CSS Syntax:

CSS (Cascading Style Sheets) is used to style HTML elements. The basic syntax consists of **selectors**, **properties**, and **values**.

Basic Syntax:

```
selector {
  property: value;
}
```

- **selector** → Selects the HTML element to style.
- **property** → Defines what to change (e.g., color, font-size).
- **value** → Specifies the effect of the property.

Example 1: Changing Text Color

```
p {
  color: blue;
}
```

This makes all <p> elements have **blue** text.

Example 2: Changing Background and Font Size

```
body {
  background-color: lightgray;
  font-size: 18px;
}
```

This sets a **light gray background** and makes the text **18px** in size.

Example 3: Styling Multiple Elements

```
h1, h2 {
  color: red;
  text-align: center;
}
```

This makes all <h1> and <h2> tags have **red** text and be **center-aligned**.

Example 4: Using Class and ID Selectors

```
/* Class selector (.) */
.green-text {
  color: green;
}

/* ID selector (#) */
#main-heading {
  font-size: 24px;
}
```

Elements with class="green-text" will have **green text**.
The element with id="main-heading" will have a **24px font size**.

Different Ways to Apply CSS (Types of CSS)

CSS can be applied to HTML in **three** different ways:

1. Inline CSS (Applied directly to an HTML element)

- Uses the style attribute inside an HTML tag.
- Applied to a **single** element only.
- **Not recommended** for larger projects.

Example:

```
<p style="color: red; font-size: 20px;">This is inline CSS.</p>
```

This paragraph will appear in **red** with a **font size of 20px**.

---

2. Internal CSS (Defined within <style> in the <head> section)

- Used for styling a **single webpage**.
- Styles are written inside the <style> tag in the <head> section of the HTML file.

Example:

```
<head>
```

```
    <style>
        h1 {
            color: green;
            text-align: center;
        }
    </style>
</head>
<body>
    <h1>Welcome to My Website</h1>
</body>
```

Effect: The <h1> text will be **green** and **center-aligned**.

---

3. External CSS (Linked via a separate .css file)

- **Best practice** for structuring large projects.
- Keeps CSS in a **separate file** (styles.css) for better **maintainability**.

Example: Create a CSS file (styles.css):

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

Link this file in an HTML document (index.html):

```
<head>
    <link rel="stylesheet" href="styles.css">
</head>
```

Effect: The entire webpage will have a **light gray background** and use the **Arial font**.

Common CSS Properties & Explanation

Property	Description	Example
color	Changes text color	color: red;
background-color	Sets background color	background-color: yellow;

Property	Description	Example
font-size	Sets the size of text	font-size: 16px;
font-family	Sets font type	font-family: Arial, sans-serif;
text-align	Aligns text	text-align: center;
margin	Adds space outside an element	margin: 10px;
padding	Adds space inside an element	padding: 5px;
border	Creates a border around an element	border: 2px solid black;

Border in CSS

CSS **borders** are used to define a visible boundary around an HTML element. You can customize the **width, style, and color** of a border.

Basic Syntax of Border

```
selector {
    border: width style color;
}
```

**Example: Basic Border on a **

```
div {
    border: 2px solid black;
}
```

Effect: The will have a **2-pixel thick black solid border**.

Border Properties in CSS

Property	Description	Example
border-width	Sets the thickness of the border	border-width: 5px;
border-style	Defines the style of the border	border-style: dashed;
border-color	Changes the color of the border	border-color: red;

Property	Description	Example
border	Shorthand for width, style, and color	border: 3px solid blue;

Example:

```
p {
   border-width: 4px;
   border-style: dotted;
   border-color: green;
}
```

Effect: The <p> element will have a **4px green dotted border**.

Different Border Styles in CSS Style	Description	Example
solid	A single solid line	border-style: solid;
dashed	A dashed line	border-style: dashed;
dotted	A dotted line	border-style: dotted;
double	Two solid lines	border-style: double;
groove	A carved effect	border-style: groove;
ridge	A 3D raised effect	border-style: ridge;
inset	Gives an inset effect	border-style: inset;
outset	Gives an outset effect	border-style: outset;
none	No border	border-style: none;

What is Responsive Web Design?

Responsive Web Design (RWD) is a technique that ensures web pages **adapt to different screen sizes** and devices automatically. Instead of creating separate designs for desktops, tablets, and mobiles, RWD makes one **dynamic** design that adjusts based on the device.

Advantages of Responsive Web Design:

Mobile-Friendly Experience

- Works on **all devices** (phones, tablets, desktops).

- Improves **user experience (UX)** with a smooth interface.

Faster Page Load Time

- No need to load separate mobile sites, making pages load faster.
- Google prioritizes fast-loading sites in search rankings.

