JavaScript

Dynamic Behavior in a Web Page

Dr. Jenila Livingston L.M.
Professor
VIT Chennai

Module 4

- Hello World Web Page
- Variables, Identifiers
- Assignment Statements and Objects
- Functions
- Document Object Model
- Forms: form Element, Controls, Buttons (Module1)
- Text Control Accessing a Form's Control Values
- reset and focus Methods
- Event Handler Attributes: onchange, onmouseover, onmouseout.

Presentation Overview

1. Introduction
2. Using JavaScript
   - The First Script
   - Internal Script
     - Event Handler
   - External Script
3. Other Elements
   - noscript
   - defer/sync/async
   - Dynamically Loaded Scripts
4. Three methods or Popup boxes
5. JAVASCRIPT INPUT-OUTPUT

1. JavaScript - Introduction

- JavaScript is a front-end scripting language developed by Netscape for dynamic content
  - Lightweight, but with limited capabilities
  - Can be used as object-oriented language
- Client-side technology
  - Embedded in your HTML page
  - Interpreted by the Web browser
- Simple and flexible
- Can read and write HTML elements and Powerful to manipulate the DOM

JavaScript Advantages

- JavaScript allows interactivity such as:
  - Implementing form validation
  - React to user actions, e.g. handle events
    - Changing an image on moving mouse over it
    - Sections of a page appearing and disappearing
  - Can handle exceptions
  - Content loading and changing dynamically
  - Performing complex calculations
  - Can access / modify browser cookies
  - Custom HTML controls, e.g. scrollable table
  - Implementing AJAX functionality (asynchronous server calls)

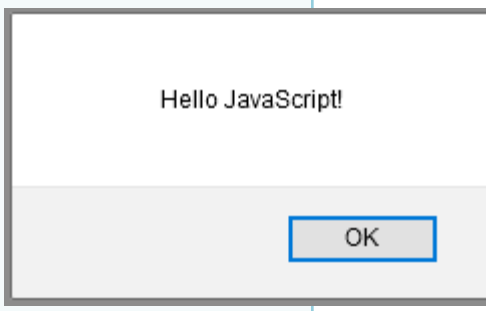
JavaScript Vs Java

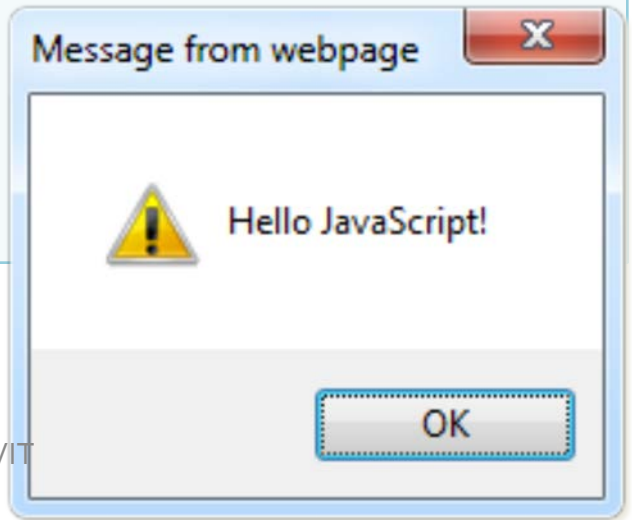
JavaScript	Java
Interpreted by the client-side computer	Compiled on the server before executed on the client machine
Dynamic binding, object references are checked at runtime	Static binding, object references must exist at compile time
No need to declare data types	Data types must be declared
Code is embedded in HTML	Code is not integrated in HTML
Limited by the browser functionality	Java applications are standalone
Can access browser objects	Java has no access to browser objects

Using JavaScript
2.1 The First Script

first-script.html

```
<html>

<head>
  <script type="text/javascript">
     alert('Hello JavaScript!');
  </script>
</head>

</html>
```





Note: type="text/javascript": Specifies that the script is JavaScript (**optional** in modern HTML, as it's the default).

Comments in JavaScript

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

let x = 5; // Comment after a statement

/* This is a **multi-line comment** spanning multiple lines */

var y = 10;

Semicolon

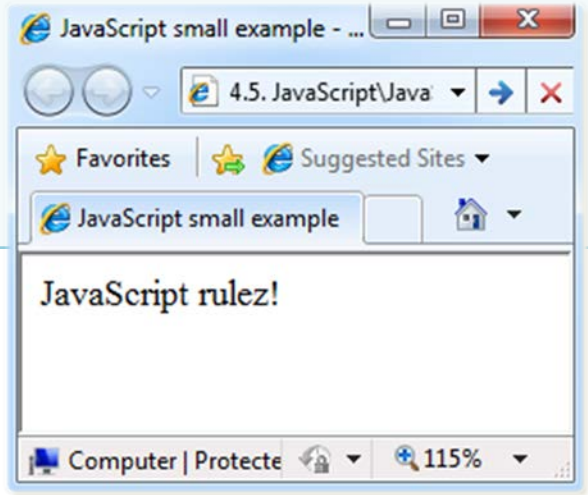
- Semicolon specifies the end of a statement.
- Semicolon can be omitted if a line break is used.

2.2 Internal Script

internal-script.html

- <script> tag in the head
- <script> tag in the body – not recommended

```
<html>

<head>
  <script type="text/javascript">
     document.write('JavaScript rulez!');
  </script>
</head>

</html>
```

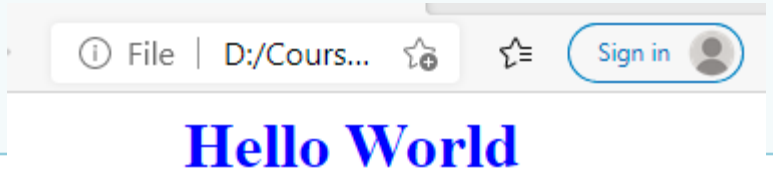


Internal Script

Embedding HTML tags with document.write

internal-script.html

```
<html>

<head>
<script>
document.write("<h1 style=color:blue;text-
align:center;>Hello World</h1>");
</script>
</head>

</html>
```



JavaScript – When is Executed?

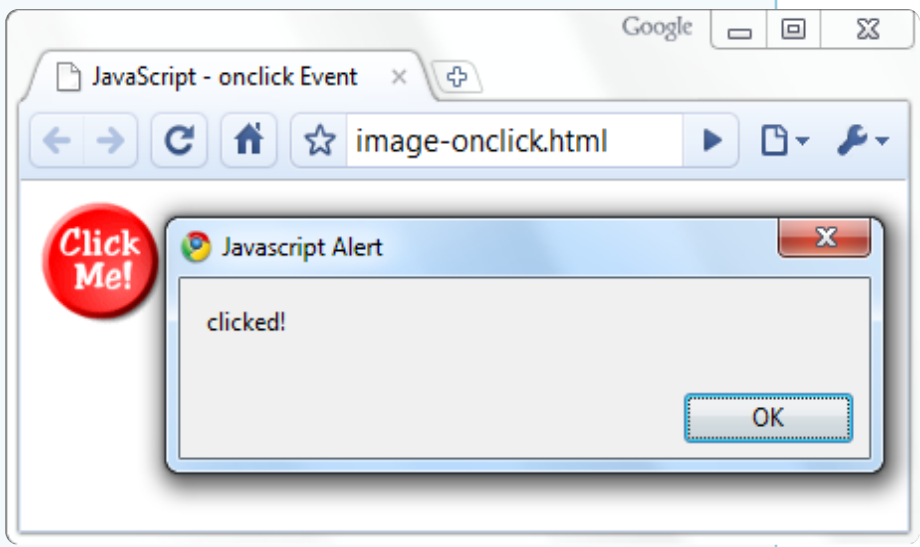
- JavaScript code is executed during the **page loading** or when the browser fires an **event**
  - All statements are executed at page loading
  - Some statements just define functions that can be called later
- Function calls or code can be attached as "event handlers" via tag attributes
  - Executed when the event is fired by the browser

```
<img src="logo.gif" onclick="alert('clicked!')" />
```

Calling a JavaScript Function from Event Handler – Example

image-onclick.html

```
<html>
<head>
<script type="text/javascript">
  function test() {
      alert('clicked!');
  }
</script>
</head>

<body>
  <img src="logo.gif"
    onclick="test()" />
</body>
</html>
```



HTML Event Attributes

Event	Description
onchange	An HTML Element has been changed
onclick	The user clicks an HTML Element
onmouseover	The user moves the mouse over the HTML Element
onmouseout	The user moves the mouse away from the HTML Element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

2.3 External Script

- The JavaScript code can be placed in:
  - External files, linked via `<script>` tag the head
    - Files usually have `.js` extension

```
<script src="scripts.js" type="text/javscript">
<!- code placed here will not be executed! -->
</script>
```

    - Highly recommended
    - The `.js` files get cached by the browser

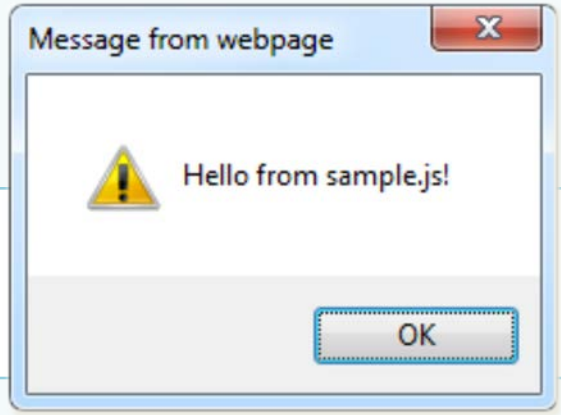
External Script – Function Call

- Using external script files:

external-JavaScript.html

```
<html>
<head>
  <script src="sample.js" type="text/javascript">
  </script>
</head>
<body>
  <button onclick="sample()" value="Call external
JavaScript function" />
</body>
</html>
```

The <script> tag is always empty.



- External JavaScript file:

```
function sample() {
  alert('Hello from sample.js!')
}
```

sample.js

Area of Rectangle– **Internal JavaScript**

arearect.html

```
<html>

<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function area() {
      length = parseInt(document.F1.t1.value);
      breadth = parseInt(document.F1.t2.value);
      area = length * breadth;
      alert(area);
    }
  </script>
</head>
```

Area of Rectangle– Internal JavaScript

Arearect.html(contd..)

```
<body>
  <form name="F1">
    <input type="text" name="t1" /> <br/>
    <input type="text" name="t2" /> <br/>
    <input type="button" value="Process"
      onclick="area()" />
  </form>
</body>

</html>
```

Area of Rectangle– **External JavaScript1**

Arearect.html

```
<html>
<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript" src="rect.js">
  </script>
</head>
<body>
  <form name="F1">
    <input type="text" name="t1" /> <br/>
    <input type="text" name="t2" /> <br/>
    <input type="button" value="AreaCalc"
      onclick="area()" />
  </form>
</body>
</html>
```

Area of Rectangle– External JavaScript1

rect.js

Using name.value

```
function area() {
      length = parseInt(document.F1.t1.value);
      breadth = parseInt(document.F1.t2.value);
      area = length * breadth;
      alert(area);
    }
```

Area of Rectangle– External JavaScript2

Arearect.html

```
<html>
<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript" src="rect.js">
  </script>
</head>
<body>
  <form name="F1">
    <input type="text" id="t1" /> <br/>
    <input type="text" id="t2" /> <br/>
    <input type="button" value="AreaCalc"
      onclick="area()" />
  </form>
</body>
</html>
```

Area of Rectangle– External JavaScript2

getElementById

rect.js

```
function area() {
   length = parseInt(document.getElementById("t1"));
   breadth = parseInt(document.getElementById("t2"));
    area = length * breadth;
    alert(area);
 }
```

Advantage of using External script

- Once an external script is downloaded it is kept in memory.
- The next time the page loads it refers to it.
- There is no need to re-download the script every time the page is loaded.
- For large scripts it suggested to make it external.

