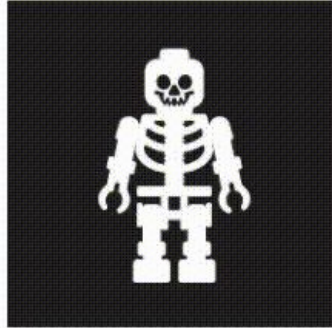


Javascript

HTML
structure



CSS
presentation/appearance



JavaScript
dynamism/action



<https://d2v4zi8pl64nxt.cloudfront.net/javascript-seo/5948abfc0e2df5.02876591.gif>

Kameswari Chebrolu

Department of CSE, IIT Bombay

JavaScript

- A growing demand for more interactive and dynamic content → Client-Side Scripting
 - Originated in Netscape Communications, initially named Mocha and later LiveScript
 - Was originally designed to run on the client side (i.e. in the browser)
 - Netscape and Sun later entered a collaboration, and renamed it Javascript
 - Why? Java of Sun Microsystems very popular, renamed to leverage popularity of Java
 - Note: JavaScript and Java are different languages with different purposes!

- Lots of nice features:
 - Light-weight
 - Cross-Platform Compatibility (across browsers, OS)
 - Can interact with Document Object Model (DOM)
 - Helps manipulate elements in the HTML document
- Standardized in 1997; led to widespread adoption

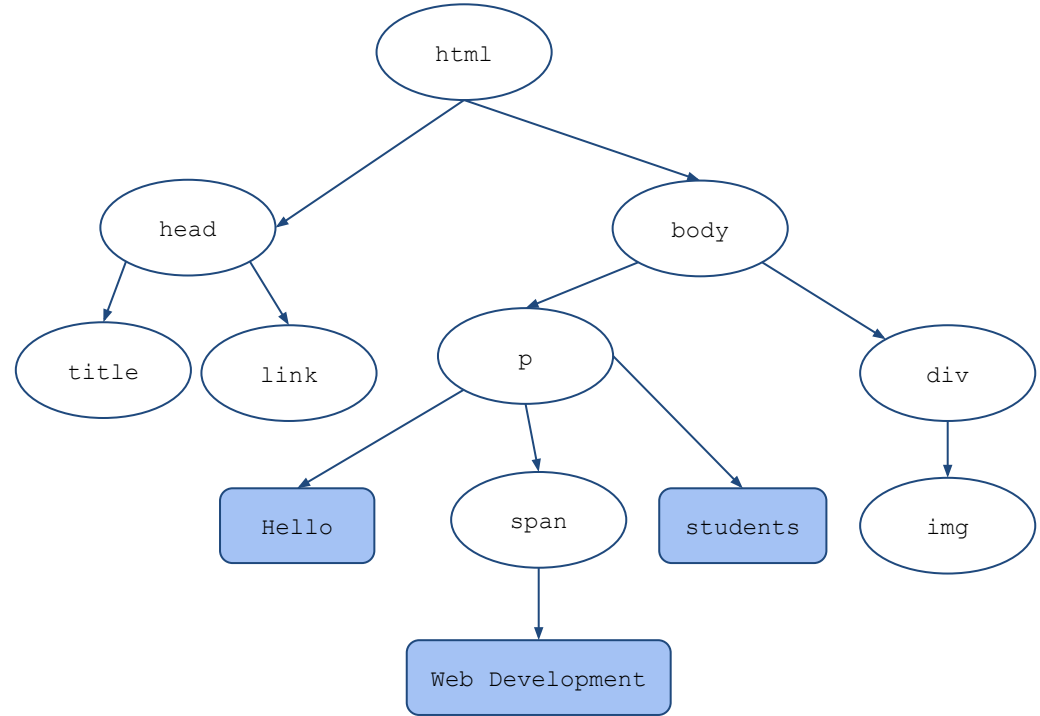
- Evolved new features
 - Libraries like jQuery
 - Simplifies DOM manipulation, event handling, animation, and Ajax interactions
 - Frameworks such as Angular, React, and Vue.js
 - Pre-written reusable code libraries or sets of tools
 - Help simplify and streamline development of web applications

Document Object Model (DOM)

- DOM is a hierarchical representation of the HTML document structure
 - Converts the page into a tree-like structure
 - Each node in the tree is an object representing a part of the document
 - E.g. elements, attributes, text content etc
 - Objects can be manipulated programmatically via JavaScript
 - Structure, style, and content of the page can be changed dynamically
- DOM essentially an API that provides a standardized interface to dynamically change the webpage

```
<html>
<head>
  <title>DOM Tutorial</title>
  <link rel="stylesheet"
href="index.css">
</head>
<body>
  <p>
    Hello
    <span>Web Development</span>
    students
  </p>
  <div>
    
  </div>
</body>
</html>
```

