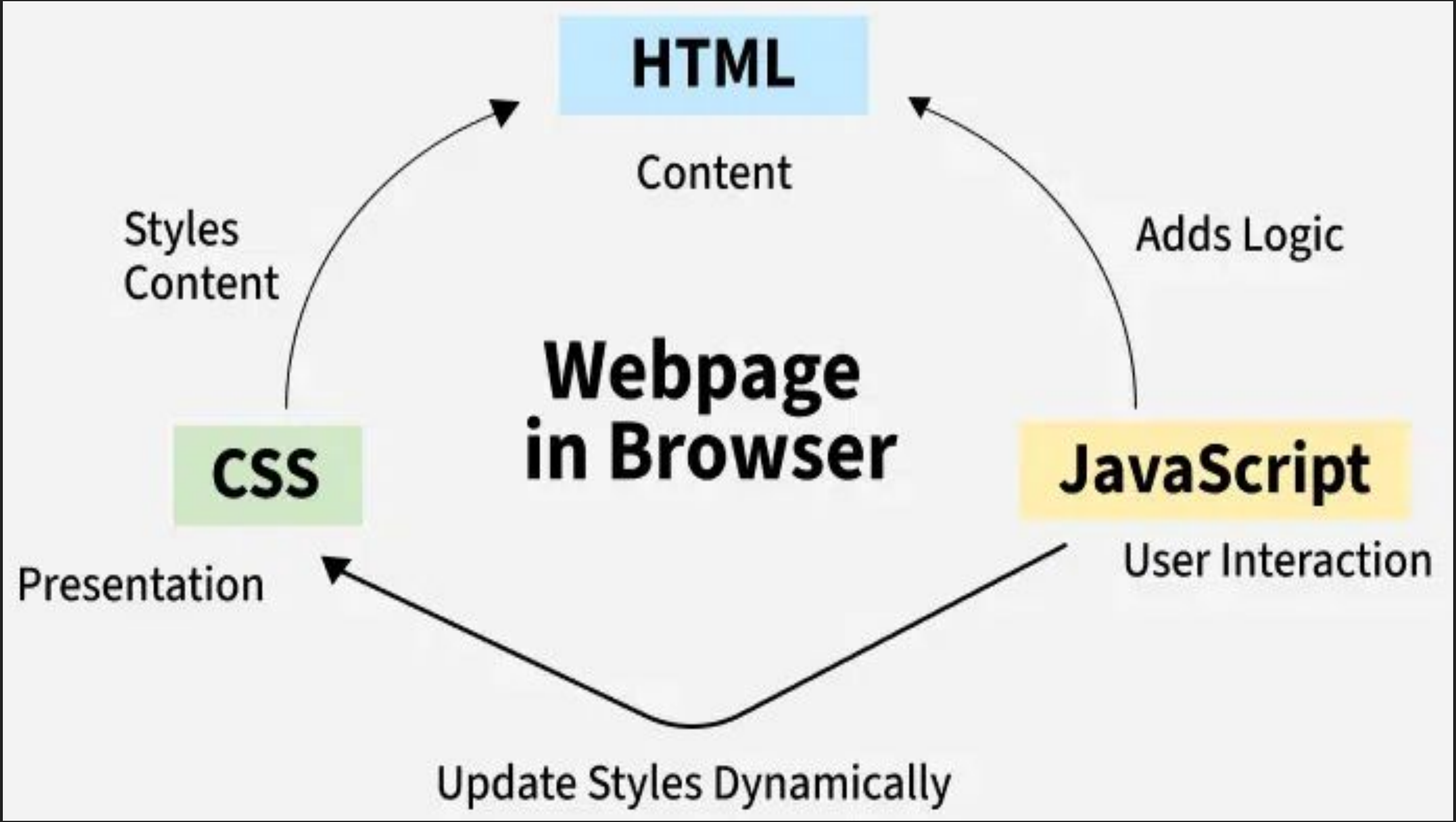
Unit-7
JavaScript
(Marks: 15)

CTEVT Diploma in Computer Engineering
Subject : Web Technology I
Prepared By: Er. Abinash Adhikari

Shree Sarwajanik Secondary Technical School


HTML
Content
Styles Content
Adds Logic
Webpage in Browser
CSS
JavaScript
Presentation
User Interaction
Update Styles Dynamically

What is JavaScript

JavaScript is a high-level, lightweight, interpreted programming language primarily used to make web pages interactive. It is client-side but can also be executed server-side (e.g., Node.js).

JavaScript is:

- ❖ Scripting language for web development.
- ❖ Dynamically typed and loosely coupled.
- ❖ Compatible with multiple browsers.
- ❖ Event-driven, enabling interactive UI features like dropdowns and form validation.

Why to Learn JavaScript?

1. JavaScript is the most popular programming language in the world, making it a programmer's great choice. Once you learn JavaScript, it helps you develop great front-end and back-end software using different JavaScript based frameworks like jQuery, Node.JS, etc.
2. JavaScript is everywhere, it comes installed on every modern web browser and so to learn JavaScript, you really do not need any special environment setup. For example, Chrome, Mozilla Firefox, Safari, and every browser you know as of today, supports JavaScript.
3. JavaScript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
4. JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as JavaScript Programmer.
5. Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.
6. Great thing about JavaScript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.
7. JavaScript is in all over the world, and companies like Google, Meta, Microsoft, PayPal, LinkedIn, etc. also use JavaScript.
8. Furthermore, JavaScript has more than 1.5 lakh libraries. It is also growing.
9. A huge community of JavaScript is available on the internet with students, developers, and mentors. So anyone can easily get support.

Advantages of JavaScript

- Fast Execution: JavaScript is executed directly in the browser without requiring a server-side interaction, improving performance.
- Versatile: It can be used for both front-end (e.g., DOM manipulation) and back-end development (e.g., Node.js).
- Rich User Interfaces: Provides features like drag-and-drop and sliders for user interaction.
- Cross-Browser Compatibility: Supported by all modern browsers.
- Event Handling: Handles events like clicks, keypresses, and mouse movements effortlessly.