3. Other Elements
3.1 <noscript> Element

- Sometimes JavaScript can be disabled in browsers.
- In such case the **alternate content** can be specified in the <noscript> block.

<noscript> Element

```
<html>
<body>
<script type="text/javascript">
alert("Hello");
</script>
<noscript>JavaScript is disabled</noscript>
</body>
</html>
```

3.2 Defer/sync/async

- The defer attribute of the script tag **delays the execution of the script** after the DOM has been loaded.
- The DOM-Document Object Model is the object representation of all tags and details of the layout page.
- Deferred script works only for external script.

Deferred script Demo

Use: defer="defer", defer="true", or just defer

```
<html>
<head>
<script type="text/javascript" src="script.js" defer="defer">
</script>
</head>
<body>
<h1>HELLO WORLD</h1>
</body>
</html>
```

script.js

```
alert("welcome");
```

defer="defer" ensures that the script is executed only after the HTML document has been fully parsed. After parsing, the browser executes the alert message.

Script without defer (sync)

By default, the browser executes the script immediately as it encounters the <script> tag in the HTML document.

```
<!DOCTYPE html> <html>
<head>
<script src="script1.js"></script>
<script src="script2.js"></script>
</head>
<body>
<h1>Hello World</h1> </body>
</html>
```

The browser:

1. Encounters script1.js, stops parsing the HTML, downloads, and executes it.
2. Moves to script2.js, downloads, and executes it.
3. Resumes parsing the HTML.

script1.js

```
console.log("Script 1 is executed");
```

script2.js

```
console.log("Script 2 is executed");
```

Output:

```
Script 1 is executed

Script 2 is executed
```

Async

- <!DOCTYPE html>
- <html> <head> <title>Async Script Example</title>
- <script src="script1.js" async></script>
- <script src="script2.js" async></script> </head>
- <body>
- <h1>Async Script Example</h1>
- Check the console for script execution order.
- </body> </html>

script1.js

```
console.log("Script 1 is executed");
```

script2.js

```
console.log("Script 2 is executed");
```

Output: Order may vary

```
Script 1 is executed
Script 2 is executed
```

```
Script 2 is executed
Script 1 is executed
```

3.3 Dynamically Loaded Scripts

- To load and run an external JavaScript we use src attribute in script tag.
- Consider a situation where you need to choose between two JavaScript files at the time of loading.
- The script file has to be loaded dynamically.

Dynamically Loaded Scripts

```
<script type="text/javascript">
function dynLoad(file) {
var selm = document.createElement("script");
selm.type = "text/javascript";
selm.src = file;
document.body.appendChild(selm);
}
</script>
<button onclick="dynLoad('script1.js');">Load script1.js</button>
<button onclick="dynLoad('script2.js');">Load script2.js</button>
```

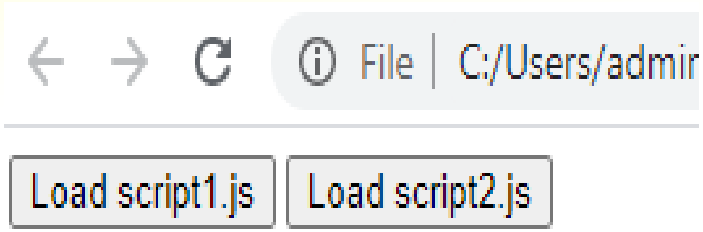
script1.js

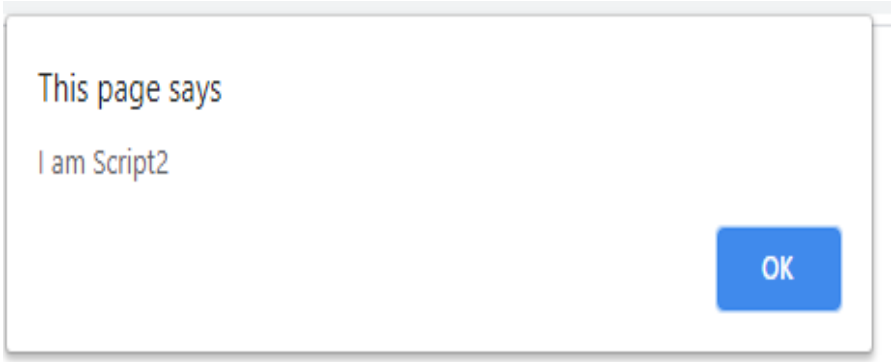
```
alert("I am Script1");
```

script2.js

```
alert("I am Script2");
```

Dynamically Loaded Scripts





4. Standard Popup Boxes

- Alert box with text and [OK] button
  - Just a message shown in a dialog box:

```
alert("Some text here");
```

- Confirmation box
  - Contains text, [OK] button and [Cancel] button:

```
confirm("Are you sure?");
```

- Prompt box
  - Contains text, input field with default value:

```
prompt ("enter amount", 10);
```

Using the alert() Method

```
<head>
<script language="JavaScript">
        alert("An alert triggered by JavaScript");
</script>
</head>
```

- It is the easiest methods to use amongst alert(), prompt() and confirm().
- You can use it to **display textual information** to the user (simple and concise).
- The user can simply click "OK" to close it.

Display value of a variable using the alert()

```
<!DOCTYPE html> <html>
<head> <title>Display Variable Value</title>
<script>
function showAlert() {
var message = "Hello, World!";
alert(message);
}
</script>
</head>
<body>
<button onclick="showAlert()">Click me</button>
</body> </html>
```

Using the confirm() Method

```
<head>
<script language="JavaScript">
        confirm("Are you happy with the class?");
</script>
</head>
```

- This box is used to give the user a choice either **OK or Cancel**.
- It is very similar to the "alert()" method.
- You can also put your message in the method.

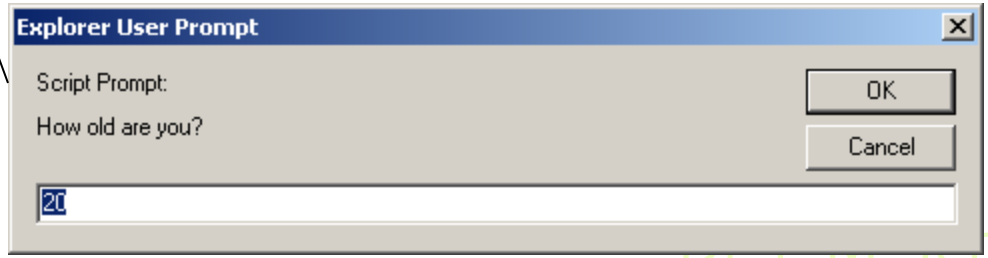
Using the prompt() Method

```
<head>
<script language="JavaScript">
        prompt("What is your student id number?");
        prompt("What is your name?","No name");
</script>
</head>
```

- This is the only one that allows the user to type in his own response to the specific question.
- You can give a default value to avoid displaying "undefined".

Three methods

```
<script language="JavaScript">
alert("This is an Alert method");
confirm("Are you OK?");
prompt("What is your name?");
prompt("How old are you?","20");
</script>
```



5. JAVASCRIPT INPUT-OUTPUT

Variable Declaration

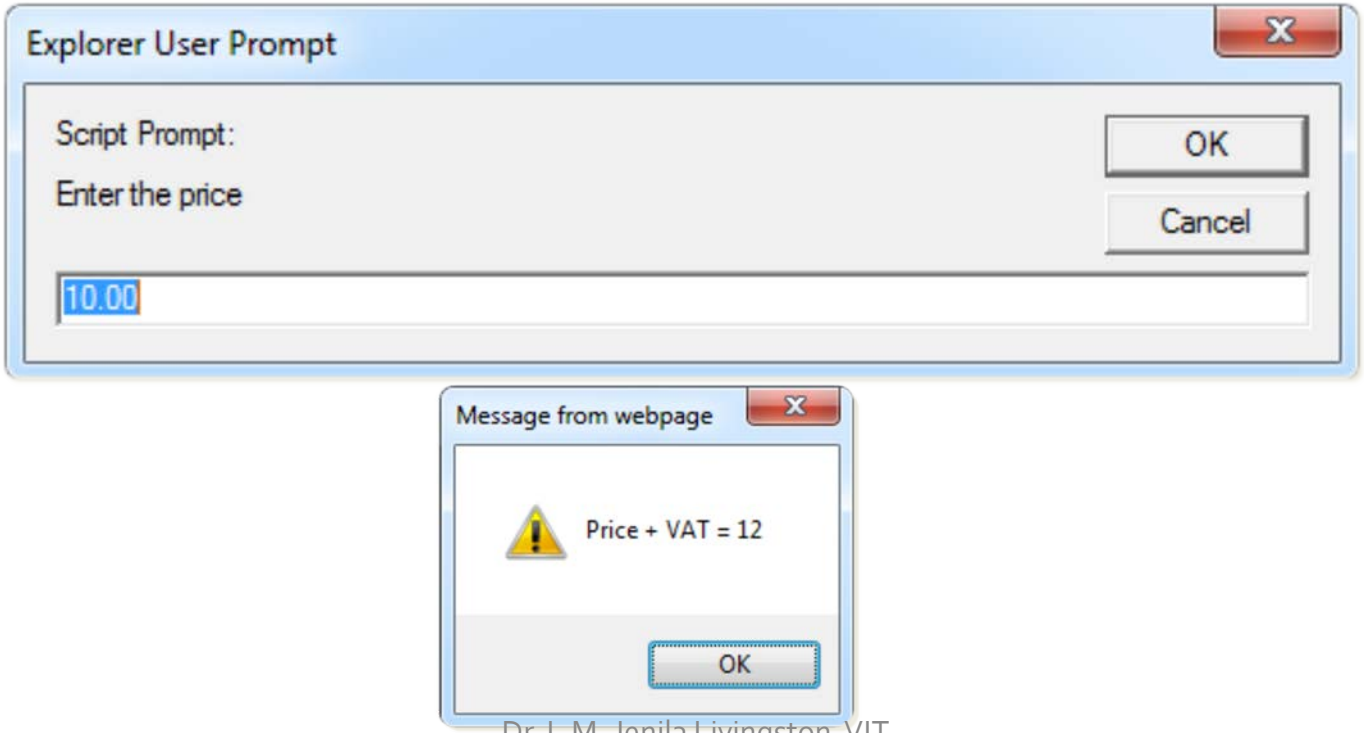
```
<head>
<script language="JavaScript">
        var id;
        id = prompt("What is your student id number?");
        alert(id);
        name = prompt("What is your name?","No name");
        alert(name);
</script>
</head>
```

Variable declaration

Input-Output using Prompt & Alert

prompt.html

```
price = prompt("Enter the price", "10.00");
alert('Price + VAT = ' + price * 1.2);
```



Input-Output using Text boxes - Sum of Numbers

sum-of-numbers.html

```
<html>

<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function calcSum() {
      value1 =
        parseInt(document.mainForm.textBox1.value);
      value2 =
        parseInt(document.mainForm.textBox2.value);
      sum = value1 + value2;
      document.mainForm.textBoxSum.value = sum;
    }
  </script>
</head>
```

Note:
parseInt: Convert string to integer
parseFloat: Convert string to float

Sum of Numbers – Contd..

sum-of-numbers.html (cont.)

```
<body>
  <form name="mainForm">
    <input type="text" name="textBox1" /> <br/>
    <input type="text" name="textBox2" /> <br/>
    <input type="button" value="Process"
      onclick="javascript:calcSum()" />
    <input type="text" name="textBoxSum"
      readonly="readonly"/>
  </form>
</body>

</html>
```

Input-Output using Text boxes - Sum of Numbers - getElementById

sum-of-numbers.html

```
<html>

<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function calcSum() {
     value1 = document.getElementById(“t1”);
     value2 = document.getElementById(“t2”);
      sum = value1 + value2;
      document.getElementById(“t3”).value = sum;
    }
  </script>
</head>
```

Note:
parseInt: Convert string to integer
parseFloat: Convert string to float

Sum of Numbers – Contd..

sum-of-numbers.html (cont.)

```
<body>
  <form name="mainForm">
    <input type="text" name="text1" id="t1" />
<br/>
    <input type="text" name="text2" id="t2" />
<br/>
    <input type="button" value="Process"
      onclick="calcSum()" />
    <input type="text" name="textBoxSum" id="t3"
      readonly="readonly"/>
  </form>
</body>

</html>
```

JavaScript OUTPUT

1. Writing into the HTML output
   using document.write()/ document.writeln.
2. Writing into an HTML element:
   - using innerHTML
   - using textContent
   - using innerText
3. Writing into text box
4. Writing into an alert box, using window.alert().