Sample.html

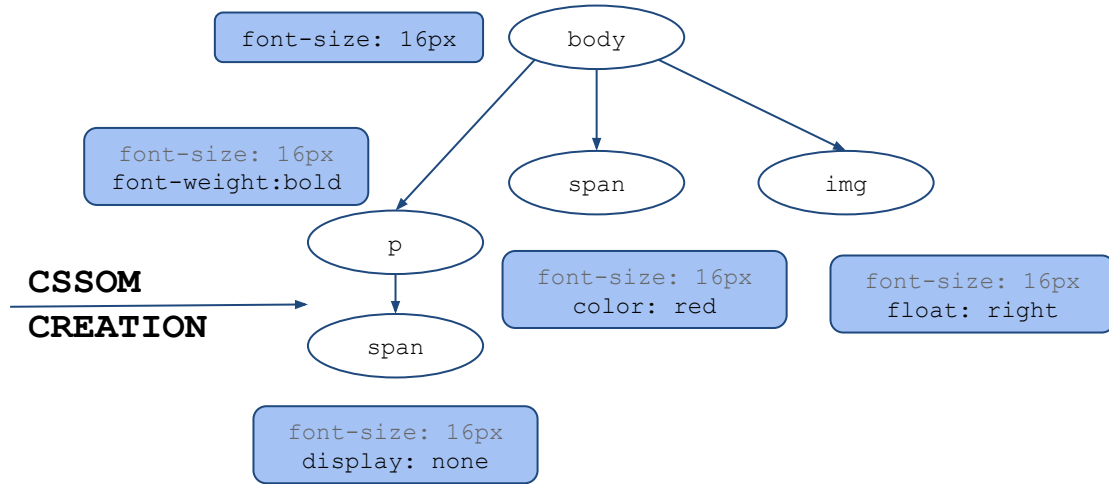


DOM Tree

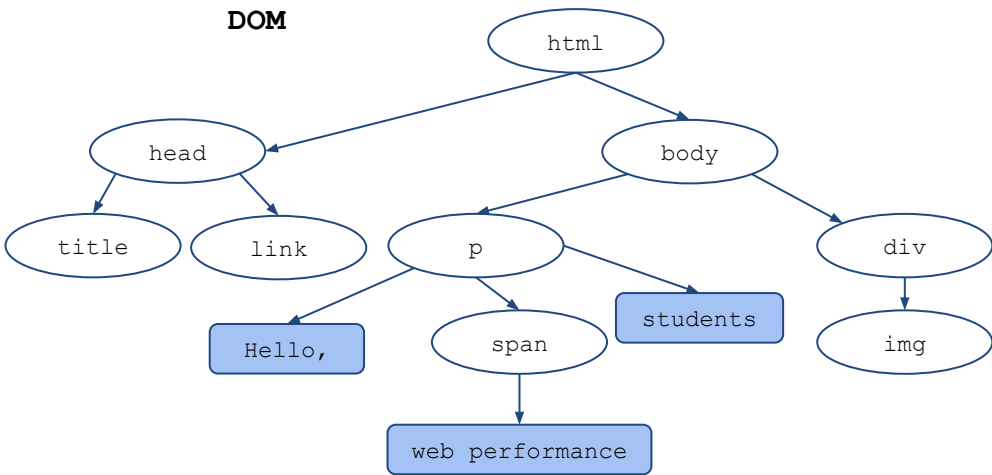
```
body {  
  font-size: 16px;  
}  
p {  
  font-weight: bold;  
}  
span {  
  color: red;  
}  
p span {  
  display: none;  
}  
img {  
  float: right;  
}
```