Applications of JavaScript

1. **Client side validation**
   - ➢ This is really important to verify any user input before submitting it to the server and JavaScript plays an important role in validating those inputs at front-end itself.
2. **Manipulating HTML Pages**
   - ➢ JavaScript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using JavaScript and modify your HTML to change its look and feel based on different devices and requirements.
3. **User Notifications**
   - ➢ You can use JavaScript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
4. **Back-end Data Loading**
   - ➢ JavaScript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
5. **Presentations**
   - ➢ JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides Reveal JS and Bespoke JS libraries to build a web-based slide presentation.

6. **Server Applications**
   - ➢ Node JS is built on Chrome's JavaScript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.
7. **Machine learning**
   - ➢ Developers can use the ML5.js library to complete the task related to machine learning.
8. **Game Developments**
   - ➢ JavaScript contains multiple libraries and NPM packages to design graphics for the game. We can use HTML, CSS, and JavaScript with libraries to develop the games.
9. **Mobile applications**
   - ➢ We can use frameworks like React Native to build feature-rich mobile applications.
10. **Internet of Things (IoT)**
    - ➢ JavaScript is used to add functionality to devices like smartwatches, earbuds, etc.
11. **Data visualization**
    - ➢ JavaScript contains the libraries like D3.js to visualize the data efficiently. The D3.js is also used to prepare high-quality charts to visualize the data.
12. **Cloud computing**
    - ➢ We can use JavaScript in serverless computing platforms like Cloudflare and AWS lambda to write and deploy functions on the cloud.

Implementing JavaScript Code in an HTML Page using SCRIPT Tag

To add JavaScript in HTML document, several methods can be used. These methods include embedding JavaScript directly within the HTML file or linking an external JavaScript file.

1. Inline JavaScript
2. Internal JavaScript (Within <script> Tag)
3. External JavaScript (Using External File)

1. **Inline JavaScript:**
   You can write JavaScript code directly inside the HTML element using the onclick, onmouseover, or other event handler attributes.
   ```
   <button onclick="alert('Button Clicked!')">
   	Click Here
   </button>
   ```

2. <u>Internal JavaScript (Within <script> Tag):</u>

You can write JavaScript code inside the <script> tag within the HTML file. This is known as internal JavaScript and is commonly placed inside the <head> or <body> section of the HTML document.

```
<script>
    function greet() {
      console.log('Welcome to JavaScript!');
    }
</script>
```

3. <u>External JavaScript (Using External File):</u>

For larger projects or when reusing scripts across multiple HTML files, you can place your JavaScript code in an external .js file. This file is then linked to your HTML document using the src attribute within a <script> tag.

```
<head>
  <script src="script.js"></script>
</head>
```

Variables in JavaScript

A JavaScript variable is simply a name of storage location. The actual value of a variable can be changed at any time. Variables are helpful in various situations such as:

- Variables as Placeholders for unknown values:
  - They can be used in places where the values they represent are unknown when the code is written.
- Variables as Code Clarifies:
  - They can make the purpose of your code cleaner.

Rules of naming variables in JavaScript:

There are some rules while declaring a JavaScript variable (also known as identifiers).

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.

JavaScript Data Type-Variant Subtypes

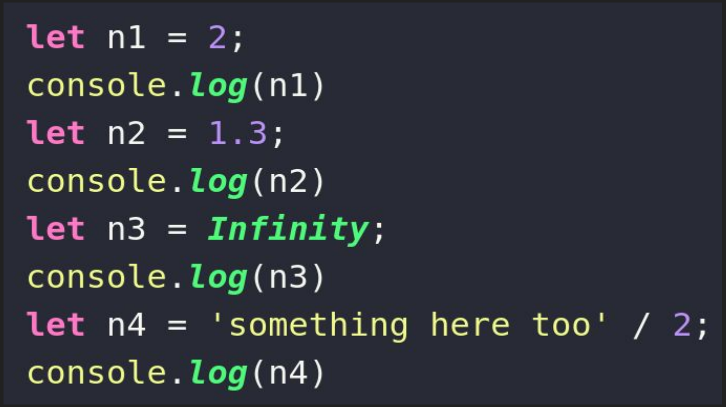
In JavaScript, each value has a data type, defining its nature (e.g., Number, String, Boolean) and operations. Data types are categorized into Primitive (e.g., String, Number) and Non-Primitive (e.g., Objects, Arrays).

1. Primitive Data Type
    a. Number
    b. String
    c. Boolean
    d. Null
    e. Undefined
    f. Symbol (Introduced in ES6)
    g. BigInt (Introduced in ES2020)
2. Non-Primitive Data Types
    a. Object
    b. Arrays
    c. Function
    d. Date Object
    e. Regular Expression

Primitive Data Type

1. **Number**

   The Number data type in JavaScript includes both integers and floating-point numbers. Special values like Infinity, -Infinity, and NaN represent infinite values and computational errors, respectively.

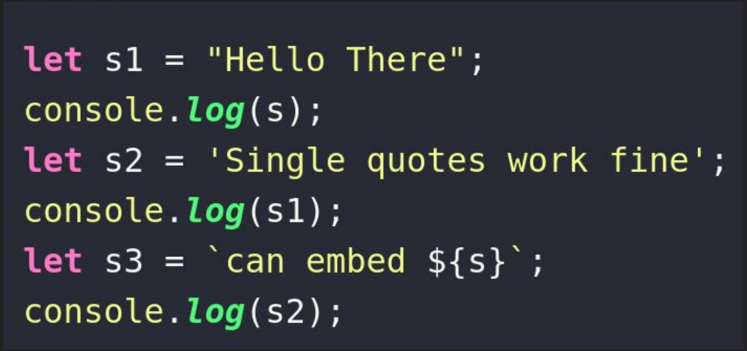


```
let n1 = 2;
console.log(n1)
let n2 = 1.3;
console.log(n2)
let n3 = Infinity;
console.log(n3)
let n4 = 'something here too' / 2;
console.log(n4)
```

2. **String**

   A String in JavaScript is a series of characters that are surrounded by quotes. There are three types of quotes in JavaScript, which are.