Document.write

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

The **writeln**() method is identical to the **document**. **write()** method, with the addition of writing a newline character after each statement.

innerHTML

- <html>
- <body>
  <h1>First Paragraph </h1>

- <h1>Second Paragraph </h1>

  <script>
  document.getElementById("demo").innerHTML = ”<h1>”+(5 + 6)+”</h1>”;
- document.getElementById("demo2").innerHTML = "New Bold Text";
  </script>
  </body></html>

JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element.

textContent

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").textContent = 5 + 6;
</script>
</body>
</html>
```

innerText

```
<html>
<body>
<h1 style="color: blue; text-align: center;">First Paragraph</h1>
<p id="demo" style="color: red; font-size: 20px; font-weight: bold;"></p>
<script>
document.getElementById("demo").innerText = 5 + 6;
</script>
</body>
</html>
```

innerHTML Vs textContent VS innerText

Method	Allows HTML Tags?	Respects CSS Styles?
`innerHTML`	✅ Yes	✅ Yes
`textContent`	❌ No	❌ No
`innerText`	❌ No	✅ Yes

✓ **Use innerHTML** when inserting **HTML formatting** inside elements.
✓ **Use textContent** when inserting **plain text only** (**safer** approach).
✓ **Use innerText** when you want to respect **CSS styling and visibility**.

Writing into text box

```
value1 = document.getElementById(“t1”);
value2 = document.getElementById(“t2”);
 sum = value1 + value2;
 document.getElementById(“t3”).value = sum;
  //document.mainForm.textBoxSum.value = sum;
```

inner.htmlOutput using innerHTML

```
<html> <head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function calc() {
    v=document.getElementById(“t1”);
    document.getElementById(‘display’).innerHTML=v;}
</head>
<body>
  <form name="mainForm">
    <input type="text" name=“text1” id=“t1” />
<input type="button" value="Process"
      onClick="calc()" />
</form>
  <p id='display'>
</body> </html>
```

write vs innerHTML

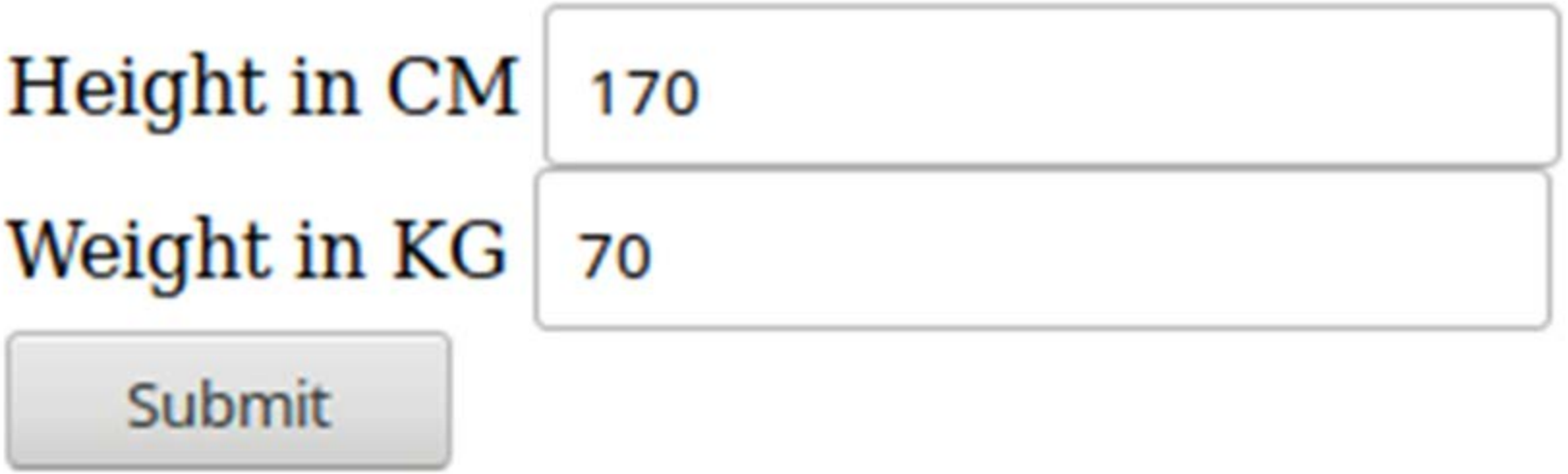
Feature	`document.write()`	`innerHTML`
Usage Timing	Best used during the initial loading of the page.	Best used after the page has fully loaded.
Effect on Document	Can overwrite the entire document if used after load.	Updates content only inside a specific element.
Flexibility	Less flexible, especially after the page has loaded.	More flexible, commonly used for dynamic content updates.
Content Insertion	Inserts content directly into the document.	Inserts/updates content inside a targeted DOM element.

Windows.alert

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

Exercise

BMI = KG / (H/100 * H/100)



Under Weight = Less than 18.6
Normal Range = 18.6 and 24.9
Overweight = Greater than 24.9

Thank You

JavaScript – Part2

Dr. Jenila Livingston L.M.
Professor
VIT Chennai

Module 4

- Hello World Web Page
- Variables, Identifiers
- Assignment Statements and Objects
- Functions
- Document Object Model
- Forms: form Element, Controls, Buttons (Module1)
- Text Control Accessing a Form's Control Values
- reset and focus Methods
- Event Handler Attributes: onchange, onmouseover, onmouseout.

Presentation Overview

JavaScript Elements

- Data Types
- Arrays
- Operators
- Functions

JavaScript Elements

- The JavaScript syntax is similar to C# and Java
  - Data Types
  - Operators (+, *, =, !=, &&, ++, …)
  - Conditional statements (`if`, `else`)
  - Looping Statements (`for`, `while`)
  - Arrays (`my_array[]`) and
    associative arrays (`my_array['abc']`)
  - Functions
  - getElemnetsBYName

Data Types

- JavaScript allows the same variable to contain different types of data values.
- Primitive data types
  - Number: integer & floating-point numbers
  - Boolean: logical values “true” or “false”
  - String: a sequence of alphanumeric characters
- Composite data types (or Complex data types)
  - Object: a named collection of data
  - Array: a sequence of values
- Special data types
  - Null: an initial value is assigned
  - Undefined: the variable has been created by not yet assigned a value

Data types

- JavaScript uses the let/var keyword to declare variables.
- An **equal sign** is used to assign values to variables.
- **Strings** are written inside **double or single quotes**. **Numbers** are written **without quotes**.
- If you put a number in quotes, it will be treated as a text string.

- JavaScript variables are Case Sensitive
- let lastname, lastName;
  lastName = "Doe";
  lastname = "Peterson";

Javascript is dynamic

- let x; // Now x is undefined
  x = 5; // Now x is a Number
  x = "John"; // Now x is a String

The typeof Operator

- JavaScript typeof operator to find the type of a JavaScript variable

```
typeof ""             // Returns "string"
typeof "John"         // Returns "string“
typeof 0              // Returns "number"
typeof 314            // Returns "number"
typeof 3.14           // Returns "number"
typeof (3)            // Returns "number"
typeof (3 + 4)        // Returns "number"
```

var, let, const

```
function demo() {
if (true) {
    var x = 10; // Function-scoped
    x = 15; // Updating function-scoped variable
    let y = 20; // Block-scoped
    const z = 30; // Block-scoped
    console.log(x); // ✅ 10 (accessible inside function)
    console.log(y); // ✅ 20 (accessible inside block
    console.log(z); // ✅ 30 (accessible inside block)
}
console.log(x); // ✅ 15 (accessible in function)
console.log(y); // ❌ Error (y is block-scoped)
console.log(z); // ❌ Error (z is block-scoped)
}
demo();
```

var, let, const

```
function demo() {
if (true) {
    var x = 10; // Function-scoped
    x = 10; // Updating function-scoped variable
    let y = 20; // Block-scoped
    const z = 30; // Block-scoped
    console.log(x); // ✅ 10 (accessible inside function)
    console.log(y); // ✅ 20 (accessible inside block
    console.log(z); // ✅ 30 (accessible inside block)
} }
demo();
console.log(x); // ❌ Error (not accessible outside the function)
console.log(y); // ❌ Error (y is block-scoped)
console.log(z); // ❌ Error (z is block-scoped)
```

Global variable

```
x = 10;
```

x=10

Without explicitly declaring x using let, const, or var, **JavaScript treats x as an implicitly global variable, even if it appears inside a function**.

var, let, const

Feature	`var`	`let`	`const`
Scope	Function	Block	Block
Redeclaration	✅ Yes	❌ No	❌ No
Reassignment	✅ Yes	✅ Yes	❌ No

```
var x = 10;
console.log(x); // ✅ 10

var x = 20; // ✅ Allowed
```

```
let y = 10;
console.log(y); // ✅ 10

let y = 20; // ❌ Error: Identifier 'y' has already been declared
```

Integer and Floating-point number example

```
<script language="JavaScript">
        var integerVar = 100;
        var floatingPointVar = 3.14;
        document.write(integerVar);
        document.write(floatingPointVar);
</script>
```

Boolean value example

```
<head>
<script language="JavaScript">
        var result;
        result = (true*10 + false*7);
        alert("true*10 + false*7 = ", result);
</script>
</head>
```

- The expression is converted to
  - (1*10 + 0*7) = 10
- They are automatically converted.

String

- String type – string of characters

- String can also be enclosed in single quotation marks (‘) or in double quotation marks (“).

```
var myName = "You can use both single or double
quotes for strings";
```

Strings example

```
<head>
<script language="JavaScript">
        document.write("This is a string.");
        document.write("This string contains 'quote'.");
        var myString = "My testing string";
        alert(myString);
</script>
</head>
```

- Unlike Java and C, JavaScript does not have a single character (char) data type.

String Operations

- The + operator joins strings

```
string1 = "fat ";
string2 = "cats";
alert(string1 + string2);  // fat cats
```

- What is "9" + 9?