index.css

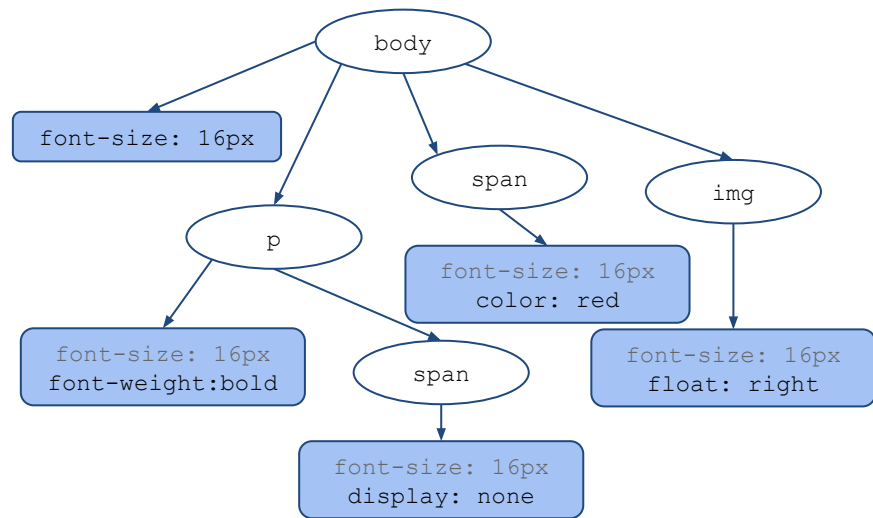
CSSOM Tree



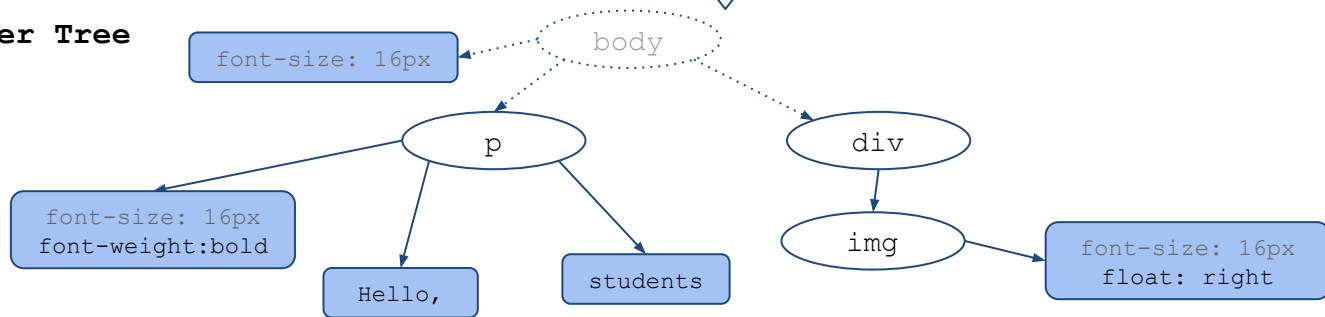
DOM

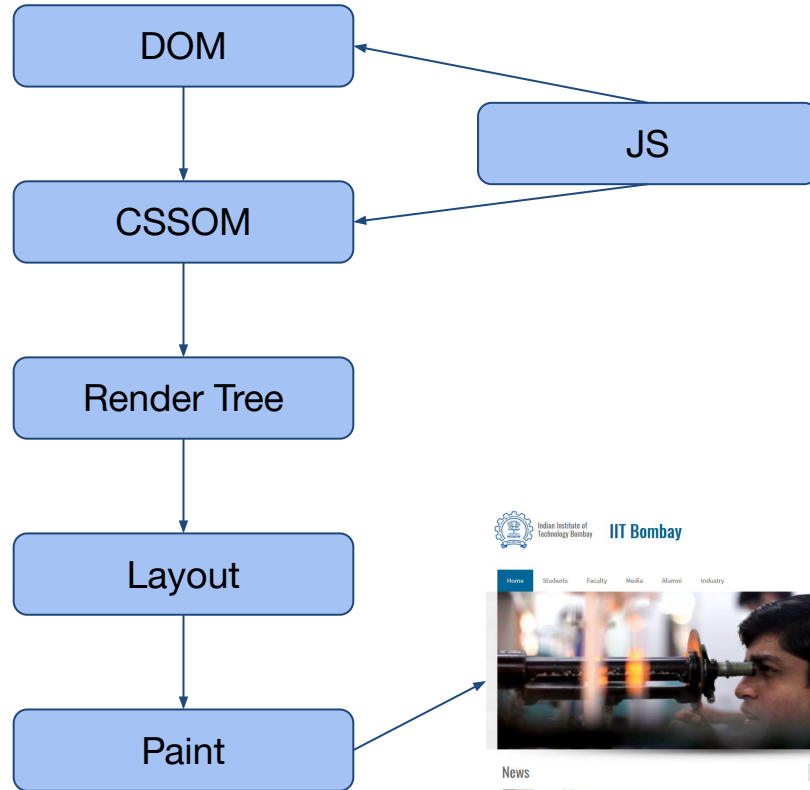


CSSOM



Render Tree





DOM Manipulation

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Power of Javascript

```
<html>
<head>
  <title>JavaScript Example</title>
  <style>
    #message {
      color: blue;
    }
    .hidden {
      display: none;
    }
  </style>
</head>
<body>
  <h1 id="title">Original Title</h1>
  <button id="changeContentBtn">Change Content</button>
  <button id="changeStyleBtn">Change Style</button>
  <button id="hideElementBtn">Hide Element</button>
```

```
<p id="message">This is a message.</p>
  <form id="exampleForm">
    <label for="email">Email: </label>
    <input type="text" id="email" name="email">
    <button type="submit">Submit</button>
  </form>

<p id="errorMessage" class="hidden">Please enter a valid email address.</p>

<script>
  // Change HTML Content
  document.getElementById("changeContentBtn").addEventListener("click", () =>
{
  document.getElementById("title").innerText = "Updated Title!";
});
```

```
// Change CSS Styling
```

```
document.getElementById("changeStyleBtn").addEventListener("click", () => {  
    const message = document.getElementById("message");  
    message.style.color = "green";  
    message.style.fontSize = "20px";  
});
```

```
// Hide Element
```

```
document.getElementById("hideElementBtn").addEventListener("click", () => {  
    document.getElementById("message").classList.add("hidden");  
});
```

```
// Validate Form
```

```
document.getElementById("exampleForm").addEventListener("submit", (event) => {  
    event.preventDefault();  
    const emailInput = document.getElementById("email").value;  
    const errorMessage = document.getElementById("errorMessage");  
  
    if (!emailInput.includes("@")) {  
        errorMessage.classList.remove("hidden");  
    } else {  
        errorMessage.classList.add("hidden");  
        alert("Form submitted successfully!");  
    }  
});
```

```
</script>
```

```
</body>
```

```
</html>
```

Original Title

Change Content

Change Style

Hide Element

This is a message.

Email:

Submit

Updated Title!

Change Content

Change Style

Hide Element

This is a message.

Email:

Submit

Please enter a valid email address.

Javascript Basics

- JavaScript has C-style syntax
 - Usual: variables, data types, operators, control structures (such as loops and conditionals), functions, and objects

- JavaScript code is inserted between `<script>` and `</script>` tags
 - Can place any number of scripts in a HTML document.
 - Scripts can be placed in `<body>`, or in `<head>` section or both
 - Scripts can also be placed in external files and included in HTML
 - JavaScript files have the file extension `.js`
 - Useful when same code is used in many different web pages
 - Separates HTML and code → make them easier to read and maintain
 - Cached JavaScript files can speed up page loads

- An external script can be referenced in 3 different ways:
 - With a full URL (a full web address)
 - `<script src="https://www.w3schools.com/js/myScript.js"></script>`
 - With a file path (like /js/)
 - `<script src="/js/myScript.js"></script>`
 - Without any path
 - `<script src="myScript.js"></script>`
 - located in the same folder as the current page
 - You don't include `<script>` tags inside external files!

```
<html lang="en">
<head>
  <title>JavaScript Example</title>
  <script>
    // Inline JavaScript
    function changeText() {
      const message = document.getElementById('inlineMessage');
      message.textContent = 'The text has been changed by inline JavaScript!';
    }
  </script>
</head>
<body>
  <h1>JavaScript Example</h1>
  <!-- Paragraph modified by external JavaScript -->
  <p id="dynamicParagraph">This paragraph is styled and modified by external JavaScript.</p>
  <!-- Button to trigger inline JavaScript -->
  <button onclick="changeText()">Change Text via Inline JavaScript</button>
  <!-- Inline message -->
  <p id="inlineMessage">This is the initial inline message.</p>
  <!-- External JavaScript file -->
  <script src="external.js"></script>
</body>
</html>
```

Contents of external.js

```
// External JavaScript
const paragraph = document.getElementById('dynamicParagraph');

// Modify the style of the paragraph immediately
paragraph.style.color = 'red';
paragraph.style.fontSize = '20px';
paragraph.style.fontWeight = 'bold';

// Add a new element to the page dynamically
const newElement = document.createElement('p');
newElement.textContent = 'This is a dynamically added paragraph from external JavaScript.';
document.body.appendChild(newElement);
```

JavaScript Example

This paragraph is styled and modified by external JavaScript.

Change Text via Inline JavaScript

This is the initial inline message.

This is a dynamically added paragraph from external JavaScript.

Display

- JavaScript can "display" data in different ways:
 - Writing into an HTML element, using `innerHTML`
 - Writing into the HTML output using `document.write()`
 - Writing into an alert box, using `window.alert()`
 - Writing into the browser console, using `console.log()`

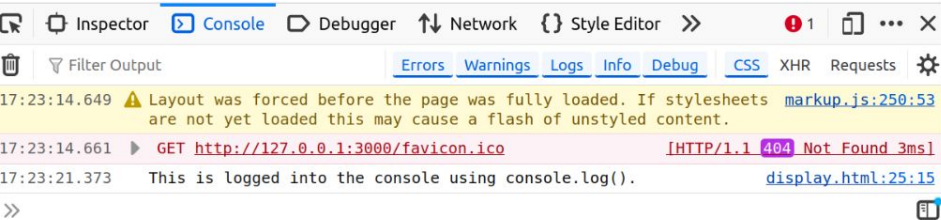
```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Display Methods</title>
</head>
<body>
  <h1>JavaScript Display Methods</h1>
  <p id="innerHTMLExample">This will be replaced using innerHTML.</p>
  <button onclick="displayExamples()">Click to See Examples</button>

  <script>
    function displayExamples() {
      // 1. Using innerHTML to modify an existing HTML element
      document.getElementById('innerHTMLExample').innerHTML = "This text was updated using innerHTML.";

      // 2. Using document.write() to write directly into the HTML output
      //This will replace everything in the page; comment it first and then uncomment it to see it in action
      //document.write("<p>This paragraph is added using document.write().</p>");

      // 3. Using alert() to show a message box
      window.alert("This is displayed in an alert box using window.alert().");

      // 4. Using console.log() to display data in the console
      console.log("This is logged into the console using console.log().");
    }
  </script>
</body>
</html>
```

JavaScript Display Methods

This text was updated using innerHTML.