Better SEO (Search Engine Optimization)

- Google recommends responsive design for better rankings.
- One URL for all devices = **Easier for search engines to index**.

Easy Maintenance & Cost-Effective

- One website for all devices = **Less work for developers**.
- No need for a separate mobile site = **Lower maintenance costs**.

More Conversions & Sales

- A better user experience means **higher engagement & sales**.
- Users won't leave because of a **bad mobile experience**.

Future-Proof Design

- New devices (like smart TVs, foldable phones) will still support the site.
- RWD makes websites **adaptable to future screen sizes**.

AngularJS

AngularJS is a **structural JavaScript framework** developed by **Google** to create **dynamic, single-page web applications (SPAs)**. It extends the capabilities of **HTML** by adding new attributes and features, making it more suitable for interactive and responsive web applications.

Unlike traditional web development, where HTML is used only for static content, AngularJS enables developers to build **dynamic views** with two-way data binding and reusable components.

Features of AngularJS

Model-View-Controller (MVC) Architecture

AngularJS follows the **MVC** pattern, which divides an application into three parts:

- **Model** → Manages application data.
- **View** → Represents the user interface.
- **Controller** → Connects the Model and View, handling business logic.

This architecture helps in organizing code efficiently, making it easy to develop and maintain applications.

Two-Way Data Binding

One of the most powerful features of AngularJS is **two-way data binding**, which ensures that changes in the model (data) are immediately reflected in the view (UI) and vice versa. This reduces the need for additional JavaScript code to update the DOM manually.

Directives

AngularJS introduces **directives**, which are custom HTML attributes that enhance the functionality of HTML elements. Some commonly used directives include:

- ng-app → Defines an AngularJS application.
- ng-model → Binds form inputs to application data.
- ng-repeat → Creates dynamic lists.
- ng-show/ng-hide → Displays or hides elements based on conditions.

Dependency Injection (DI)

AngularJS has a **built-in Dependency Injection (DI)** system that automatically provides necessary services where required. This improves **modularity, flexibility, and testability** of applications.

Routing

AngularJS allows developers to create **single-page applications (SPAs)** using built-in **routing mechanisms**. This enables navigation between different views without reloading the entire webpage, resulting in a **smoother user experience**.

Filters

Filters in AngularJS allow modification of data before displaying it. Common filters include:

- uppercase → Converts text to uppercase.

- lowercase → Converts text to lowercase.
- currency → Formats numbers as currency.
- date → Formats date values.

Built-in Services

AngularJS provides many **predefined services** to handle common functionalities, such as:

- $http → Handles AJAX requests for fetching data.
- $location → Accesses and manipulates browser URLs.
- $timeout → Executes functions after a specified delay.
- $route → Manages application routing.

Advantages of AngularJS

- **Enhances HTML** → Adds powerful features like two-way data binding and directives.
- **Reduces Code Complexity** → Eliminates the need for extensive JavaScript coding.
- **Supports Single Page Applications (SPAs)** → Enables fast, interactive web apps.
- **Reusable Components** → Improves development efficiency.
- **Cross-Browser Compatibility** → Works on all modern web browsers.
- **Testing Support** → Built-in unit testing capabilities.

Write a JavaScript program to validate an Email address.

```
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Email Validation</title>
    <script>
        function validateEmail() {
            var email = document.getElementById("email").value;
            var emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-
Z]{2,}$/;

            if (emailPattern.test(email)) {
                alert("Valid Email Address!");
            } else {
```

```
                alert("Invalid Email Address! Please enter a correct email.");
            }
        }
    </script>
</head>
<body>
    <h2>Email Validation in JavaScript</h2>
    <input type="text" id="email" placeholder="Enter your email">
    <button onclick="validateEmail()">Check Email</button>
</body>
</html>
```

What is the use of JavaScript? List and discuss the advantages of JavaScript.

JavaScript (JS) is a high-level, interpreted programming language primarily used for web development. It allows developers to create interactive and dynamic web pages by adding functionalities like form validation, animations, pop-ups, event handling, and AJAX requests.

Uses of JavaScript

1. Front-End Development (Client-Side Scripting)

- Enhances HTML and CSS to create interactive web pages.
- Used in frameworks like **React.js, Angular, and Vue.js**.
- Example: **Form validation, image sliders, interactive buttons**.

2. Back-End Development (Server-Side Scripting)

- JavaScript can be used on the server-side with **Node.js**.
- Helps build full-stack web applications.
- Example: **Handling database queries, authentication, APIs**.

3. Web and Mobile App Development

- Used in hybrid mobile app development frameworks like **React Native, Ionic**.
- Example: **Cross-platform apps with a single codebase**.

4. Game Development

- JavaScript is used with **HTML5 Canvas** to build browser-based games.
- Example: **2D/3D games with Phaser.js or Three.js**.