```
alert("9" + 9);  // 99
```

- Converting string to number:

```
alert(parseInt("9") + 9);  // 18
```

Strings: Extracting part of a string

There are 3 methods for extracting a part of a string:

I. slice(start, end)
II. substring(start, end)
III. substr(start, length)

Strings - slice

- var str = "Apple, Banana, Kiwi";
  var res = str.slice(7, 13);
- o/p: Banana
- If a parameter is negative, the position is counted from the end of the string.
- var str = "Apple, Banana, Kiwi";
  var res = str.slice(-12, -6);
- o/p: Banana
- slice(2,-1) extracts the third element through the second-to-last element in the sequence.

- The **slice**() method does not include the given end argument.
- slice starts from the index o

Strings - substr

- var str = "Apple, Banana, Kiwi";
  var res = str.substr(7, 6);
- o/p: Banana
- substr() is similar to slice().
- The difference is that the second parameter specifies the length of the extracted part.

Strings - substring

- substring() is similar to slice().
- The difference is that substring() cannot accept negative indexes.

Strings: Replace

- The replace() method replaces a specified value with another value in a string:
- str = "Please visit Microsoft!";
  var n = str.replace("Microsoft", “VIT");

Strings: charAt

- The charAt() method returns the character at a specified index (position) in a string:
- var str = "HELLO WORLD";
  str.charAt(0); // returns H

Strings – UpperCase and LowerCase

- A string is converted to upper case with toUpperCase():
- var text1 = "Hello World!"; // String
  var text2 = text1.toUpperCase(); // text2 is text1 converted to upper
- A string is converted to lower case with toLowerCase():
- var text1 = "Hello World!"; // String
  var text2 = text1.toLowerCase(); // text2 is text1 converted to lower

String: indexOf() and lastIndexOf()

- Search a string
- var str = "Hello world, welcome to the universe.";
  var n = str.indexOf("welcome");
- The result of *n* will be: 13

- Search a string for the last occurrence of a string
- var str = "Hello planet earth, you are a great planet.";
  var n = str.lastIndexOf("planet");
- The result of *n* will be: 36

Both methods return -1 if the value to search for never occurs.

String: Spilt

- **str.split()** function is used to split the given string into array of strings by separating it into substrings using a specified separator provided in the argument.
- **str.split(separator[, limit])**
- var str = 'It is a great Day.'
- var array1 = str.split(" ");
- var array2 = str.split(" ",2);
- Document.write(array1);
- Document.write(array1);

Everything is Object

- Every variable can be considered as object
  - For example strings and arrays have member functions:

objects.html

```
var test = "some string";
alert(test[7]); // shows letter 'r'
alert(test.charAt(5)); // shows letter 's'
alert("test".charAt(1)); //shows letter 'e'
alert("test".substring(1,3)); //shows 'es'
```

```
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7); // appends 7 to end of array
alert (arr[3]); // shows 7
```

Arrays

- An **array** in JavaScript is a data structure that stores multiple values in a single variable. Arrays can hold different data types and are zero-indexed.

- Creating an Array

```
// Using array literal
let my_array = [1, 5.3, "aaa"];
let arr = [1, 2, 3, 4, 5];

// Using Array constructor
let numbers = new Array(10, 20, 30);

// Empty array let empty
var arr = new Array();
var arr = new Array(3);
Array = [];
```

Arrays Operations and Properties

- Modifying Array element

```
fruits[1] = "Orange";
```

- Appending an element

```
arr.push(3); //Adds to an end
fruits.unshift("Pineapple"); // Adds to the beginning
```

- Removing an element:

```
var element = arr.pop(); //Removes from the end
fruits.shift(); // Removes from the beginning
```

- Reading the number of elements (array length):

```
arr.length;
```

- Finding element's index in the array:

```
arr.indexOf(1);
```

Accessing An Array

```
<script language="JavaScript">
        var Car = new Array(3);
        Car[0] = "Ford";
        Car[1] = "Toyota";
        Car[2] = "Honda";
        document.write(Car[0] + "<br>");
        document.write(Car[1] + "<br>");
        document.write(Car[2] + "<br>");
</script>
```

- You can also declare arrays with variable length.
  - arrayName = new Array();
  - Length = 0, allows automatic extension of the length.

Other Operations

- **Finding Elements**

```
fruits.indexOf("Orange"); // 1
fruits.includes("Mango"); // true
```

- **Slicing and Splicing**

```
let sliced = fruits.slice(1, 3); // Extracts elements from index 1 to 2
fruits.splice(1, 1, "Strawberry"); // Removes 1 element at index 1 and adds "Strawberry“
```

- **Joining and Splitting**

```
let joined = fruits.join(" - ");
let splitArray = joined.split(" - ");
```

- **Sort and Reverse**

```
nums.sort();
nums.reverse();
```

Looping through Array

```
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}

fruits.forEach(fruit => console.log(fruit));

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

Arrays

- **Associative arrays (hash tables)**

  Key:value pair

```
let harr = {a:2, b:3, c:"text"};
harr["c"] //text
```

```
let person = {
    name: "John Doe",
    age: 30,
    city: "New York"
};
```

- **Multi-Dimensional Array**

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
```

Get input from text field

```
<input type="text" id="t1" placeholder="Enter value">
<button onclick="addToArray()">Add</button>
<script>
let myArray = []; // Initialize empty array
function addToArray() {
let arr = document.getElementById("t1").value;
if (arr.trim() !== "") { // Ensure it's not empty
        myArray.push(arr); // Add to array
        document.getElementById("t1").value = ""; // Clear input field
else {
        alert("Please enter a value!");
} } </script>
```

Display array element

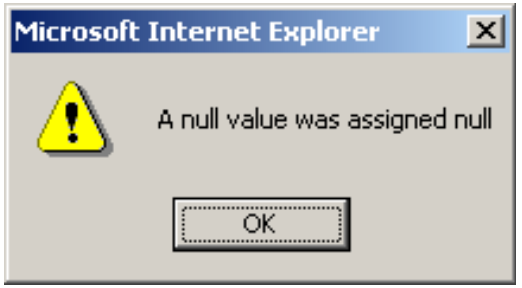
```
<h3>Array Elements:</h3>
<div id="arrayDisplay"></div>

function displayArray() {
let disp = document.getElementById("arrayDisplay");
disp.innerHTML = ""; // Clear previous content
let result = "<ul>"; // Start unordered list
for (i = 0; i < myArray .length; i++) {
        result += "<li>" + myArray [i] + "</li>"; }
result += "</ul>"; // Close unordered list
displ.innerHTML = result; // Set content inside div
} displayArray()
```

Null & Undefined

- An “undefined” value is returned when you attempt to use a variable that has not been defined or you have declared but you forgot to provide with a value.
- Null refers to “nothing”
- You can declare and define a variable as “null” if you want absolutely nothing in it, but you just don’t want it to be “undefined”.

Null & Undefined example

```
<html>
<head>
<title> Null and Undefined example </title>
<script language="JavaScript">
            var test1, test2 = null;
            alert("No value assigned to the variable" + test1);
            alert("A null value was assigned" + test2);
</script>
</head>
<body> … </body>
</html>
```



JavaScript Special Characters

Character	Meaning
\b	Backspace
\f	Form feed
\t	Horizontal tab
\n	New line
\r	Carriage return
\\	Backslash
\’	Single quote
\”	Double quote

Operators

- Arithmetic operators
- Logical operators
- Comparison operators
- String operators
- Bit-wise operators
- Assignment operators
- Conditional operators

Arithmetic operators

- left_operand “operator” right_operand

Operator	Name	Description	Example
+	Addition	Adds the operands	3 + 5
-	Subtraction	Subtracts the right operand from the left operand	5 - 3
*	Multiplication	Multiplies the operands	3 * 5
/	Division	Divides the left operand by the right operand	30 / 5
%	Modulus	Calculates the remainder	20 % 5

Unary and Binary Operators

- Binary operators take two operands.
- Unary type operators take only one operand.

Name	Example
Post Incrementing operator	Counter++
Post Decrementing operator	Counter--
Pre Incrementing operator	++counter
Pre Decrementing operator	--counter

Logical operators

- Used to perform Boolean operations on Boolean operands

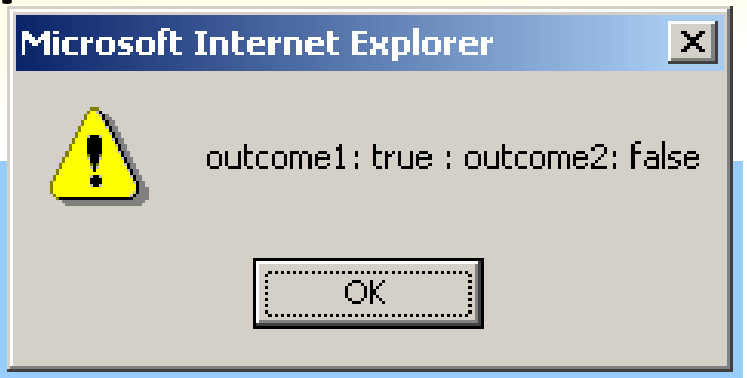
Operator	Name	Description	Example
&&	Logical and	Evaluate to “true” when both operands are true	3>2 && 5<2
||	Logical or	Evaluate to “true when either operand is true	3>1 || 2>5
!	Logical not	Evaluate to “true” when the operand is false	5 != 3

Comparison operators

- Used to compare two numerical values

Operator	Name	Description	Example
==	Equal	Perform type conversion before checking the equality	“5” == 5
===	Strictly equal	No type conversion before testing	“5” === 5
!=	Not equal	“true” when both operands are not equal	4 != 2
!==	Strictly not equal	No type conversion before testing nonequality	5 !== “5”
>	Greater than	“true” if left operand is greater than right operand	2 > 5
<	Less than	“true” if left operand is less than right operand	3 < 5
>=	Greater than or equal	“true” if left operand is greater than or equal to the right operand	5 >= 2
<=	Less than or equal	“true” if left operand is less than or equal to the right operand	5 <= 2

Strict Equality Operators




```
<script language="JavaScript">
        var currentWord="75";
        var currentValue=75;
        var outcome1=(currentWord == currentValue);
        var outcome2=(currentWord === currentValue);
        alert("outcome1: " + outcome1 + " : outcome2: " + outcome2);
</script>
```

String operator

- JavaScript only supports one string operator for joining two strings.

Operator	Name	Description	Return value
+	String concatenation	Joins two strings	"HelloWorld"

```
<script language="JavaScript">
        var myString = "";
        myString = "Hello" + "World";
        alert(myString);
</script>
```

Bit Manipulation operators

- Perform operations on the bit representation of a value, such as shift left or right.

Operator	Name	Description
&	Bitwise AND	Examines each bit position
|	Bitwise OR	If either bit of the operands is 1, the result will be 1
^	Bitwise XOR	Set the result bit, only if either bit is 1, but not both
<<	Bitwise left shift	Shifts the bits of an expression to the left
>>	Bitwise signed right shift	Shifts the bits to the right, and maintains the sign
>>>	Bitwise zero-fill right shift	Shifts the bits of an expression to right

Assignment operators

- Used to assign values to variables

Operator	Description	Example
=	Assigns the value of the right operand to the left operand	A = 2
+=	Add the operands and assigns the result to the left operand	A += 5
-=	Subtracts the operands and assigns the result to the left operand	A -= 5
*=	Multiplies the operands and assigns the result to the left operand	A *= 5
/=	Divides the left operands by the right operand and assigns the result to the left operand	A /= 5
%=	Assigns the remainder to the left operand	A %= 2

Functions

- Code structure – splitting code into parts
- Data comes in, processed, result returned

```
function name(parameter1, parameter2) // Formal parameters
{
   //code to be executed
  // Optional return statement
}
//function call

name(arg1, arg2); // Actual parameters
```

Functions - Example

```
function average(a, b, c)
{
    let total;
    total = a+b+c;
    return total/3;
}
alert(average(5,6,7));
```

Parameters come in here.

Declaring variables is optional.

Value returned here.