   There’s no difference between ‘single’ and “double” quotes in JavaScript. Backticks provide extra functionality as with their help of them we can embed variables inside them.



```
let s1 = "Hello There";
console.log(s);
let s2 = 'Single quotes work fine';
console.log(s1);
let s3 = `can embed ${s}`;
console.log(s2);
```

3. Boolean

The boolean type has only two values i.e. true and false.

```
let b1 = true;
console.log(b1);
let b2 = false;
console.log(b2);
```

4. Null

The special null value does not belong to any of the default data types. It forms a separate type of its own which contains only the null value.
The ‘null’ data type defines a special value that represents nothing, or empty value.

```
let age = null;
console.log(age)
```

5. Undefined

A variable that has been declared but not initialized with a value is automatically assigned the undefined value. It means the variable exists, but it has no value assigned to it.

```
let a;
console.log(a);
```

6. Symbol (Introduced in ES6)

Symbols, introduced in ES6, are unique and immutable primitive values used as identifiers for object properties. They help create unique keys in objects, preventing conflicts with other properties.

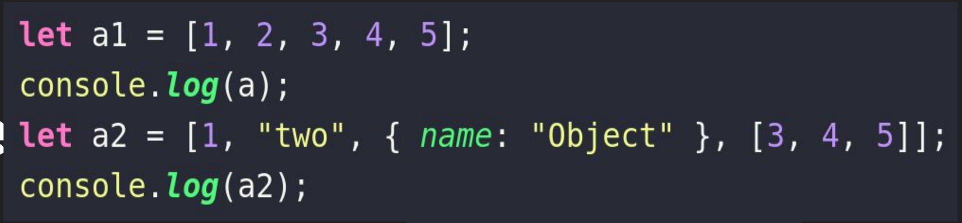
```
let s1 = Symbol("Geeks");
let s2 = Symbol("Geeks");
console.log(s1 == s2);
```

7. BigInt (Introduced in ES2020)

BigInt is a built-in object that provides a way to represent whole numbers greater than 253. The largest number that JavaScript can reliably represent with the Number primitive is 253, which is represented by the MAX_SAFE_INTEGER constant.

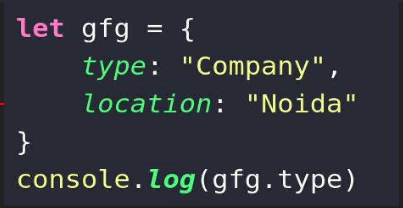
```
let b = BigInt("0b1010101001010101001111111111111111");
console.log(b);
```

Non-Primitive Data Type



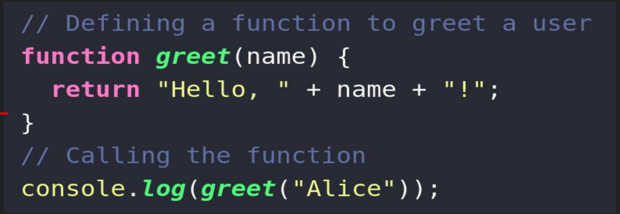
```
let a1 = [1, 2, 3, 4, 5];
console.log(a);
let a2 = [1, "two", { name: "Object" }, [3, 4, 5]];
console.log(a2);
```

- Object
  - JavaScript objects are key-value pairs used to store data, created with {} or the new keyword. They are fundamental as nearly everything in JavaScript is an object.



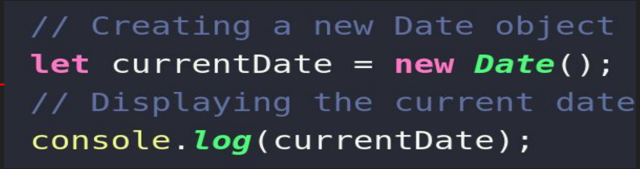
```
let gfg = {
    type: "Company",
    location: "Noida"
}
console.log(gfg.type)
```

- Arrays
  - An Array is a special kind of object used to store an ordered collection of values, which can be of any data type.
- Function
  - A function in JavaScript is a block of reusable code designed to perform a specific task when called.



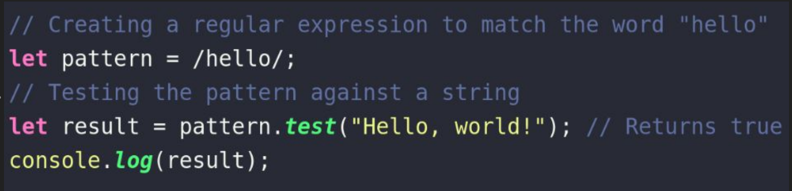
```
// Defining a function to greet a user
function greet(name) {
  return "Hello, " + name + "!";
}
// Calling the function
console.log(greet("Alice"));
```

- Date Object
  - The Date object in JavaScript is used to work with dates and times, allowing for date creation, manipulation, and formatting.



```
// Creating a new Date object
let currentDate = new Date();
// Displaying the current date
console.log(currentDate);
```

- Regular Expression
  - A RegExp (Regular Expression) in JavaScript is an object used to define search patterns for matching text in strings.



```
// Creating a regular expression to match the word "hello"
let pattern = /hello/;
// Testing the pattern against a string
let result = pattern.test("Hello, world!"); // Returns true
console.log(result);
```

Data Type	Description	Example
String	Textual data.	'hello', "hello world!", etc.
Number	An integer or a floating-point number.	3, 3.234, 3e-2, etc.
BigInt	An integer with arbitrary precision.	900719925124740999n, 1n, etc.
Boolean	Any of two values: true or false.	true and false
undefined	A data type whose variable is not initialized.	let a;
null	Denotes a null value.	let a = null;
Symbol	A data type whose instances are unique and immutable.	let value = Symbol('hello');
Object	Key-value pairs of collection of data.	let student = {name: "John"};

JavaScript Functions

Reference Links:

1. http://programiz.com/javascript/function
2. https://www.w3schools.com/js/js_functions.asp
3. https://www.geeksforgeeks.org/functions-in-javascript/

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

Functions are useful in organizing the different parts of a script into the several tasks that must be completed. There are mainly two advantages of JavaScript functions.

- ❖ **Code reusability**: We can call a function several times in a script to perform their tasks so it saves coding.
- ❖ **Less coding:** It makes our program compact. We don't need to write many lines of code each time to perform a common task.

Rules for naming functions:

- ❖ It must be case sensitive.
- ❖ It must be start with alphabetical character (A-Z) or an underscore symbol.
- ❖ It cannot contain spaces.
- ❖ It cannot be use as a reserve words.

How to declare a Function:

To declare a function we have to use the reserved keyword "function", followed by its name, and a set of arguments.