[Click to See Examples](#)

Variables and Operators

- JavaScript Variables can be declared in 4 ways: Automatically; Using var; Using let; Using const
 - Var is not used any more
 - Good practice to declare variables, so don't do it automatically
 - Always use const if the value should not be changed
- Operators:
 - Very similar to C (+, -, *, /, ++, --, **, &&, ! etc)
 - Comparison also similar to C (<, >, >=, !=, == etc)
 - “==” equal to vs “===” equal value and equal type

```
<html>
<head>
  <title>JavaScript Variables and Operations</title>
</script>
  // Using const (for values that should not be changed)
  const pi = 3.14159;
  console.log("Value of pi:", pi); // Outputs: 3.14159

  // Using let (for values that may change)
  let age = 25;
  console.log("Age before increment:", age); // Outputs: 25

  // Using var (old way, not recommended)
  var oldVariable = "I'm a legacy variable";
  console.log(oldVariable); // Outputs: I'm a legacy variable

  // Automatically declared (not recommended, not good practice)
  x = 10; // This is an implicit global variable
  console.log(x); // Outputs: 10
```

```
// Arithmetic Operators (+, -, *, /, %, **)
let sum = 10 + 5;      // Addition
let difference = 10 - 5; // Subtraction
let product = 10 * 5;   // Multiplication
let quotient = 10 / 5;  // Division
let remainder = 10 % 3; // Modulus (remainder)
let power = 2 ** 3;     // Exponentiation

console.log("Sum:", sum); // Outputs: 15
console.log("Difference:", difference); // Outputs: 5
console.log("Product:", product); // Outputs: 50
console.log("Quotient:", quotient); // Outputs: 2
console.log("Remainder:", remainder); // Outputs: 1
console.log("Power:", power); // Outputs: 8

// Increment and Decrement (++ , --)
age++; // Increment age by 1
console.log("Age after increment:", age); // Outputs: 26

age--; // Decrement age by 1
console.log("Age after decrement:", age); // Outputs: 25
```

```
// Comparison Operators (<, >, <=, >=, ==, !=)

let isGreaterThan = 10 > 5; // true
let isLessThan = 5 < 10; // true
let isEqual = 5 == '5'; // true (loose equality, because '5' is converted to 5)
let isStrictEqual = 5 === '5'; // false (strict equality, types do not match)
let isNotEqual = 5 != '5'; // false (loose inequality)


console.log("isGreaterThan:", isGreaterThan); // true
console.log("isEqual (loose):", isEqual); // true
console.log("isStrictEqual:", isStrictEqual); // false
console.log("isNotEqual:", isNotEqual); // false


// Logical Operators (&&, ||, !)

let hasPermission = true;
let isAdmin = false;


let canAccess = hasPermission && isAdmin; // false, both must be true for && to be true
let isAllowed = hasPermission || isAdmin; // true, one of them must be true for || to be true
let notAdmin = !isAdmin; // true, negates the value of isAdmin

console.log("Can Access (has permission and is admin):", canAccess); // false
console.log("Is Allowed (has permission or is admin):", isAllowed); // true
console.log("Is not Admin:", notAdmin); // true
```






















```
// Conditional Operator (Ternary operator)
let message = age >= 18 ? "You are an adult." : "You are a minor.";
console.log(message); // Outputs: You are an adult.

// Using functions to demonstrate variable scope (let and const)
function demoScope() {
    const constantValue = 100; // Block-scoped constant
    let variableValue = 50; // Block-scoped variable

    console.log("Inside function, constantValue:", constantValue); // 100
    console.log("Inside function, variableValue:", variableValue); // 50

    // Updating the variable
    variableValue = 75;
    console.log("Inside function after update, variableValue:", variableValue); // 75
}

demoScope();
// Outside the function, we cannot access constantValue or variableValue
// console.log(constantValue); // Error: constantValue is not defined
// console.log(variableValue); // Error: variableValue is not defined
</script>
</head>
<body>
    <h1>JavaScript Variables and Operations</h1>
    <p>Check the browser's developer console for output!</p>
</body>
</html>
```

  Inspector  Console  Debugger  Network  Style Editor  Performance  Memory  ... 		
 Filter Output  Errors  Warnings  Logs  Info  Debug  CSS  XHR  Requests 		
13:41:24.006	 This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use variables-operators.html " <code><!DOCTYPE html></code> ". [Learn More]	
13:41:24.031	Value of pi: 3.14159	variables-operators.html:8:17
13:41:24.032	Age before increment: 25	variables-operators.html:12:17
13:41:24.032	I'm a legacy variable	variables-operators.html:16:17
13:41:24.032	10	variables-operators.html:20:17
13:41:24.032	Sum: 15	variables-operators.html:30:17
13:41:24.032	Difference: 5	variables-operators.html:31:17
13:41:24.032	Product: 50	variables-operators.html:32:17
13:41:24.032	Quotient: 2	variables-operators.html:33:17
13:41:24.032	Remainder: 1	variables-operators.html:34:17
13:41:24.032	Power: 8	variables-operators.html:35:17
13:41:24.032	Age after increment: 26	variables-operators.html:39:17
13:41:24.032	Age after decrement: 25	variables-operators.html:42:17
13:41:24.032	isGreaterThan: true	variables-operators.html:51:17
13:41:24.032	isEqual (loose): true	variables-operators.html:52:17
13:41:24.032	isStrictEqual: false	variables-operators.html:53:17
13:41:24.032	isNotEqual: false	variables-operators.html:54:17
13:41:24.032	Can Access (has permission and is admin): false	variables-operators.html:64:17
13:41:24.032	Is Allowed (has permission or is admin): true	variables-operators.html:65:17
13:41:24.032	Is not Admin: true	variables-operators.html:66:17
13:41:24.032	You are an adult.	variables-operators.html:70:17
13:41:24.032	Inside function, constantValue: 100	variables-operators.html:77:21
13:41:24.032	Inside function, variableValue: 50	variables-operators.html:78:21
13:41:24.032	Inside function after update, variableValue: 75	variables-operators.html:82:21

