# DHTML

- DHTML, or Dynamic HTML, is a web-development technique combining HTML, CSS, and JavaScript to create dynamic, interactive web pages and complex web applications.
- It uses the Document Object Model (DOM) to manipulate content spontaneously and respond to user interactions in real time without reloading the entire page.

# Components of DHTML

- HTML (Hypertext Markup Language): The standard markup language for creating web pages and web applications.
- CSS (Cascading Style Sheets): A style sheet language used for describing the presentation of a document written in HTML or XML.
- JavaScript: A programming language that allows you to implement complex features on web pages, such as interactive forms, animations, and real-time updates.

# Frames

- <html>
- <head>
- <title>Two Rows and Two Columns Frameset</title>
- </head>
- <frameset rows="50%,50%">
- <frame src="top.html" name="topFrame">
- <frameset cols="50%,50%">
- <frame src="left.html" name="leftFrame">
- <frame src="right.html" name="rightFrame">
- </frameset>
- </frameset>
- </html>

# Table

```
<html>
<body>
<table border=1>
<tr>
<td rowspan=2> 1 </td>
<td> 2 </td>

<tr>
<td> 3 </td>
</tr>

<tr>
<td> 4 </td>
<td> 5 </td>
</tr>

<tr>
<td colspan=2>6</td>
</tr>

<tr>
<td>
7
</td>
<td> 8 </td>
</tr>
</table>
</body>
</html>
```

# Forms

- Textboxes
- Radio button
- Check boxes
- Submit
- TextArea

# CSS

- CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation of a document written in HTML or XML.
- CSS controls the layout, formatting, and visual presentation of web pages. By separating content from design, CSS allows for more flexibility and control in the specification of presentation characteristics.

# CSS Syntax

- CSS consists of selectors and declarations. A declaration block contains one or more declarations separated by semicolons. Each declaration includes a property name and a value, separated by a colon.

Selector

{ property: value;

property: value;

}

# Inline Style sheet

- An inline style sheet allows you to include CSS (Cascading Style Sheets) directly within an HTML element using the style attribute. This method is useful for applying quick, one-off styles to specific elements without affecting the rest of the document.

```
<html>
<head>
  <title>Inline Style Example</title>
</head>
<body>
  <h1 style="color: blue; font-family: Arial, sans-serif;">This is a heading</h1>
  <p style="color: green; font-size: 20px;">This is a paragraph.</p>
</body>
</html>
```

# Internal Styles Example

```
<html>
<head>
   <title>Internal Style Sheet Example</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         background-color: #f0f0f0;
      }

      .container {
               background-color: white;
      }
   </style>
</head>
```

Style.css

```
<body>
   <h1>This is a heading</h1>
   <p class =“container”>This is a paragraph.</p>
</body>
</html>
```

# External Styles Example

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a sample paragraph.</p>
</body>
</html>
```

Style.css

```
body {
  background-color: #f0f0f0;
  font-family: Arial, sans-serif;
}

h1 {
  color: #333;
  text-align: center;
}

p {
  color: #666;
  font-size: 16px;
  line-height: 1.5;
}
```

# External Styles Example

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a sample paragraph.</p>
</body>
</html>
```

Style.css

```
body {
  background-color: #f0f0f0;
  font-family: Arial, sans-serif;
}

h1 {
  color: red;
  text-align: center;
}

p {
  color: blue;
  font-size: 16px;
  line-height: 1.5;
}
```

# Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1 id="mainTitle">Welcome to My Website</h1>
  <p>This is a sample paragraph.</p>
  <p id="specialParagraph">This paragraph has a special style.</p>
</body>
</html>
```

Style.css