JavaScript Function Syntax

```
        function functionName([arg1, arg2, ...argN]){
                //code to be executed
        }
```

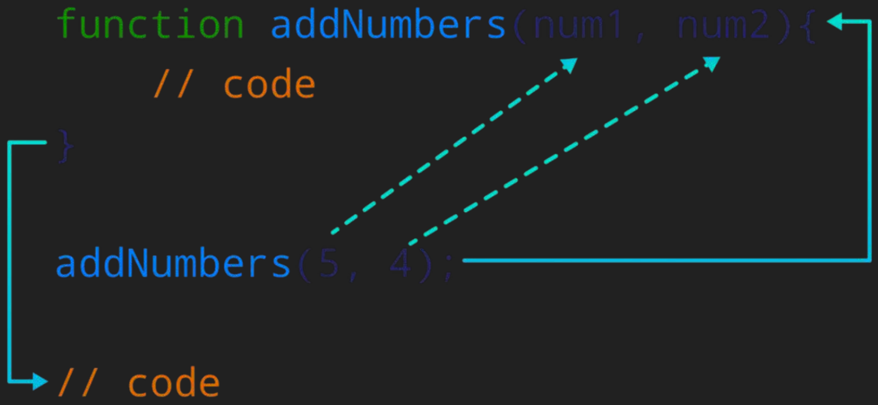
In the above syntax, function is a reserved keyword and "functionName" is a name given to the function. JavaScript Functions can have 0 or more arguments.

JavaScript Function Example

```
<script>
        function msg(){
                alert("hello! this is message");
        }
</script>
<input type=button onclick=msg() value="call function"/>
```

Why Functions?

1. Functions can be used multiple times, reducing redundancy.
2. Break down complex problems into manageable pieces.
3. Manage complexity by hiding implementation details.
4. Can call themselves to solve problems recursively.



```
function addNumbers(num1, num2){
    // code
}

addNumbers(5, 4);

// code
```

- **Function Declaration**
  - A function declaration defines a function with the function keyword.
    ```
    function greet(name) {
        return `Hello, ${name}!`;
    }
    ```
  - console.log(greet("Alice")); // Output: Hello, Alice!
  - Can be called before its definition due to hoisting.
  - Always has a function name.
- **Function Expression**
  - A function expression defines a function inside an expression.
    ```
    const greet = function(name) {
        return `Hello, ${name}!`;
    };
    ```
  - console.log(greet("Bob")); // Output: Hello, Bob!
  - Anonymous function assigned to a variable.
  - Cannot be called before declaration (not hoisted).

- **Arrow Function (ES6)**
  - A shorter syntax for writing functions.
    ```
    const greet = (name) => `Hello, ${name}!`;
    console.log(greet("Charlie")); // Output: Hello, Charlie!
    ```
  - Implicit return if using one line.
  - No this binding (useful in callbacks and event handlers).
- **Immediately Invoked Function Expression (IIFE)**
  - A function that runs immediately after it is defined.
    ```
    (function() {
        console.log("I am an IIFE!");
    })();
    ```
  - Prevents polluting the global scope.
  - Often used for initialization logic.
- **Higher-Order Functions**
  - A function that takes another function as a parameter or returns a function.
    ```
    function operate(a, b, operation) {
        return operation(a, b);
    }
    const add = (x, y) => x + y;
    console.log(operate(5, 3, add)); // Output: 8
    ```
  - Used in array methods like .map(), .filter(), .reduce().

- **Callback Functions**
  - A function passed as an argument to another function.

    ```
    function fetchData(callback) {
        setTimeout(() => {
            callback("Data received");
        }, 1000);
    }
    fetchData((message) => console.log(message)); // Output (after 1s): Data
    received
    ```
  - Used in asynchronous operations like API calls.

- **Closures**
- A function that remembers variables from its outer scope even after execution.

```
function counter() {
    let count = 0;
    return function() {
                count++;
                console.log(count);
            };
}
const increment = counter();
increment(); // Output: 1
increment(); // Output: 2
```

- Used in data privacy and function factories.

- **Function Parameters and Default Values**
    - We can assign default values to function parameters.

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

- **Rest and Spread Operators in Functions**
  - Rest Operator (...args) allows a function to take multiple arguments.

```
function sum(...numbers) {
    return numbers.reduce((acc, num) => acc + num, 0);
}
console.log(sum(1, 2, 3, 4)); // Output: 10
```

    Spread Operator (...) can be used to pass array elements as function arguments.

```
const nums = [1, 2, 3];
console.log(Math.max(...nums)); // Output: 3
```

Function Scope and this Keyword

- Functions have access to variables within their own scope, parent scope, and global scope.
- Arrow functions do not have their own this context.

```
const person = {
    name: "Alice",
    greet: function() {
        console.log(`Hello, ${this.name}`);
    }
};
person.greet(); // Output: Hello, Alice
```

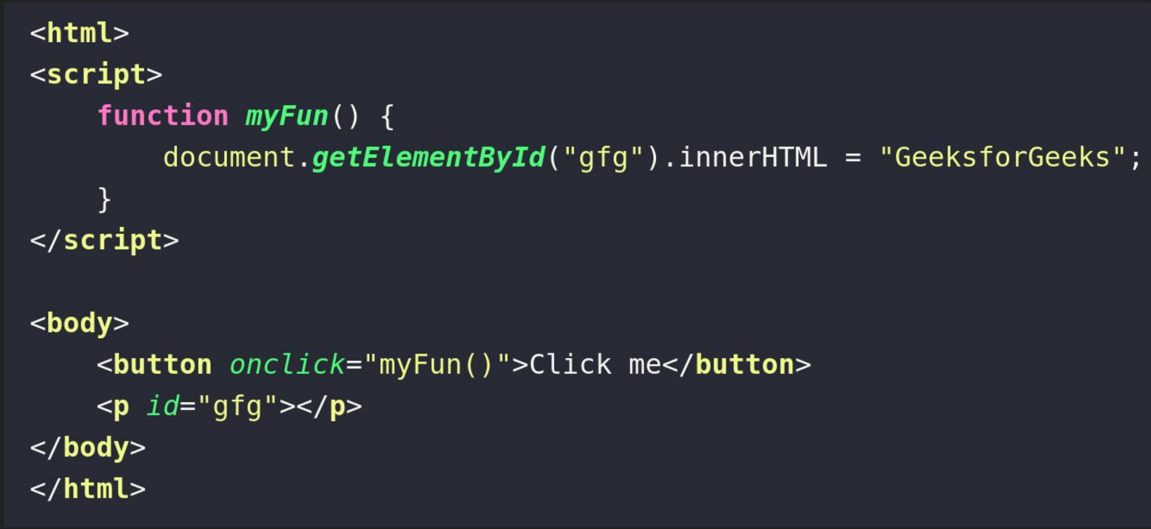
JavaScript Events

Reference Links:

1. https://www.geeksforgeeks.org/javascript-events/
2. https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers
3. https://www.javatpoint.com/javascript-events

JavaScript Events

JavaScript Events are actions or occurrences that happen in the browser. They can be triggered by various user interactions or by the browser itself.



```
<html>
<script>
    function myFun() {
        document.getElementById("gfg").innerHTML = "GeeksforGeeks";
    }
</script>

<body>
    <button onclick="myFun()">Click me</button>
    <p id="gfg"></p>
</body>
</html>
```

- ❖ The onclick attribute in the <button> calls the myFun() function when clicked.
- ❖ The myFun() function updates the <p> element with id=”gfg” by setting its innerHTML to “GeeksforGeeks”.
- ❖ Initially, the <p> is empty, and its content changes dynamically on button click.

Event Types

JavaScript supports a variety of event types. Common categories include:

1. Mouse Events: click, dblclick, mousemove, mouseover, mouseout
2. Keyboard Events: keydown, keypress, keyup
3. Form Events: submit, change, focus, blur
4. Window Events: load, resize, scroll

Common JavaScript Events

Event Attribute	Description
onclick	Triggered when an element is clicked.
onmouseover	Fired when the mouse pointer moves over an element.
onmouseout	Occurs when the mouse pointer leaves an element.
onkeydown	Fired when a key is pressed down.
onkeyup	Fired when a key is released.
onchange	Triggered when the value of an input element changes.
onload	Occurs when a page has finished loading.
onsubmit	Fired when a form is submitted.
onfocus	Occurs when an element gets focus.
onblur	Fired when an element loses focus.

```
// Mouse Event
document.addEventListener("mousemove", (e) => {
  console.log(`Mouse moved to (${e.clientX}, ${e.clientY})`);
});
// Keyboard Event
document.addEventListener("keydown", (e) => {
  console.log(`Key pressed: ${e.key}`);
});
```

- ❖ The mousemove event tracks cursor movement.
- ❖ The keydown event captures key presses.

Event Handling Methods

1. Inline HTML Handlers

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

2. DOM Property Handlers

```
let btn = document.getElementById("myButton");
btn.onclick = () => {
    alert("Button clicked!");
};
```

3. addEventListener()

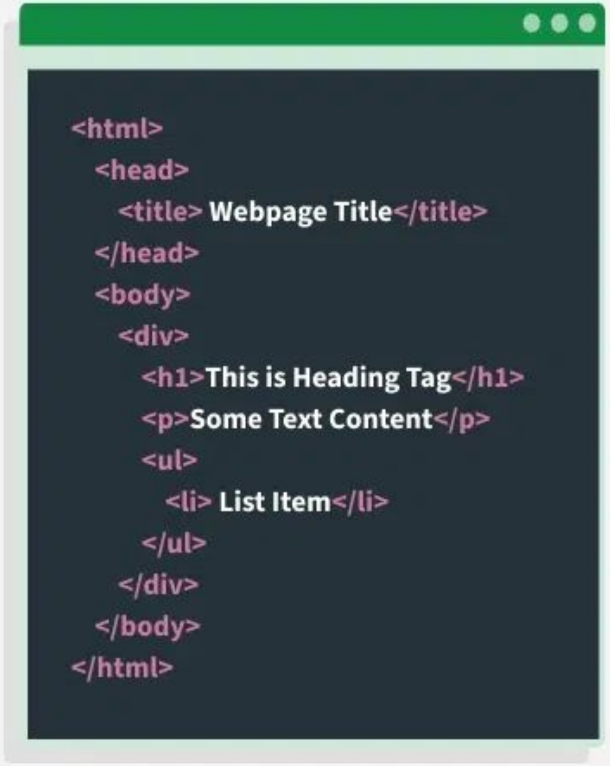
```
btn.addEventListener("click", () => {
  alert("Button clicked using addEventListener!");
});
```

Document Object Model

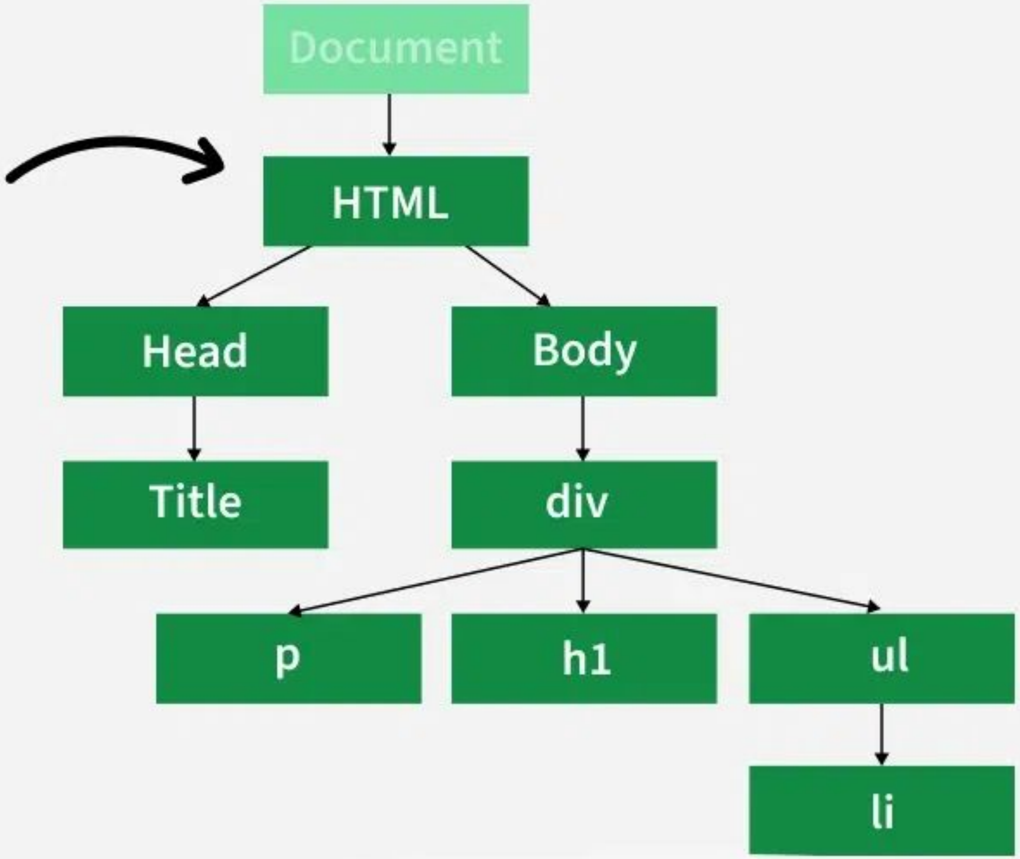
Reference Links:

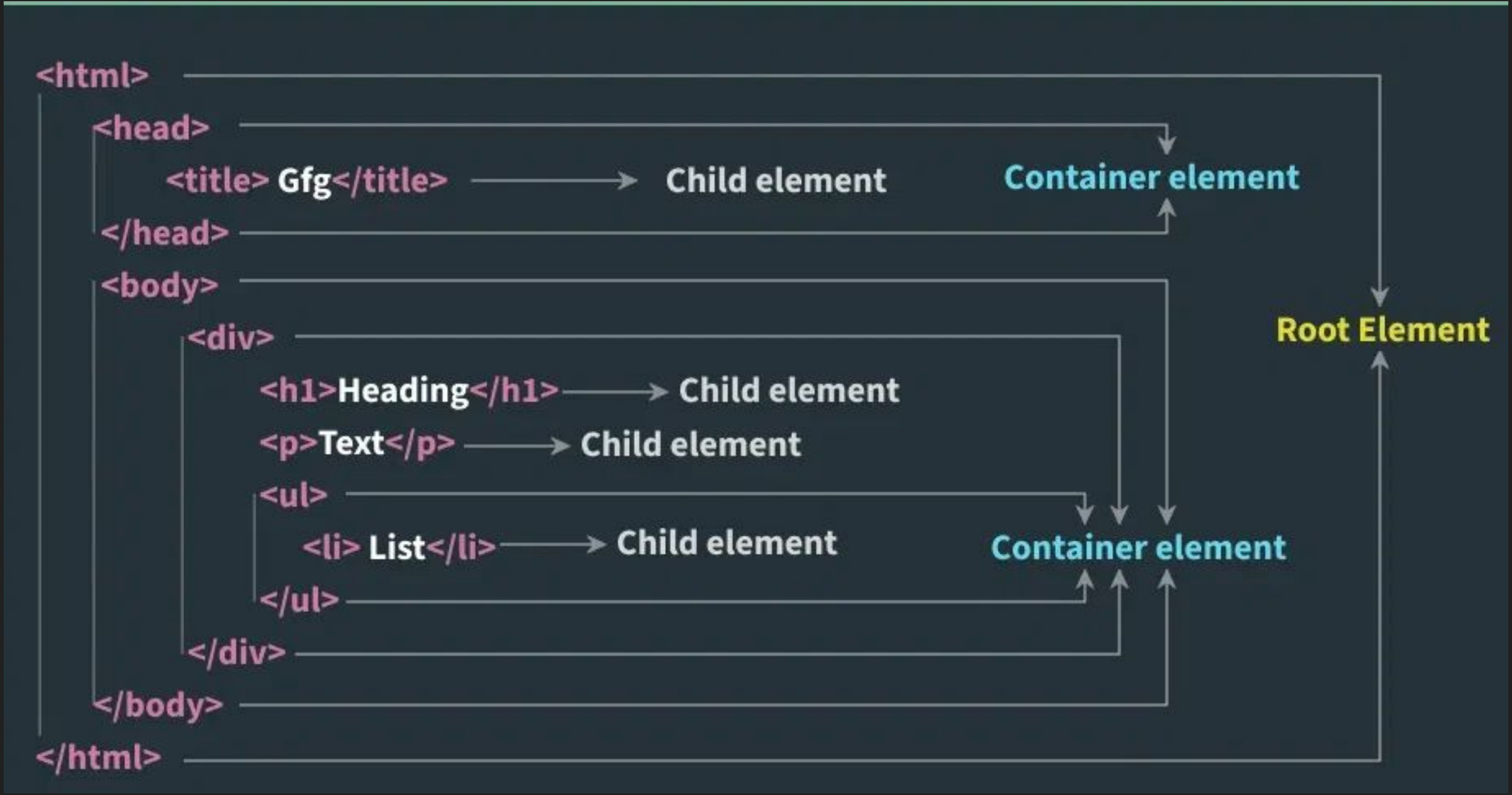
1. https://www.geeksforgeeks.org/dom-document-object-model/
2. https://www.w3schools.com/js/js_htmldom.asp
3. https://www.javatpoint.com/document-object-model

The “DOM Tree”



```
<html>
  <head>
    <title> Webpage Title</title>
  </head>
  <body>
    <div>
      <h1>This is Heading Tag</h1>
      <p>Some Text Content</p>
      <ul>
        <li> List Item</li>
      </ul>
    </div>
  </body>
</html>
```




```html
<html>
  <head>
    <title> Gfg</title>
  </head>
  <body>
    <div>
      <h1>Heading</h1>
      <p>Text</p>
      <ul>
        <li> List</li>
      </ul>
    </div>
  </body>
</html>
```

HTML DOM (Document Object Model)

The HTML DOM (Document Object Model) is a programming interface that represents the structure of a web page in a way that programming languages like JavaScript can understand and manipulate.

Think of it as a tree of objects where each part of your HTML document (elements, attributes, text) is represented as a node, allowing you to dynamically change or interact with the content and structure of the page.

What Does the HTML DOM Look Like?

Imagine your webpage as a tree

1. The document is the root.
2. HTML tags like <html>, <head>, and <body> are branches.
3. Attributes, text, and other elements are the leaves.

Why is DOM Required?