Data Types

- JavaScript has 8 Datatypes:
 - String, Number, BigInt, Boolean, Undefined, Null, Symbol, Object
 - Object data type can contain:
 - An object, array or date
- JavaScript has dynamic types
 - Type of a variable can change during execution of the program
 - Same variable can be used to hold different data types
 - When adding a number and a string, JavaScript will treat the number as a string
 - Evaluates expressions from left to right


```
<html>

<head>


  <title>JavaScript Data Types</title>

  <style>
    body {
      font-family: Arial, sans-serif;
    }
    #output {
      margin-top: 20px;
      font-size: 18px;
      white-space: pre-line;
    }
  </style>
  <script>
    window.onload = function () {
      // 1. String Data Type
      let name = "Alice";
      console.log("String:", name); // String: Alice
      document.getElementById("output").innerText += "String: " + name + "\n";
    }
  </script>
</html>
```

```
// 2. Number Data Type
let age = 30;
console.log("Number:", age); // Number: 30
document.getElementById("output").innerText += "Number: " + age + "\n";

// 3. BigInt Data Type (for large integers)
let bigNumber = 9007199254740991n;
console.log("BigInt:", bigNumber); // BigInt: 9007199254740991n
document.getElementById("output").innerText += "BigInt: " + bigNumber + "\n";

// 4. Boolean Data Type
let isAdult = true;
console.log("Boolean:", isAdult); // Boolean: true
document.getElementById("output").innerText += "Boolean: " + isAdult + "\n";

// 5. Undefined Data Type
let undefinedVariable;
console.log("Undefined:", undefinedVariable); // Undefined: undefined
document.getElementById("output").innerText += "Undefined: " + undefinedVariable + "\n";
```

```
// 6. Null Data Type
```

```
let emptyValue = null;
```

```
console.log("Null:", emptyValue); // Null: null
```

```
document.getElementById("output").innerText += "Null: " + emptyValue + "\n";
```

```
// 7. Symbol Data Type (used for unique identifiers)
```

```
let uniqueSymbol = Symbol("id");
```

```
console.log("Symbol:", uniqueSymbol); // Symbol: Symbol(id)
```

```
document.getElementById("output").innerText += "Symbol: " + uniqueSymbol.toString() + "\n";
```

```
// 8. Dynamic Typing Example
```

```
let dynamicVar = "I am a string now!";
```

```
console.log("Dynamic Variable (String):", dynamicVar); // Dynamic Variable (String): I am a  
string now!
```

```
document.getElementById("output").innerText += "Dynamic Variable (String): " + dynamicVar +  
"\n";
```

```
dynamicVar = 42; // Now it's a number
```

```
console.log("Dynamic Variable (Now a Number):", dynamicVar); // Dynamic Variable (Now a  
Number): 42
```

```
document.getElementById("output").innerText += "Dynamic Variable (Now a Number): " + dynamicVar  
+ "\n";
```

```
// 9. Operations: Adding Number and String (JavaScript treats number as a string)
    // "5" and '5' are both strings.
    // JavaScript allows both double quotes (") and single quotes (') to denote string literals, and
they are equivalent.

    let result = 10 + "5"; // Concatenation, not addition
    console.log("Result of 10 + '5' (String concatenation):", result); // Result of 10 + '5'
(String concatenation): 105

    document.getElementById("output").innerText += "Result of 10 + '5' (String concatenation): " +
result + "\n";

// 10. Evaluating expressions from left to right
let expressionResult = 5 + 3 * 2; // Multiplies first (3 * 2) then adds 5
console.log("Expression result (5 + 3 * 2):", expressionResult); // Expression result (5 + 3 *
2): 11

    document.getElementById("output").innerText += "Expression result (5 + 3 * 2): " +
expressionResult + "\n";
```

```
// 11. Comparison Example: Loose equality (==) vs Strict equality (===)
    let isEqual = 5 == "5"; // Loose equality, checks value
    let isStrictEqual = 5 === "5"; // Strict equality, checks both value and type
    console.log("5 == '5' (Loose Equality):", isEqual); // 5 == '5' (Loose Equality): true
    console.log("5 === '5' (Strict Equality):", isStrictEqual); // 5 === '5' (Strict Equality):
false

    document.getElementById("output").innerText += "5 == '5' (Loose Equality): " + isEqual + "\n";
    document.getElementById("output").innerText += "5 === '5' (Strict Equality): " + isStrictEqual
+ "\n";
    };
</script>

</head>
<body>
    <h1>JavaScript Data Types Example</h1>
    <p>Check the browser's console for the output, and see the page below:</p>
    <div id="output"></div>
</body>
</html>
```

JavaScript Data Types Example

Check the browser's console for the output, and see the page below:

String: Alice

Number: 30

BigInt: 9007199254740991

Boolean: true

Undefined: undefined

Null: null

Symbol: Symbol(id)

Dynamic Variable (String): I am a string now!

Dynamic Variable (Now a Number): 42

Result of 10 + '5' (String concatenation): 105

Expression result (5 + 3 * 2): 11

5 == '5' (Loose Equality): true

5 === '5' (Strict Equality): false

Functions

- Defined with “function” keyword, followed by a name, followed by parentheses ()
 - Can take arguments too!
- Code to be executed, by the function, is placed inside curly brackets {}

```
<html>
<head>
  <title>JavaScript Function with Arguments</title>
  <script>
    // Define the function that takes arguments
    function greet(name, age) {
      var greeting = "Hello, " + name + "! You are " + age + " years old.";
      document.getElementById("message").innerText = greeting;
    }
  </script>
</head>
<body>
  <h1>JavaScript Function with Arguments</h1>
  <button onclick="greet('Ravi', 17)">Click me to greet Ravi</button>
  <button onclick="greet('Rashmi', 18)">Click me to greet Rashmi</button>
  <p id="message"></p>
</body>
</html>
```


Arrays

- An array is a special variable, which can hold more than one value
- Real strength of JavaScript arrays are the built-in array properties and methods
 - length property of an array returns the length of an array
 - add a new element to an array using the push() method:
 - pop() method removes the last element from an array:
 - sort() method sorts an array alphabetically:
 - reverse() method reverses the elements in an array.

```
const cars = ["Saab",  
"Volvo", "BMW"];
```

```
const cars = [  
  "Saab",  
  "Volvo",  
  "BMW"  
];
```

```
const cars = [];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";
```

```
<html>
<head>
  <title>JavaScript Array Example</title>
  <script>
    // Define the function that works with arrays
    function arrayOperations() {
      // Create an array
      var fruits = ["Apple", "Banana", "Orange", "Mango"];
      var output = "Initial array: " + fruits.join(", ") + "\n";

      // Use the length property to get the length of the array
      output += "Length: " + fruits.length + "\n\n";

      // Add a new element using push()
      fruits.push("Pineapple");
      output += "After push(): " + fruits.join(", ") + "\n";

      // Remove the last element using pop()
      fruits.pop();
      output += "After pop(): " + fruits.join(", ") + "\n";
    }
  </script>
</head>
</html>
```

```
// Sort the array alphabetically using sort()
    output += "\nBefore sort(): " + fruits.join(", ") + "\n";
    fruits.sort();
    output += "After sort(): " + fruits.join(", ") + "\n";

    // Reverse the array using reverse()
    output += "\nBefore reverse(): " + fruits.join(", ") + "\n";
    fruits.reverse();
    output += "After reverse(): " + fruits.join(", ") + "\n";

    // Display the results
    document.getElementById("message").innerText = output;
}
</script>
</head>
<body>
    <h1>JavaScript Array Example</h1>
    <button onclick="arrayOperations()">Click me to perform array operations</button>
    <pre id="message"></pre>
</body>
</html>
```

JavaScript Array Example

Click me to perform array operations


Initial array: Apple, Banana, Orange, Mango
Length: 4

After push(): Apple, Banana, Orange, Mango, Pineapple
After pop(): Apple, Banana, Orange, Mango

Before sort(): Apple, Banana, Orange, Mango
After sort(): Apple, Banana, Mango, Orange