```
#mainTitle {
  color: blue;
  text-align: center;
}

#specialParagraph {
  color: green;
  font-weight: bold;
}
```

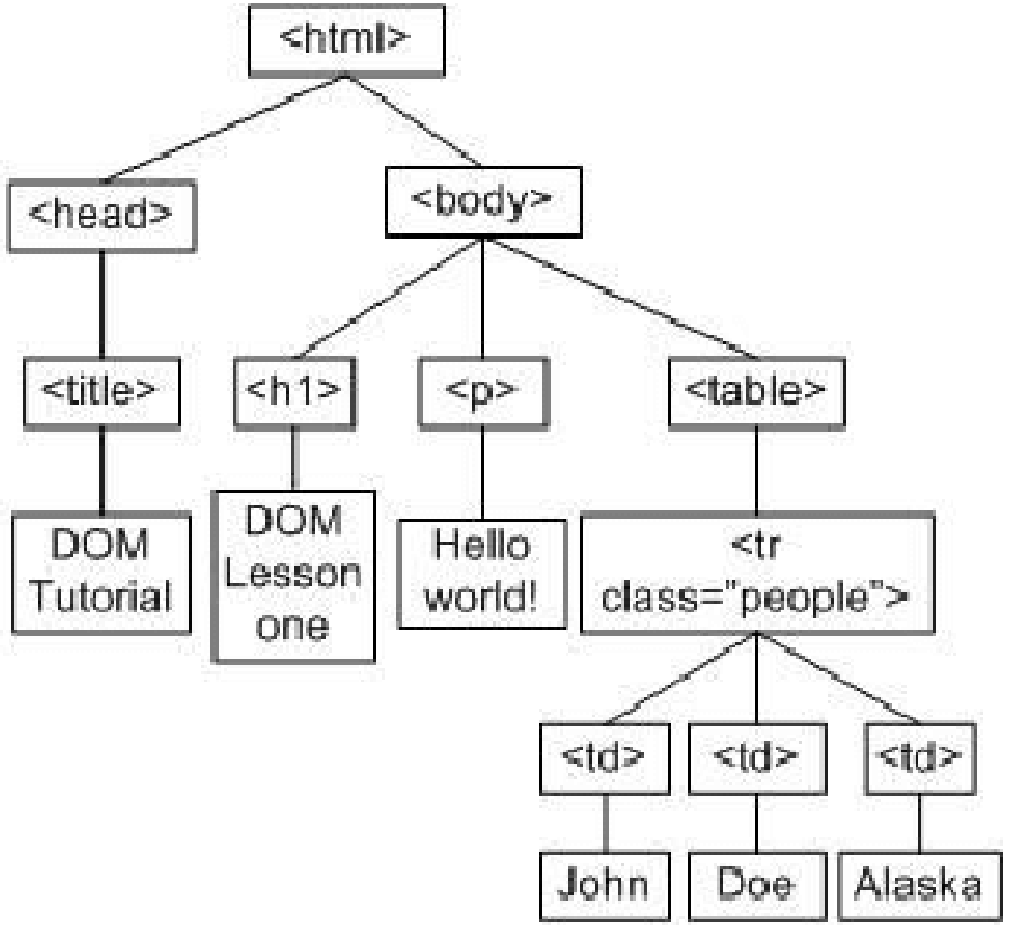
# DOM

- The Document Object Model (DOM) is the skeleton that gives it structure. It's a tree-like representation where every element, attribute, and text content is a node.
- Key Components of a DOM Tree:
  - Root Node: The topmost node, representing the entire document (usually the <html> element).
  - Child Nodes: Elements nested within a parent node.
  - Parent Node: The element that contains a child node.
  - Sibling Nodes: Elements sharing the same parent.
  - Leaf Nodes: Nodes without children (usually text nodes).

- HTML is used to **structure** the web pages and **Javascript** is used to add **behavior** to our web pages.
- When an HTML file is loaded into the browser, the JavaScript can not understand the HTML document directly. So it interprets and interacts with the Document Object Model (DOM), which is created by the browser based on the HTML document.

```
<html>
   <head>
      <title>DOM Tutorial</title>
   </head>
   <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
      <table>
         <tr class="people">
            <td>John</td>
            <td>Doe</td>
            <td>Alaska</td>
         </tr>
      </table>
   </body>
</html>
```

test.html




DOM Tree

# JavaScript

- JavaScript is a versatile programming language primarily used to create and control dynamic website content. It enables interactive web pages and is an essential part of web development, alongside HTML and CSS.

# Key Features of JavaScript:

- **Interactivity:**
  - JavaScript makes web pages interactive, allowing them to respond to user actions such as clicks, form submissions, and mouse movements.
- **Client-Side Execution:**
  - JavaScript is executed on the client side, meaning it runs in the user's web browser, reducing the need for server-side processing and making web pages more responsive.
- **Dynamic Content:**
  - JavaScript can modify HTML and CSS to update the content, structure, and style of web pages in real-time without needing to reload the page.
- **Event Handling:**
  - JavaScript can capture and respond to events such as user inputs, mouse actions, and keyboard events, providing a rich user experience.

# Simple javascript