The DOM is essential because

1. Dynamic Content Updates: Without reloading the page, the DOM allows content updates (e.g., form validation, AJAX responses).
2. User Interaction: It makes your webpage interactive (e.g., responding to button clicks, form submissions).
3. Flexibility: Developers can add, modify, or remove elements and styles in real-time.
4. Cross-Platform Compatibility: It provides a standard way for scripts to interact with web documents, ensuring browser compatibility.

How the DOM Works?

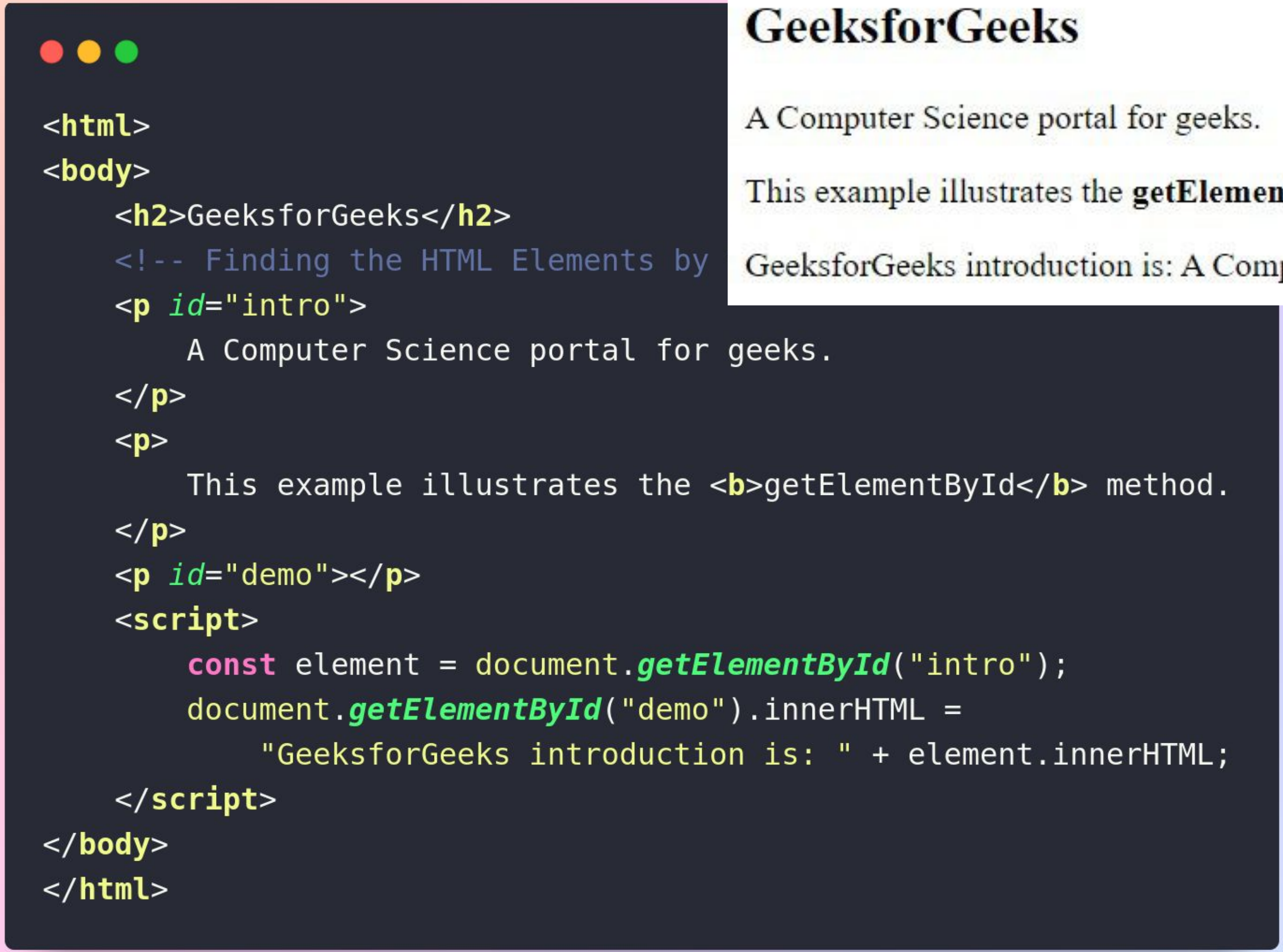
1. The DOM connects your webpage to JavaScript, allowing you to:
2. Access elements (like finding an <h1> tag).
3. Modify content (like changing the text of a <p> tag).
4. React to events (like a button click).
5. Create or remove elements dynamically.

Properties of the DOM

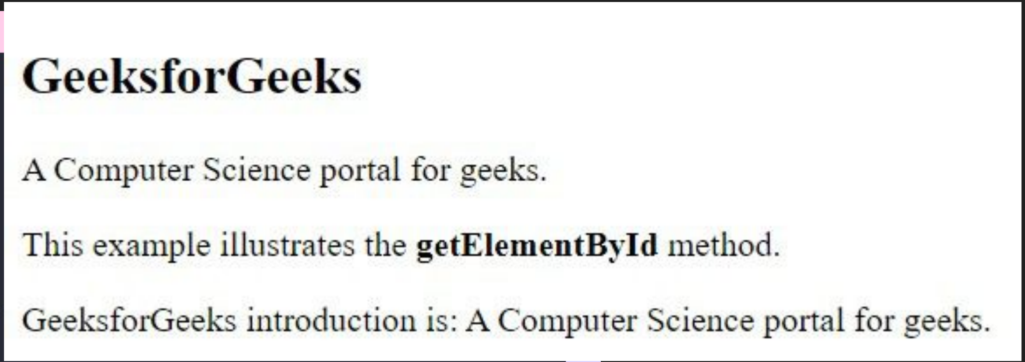
1. Node-Based: Everything in the DOM is represented as a node (e.g., element nodes, text nodes, attribute nodes).
2. Hierarchical: The DOM has a parent-child relationship, forming a tree structure.
3. Live: Changes made to the DOM using JavaScript are immediately reflected on the web page.
4. Platform-Independent: It works across different platforms, browsers, and programming languages.

