to the event handler containing information about the event.

```
document.addEventListener("click", function(event) {
  console.log(event.type);          // Type of event
  console.log(event.target);        // Element that triggered event
  console.log(event.clientX);       // X coordinate of mouse
  console.log(event.clientY);       // Y coordinate of mouse
  console.log(event.keyCode);       // Code of key pressed
  console.log(event.key);           // Key pressed
  console.log(event.altKey);        // Whether Alt key was pressed
  console.log(event.ctrlKey);       // Whether Ctrl key was pressed
  console.log(event.shiftKey);      // Whether Shift key was pressed
});
```

Mouse Events Example:

```
var box = document.getElementById("box");

// Click event
box.addEventListener("click", function() {
  console.log("Box clicked!");
  this.style.backgroundColor = "green";
});

// Double-click event
box.addEventListener("dblclick", function() {
  console.log("Box double-clicked!");
  this.style.backgroundColor = "blue";
});

// Mouse over event
box.addEventListener("mouseover", function() {
  this.style.backgroundColor = "yellow";
  this.textContent = "Mouse is over";
});

// Mouse out event
box.addEventListener("mouseout", function() {
  this.style.backgroundColor = "gray";
  this.textContent = "Mouse left";
});

// Mouse move event (track cursor position)
box.addEventListener("mousemove", function(e) {
  console.log("Cursor at: " + e.clientX + ", " + e.clientY);
});

// Mouse down event
box.addEventListener("mousedown", function() {
  console.log("Mouse button pressed");
});

// Mouse up event
box.addEventListener("mouseup", function() {
  console.log("Mouse button released");
});
```

Keyboard Events Example:

```
var input = document.getElementById("inputField");

// Key down event
input.addEventListener("keydown", function(e) {
  console.log("Key pressed: " + e.key);
  console.log("Key code: " + e.keyCode);
});

// Key up event
input.addEventListener("keyup", function(e) {
  console.log("Key released: " + e.key);
});

// Key press event
input.addEventListener("keypress", function(e) {
  console.log("Character: " + String.fromCharCode(e.keyCode));
});
```

Form Events Example:

```
var form = document.getElementById("myForm");
var input = document.getElementById("inputField");

// Form submit event
form.addEventListener("submit", function(e) {
  e.preventDefault();  // Prevent default form submission
  console.log("Form submitted!");
});

// Input change event
input.addEventListener("change", function() {
  console.log("Input value changed: " + this.value);
});

// Input focus event
input.addEventListener("focus", function() {
  console.log("Input focused");
  this.style.border = "2px solid blue";
});

// Input blur event
input.addEventListener("blur", function() {
  console.log("Input lost focus");
  this.style.border = "1px solid gray";
```

```
});
```

Page Load Events:

```
// When page has finished loading
window.addEventListener("load", function() {
  console.log("Page fully loaded!");
});

// When page is about to unload
window.addEventListener("beforeunload", function() {
  console.log("Page is closing");
});

// When page unloads
window.addEventListener("unload", function() {
  console.log("Page unloaded");
});
```

Controlling the Browser

Browser Window Control:

```
// Get window dimensions
console.log(window.innerWidth);   // Width of browser window
console.log(window.innerHeight);  // Height of browser window

// Open new window
window.open("https://example.com", "example", "width=800,height=600");

// Close window
window.close();

// Scroll to position
window.scrollTo(0, 100);  // Scroll to top: 100px
window.scrollBy(0, 50);   // Scroll by 50px

// Get scroll position
console.log(window.scrollX);  // Horizontal scroll
console.log(window.scrollY);  // Vertical scroll
```

History Navigation:

```
// Go back to previous page
history.back();
```

```
// Go forward to next page
history.forward();

// Go to specific page in history
history.go(-1);   // Back one page
history.go(1);    // Forward one page
history.go(0);    // Reload current page
```