Before reverse(): Apple, Banana, Mango, Orange
After reverse(): Orange, Mango, Banana, Apple

Objects

Object	Properties	Methods
	<code>car.name = Fiat</code> <code>car.model = 500</code> <code>car.weight = 850kg</code> <code>car.color = white</code>	<code>car.start()</code> <code>car.drive()</code> <code>car.brake()</code> <code>car.stop()</code>

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

- Objects are variables but can contain many values and also methods!
 - values are written as name:value pairs (called properties)
 - Can be accessed via `objectName.propertyName` or `objectName["propertyName"]`
 - Methods are actions that can be performed on objects
 - Are stored in properties as function definitions
 - “this” keyword refers to an object
 - which object depends on how this is being invoked (used or called)?
 - When a JavaScript variable is declared with the keyword "new", variable is created as an object

```
<html>

<head>

  <title>JavaScript Object Example</title>

  <script>

    // Define the object structure
    var person = {
      firstName: "",
      lastName: "",
      age: 0,
      fullName: function() {
        return this.firstName + " " + this.lastName; // Using "this" keyword
      },
      incrementAge: function() {
        this.age++; // Incrementing age using "this"
      }
    };
  </script>
</head>

</html>
```

```
// Define a function to work with the object
function objectExample() {
    // Assign values to the object
    person.firstName = "Alice";
    person.lastName = "Smith";
    person.age = 25;

    // Access and display initial object properties
    var output = "Initial Person Object:\n";
    output += "Full Name: " + person.fullName() + "\n";
    output += "Age: " + person.age + "\n\n";

    // Modify a property directly
    person.age = 26;
    output += "After modifying age directly:\n";
    output += "Age: " + person.age + "\n\n";

    // Call the method to increment age
    person.incrementAge();
    output += "After calling incrementAge():\n";
    output += "Age: " + person.age + "\n";

    // Display the results
    document.getElementById("message").innerText = output;
}
</script>
</head>
```



```
<body>  
  <h1>JavaScript Object Example</h1>  
  <button onclick="objectExample()">Click me to demonstrate object properties and methods</button>  
  <pre id="message"></pre>  
</body>  
</html>
```

JavaScript Object Example

Click me to demonstrate object properties and methods

Initial Person Object:
Full Name: Alice Smith
Age: 25

After modifying age directly:
Age: 26

After calling incrementAge():
Age: 27

Javascript Errors

- “try” statement allows you to define a block of code to be tested for errors while it is being executed.
- “catch” statement allows you to define a block of code to be executed, if an error occurs in the try block
 - JavaScript statements try and catch come in pairs
- “throw” statement allows you to create a custom error

```
<html>
<head>
  <title>JavaScript try-catch-throw and const Example</title>
</script>
  function tryCatchExample() {
    // Define a constant array
    const people = ["Aarav", "Isha", "Rajesh"];

    // Output the initial array
    //The join() method combines all elements of an array into a string, placing the provided delimiter
    between each element.
    let output = "Initial Array:\n" + people.join(", ") + "\n\n";

    try {
      // Adding a new item at the end of the array (allowed with const)
      people.push("Neha");
      output += "After adding an item:\n" + people.join(", ") + "\n\n";

      // Attempting to assign a new array to the const variable (not allowed)
      people = ["Priya", "Vikram"];
    } catch (error) {
      // Handle the error and display it
      output += "Error: " + error.message + "\n\n";
    }
  }
}
```

```
// Demonstrate throwing an error
try {
  if (people.length > 3) {
    throw "Too many people in the array!";
  }
} catch (error) {
  // Catch the thrown error
  output += "Custom Error: " + error + "\n\n";
}

// Output the final state of the array
output += "Final Array:\n" + people.join(", ");

// Display the results
document.getElementById("message").innerText = output;
}
</script>
</head>
<body>
  <h1>JavaScript try-catch-throw and const Example</h1>
  <button onclick="tryCatchExample()">Click me to run the example</button>
  <pre id="message"></pre>
</body>
</html>
```

Classes

- Classes are not objects, they are just a template for objects!
- Use keyword class to create a class
 - Defined using the class keyword, followed by the class name.
 - Body contains the constructor() method and methods that define the behavior of objects created
 - Constructor is executed automatically when a new object is created and is used to initialize object properties

- Can create an instance of a class using the new keyword
 - Invokes the constructor and creates a new object with the properties and methods defined in the class
 - Properties and methods defined in the class can be accessed via “this” inside the class
 - Outside the class, they can be accessed using the dot notation on the instance
- Methods defined inside the class are shared by all instances (objects) of the class
- Getters allow you to define methods that retrieve property values
- Setters allow you to define methods that update property values
- Getters and setters allow for controlled access to properties

```
<html>
<head>
  <title>JavaScript Class Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .output {
      margin-top: 20px;
    }
  </style>
```

```
<script>
    // Define a class
    class Person {
        constructor(firstName, lastName, age) {
            this.firstName = firstName; // Instance variable
            this.lastName = lastName;   // Instance variable
            this._age = age;             // _age is used for the getter/setter
        }

        // Getter for age (computed property)
        get age() {
            return this._age;
        }

        // Setter for age (validates and modifies property)
        set age(newAge) {
            if (newAge < 0) {
                document.getElementById( 'output' ).innerHTML += "<p>Age cannot be negative!</p>" ;
            } else {
                this._age = newAge;
            }
        }

        // Method to return full name
        fullName() {
            return `${this.firstName} ${this.lastName}`;
        }

        // Static method (no need for an instance)
        static greet() {
            document.getElementById( 'output' ).innerHTML += "<p>Hello from the Person class!</p>" ;
        }
    }
}
```



```
// Create two instances of the Person class
const person1 = new Person("Aarav", "Sharma", 25);
const person2 = new Person("Arti", "Shah", 18);

// Display full name
window.onload = function () {
  document.getElementById('output').innerHTML += `

Full Name: ${person1.fullName()}</p>`;

  // Display age using getter
  document.getElementById('output').innerHTML += `

Age of Person1: ${person1.age}</p>`;

  // Set new age using setter
  person1.age = 30;
  document.getElementById('output').innerHTML += `

Updated Age: ${person1.age}</p>`;

  // Set an invalid age (negative)
  person1.age = -5;

  // Call the static method
  Person.greet();

  // Display age using getter
  document.getElementById('output').innerHTML += `

Age of Person2: ${person2.age}</p>`;

}
</script>
</head>
<body>
  <h1>JavaScript Class Example</h1>
  <div id="output" class="output"></div>
</body>
</html>


```

JavaScript Class Example

Full Name: Aarav Sharma

Age of Person1: 25

Updated Age: 30

Age cannot be negative!

Hello from the Person class!