Commonly Used DOM Methods

Methods	Description
getElementById(id)	Selects an element by its ID.
getElementsByClassName(class)	Selects all elements with a given class.
querySelector(selector)	Selects the first matching element.
querySelectorAll(selector)	Selects all matching elements.
createElement(tag)	Creates a new HTML element.
appendChild(node)	Adds a child node to an element.
remove()	Removes an element from the DOM.
addEventListener(event, fn)	Attaches an event handler to an element.



```
<html>
<body>
    <h2>GeeksforGeeks</h2>
    <!-- Finding the HTML Elements by
    <p id="intro">
        A Computer Science portal for geeks.
    </p>
    <p>
        This example illustrates the <b>getElementById</b> method.
    </p>
    <p id="demo"></p>
    <script>
        const element = document.getElementById("intro");
        document.getElementById("demo").innerHTML =
            "GeeksforGeeks introduction is: " + element.innerHTML;
    </script>
</body>
</html>
```



GeeksforGeeks

A Computer Science portal for geeks.

This example illustrates the **getElementById** method.

GeeksforGeeks introduction is: A Computer Science portal for geeks.

Browser Objects and Events

The Browser Object Model (BOM) interacts with the browser:

- window: Represents the browser window (global object).
- navigator: Browser information.
- screen: Screen size and resolution.
- Example:
  - alert(window.innerHeight); // Height of the browser window

Document Objects and Events

Document Object: Represents the webpage.

Access elements using:

```
document.getElementById()
document.querySelector()
```

Example:

```
let element = document.getElementById("header");
element.style.color = "blue";
```

Events: Handle interactions like onclick or onchange.

Form Objects and Events

Form Objects: Represent <form> elements.

Example:

```
let form = document.forms["myForm"];
alert(form["username"].value);
```

Events: Handle submission or input changes.

Example:

```
document.getElementById("myForm").onsubmit = function (e) {
  e.preventDefault();
  alert("Form submitted!");
};
```

Dialog Boxes Supported by JavaScript

JavaScript provides built-in dialog boxes:

❖ **Alert Box:**

```
alert("Hello, World!");
```

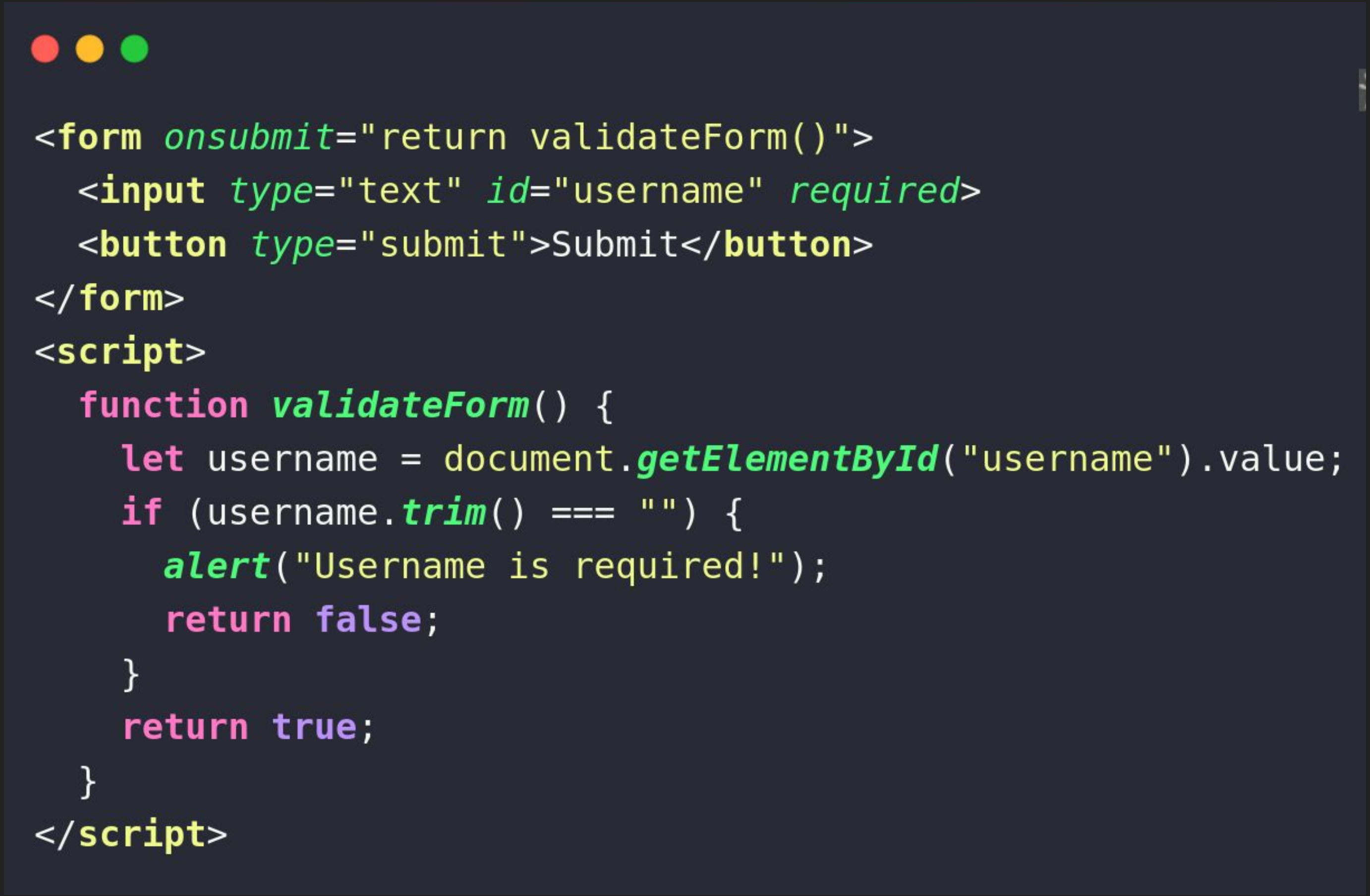
❖ **Confirm Box:**

```
if (confirm("Are you sure?")) {
  console.log("Confirmed!");
}
```

❖ **Prompt Box:**

```
let name = prompt("What is your name?");
alert(`Hello, ${name}!`);
```

Form Validation



```
<form onsubmit="return validateForm()">
  <input type="text" id="username" required>
  <button type="submit">Submit</button>
</form>
<script>
  function validateForm() {
    let username = document.getElementById("username").value;
    if (username.trim() === "") {
      alert("Username is required!");
      return false;
    }
    return true;
  }
</script>
```