Timer Functions:

setTimeout - Execute after delay

```
// Execute function after 2000 milliseconds (2 seconds)
setTimeout(function() {
  console.log("This runs after 2 seconds");
}, 2000);

// Cancel timeout
var timeoutId = setTimeout(function() {
  console.log("This might not run");
}, 2000);

clearTimeout(timeoutId);  // Cancel before it executes
```

setInterval - Execute repeatedly

```
// Execute function every 1000 milliseconds (1 second)
var intervalId = setInterval(function() {
  console.log("This runs every 1 second");
}, 1000);

// Stop interval
clearInterval(intervalId);
```

Buttons and Interactive Controls:

Basic Button Click Handler

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

<script>
var btn = document.getElementById("myBtn");

btn.addEventListener("click", function() {
  alert("Button was clicked!");
  console.log("Button clicked");
```

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

Toggle Button Example

```
<button id="toggleBtn">Toggle Light</button>
<div id="light" style="width: 100px; height: 100px; background-color: white; border: 1px sol

<script>
var toggleBtn = document.getElementById("toggleBtn");
var light = document.getElementById("light");
var isOn = false;

toggleBtn.addEventListener("click", function() {
  if (isOn) {
    light.style.backgroundColor = "white";
    toggleBtn.textContent = "Turn On";
    isOn = false;
  } else {
    light.style.backgroundColor = "yellow";
    toggleBtn.textContent = "Turn Off";
    isOn = true;
  }
});
</script>
```

Dynamic Content Update

```
<button id="updateBtn">Update Content</button>
<div id="content">Original content</div>

<script>
var updateBtn = document.getElementById("updateBtn");
var content = document.getElementById("content");

updateBtn.addEventListener("click", function() {
  content.innerHTML = "<h2>Content Updated!</h2>";
  content.style.color = "blue";
});
</script>
```

Progress Tracking:

```
var count = 0;
var countDisplay = document.getElementById("count");
var startBtn = document.getElementById("startBtn");
var stopBtn = document.getElementById("stopBtn");
```

```
var intervalId;

startBtn.addEventListener("click", function() {
  intervalId = setInterval(function() {
    count++;
    countDisplay.textContent = count;

    if (count >= 100) {
      clearInterval(intervalId);
      alert("Reached 100!");
    }
  }, 100);
});

stopBtn.addEventListener("click", function() {
  clearInterval(intervalId);
  console.log("Stopped at: " + count);
});
```

Complete DHTML Example: Interactive Form

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }

    .form-group {
      margin-bottom: 15px;
    }

    label {
      display: block;
      margin-bottom: 5px;
      font-weight: bold;
    }

    input, textarea {
      width: 300px;
      padding: 8px;
      border: 1px solid #ccc;
```

```
      border-radius: 4px;
    }

    input:focus {
      outline: none;
      border-color: blue;
      background-color: #f0f0ff;
    }

    .error {
      color: red;
      font-size: 12px;
      margin-top: 3px;
    }

    button {
      padding: 10px 20px;
      background-color: #4CAF50;
      color: white;
      border: none;
      border-radius: 4px;
      cursor: pointer;
      margin-right: 10px;
    }

    button:hover {
      background-color: #45a049;
    }

    .success {
      color: green;
      padding: 10px;
      background-color: #f0fff0;
      border-radius: 4px;
      display: none;
    }
  </style>
</head>
<body>

<h1>Interactive Registration Form</h1>

<form id="registrationForm">
  <div class="form-group">
    <label for="name">Full Name:</label>
    <input type="text" id="name" placeholder="Enter your name">
```

```
    <div class="error" id="nameError"></div>
  </div>

  <div class="form-group">
    <label for="email">Email:</label>
    <input type="email" id="email" placeholder="Enter your email">
    <div class="error" id="emailError"></div>
  </div>

  <div class="form-group">
    <label for="password">Password:</label>
    <input type="password" id="password" placeholder="Enter password (min 6 chars)">
    <div class="error" id="passwordError"></div>
  </div>

  <button type="button" onclick="validateForm()">Submit</button>
  <button type="reset">Reset</button>
</form>

<div class="success" id="successMsg">
  Form submitted successfully!
</div>

<script>
function validateForm() {
  // Get form values
  var name = document.getElementById("name").value.trim();
  var email = document.getElementById("email").value.trim();
  var password = document.getElementById("password").value;

  // Clear previous errors
  document.getElementById("nameError").textContent = "";
  document.getElementById("emailError").textContent = "";
  document.getElementById("passwordError").textContent = "";

  let isValid = true;

  // Validate name
  if (name === "") {
    document.getElementById("nameError").textContent = "Name is required";
    isValid = false;
  } else if (name.length < 3) {
    document.getElementById("nameError").textContent = "Name must be at least 3 characters";
    isValid = false;
  }
```

```
  // Validate email
  var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (email === "") {
    document.getElementById("emailError").textContent = "Email is required";
    isValid = false;
  } else if (!emailRegex.test(email)) {
    document.getElementById("emailError").textContent = "Invalid email format";
    isValid = false;
  }

  // Validate password
  if (password === "") {
    document.getElementById("passwordError").textContent = "Password is required";
    isValid = false;
  } else if (password.length < 6) {
    document.getElementById("passwordError").textContent = "Password must be at least 6 cha
    isValid = false;
  }

  // If all valid, show success message
  if (isValid) {
    document.getElementById("successMsg").style.display = "block";
    document.getElementById("registrationForm").reset();

    // Hide success message after 3 seconds
    setTimeout(function() {
      document.getElementById("successMsg").style.display = "none";
    }, 3000);
  }
}

// Real-time validation on input
document.getElementById("email").addEventListener("blur", function() {
  var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(this.value) && this.value !== "") {
    document.getElementById("emailError").textContent = "Invalid email format";
  } else {
    document.getElementById("emailError").textContent = "";
  }
});

// Add focus effects
document.getElementById("name").addEventListener("focus", function() {
  this.style.backgroundColor = "#f0f0ff";
});
```

```
document.getElementById("name").addEventListener("blur", function() {
  this.style.backgroundColor = "white";
});
</script>

</body>
</html>
```