Age of Person2: 18

Key Differences Between `var` and `class` for Creating Objects

Aspect	<code>var</code> (Plain Object)	<code>class</code> (Blueprint)
Purpose	Used for creating a single object.	Defines a blueprint for creating multiple objects.
Reusability	You can create only a single object with <code>var</code> .	You can create many objects (instances) from the class.
Syntax	Directly assigns an object to a variable.	Uses <code>class</code> keyword, and objects are created via <code>new</code> .
Methods	Methods are defined directly in the object.	Methods are defined within the class, and objects inherit them.
Constructor	No constructor function. Properties are added directly.	A constructor method (<code>constructor()</code>) is used to initialize properties.
Inheritance	Cannot easily be inherited or extended.	Classes can extend other classes, enabling inheritance.
Example Use Case	Suitable for single, simple objects.	Suitable for creating multiple objects with similar properties and behaviors.

Conditionals and Loops

- Conditionals are used to decide which block of code to execute based on certain conditions (e.g., if, else)
- Loops are used to repeat a block of code multiple times (e.g., for, while, do-while, for...of)

```
<html>

<head>
  <title>Conditionals and Loops Example</title>
</head>

<body>

  <div id="output"></div>

  <script>
    // Numeric Variable
    let number = 5;

    // Checking if the number is even or odd
    // Backticks allow you to use template literals, which allows String Interpolation:
    // The ${} syntax inside backticks is used for string interpolation
    // Allows embedding expressions, such as variables, directly into strings without needing to concatenate them
    with +

    if (number % 2 === 0) {
      document.getElementById('output').innerHTML += `<p>The number ${number} is even.</p>`;
    } else {
      document.getElementById('output').innerHTML += `<p>The number ${number} is odd.</p>`;
    }
  </script>

```

```
// Array of numbers (numeric array)
let numbers = [1, 3, 5, 7, 9, 10, 12, 15];
document.getElementById( 'output' ).innerHTML += `<p>Original numeric array:  ${numbers.join( ", " )}</p>`;

// Using a 'for' loop to go through the numeric array
let evenNumbers = [];
for (let i = 0; i < numbers.length; i++) {
    if (numbers[i] % 2 === 0) {
        evenNumbers.push(numbers[i]); // Collecting even numbers
    }
}
document.getElementById( 'output' ).innerHTML += `<p>Even numbers (using 'for' loop):  ${evenNumbers.join( ", " )}</p>`;

// Using a 'while' loop to print odd numbers from the numeric array
let oddNumbers = [];
let j = 0;
while (j < numbers.length) {
    if (numbers[j] % 2 !== 0) {
        oddNumbers.push(numbers[j]); // Collecting odd numbers
    }
    j++;
}
document.getElementById( 'output' ).innerHTML += `<p>Odd numbers (using 'while' loop):  ${oddNumbers.join( ", " )}</p>`;
```

```
let fruits = ["apple", "banana", "cherry", "mango", "orange"];

// Using a for...in loop to iterate over the array (indexes)
let output = "";
for (let x in fruits) {
    output += `<p>Fruit at index ${x}: ${fruits[x]}</p>`;
}
document.getElementById('output').innerHTML += output;


// Using a 'while' loop to check for fruits that have more than 5 letters
let longFruits = [];
let k = 0;
while (k < fruits.length) {
    if (fruits[k].length > 5) {
        longFruits.push(fruits[k]); // Collecting fruits with more than 5 letters
    }
    k++;
}
document.getElementById('output').innerHTML += `<p>Fruits with more than 5 letters (using 'while' loop):
${longFruits.join(", ")}</p>`;
</script>

</body>

</html>
```

The number 5 is odd.

Original numeric array: 1, 3, 5, 7, 9, 10, 12, 15

Even numbers (using 'for' loop): 10, 12

Odd numbers (using 'while' loop): 1, 3, 5, 7, 9, 15

Fruit at index 0: apple

Fruit at index 1: banana

Fruit at index 2: cherry

Fruit at index 3: mango

Fruit at index 4: orange

Fruits with more than 5 letters (using 'while' loop): banana, cherry, orange

Event Handling

- Process of capturing user interactions (such as clicks, key presses, mouse movements, etc.) and responding to them by executing code
 - Allow web pages to become interactive
 - Syntax: `<element event='some JavaScript'>`
 - E.g. `<button onclick="alert('Button clicked!')">Click Me</button>`

- Event handlers are JavaScript functions that execute when an event occurs
- Three main ways to assign event handlers:
 - Inline Event Handling (in HTML); example earlier
 - Using DOM Properties (in JavaScript)
 - Using `addEventListener()` (Preferred Method)
 - Attach an event listener to an element
 - Allows multiple event listeners for the same event

Event Types

UI Events

- Click
- Double click
- Mouseover
- Mouseout
- Keydown, keyup
- Focus, blur

Form Events:

- Submit
- Change
- Input

Document/Window Events:

- Load
- Resize
- Scroll

```
<html>
<head>
  <title>Comprehensive UI Events</title>
  <style>
    button, input {
      padding: 10px 20px;
      font-size: 16px;
      margin: 10px;
      cursor: pointer;
    }
    #output {
      font-size: 16px;
      margin-top: 20px;
    }
    #eventButton:hover {
      background-color: #e0e0e0;
    }
  </style>
</head>
<body>
```

<h1>Comprehensive UI Events Example</h1>

<!-- Button for Mouse and Click Events -->

<button id="eventButton">Hover or Click Me!</button>

<!-- Input Field for Keyboard and Focus Events -->

<input id="textInput" type="text" placeholder="Type something here..." />

<!-- Output area -->

<div id="output"></div>

<script>

const button = document.getElementById('eventButton');

const textInput = document.getElementById('textInput');

const output = document.getElementById('output');

// Click event

//This is an arrow function, a shorthand way to write functions in JavaScript.

//Alternative would have been

//function handleClick() {

//output.innerHTML += `<p>Button clicked!</p>`;

//}

//button.addEventListener('click', handleClick);

button.addEventListener('click', () => {

output.innerHTML += `<p>Button clicked!</p>`;

});

```
// Double click event

button.addEventListener('dblclick', () => {
  output.innerHTML += `

Button double-clicked!</p>`;
});

// Mouseover event

button.addEventListener('mouseover', () => {
  output.innerHTML += `

Mouse is over the button.</p>`;
});

// Mouseout event

button.addEventListener('mouseout', () => {
  output.innerHTML += `

Mouse left the button.</p>`;
});

// Keydown event

textInput.addEventListener('keydown', (event) => {
  output.innerHTML += `

Key pressed down: ${event.key}</p>`;
});


```

```
// Keyup event
textInput.addEventListener('keyup', (event) => {
  output.innerHTML += `<p>Key released: ${event.key}</p>`;
});

// Focus event
textInput.addEventListener('focus', () => {
  output.innerHTML += `<p>Input field focused.</p>`;
});

// Blur event
textInput.addEventListener('blur', () => {
  output.innerHTML += `<p>Input field lost focus.</p>`;
});
</script>
</body>
</html>
```

Comprehensive UI Events Example

Hover or Click Me!

s

Mouse is over the button.