```
function checkEligibility(age) {
    return age >= 18 ? "Eligible to vote" : "Not eligible to vote";
}

alert(checkEligibility(20)); // Output: Alert box with "Eligible to vote"
```

Function Expression

```
const doubleAge = function(age) {
return age * 2;
};
alert(doubleAge(15)); // Output: Alert box with "30“
```

Arrow function syntax (**Shorter and** Modern form):

```
const doubleAge = (age) => age * 2;
alert(doubleAge(15));
```

Function Object

```
function disp(){}
alert(typeof disp);
```

Example

```
<html1
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(77) + " Celsius";

function toCelsius(fahrenheit)
{
   return (5/9) * (fahrenheit-32);
}
</script>
</body>
</html>
```

Output:
The temperature is 25 Celsius

Function Arguments

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
  - The function has access to all the arguments passed via **`arguments` array**

```
function sum() {
  var sum = 0;
  for (var i = 0; i < arguments.length; i ++)
    sum += parseInt(arguments[i]);
  return sum;
}
alert(sum(1, 2, 4));
//returns 7
```

functions-demo.html

Function – Return and store a result

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

// Function call
let result = add(2, 3, 5);
alert(result); // Output: 10
```

Thank You

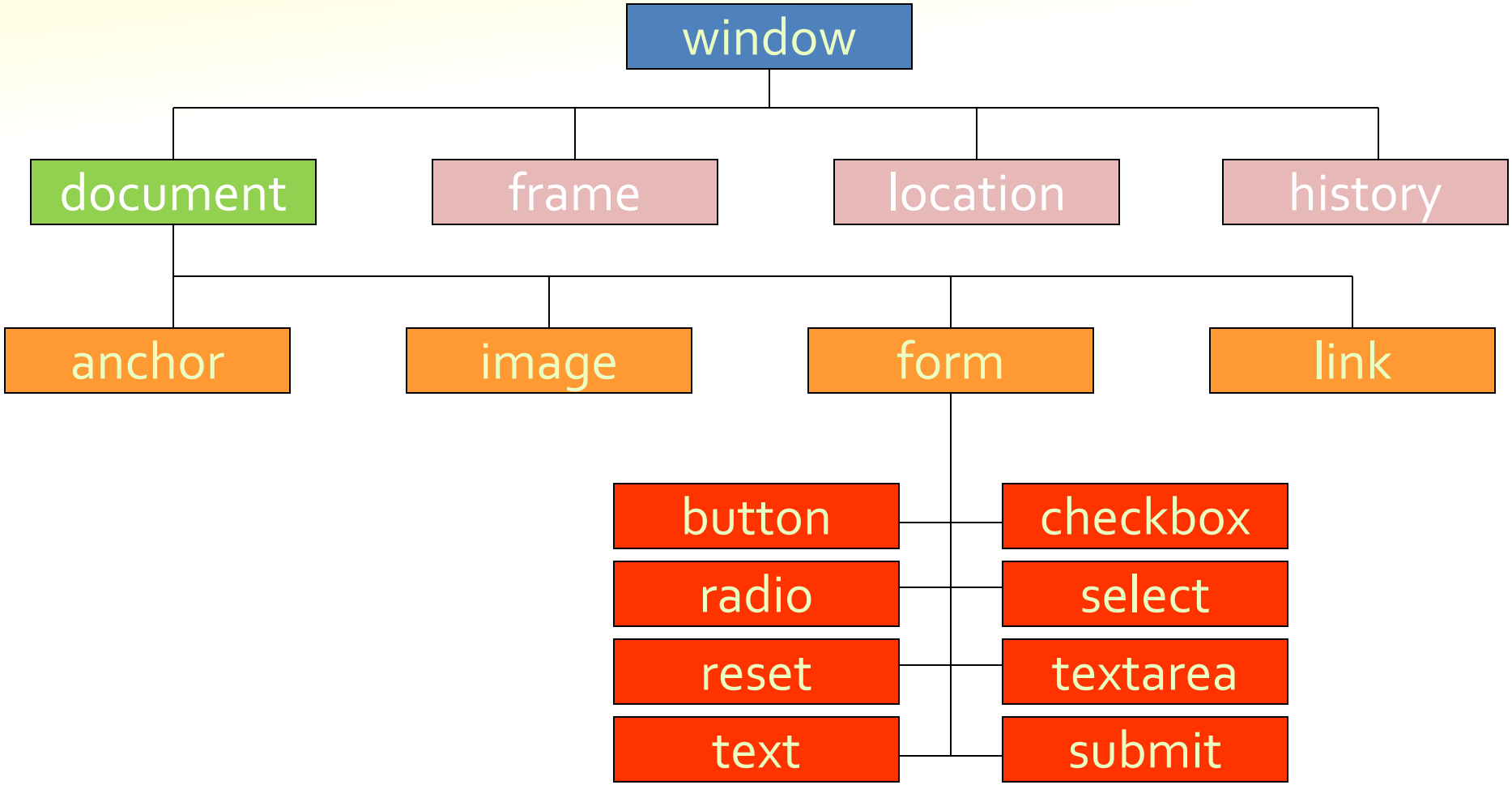
DOM & Selection Methods

Dr. Jenila Livingston L.M.
Professor
VIT Chennai

DOM

- JavaScript also provides you with **objects that can control and manipulate the displays** of browsers.
  - More dynamic and interactive.
- DOM is an object-oriented model that describes **how all elements** in an HTML page **are arranged**.
- The HTML DOM is a standard for how to **get, change, add, or delete** HTML elements.

DOM


window
document
frame
location
history
anchor
image
form
link
button
radio
reset
text
checkbox
select
textarea
submit

How the DOM works?

```
<head><script>
function toggle()
{ document.img button1.src="button_on.gif"; }
</script></head>
<body>
<a href="test.html" onmouseover="toggle()"> <img name="button1"
src="button_off.gif"></a>
</body>
```

reaction

action

Action → Event → JavaScript → DOM → Reaction

src="button_off.gif" | onmouseover | toggle() | document.img.button1 | src="button_on.gif"

1) User moves mouse over object
2) Event senses that something happened to the object
3) JavaScript tells the object what to do (Even handler)
4) Locates object on the web page
5) Object's image source is changed

HTML DOM

- HTML DOM **methods (fuction)** are **actions** you can perform (on HTML Elements).
- HTML DOM **properties** are **values** (of HTML Elements) that you can set or change.

DOM Selection methods

Method	Description
getElementById(id)	Find an element-by-element id
getElemnetsBYName(name)	Find elements by Name
getElementsByTagName(tag)	Find elements by tag name
getElementsByClassName(class_name)	Find elements by class name
querySelector(selector)	Find element by CSS selector
querySelectorAll(selector)	Find elements by CSS selector Alternate for getElementsByName

DOM Selection methods

Method	Selects By	Returns	Use Case
`getElementById(id)`	`id`	Single element	Fast, unique elements with `id`.
`getElementsByName(name)`	`name`	Live NodeList	Grouped form elements with the same `name`.
`getElementsByTagName(tag)`	Tag name	Live NodeList	Multiple elements with the same tag (e.g., `<div>`).
`getElementsByClassName(class)`	Class name	Live NodeList	Multiple elements with the same class.
`querySelector(selector)`	CSS selector	Single element	First matching element using a CSS selector.
`querySelectorAll(selector)`	CSS selector	Static NodeList	All matching elements using a CSS selector.

Note: NodeList: Array like Collection. First element index is o

A **Static NodeList** does **not** update when the document changes.

Which One Should You Use?

Live NodeLists or Static NodeLists

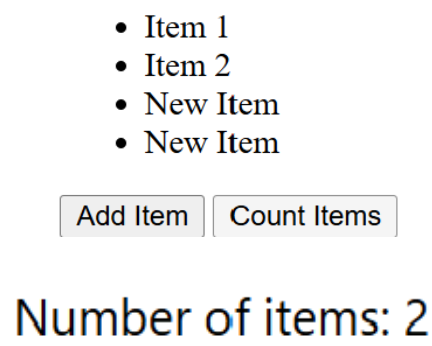
- **Use Live NodeLists** (getElementsByName(), etc.) when you want to track changes dynamically.

- **Use Static NodeLists** (querySelectorAll()) when you don't want automatic updates (better performance in large documents).

Static NodeList

```
<body>

    <ul id="myList">
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>

    <button onclick="addNewItem()">Add Item</button>
    <button onclick="countItems()">Count Items</button>

    <script>
        // Creates a static NodeList
        let items = document.querySelectorAll("#myList li");

        function addNewItem() {
            let newItem = document.createElement("li");
            newItem.textContent = "New Item";
            document.getElementById("myList").appendChild(newItem);
        }

        function countItems() {
          // This will NOT increase when new items are added
            alert("Number of items: " + items.length);
        }
    </script>

</body>
```

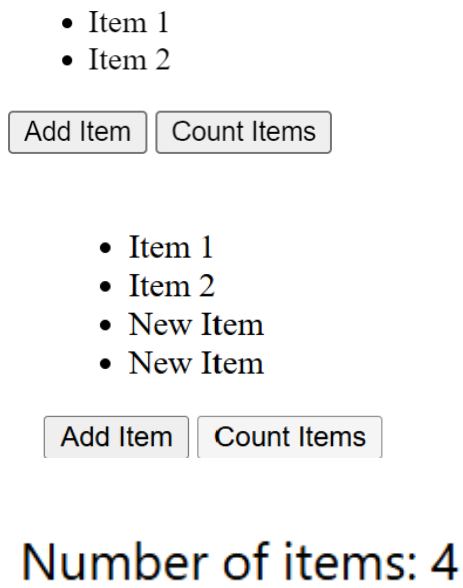
- **querySelectorAll("#myList li") creates a static NodeList** when the page loads.
- When clicking "Add Item", a new <li> element is added dynamically.
- Clicking "Count Items" will always show **2**, even after adding new items.
- The NodeList does not update because it is **static**.



Dynamic NodeLIst

```
<body>
    <ul id="myList">
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>

    <button onclick="addNewItem()">Add Item</button>
    <button onclick="countItems()">Count Items</button>

    <script>
        // Creates a static NodeList
       // let items = document.querySelectorAll("#myList li");
        let items = document.getElementsByTagName("li");

        function addNewItem() {
            let newItem = document.createElement("li");
            newItem.textContent = "New Item";
            document.getElementById("myList").appendChild(newItem);
        }

        function countItems() {
          // This will increase when new items are added
            alert("Number of items: " + items.length);
        }
    </script>
</body>
```

updates the count when new items are added



Accessing Form Fields

```
<form name="F1">
  Name: <input type="text" id="name" name="username"><br>
  PAN: <input type="text" id="pan" name="pan"><br>
  Age: <input type="number" id="age" name="age"><br>
  <button type="button" onclick="getValues()">Submit</button>
</form>

<script>
  function getValues() {
    var form = document.forms["F1"];
    var name = form["username"].value;
    var pan = form["pan"].value;
    var age = form["age"].value;

    alert("Name: " + name + "\nPAN: " + pan + "\nAge: " + age);
  }
</script>
```

Accessing Input Field Values (Text, Number, Password, etc.)