5. Data Visualization & Machine Learning

- Libraries like **D3.js, Chart.js** for data visualization.
- Used in AI/ML applications with **TensorFlow.js**.

6. Internet of Things (IoT)

- JavaScript helps control hardware devices using **Node.js**.

Advantages of JavaScript

1. Easy to Learn and Use
JavaScript has a **simple syntax** similar to C and Java, making it easy to learn and implement.
2. Client-Side Execution (Fast Performance)
JavaScript runs **directly in the browser**, reducing server load and providing **fast execution**.
3. Cross-Browser Compatibility
Most modern browsers **(Chrome, Firefox, Edge, Safari)** support JavaScript, making it widely accessible.
4. Enhances Web Interactivity
JavaScript allows **real-time updates, animations, and event handling**, improving user experience.
5. Versatility (Front-End + Back-End)
With **Node.js**, JavaScript can be used for both **client-side and server-side development**.
6. Rich Ecosystem of Libraries & Frameworks
Thousands of frameworks/libraries like **React.js, Angular.js, Vue.js, jQuery** simplify development.
7. Reduces Server Load
Since JavaScript runs in the browser, it processes **client-side logic** without excessive server requests.
8. Large Community Support
JavaScript has **millions of developers worldwide**, offering strong community support and frequent updates.

Single Page Application (SPA) in AngularJS
A Single Page Application (SPA) is a web application that loads a single HTML page and dynamically updates the content as the user interacts with the app. Unlike traditional multi-page applications, where each user action triggers a full-page reload, SPAs use JavaScript to fetch and update content without requiring a complete reload.
AngularJS provides built-in support for creating SPAs through its routing mechanism, which allows developers to define multiple views within a single page. The $routeProvider service in AngularJS enables navigation between different views without reloading the entire page, leading to a faster and smoother user experience. SPAs also enhance performance by reducing server requests and improving the responsiveness of web applications.

Advantages of AngularJS

1. **Supports Single Page Applications (SPAs)**
   AngularJS is specifically designed to develop SPAs, allowing smooth transitions between views and reducing the need for multiple server requests.
2. **Two-Way Data Binding**
   AngularJS automatically synchronizes data between the model and the view, reducing the need for manual DOM manipulation and making development more efficient.
3. **Model-View-Controller (MVC) Architecture**
   The MVC pattern in AngularJS helps in organizing code systematically, making it easier to develop and maintain large applications.
4. **Dependency Injection (DI)**
   AngularJS has a built-in dependency injection system, which simplifies component management and enhances code modularity.
5. **Directives for HTML Enhancement**
   AngularJS extends the functionality of HTML using directives such as ng-model, ng-repeat, and ng-show, enabling developers to create dynamic and interactive web applications.

6. **Faster Development with Less Code**
   AngularJS reduces the amount of JavaScript code required by handling many common functionalities, such as data binding and dependency management, automatically.
7. **Reusable Components**
   AngularJS allows developers to create reusable components, improving efficiency and reducing redundancy in code development.
8. **RESTful API Integration**
   AngularJS provides built-in support for working with RESTful APIs using the $http service, making it easy to fetch and update data from a server.

Dialog Boxes in JavaScript

JavaScript provides three types of dialog boxes to interact with users:

1. **Alert Box**
2. **Confirm Box**
3. **Prompt Box**

Each dialog box serves a specific purpose in user interaction and provides a simple way to display messages or gather input from users.

1. Alert Box

The alert() method is used to display a simple message to the user. It contains a message and an "OK" button. This is typically used to show important information or warnings.

Example:

```
alert("This is an alert box!");
```

Explanation:

- When executed, a pop-up window appears with the message "This is an alert box!" and an "OK" button.
- The user must click "OK" to close the alert.

2. Confirm Box

The confirm() method is used to display a message with two options: "OK" and "Cancel". It is generally used to get confirmation from the user before proceeding with an action.

Example:

```
var result = confirm("Are you sure you want to delete this?");
if (result) {
   alert("Item deleted!");
} else {
   alert("Action canceled!");
}
```

Explanation:

- If the user clicks "OK", the confirm() method returns true, and the "Item deleted!" message appears.
- If the user clicks "Cancel", it returns false, and "Action canceled!" is displayed.

3. Prompt Box

The prompt() method is used to take input from the user. It displays a text box where the user can enter a value.

Example:

```
var name = prompt("Enter your name:");
if (name) {
   alert("Hello, " + name + "!");
} else {
   alert("You did not enter a name.");
}
```

Explanation:

- The user is prompted to enter their name.
- If they enter a name, an alert appears with "Hello, [name]!".
- If they leave it blank or cancel, an alert appears saying "You did not enter a name."