Summary of Key Concepts

Client-Side Scripting Benefits:

- Executes on client browser without server involvement
- Reduces server load and bandwidth usage
- Provides immediate user feedback
- Enables form validation before submission
- Creates interactive and dynamic web pages

JavaScript Fundamentals:

- **Variables**: Containers for data using `var`, `let`, or `const`
- **Data Types**: Primitives (string, number, boolean, null, undefined, symbol, bigint) and objects
- **Functions**: Reusable code blocks with parameters and return values
- **Conditionals**: if, if-else, switch statements for decision making
- **Loops**: for, while, do-while, for-in, for-of for repetitive tasks
- **Objects**: Collections of properties and methods

DOM Manipulation:

- Select elements using getElementById, querySelector, etc.
- Modify content, attributes, and styles dynamically
- Create, delete, and navigate DOM elements
- Handle events (click, mouseover, keyboard, form)

DHTML Implementation:

- Combines HTML, CSS, JavaScript, and DOM
- Create interactive and animated web pages
- Real-time changes without page reload
- Form validation and data processing

Best Practices:

- Use meaningful variable and function names
- Separate JavaScript from HTML (use external files)

- Use addEventListener instead of inline handlers
- Always validate user input on both client and server
- Handle errors gracefully
- Optimize performance and minimize DOM manipulation
- Test across different browsers

Practice Problems

1. **Create a calculator** that takes two numbers and operation (+, -, *, /) and returns result
2. **Build a to-do list** where users can add, delete, and mark items as complete
3. **Create a color picker** that displays color codes when clicked
4. **Build a password strength checker** that shows real-time feedback
5. **Create a countdown timer** with start, pause, and reset buttons
6. **Build an image gallery** with next/previous navigation
7. **Create a quiz application** with scoring and result display
8. **Build a weather app** that displays dynamic information

Conclusion

JavaScript is a powerful client-side scripting language that enables dynamic, interactive web applications. Combined with HTML and CSS through DHTML concepts, it transforms static web pages into engaging user experiences. Understanding these fundamentals—variables, functions, loops, conditionals, DOM manipulation, and event handling—is essential for web development success.

Master these concepts through practice and experimentation, and you'll be well-equipped to build sophisticated web applications!