- **document.forms[“formname”][“fieldname”].value**
- **document.formname.filedname.value**
- **getElementById**

```
<body><form name="F1">
  Name: <input type="text" id="name" name="username"><br>
  PAN: <input type="text" id="pan" name="pan"><br>
  Age: <input type="number" id="age" name="age"><br>
  <button type="button" onclick="getValues()">Submit</button>
</form>

<script>
  function getValues() {
    var name = document.forms["F1"]["username"].value;
    var pan = document.F1.pan.value;
    var age = document.getElementById("age").value;
    alert("Name: " + name + "\nPAN: " + pan + "\nAge: " + age);
  }
</script></body>
```

Accessing Dropdown (Select) Values

getElementById

getElementById for fetching form values instead of document.forms. This ensures clarity and consistency when accessing input fields.

```
<form name="myForm">
  Country:
  <select id="country">
    <option value="India">India</option>
    <option value="USA">USA</option>
    <option value="UK">UK</option>
  </select>
  <button type="button" onclick="getDropdownValue()">Get Country</button>
</form>

<script>
  function getDropdownValue() {
    var country = document.getElementById("country").value;
    alert("Selected Country: " + country);
  }
</script>
```

Accessing Textarea Value

getElementById

```
<form name="myForm">
  Comments: <br>
  <textarea id="comment" rows="4" cols="30"></textarea>
  <button type="button" onclick="getValue()">Get Comment</button>
</form>

<script>
  function getValue() {
    var comment = document.getElementById("comment").value;
    alert("Your Comment: " + comment);
  }
</script>
```

getElemnetsByName

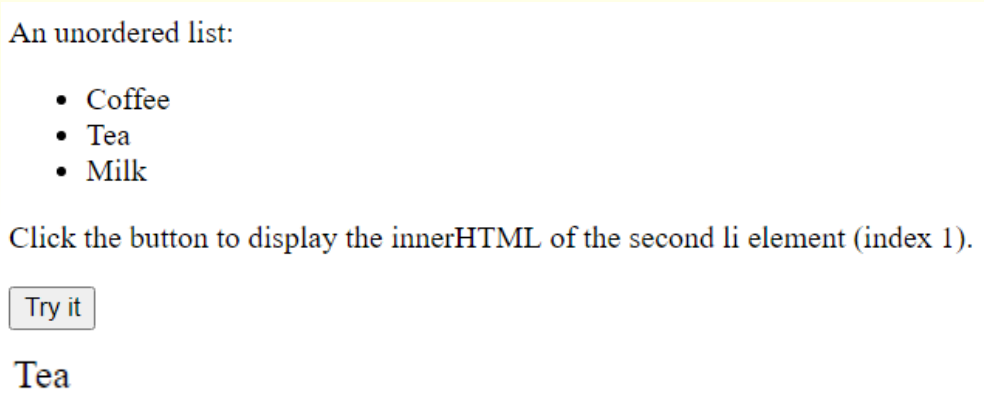
- The getElementsByName() method returns a collection of all elements in the document with the specified name (the **value** of the name attribute), as an HTMLCollection object.
- The HTMLCollection object represents a collection of nodes. The nodes can be accessed by index numbers. **The index starts at 0**.
- **Tip:** You can use the **length property** to determine the number of elements, then **loop through all elements** and fetch the info.

Accessing radio button and combo box

```
<html>  <head>  <title> GetElementsByName </title> </head>
 <body> <p> Select a radio button and an option from Combo box. </p>
    Gender:
   <input type="radio" name="gender" value="Male">Male
   <input type="radio" name="gender" value="Female">Female
   <input type="radio" name="gender" value="Others">Others
   <br> Select Your Degree<Br>
   <select>
          <option name="degree" value="CSE">CSE
          <option name="degree" value="ECE">ECE
          <option name="degree" value="EEE">EEE
          <option name="degree" value="Mech">Mech
          <option name="degree" value="Civil">Civil
          </select><Br>
   <button type="button" onclick="displayValue()"> Submit </button> <br>
   <div id="result"></div>
   <div id="result2"></div>
```

```
<script>
    function displayValue() {
      var ele1 = document.getElementsByName('gender'); //creates an array
      var ele2= document.getElementsByName('degree');
      for(i = 0; i < ele1.length; i++) {
        if(ele1[i].checked)
        document.getElementById("result").innerHTML
              = "Gender: "+ele1[i].value;
      }
        for(i = 0; i < ele2.length; i++) {
        if(ele2[i].selected)
        document.getElementById("result2").innerHTML
              = "Degree: "+ele2[i].value;
      }
    }
  </script> </body>
</html>
```

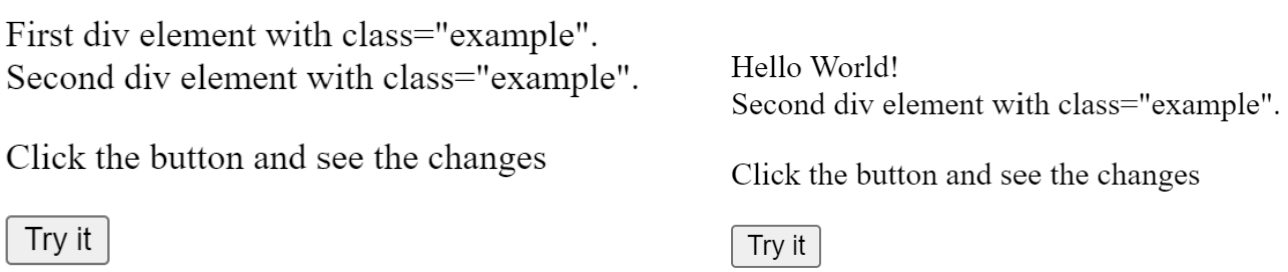
getElementsByTagName




```
<!DOCTYPE html>
<html>
<body>
<p>An unordered list:</p>
<ul>
  <LI>Coffee</LI>
  <LI>Tea</LI>
  <LI>Milk</LI>
</ul>
<p>Click the button to display the innerHTML of the second li element (index 1).</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var x = document.getElementsByTagName("LI");
  document.getElementById("demo").innerHTML = x[1].innerHTML;
}
</script>
</body>
</html>
```

document.getElementsByClassName

```
<!DOCTYPE html>
<html>
<body>
<div class="example">First div element with class="example".</div>
<div class="example">Second div element with class="example".</div>
<p>Click the button and see the changes</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  var x = document.getElementsByClassName("example");
  x[0].innerHTML = "Hello World!";
}
</script>
</body>
</html>
```



querySelector and querySelectorAll

Syntax:

```
var element = document.querySelector("selector");
var elements = document.querySelectorAll("selector");
```

Examples:

```
var nameInput = document.querySelector("#name"); // Selects the element with ID 'name'
var firstItem = document.querySelector(".item"); // Selects the first element with class
var firstInput = document.querySelector("input"); // Selects the first <input> element
var nameInput = document.querySelector("form[name='F1'] input[name='username']");
var nameInput = document.querySelector("input[name='username']");
```

Tag

// it selects the **first** <input> element with the attribute name="username".

```
var inputs = document.querySelectorAll("input"); // Selects all input fields
var ch = document.querySelectorAll('input:checked');
var ch = document.querySelectorAll('input[name="skills"]:checked');
```

querySelector

```
<form name="F1">
  Name: <input type="text" id="username"><br>
  Age: <input type="number" id="age"><br>
  <button type="button" onclick="getValues()">Submit</button>
</form>

<script>
  function getValues() {
    var nameInput = document.querySelector("#username");
    var ageInput = document.querySelector("#age");

    alert("Name: " + nameInput.value + "\nAge: " + ageInput.value);
  }
</script>
```

querySelector

```
<form name="F1">
  Name: <input type="text" name="username"><br>
  Age: <input type="number" name="age"><br>
  <button type="button" onclick="getValues()">Submit</button>
</form>

<script>
  function getValues() {
    var nameInput = document.querySelector("input[name='username']");
    var ageInput = document.querySelector("input[name='age']");

    alert("Name: " + nameInput.value + "\nAge: " + ageInput.value);
  }
</script>
```

Accessing Radio Button Values

querySelector

```
<form name="myForm">
    Gender:
    <input type="radio" name="gender" value="Male"> Male
    <input type="radio" name="gender" value="Female"> Female
    <input type="radio" name="gender" value="Other"> Other
    <button type="button" onclick="getRadioValue()">Check Gender</button>
</form>

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

<script>
    function getRadioValue() {
        var sel = document.querySelector('input[name="gender"]:checked');

        var opt = sel ? sel.value : "No selection";

        document.getElementById("output").innerHTML = `<b>My Gender:</b> ${opt}`;
    }
</script>
```

Gender: ○ Male ◉ Female ○ Other [Check Gender]

My Gender: Female

```
console.log(`My Gender: ${opt}`);
```

or

```
console.log("My Gender: " + opt);
```

Accessing Checkbox Values

querySelectorAll

```
<form name="myForm">
  Skills:
  <input type="checkbox" name="skills" value="HTML"> HTML
  <input type="checkbox" name="skills" value="CSS"> CSS
  <input type="checkbox" name="skills" value="JavaScript"> JavaScript
  <button type="button" onclick="getCheckboxValues()">Get Skills</button>
</form>

<script>
  function getCheckboxValues() {
    // var ch = document.querySelectorAll('input:checked');
    var ch = document.querySelectorAll('input[name="skills"]:checked');

    var arr = [];
    for (var i = 0; i < ch.length; i++) {
      arr.push(ch[i].value);
    }
    alert("Selected Skills: " + arr.join(", "));
  }
</script>
```

Accessing Checkbox Values
querySelectorAll

```
<form name="myForm">
  Skills:
  <input type="checkbox" name="skills" value="HTML"> HTML
  <input type="checkbox" name="skills" value="CSS"> CSS
  <input type="checkbox" name="skills" value="JavaScript"> JavaScript
  <button type="button" onclick="getCheckboxValues()">Get Skills</button>
</form>

<script>
  function getCheckboxValues() {
    var checkboxes = document.querySelectorAll('input[name="skills"]:checked');
    var values = Array.from(checkboxes).map(cb => cb.value);
    alert("Selected Skills: " + values.join(", "));
  }
</script>
```

Thank You

DOM – Dynamic Elements, Timerout Function, Form Validation & Regular Expression

Dr. Jenila Livingston L.M.
Professor
VIT Chennai

Overview

- **Dynamic Elements**
  - **Dynamic Table**
  - **Dynamic Nodes**
- **Timeout Function**
- **Form Validation & Regular Expression**

Dynamic Table

- Creating a **dynamic table** in JavaScript involves generating table rows and columns dynamically using **DOM manipulation**.

Table Object: Dynamic Table

- The **insertRow()** method creates an empty <tr> element and adds it to a table. It inserts new row(s) at the specified index in the table.
- **Note:** A <tr> element must contain one or more <th> or <td> elements.

- Use the **deleteRow()** method to remove a row.
- table.deleteRow(0); // Deletes the first row

Dynamic Table

Creates a table and inserting a row

```
let table = document.createElement("table");
table.border = "1"; // Set table border

let row = table.insertRow();
let cell1 = row.insertCell(0);
let cell2 = row.insertCell(1);

cell1.textContent = "Name";
cell2.textContent = "Age";

document.body.appendChild(table);
```

Deleting a row

```
// Check if the table has at least one row
if (table.rows.length > 0) {
    // Hides the row (alternative to deletion)
    table.rows[0].setAttribute("hidden", "true");
    table.deleteRow(0); // Deletes the first row
}
```

Dynamic Table

```
<!DOCTYPE html>
<html>
<head>Dynamic Table
</head>
<body>
<table id="myTable" border=1>
  <tr>
    <th>Heading1</th>
    <th>Heading2</th>
  </tr>
 </table>
<br>
<button onclick="myCreateFunction()">Create row</button>
<button onclick="myDeleteFunction()">Delete row</button>
```

Dynamic Table

```
<script>
var n=1
var cellCounter=1
function myCreateFunction() {
  var tab = document.getElementById("myTable");
  var row = tab.insertRow(n);
  var cell1 = row.insertCell(0);
  var cell2 = row.insertCell(1);
  cell1.innerHTML = `NEW CELL${cellCounter}`;
  cell2.innerHTML = `NEW CELL${cellCounter + 1}`;
  cellCounter += 2;
  n=n+1;
}
function myDeleteFunction() {
  m=prompt("Insert which row you want to delete, starting from 1");
  document.getElementById("myTable").deleteRow(m);
  n=n-1;
}
</script></body></html>
```

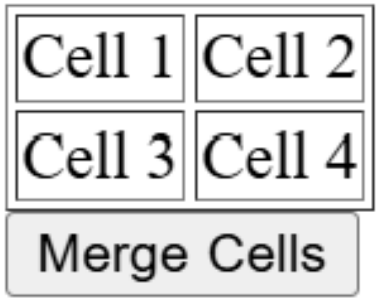
Heading1	Heading2
NEW CELL1	NEW CELL2
NEW CELL3	NEW CELL4
NEW CELL5	NEW CELL6