Mouse left the button.

Input field focused.

Key pressed down: s

Key released: s

Input field lost focus.


```
<html>
<head>
  <title>Event Handling Example</title>
</head>
<body>
  <h1>Event Handling in JavaScript</h1>
  <form id="sampleForm">
    <p>Name:</p>
    <input type="text" id="name" name="name"><br><br>

    <p>Favorite Color:</p>
    <select id="color" name="color">
      <option value="">Select</option>
      <option value="Red">Red</option>
      <option value="Blue">Blue</option>
      <option value="Green">Green</option>
    </select><br><br>

    <button type="submit">Submit</button>
  </form>

  <div id="output"></div>
  <div style="height: 1500px;">Scroll down to see the scroll event in action!</div>
```

```
<script>

  const form = document.getElementById('sampleForm');
  const nameInput = document.getElementById('name');
  const colorSelect = document.getElementById('color');
  const output = document.getElementById('output');

  // Form Events
  // 1. Submit event
  form.addEventListener('submit', (event) => {
    event.preventDefault(); // Prevents the form from reloading the page
    const name = nameInput.value;
    const color = colorSelect.value;
    output.innerHTML += `

Form submitted! Name: ${name}, Favorite Color: ${color}</p>`;
  });

  // 2. Change event (on <select>)
  colorSelect.addEventListener('change', () => {
    const selectedColor = colorSelect.value;
    output.innerHTML += `

Color changed to: ${selectedColor}</p>`;
  });


```

```
// 3. Input event (on <input>)
nameInput.addEventListener('input', () => {
  output.innerHTML += `<p>Typing: ${nameInput.value}</p>`;
});

// Document/Window Events
// 1. Load event
window.addEventListener('load', () => {
  output.innerHTML += `<p>Page fully loaded!</p>`;
});

// 2. Resize event
window.addEventListener('resize', () => {
  output.innerHTML += `<p>Window resized to: ${window.innerWidth} x ${window.innerHeight}</p>`;
});

// 3. Scroll event
window.addEventListener('scroll', () => {
  output.innerHTML += `<p>Page scrolled to position: ${window.scrollY}px</p>`;
});
</script>
</body>
</html>
```

Event Handling in JavaScript

Name:

Favorite Color:

Green ▾

Submit

Page fully loaded!

Typing: k

Color changed to: Red

Form submitted! Name: k, Favorite Color: Red

Color changed to: Green

Page scrolled to position: 1px

Page scrolled to position: 2.5px

Page scrolled to position: 3.5px

Page scrolled to position: 4.5px

Page scrolled to position: 6px

Page scrolled to position: 7px

Page scrolled to position: 8px

Window resized to: 742 x 376

Window resized to: 794 x 428

DOM

- JavaScript and Document Object Model (DOM) are closely intertwined
- Help building interactive and dynamic web pages

- In DOM, all HTML elements are defined as objects
- Example:
`document.getElementById("demo").innerHTML = "Hello World!";`
 - Document is the object; `getElementById` is a method of document object, while `innerHTML` is a property

Power of Javascript

```
<html>
<head>
  <title>JavaScript Example</title>
  <style>
    #message {
      color: blue;
    }
    .hidden {
      display: none;
    }
  </style>
</head>
<body>
  <h1 id="title">Original Title</h1>
  <button id="changeContentBtn">Change Content</button>
  <button id="changeStyleBtn">Change Style</button>
  <button id="hideElementBtn">Hide Element</button>
```

```
<p id="message">This is a message.</p>
  <form id="exampleForm">
    <label for="email">Email: </label>
    <input type="text" id="email" name="email">
    <button type="submit">Submit</button>
  </form>

<p id="errorMessage" class="hidden">Please enter a valid email address.</p>

<script>
  // Change HTML Content
  document.getElementById("changeContentBtn").addEventListener("click", () =>
{
  document.getElementById("title").innerText = "Updated Title!";
});
```



```
// Change CSS Styling
```

```
document.getElementById("changeStyleBtn").addEventListener("click", () => {  
    const message = document.getElementById("message");  
    message.style.color = "green";  
    message.style.fontSize = "20px";  
});
```

```
// Hide Element
```

```
document.getElementById("hideElementBtn").addEventListener("click", () => {  
    document.getElementById("message").classList.add("hidden");  
});
```

```
// Validate Form
```

```
document.getElementById("exampleForm").addEventListener("submit", (event) => {  
    event.preventDefault();  
    const emailInput = document.getElementById("email").value;  
    const errorMessage = document.getElementById("errorMessage");  
  
    if (!emailInput.includes("@")) {  
        errorMessage.classList.remove("hidden");  
    } else {  
        errorMessage.classList.add("hidden");  
        alert("Form submitted successfully!");  
    }  
});
```

```
</script>
```

```
</body>
```

```
</html>
```

Original Title

Change Content

Change Style

Hide Element

This is a message.

Email:

Submit

```
<html>
<head>
  <title>DOM Manipulation Example</title>
</head>
<body>
  <h1>Form Manipulation and DOM Updates</h1>

  <form id="textForm">
    <p>Enter Text:</p>
    <input type="text" id="userInput" placeholder="Type something...">
    <button type="button" id="addTextButton">Add to Page</button>
  </form>

  <div id="output"></div>

  <script>
    const userInput = document.getElementById('userInput');
    const addTextButton = document.getElementById('addTextButton');
    const output = document.getElementById('output');
```

```
// 1. Convert input text to uppercase in real-time
userInput.addEventListener('input', () => {
  userInput.value = userInput.value.toUpperCase();
});

// 2. Add new element with the input text to the DOM
addTextButton.addEventListener('click', () => {
  const text = userInput.value;

  // Check if input is not empty
  if (text.trim() !== '') {
    // Create a new paragraph element
    const newParagraph = document.createElement('p');
    newParagraph.textContent = `You added: ${text}`;

    // Append the new paragraph to the output div
    output.appendChild(newParagraph);

    // Clear the input field after adding
    userInput.value = '';
  } else {
    alert('Please enter some text!');
  }
});
</script>
</body>
</html>
```

Form Manipulation and DOM Updates

Enter Text: Add to Page

You added: K

🌐 127.0.0.1:3002

Please enter some text!

OK

References

- Javascript in depth:

<https://www.w3schools.com/js/default.asp>

Summary

- Web Pages are written in HTML
- CSS specifies the presentation and styling of the HTML (XML) document
- Javascript makes web pages interactive
 - AJAX helps update web pages asynchronously