```
<html>
<head>
   <title>Simple JavaScript Program</title>
</head>
<body>
   <script>
       var num1 = 10;
       var num2 = 20;
      var sum = num1 + num2;
       alert(sum);
   </script>
</body>
</html>
```

```
<html>
<head>
  <script>
    function greetUser() {
      // Prompt the user for their name and age
      var name = prompt("Please enter your name:");
      var age = prompt("Please enter your age:");

      // Display a greeting message
      alert("Hello, " + name + "! You are " + age + " years old.");
    }
  </script>
</head>
<body>
  <h1>Welcome to the JavaScript Program</h1>
  <button onclick="greetUser()">Click Me!</button>
</body>
</html>
```

# Example

```
<html>
<head>
<style>
  #myParagraph {
    color: black;
  }
</style>
<script>
  function changeColor() {
    document.getElementById("myParagraph").style.color =
"blue";
  }
</script>
</head>
<body>

<p id="myParagraph">This is a paragraph.</p>
<button onclick="changeColor()">Change
Color</button>

</body>
</html>
```

# Form Validation Program

```
<html>
<head>
  <script>
    function validateForm() {
      var regNumber = document.f1.regNumber.value;
      var name = document.f1.name.value;
      var department = document.f1.department.value;
      var gender = document.f1.gender.value;
      var hobbies = document.f1.hobbies;
      var address = document.f1.address.value;
                                                  var hobbyChecked = false;


      if (regNumber == "") {
        alert("Registration Number must be filled out");
        return false;
      }

      if (name == "") {
        alert("Name must be filled out");
        return false;
      }

      if (department == "Select") {
        alert("Please select a department");
        return false;
      }

      if (!gender) {
        alert("Please select a gender");
        return false;
      }

      // Check if at least one hobby is selected
     for (var i = 0; i < hobbies.length; i++) {
                                                                  if (hobbies[i].checked) {
                                                                                  hobbyChecked = true;
                                                                                  break;          }

                                                                   }
      if (!hobbyChecked) {
       alert("Please select at least one hobby");
        return false;
                }

      if (address == "") {
        alert("Address must be filled out");
        return false;
      }

      alert("Form submitted successfully!");
      return true;
    }
  </script>
</head>
<body>
  <h2>Form Validation Example</h2>
  <form name="f1" onsubmit="return validateForm()">
```

# Regular Expression

- Regular expressions (regex) in JavaScript are patterns used to match character combinations in strings. They are a powerful tool for tasks such as validation, search, and text manipulation.

- Syntax

/^reg_ex$/;

- Regular Expression:
- ^ asserts the start of the string.
- \d+ matches one or more digits.
- [A-Za-z]+ matches one or more alphabetic characters (both uppercase and lowercase).
- \w: Matches any word character,
- which includes:Alphanumeric characters: a-z, A-Z, 0-9 Underscore: _

  In other words, \w matches letters, digits, and underscores.
- $ asserts the end of the string.
- Validation: Uses the test() method to check if the input value matches the pattern.

- Email pattern:  ^[a-zA-Z0-9]+@gmail\.com$

- var emailPattern = /^\w+@gmail\.com$/;

```
<html>
<head>
<script>

function validateForm() {
    // Get the input fields
    var name = document.f1.name.value;
    var phone = document.f1.phone.value;

    // Regular expressions
    var nameRegex = /^[a-zA-Z]+$/;
    var phoneRegex = /^\d+$/;

    // Validate the name field
    if (!nameRegex.test(name)) {
        alert("Name must contain only letters.");
        return false;
    }

    // Validate the phone field
    if (!phoneRegex.test(phone)) {
        alert("Phone number must contain only numbers.");
        return false;
    }

    // If both validations pass
    alert("Form is valid!");
    return true;
}
</script>
```

```
</head>

<body>
<form name="f1">
Name <input type=text name="name"></br>
Phone <input type=text name="phone"></br>
<button onclick=validateForm()>Click</button>
</body>
</html>
```