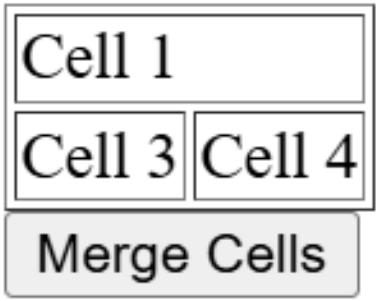
Create Row | Delete Row

Backticks ` **allow embedding expressions** inside a string using **${}**

Merging of cells

```
<table id="myTable" border="1">
    <tr>
        <td>Cell 1</td>
        <td>Cell 2</td>
    </tr>
    <tr>
        <td>Cell 3</td>
        <td>Cell 4</td>
    </tr>
</table>

<button onclick="mergeCells()">Merge Cells</button>

<script>
    function mergeCells() {
        let table = document.getElementById("myTable");
        let row = table.rows[0];

        // Set attribute for colspan
        row.cells[0].setAttribute("colspan", "2");

        // Remove the second cell
        row.deleteCell(1);
    }
</script>
```





DOM-Dynamic nodes

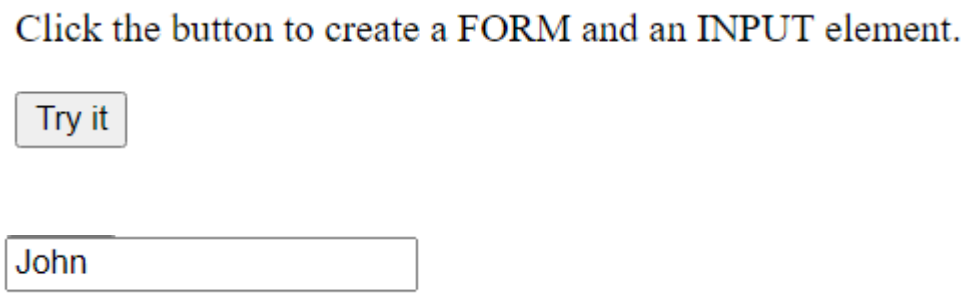
- **`createElement():`** creates an element node.
  `let var = document.createElement("tagName");`

- **`appendChild():`** Append a new child node to an existing parent node
  `parentNode.appendChild(childNode);`

- **`setAttribute("type", “value"):`** to set the attribute
  type - attribute name

Dynamic Form and Form Elements

- You can create a <form> element by using the **document.createElement()** method:



Dynamic Form and Form Elements

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to create a FORM and an INPUT element.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  let x = document.createElement("FORM");
  x.setAttribute("id", "myForm");
  document.body.appendChild(x);

  let y = document.createElement("INPUT");
  y.setAttribute("type", "text");
  y.setAttribute("value", “John");
  document.getElementById("myForm").appendChild(y);
}
</script></body></html>
```

Dynamic Form – appendChild, cloneNode

```
<head><script>
function myFunction() {
 var x = document.createElement("FORM");
 x.setAttribute("name", "myForm");
 document.body.appendChild(x);

 var y = document.createElement("INPUT");
 y.setAttribute("type", "text");
 y.setAttribute("style", "color:white;background-color:blue;");
 y.setAttribute("onfocus", "changecolor(this)");
 y.setAttribute("onblur", "changecol(this)");

 var z = document.createElement("INPUT");
 z.setAttribute("type", "button");
 z.setAttribute("value", "Click Me");
 z.setAttribute("style", "color:white;background-color:green;");
 z.setAttribute("onmouseover", "changebcolor(this)");
 z.setAttribute("onmouseout", "changebcol(this)");

 var br = document.createElement("br");

 myForm.appendChild(y);
 myForm.appendChild(br);
 myForm.appendChild(br.cloneNode());
 myForm.appendChild(br.cloneNode());
 myForm.appendChild(z);
}
```

```
function changecolor(y) {
y.style.background="pink";
y.value="";
}
function changecol(y) {
y.style.background="blue";
y.value="Hi! Welcome";
}

function changebcolor(y) {
y.style.background="blue";
}

function changebcol(y) {
y.style.background="green";
}
</script>
</head>
<body onload="myFunction()">
</body>
```

Dynamic Image

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to create a FIGURE element containing an IMG element.</p>
<button onclick="myFunction()">Try it</button><br><br>
<script>
function myFunction() {
  var x = document.createElement("FIGURE");
  x.setAttribute("id", "myFigure");
  document.body.appendChild(x);

  var y = document.createElement("IMG");
  y.setAttribute("src", "img_pulpit.jpg");
  y.setAttribute("width", "304");
  y.setAttribute("height", "228");
  y.setAttribute("alt", "The Pulpit Rock");
  x.appendChild(y);
}
</script></body></html>
```

Dynamic <HR>

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to create a HR element.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  var x = document.createElement("HR");
  document.body.appendChild(x);
}
</script>
</body>
</html>
```

DOM <H1>

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

<p>Click the button to create a H1 element with some text.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var x = document.createElement("H1");
  var t = document.createTextNode("Welcome to My Homepage");
  x.appendChild(t);
  document.body.appendChild(x);
}
</script>

</body>
</html>
```

SetTimeout() and clearTimeout():

- SetTimeout(): Takes two parameters
  - Function name
  - Time (how much time)
- clearTimeout(): clears the Timeout

Example 1 (setTimeout for 3000 milliseconds = 3 seconds)

```
function fun(){ alert("Hello"); }

function myFunction() {
  var myVar = setTimeout(fun, 3000);
}

function myStopFunction() {
  clearTimeout(myVar);
}
```

Example 2

```
<html>
<head>
<script>
function startTime() {
  var d = new Date();
  var h = d.getHours();
  var m = d.getMinutes();
  var s = d.getSeconds();
  document.getElementById('txt').innerHTML =
  h + ":" + m + ":" + s;
  var t = setTimeout(startTime, 1000);
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>
</html>
```

Example 3

```
function color() {
        if (document.getElementById('btn').style.color == 'blue')
         {
           document.getElementById('btn').style.color = 'green';
           } else {
           document.getElementById('btn').style.color = 'blue';
          }
       }
      function funstart() {
       t = setTimeout(color, 3000); }
      function stop() {
       clearTimeout(t);}
```

Validation

- Basic Validation – First of all, the form must be checked to make sure all the **mandatory fields are filled in**. It would require just a loop through each field in the form and check for data.
- Data Format Validation – Secondly, the data that is entered must be **checked for correct form and value**. Your code must include appropriate logic to test correctness of data.

Basic Validation – not empty

```
function validate()
{
if( document.myForm.Name.value == " " )
{
alert( "Please provide your name!" );
document.myForm.Name.focus() ;
return false;
}
```

Basic Validation – not empty

```
if( document.myForm.Country.value == "-1" )
 {
alert( "Please provide your country!" );
return false;
}
```

Data Format Validation

```
if(document.myForm.Zip.value == "" ||
isNaN( document.myForm.Zip.value ) ||
   document.myForm.Zip.value.length != 6 )
{
alert( "Please provide a zip in the format #####." );
document.myForm.Zip.focus() ; return false;
}
```

Form Validation Events

- <input type="text" onfocusout="myFunction()">
- <input type="text" ID="textPanNo" MaxLength="10" onblur="ValidatePAN(this);">
- <input type="button" value="Check" onclick="ValidatePAN();">

String: indexOf() and lastIndexOf()

- Search a string
- var str = "Hello world, welcome to the universe.";
  var n = str.indexOf("welcome");
- The result of *n* will be: 13

- Search a string for the last occurrence of a string
- var str = "Hello planet earth, you are a great planet.";
  var n = str.lastIndexOf("planet");
- The result of *n* will be: 36

Both methods return -1 if the value to search for never occurs.

Data Format Validation

```
function validateEmail()
{
   var emailID = document.myForm.EMail.value;
   atpos = emailID.indexOf("@");
   dotpos = emailID.lastIndexOf(".");

   if (atpos < 1 || ( dotpos - atpos < 2 ))
   {
      alert("Please enter correct email ID")
      document.myForm.EMail.focus() ;
      return false;
   }
   return( true );
}
```

Form Validation Script

```
<html><head>
<title>Form Example</title>
<script LANGUAGE="JavaScript">
function validate() {
    if (document.form1.yourname.value.length < 1)
{
       alert("Please enter your name.");
       return false;
    }
   if (document.form1.address.value.length < 3) {
      alert(" Invalid address.");
      return false;
   }
   if (document.form1.phone.value.length < 10) {
      alert("Invalid phone number.");
      return false;
   }
   return true;
}
</script>
</head>
```

```
<body>
<h1>Form Example</h1>
<p>Enter the following information. When you press the Display
button, the data you entered will be validated, then sent by email.</p>

<form name="form1" action="mailto:user@host.com"
enctype="text/plain" onSubmit="validate();">

<p><b>Name:</b> <input type="text" length="20" name="yourname">
</p>
<p><b>Address:</b> <input type="text" length="30" name="address">
</p>
<p><b>Phone: </b> <input type="text" length="15" name="phone">
</p>
<p><input type="submit" value="Submit"></p>

</form>
</body>
</html>
```

Form Validation – isInt()

```
function checkRegistration() {
  var ageField = document.registerForm.ageField;
  if (!isInt(ageField.value)) {
    alert("Age must be an integer.");
    return(false);
  }
```

Pattern Matching

JavaScript provides two ways to do pattern matching:

1. Using RegExp objects
2. Using methods on String objects

Regular Expression

- var p = new RegExp(p, attributes);

or

- var p = /pattern/;

or

- var p = /pattern/modifiers;

- pattern – A string that specifies the pattern of the regular expression or another regular expression.
- modifiers – An optional string containing any of the "g", "i", and "m" modifiers that specify global, case-insensitive, and multiline matches, respectively.

RegExp - Modifiers

- To perform case-insensitive more global searches **Modifiers** can be used :

Modifier	Description
i	Case-insensitive matching is performed. /oak/i matches "OAK" and "Oak"
g	A global match is performed (Rather than stopping after the first match all matches are found)
m	multiline matching is performed
ig	Makes the match insensitive and global

RegExp Patterns - brackets

- A range of characters is found by using brackets:

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets. [0-3] match any decimal digit ranging from 0 through 3
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-z]	It matches any character from lowercase a through lowercase z. [b-v] match any lowercase character ranging from b through v.
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
[aeiou]	matches a single character in the given set
(x|y)	Find any of the alternatives separated with |

Quantifiers

- Defines quantities:

Quantifier	Description
p+	Matches any string that contains at least one p
p*	Matches any string that contains zero or more occurrences of p
p?	Matches any string that contains zero or one occurrences of p
p{N}	It matches any string containing a sequence of N p's
p{min,max}	p{2,3} It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it
^p	It matches any string with p at the beginning of it.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.

Abbreviation

Abbreviation	*Equiv. Pattern*	*Matches*
\d	[0-9]	a digit
\D	[^0-9]	not a digit
\w	[A-Za-z_0-9]	a word char.
\W	[^A-Za-z_0-9]	not a word char.
\s	[\r\t\n\f]	a whitespace char.
\S	[^ \r\t\n\f]	not a whitespace char.

PAN Validation

var panPatt = /^([a-zA-Z]){5}([0-9]){4}([a-zA-Z]){1}?$/;

```
<script>
    function validatePAN() {
        let panInput = document.getElementById("pan");
        let panPattern = /^[A-Z]{5}[0-9]{4}[A-Z]{1}$/; // Regex for PAN format
        let message = document.getElementById("message");

        if (panPattern.test(panInput.value)) {
            message.textContent = "✅ Valid PAN Number!";
            message.style.color = "green";
            return true;
        } else {
            message.textContent = "❌ Invalid PAN Number! Format: ABCDE1234F";
            message.style.color = "red";
            return false;
        }
    }
</script>
```

The **test() method** is used to check if the user's input **matches** the PAN card format.

Matching in String Methods

- **String methods**: search(), match() and replace()
- The search() method returns the position of the given string

string.search(reg-exp) searches the string for the first match

- match() method compares a RE and a string to see whether they match.

match(pattern)

- The replace() method replaces one string with another and returns the modified string

replace(RE_pattern, string)

PAN Pattern

```
<script>
function ValidatePAN() {
var Obj = document.getElementById("textPanNo");
      if (Obj.value != "") {
         pan = Obj.value;
         var panPatt = /^([a-zA-Z]{5})(\d{4})([a-zA-Z]{1})$/;
         if (pan.search(panPatt) == -1) {
            alert("Invalid Pan No");
            return false;
         }
        else {
           alert("Correct Pan No");
           }
      }
 }
 </script>
<input type="text" ID="textPanNo" MaxLength="10">
<input type="button" value="Check" onClick="ValidatePAN();">
```

ABCDE1234J Check

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

pattern attribute

```
<input type="text" name="fieldname" pattern="regex" title="error message">
```

- **Works without JavaScript** for **basic validation**.
- **Does not provide instant feedback** (requires form submission).

Example:
Phone Number (10 digits):
<input type="tel" id="phone" name="phone" **pattern="[0-9]{10}**" title="Please enter a valid 10-digit phone number." required>

Validating a PAN Card Number

```
<body>

    <h2>PAN Card Validation (Using Pattern Attribute)</h2>

    <form>
        <label for="pan">Enter PAN Card Number:</label>
        <input type="text" id="pan" name="pan" maxlength="10"
               pattern="[A-Z]{5}[0-9]{4}[A-Z]{1}"
               placeholder="ABCDE1234F"
               required
               title="Enter a valid PAN format: ABCDE1234F">
        <input type="submit" value="Submit">
    </form>

</body>
```

HTML Pattern attribute

An HTML form with an input field that can contain only three letters **country code** (no numbers or special characters):

```
<form action="/action_page.php">
  <label for="country_code">Country code:</label>
  <input type="text" id="country_code" name="coun
try_code“ pattern="[A-Za-z]{3}" title="Three
letter country code"><br><br>
  <input type="submit">
</form>
```

HTML Pattern attribute

An <input> element with type="**password**" that must contain 8 or more characters that are of at least one number, and one uppercase and lowercase lette

```
<form action="/action_page.php">
  <label for="pwd">Password:</label>
  <input type="password" id="pwd" name="pwd"
  pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
  title="Must contain at least one  number and
one uppercase and lowercase letter, and at
least 8 or more characters">
  <input type="submit">
</form>
```

Thank You

DOM-
Event Handler

Dr. Jenila Livingston L.M.
Professor
VIT Chennai

Event Handler

Event handler attributes allow you to execute JavaScript functions when specific events occur on an HTML element. Some commonly used events are:

Event Handler	Triggers When...
`onclick`	The element is clicked
`ondblclick`	The element is double-clicked
`onchange`	The value of an input field changes
`onmouseover`	The mouse hovers over an element
`onmouseout`	The mouse leaves an element
`onfocus`	The element (input field) gains focus
`onblur`	The element loses focus
`onkeydown`	A key is pressed down
`onkeyup`	A key is released
`onsubmit`	A form is submitted
`onreset`	A form is reset

Common DOM Events

- Mouse events:
  - `onclick`, `onmousedown`, `onmouseup`
  - `onmouseover`, `onmouseout`, `onmousemove`
- Key events:
  - `onkeypress`, `onkeydown`, `onkeyup`
  - Only for input fields
- Interface events:
  - `onblur`, `onfocus`
  - `onscroll`

Common DOM Events (2)

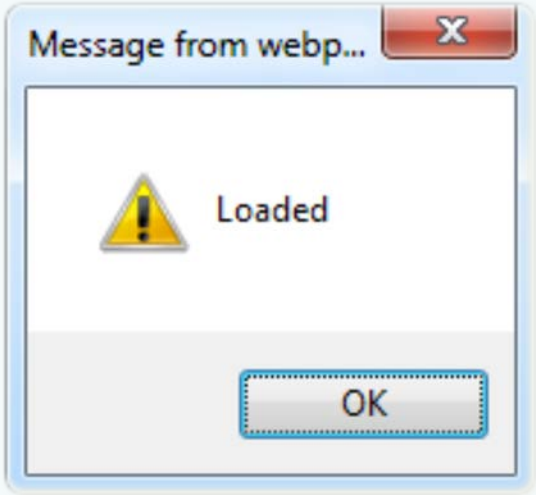
- Form events
  - `onchange` – for input fields
  - `onsubmit`
    - Allows you to cancel a form submission
    - Useful for form validation
- Miscellaneous events
  - `onload`, `onunload`
    - Allowed only for the `<body>` element
    - Fires when all content on the page was loaded / unloaded

onload Event – Example

- onload event

onload.html

```
<html>

<head>
  <script type="text/javascript">
    function greet() {
      alert("Loaded");
    }
  </script>
</head>

<body onload="greet()" >
</body>

</html>
```



onclick

The **onclick** event triggers when a user clicks on an HTML element. It is commonly used with buttons, links, images, and other interactive elements.

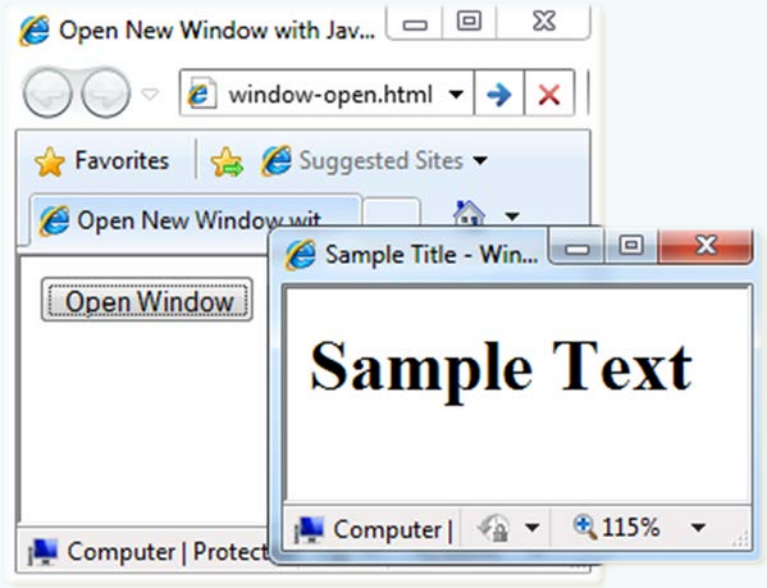
```
<button onclick="showMessage()">Click Me</button>
<p id="output"></p>

<script>
  function showMessage() {
    document.getElementById("output").innerHTML = "Button Clicked!";
  }
</script>
```

Opening a New Window – Example

- window.open()

window-open.html

```
<button onclick="openNewWindow()">Open Window</button>
<script>
  function openNewWindow() {
   var newWindow = window.open("", "sampleWindow",
  "width=300, height=100, menubar=yes,
  status=yes, resizable=yes");

  newWindow.document.write(
  "<html><head>
  <title>Sample Title</title>
  </head>
  <body><h1>SampleText</h1>
  </body>");
  newWindow.document.close();
 }</script>
```



window.open() Attributes

Attribute	Description
toolbar	Creates the standard toolbar
location	Creates the location entry field
directories	Creates standard directory buttons
status	Creates the status bar
menubar	Creates the menu bar at the top of a window
scrollbars	Creates scrollbars when the document exceeds the window size
resizable	Enables the user to resize the window
width	Specifies the width of the window
height	Specifies the height of the window

Properties and methods of the "document" Object

Property	Description
bgColor	A string value representing the background color of a document
alinkColor	A string value representing the color for active links
location	A string value representing the current URL
title	A string value representing the text specified by <title> tag

Method	Description
clear()	Clears the document window
write(content)	Writes the text of content to a document
writeln()	Writes the text and followed by a carriage return
open()	Open a document to receive data from a write() stream
close()	Closes a write() stream

Form Element-Based Objects

Each object in the form has its own properties and methods.

- Text fields, Textarea fields
- Radio buttons
- Check box buttons
- Hidden fields
- Password fields
- Combo box select menu
- List select menu

Working with Input Fields

Property	Description
defaultvalue	The default value that is initially displayed in the field
form	References the form containing the field
maxlength	The maximum number of characters allowed in the field
name	The name of the field
size	The width of the input field in characters
type	The type of input field (button, check box, file, hidden, image, password, radio, reset, submit, text)
value	The current value of the field

Method	Description
blur()	Remove the focus from the field
focus()	Give focus to the field
select()	Select the field

reset and focus

Method	Purpose
`.reset()`	Clears all form fields and restores default values
`.focus()`	Places the cursor inside a specific input field

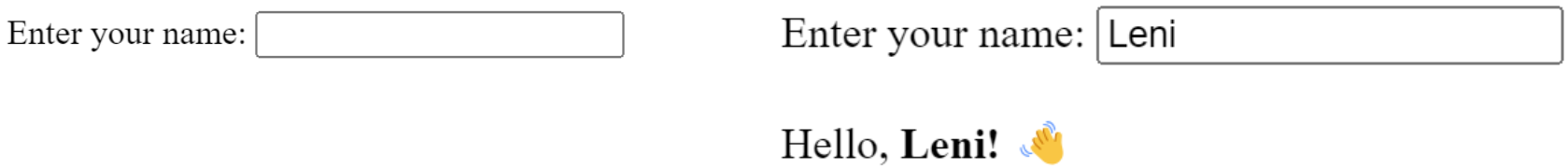
```
<form id="myForm">
  Name: <input type="text" id="name"><br>
  Email: <input type="email" id="email"><br>
  <button type="button" onclick="resetAndFocus()">Reset & Focus</button>
</form>

<script>
  function resetAndFocus() {
    document.getElementById("myForm").reset(); // Clears all fields
    document.getElementById("name").focus();   // Places cursor in Name field
  }
</script>
```

onchange, onmouseover, onmouseout

Event	Triggered When
`onchange`	An input value changes and loses focus
`onmouseover`	Mouse pointer moves over an element
`onmouseout`	Mouse pointer leaves an element

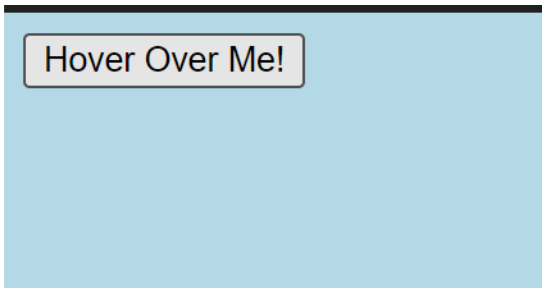
onchange

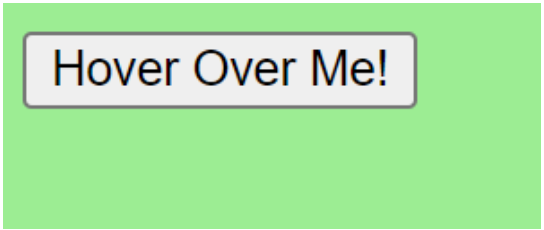
```
Enter your name: <input type="text" id="nameInput" onchange="showName()">
<p id="output"></p>

<script>
  function showName() {
    var name = document.getElementById("nameInput").value;
    document.getElementById("output").innerHTML = `Hello, <b>${name}!</b> 👋`;
  }
</script>
```



Onmouseover/onmouseout

```
<button id="colorButton" onmouseover="changeBgColor()" onmouseout="resetBgColor()">
  Hover Over Me!
</button>

<script>
  function changeBgColor() {
    document.body.style.backgroundColor = "lightblue";
  }

  function resetBgColor() {
    document.body.style.backgroundColor = "lightgreen";
  }
</script>
```





Thank You