INTRODUCTION TO WEB

Web Technology refers to the various tools and techniques that are utilized in the process of communication between different types of devices over the Internet. A web browser is used to access web pages. Web browsers can be defined as programs that display text, data, pictures, animation, and video on the Internet. Hyperlinked resources on the World Wide Web can be accessed using software interfaces provided by Web browsers.

Web Technology can be Classified into the Following Sections:

- **World Wide Web (WWW):** The World Wide Web is based on several different technologies: Web browsers, Hypertext Markup Language (HTML), and Hypertext Transfer Protocol (HTTP).
- **Web Browser:** The web browser is an application software to explore www (World Wide Web). It provides an interface between the server and the client and requests to the server for web documents and services.
- **Web Server:** Web server is a program which processes the network requests of the users and serves them with files that create web pages. This exchange takes place using Hypertext Transfer Protocol (HTTP).
- **Web Pages:** A webpage is a digital document that is linked to the World Wide Web and viewable by anyone connected to the internet has a web browser.
- **Web Development:** Web development refers to the building, creating, and maintaining of websites. It includes aspects such as web design, web publishing, web programming, and database management. It is the creation of an application that works over the internet i.e. websites.

World Wide Web (WWW)

- The **World Wide Web (WWW)** is a system of interconnected documents and resources, accessible via the internet. It was invented by **Tim Berners-Lee** in **1989** and made publicly available in **1991**.

Key Components of the WWW

1. **Web Pages & Websites** – Documents written in **HTML** that can contain text, images, videos, and links.
2. **URLs (Uniform Resource Locators)** – Addresses that specify the location of web resources (e.g., https://www.example.com).
3. **Web Browsers** – Applications (e.g., Chrome, Firefox) that retrieve and display web content.
4. **HTTP/HTTPS (Hypertext Transfer Protocol)** – The communication protocol used for transferring web data between clients and servers.
5. **Web Servers** – Computers that store and serve web pages upon request.

How the WWW Works?

1. A user enters a URL in a web browser.
2. The browser sends an HTTP request to the web server hosting the page.
3. The server processes the request and sends back the requested webpage.
4. The browser interprets the HTML, CSS, and JavaScript to display the content.

Difference Between the Internet and the WWW

- **Internet**: A global network of computers that allows data transmission.
- **WWW**: A service that runs on the internet, enabling access to web pages via browsers.

WEB BROWSER

A web browser is a software application that enables users to access and interact with content on the World Wide Web. When you enter a website's URL into a browser, it retrieves the necessary data from a web server and displays the page on your device. Browsers are essential tools for navigating the internet, allowing you to view text, images, videos, and other multimedia content.

Popular Web Browsers:

- **Google Chrome**: Developed by Google, Chrome is known for its speed, simplicity, and seamless integration with Google services.
- **Mozilla Firefox**: An open-source browser from Mozilla, Firefox emphasizes user privacy and offers robust customization options.
- **Microsoft Edge**: Microsoft's browser, based on the Chromium engine, offers integration with Windows devices and features like AI-assisted browsing.
- **Safari**: Apple's default browser for macOS and iOS devices, Safari is optimized for performance and energy efficiency on Apple hardware.
- **Opera**: Known for its built-in ad blocker, free VPN, and other integrated features, Opera offers a unique browsing experience.
- **Brave**: A privacy-focused browser that blocks ads and trackers by default, Brave aims to provide a faster and more secure browsing experience.

- **Zen Browser**: An open-source browser based on Firefox, Zen Browser focuses on privacy, customizability, and a unique design with vertical tabs.

Key Features of Modern Web Browsers:

- **Tabbed Browsing**: Allows users to open multiple web pages in a single window, each in its own tab.
- **Bookmarks**: Enable users to save and organize links to their favorite websites for easy access.
- **Private Browsing Modes**: Also known as "Incognito Mode" in some browsers, this feature prevents the browser from storing history, cookies, and other data during a session.
- **Extensions/Add-ons**: Small software modules that add specific functionalities to a browser, such as ad blockers, password managers, and more.
- **Synchronization**: Allows users to sync their bookmarks, history, passwords, and settings across multiple devices using a user account.
-

Security Considerations:

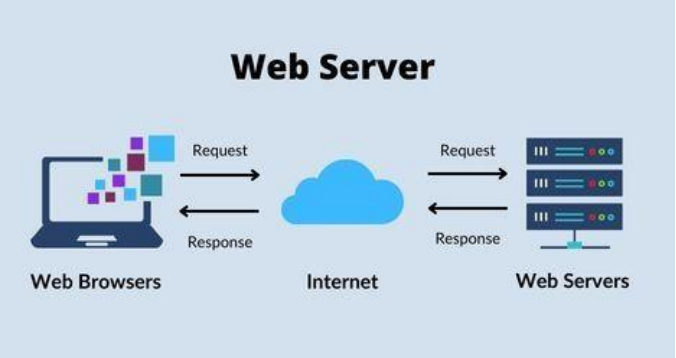
Web browsers are common targets for cyberattacks. To enhance security:

- **Keep Your Browser Updated**: Regular updates patch security vulnerabilities and provide new features.
- **Use Trusted Extensions**: Install extensions from reputable sources to minimize security risks.
- **Enable HTTPS-Only Mode**: This ensures that your browser connects to websites securely whenever possible.

Choosing the right web browser depends on your specific needs, including considerations like speed, security, privacy, and compatibility with your devices.

WEB SERVERS

- A web server is a software application or hardware device that stores, processes, and serves web content to users over the internet. It plays a critical role in the client-server model of the World Wide Web, where clients (typically web browsers) request web pages and resources, and servers respond to these requests by delivering the requested content.
- Web servers operate on the Hypertext Transfer Protocol (HTTP), which is the foundation of data communication on the World Wide Web. When you enter a website's URL into your browser, it sends an HTTP request to the web server hosting that website, which then sends back the web page you requested, allowing you to view it in your browser.



Types of Web Servers Softwares:

- There are several types of web servers, each designed for specific purposes:

  a. Apache HTTP Server : Apache is one of the most popular open-source web servers globally, known for its flexibility and robustness. It's highly customizable and supports a wide range of modules and extensions.

  b. Nginx : Nginx is another widely used web server known for its speed and efficiency in handling concurrent connections.

  c. Microsoft Internet Information Services (IIS) : IIS is a web server developed by Microsoft for Windows servers. It's commonly used for hosting websites and web applications built on Microsoft technologies like ASP.NET.

d. LiteSpeed : LiteSpeed is a commercial web server known for its high performance and security features. It's often used in hosting environments where speed and security are paramount.

Features of Web Servers

Web servers offer a range of features, including:

- **Content Hosting :** They store and serve web content, including HTML pages, images, videos, and other multimedia files.
- **Security :** Web servers implement various security mechanisms to protect against unauthorized access and cyberattacks.
- **Load Balancing :** Some web servers can distribute incoming traffic across multiple server instances to ensure optimal performance and availability.
- **Logging and Monitoring :** They provide tools to track and analyze server performance, user access, and error logs.
- **Caching :** Web servers can cache frequently accessed content to reduce server load and improve response times.

Benefits of Web Servers

Using web servers offers several advantages, including:

- **Scalability :** Web servers can handle a large number of simultaneous connections, making them suitable for high-traffic websites.
- **Reliability :** They are designed for continuous operation and can recover from failures gracefully.
- **Security :** Web servers include security features to protect against common web threats like DDoS attacks and SQL injection.
- **Customization :** Web server configurations can be tailored to specific application requirements.

Uses of Web Server:

- **Hosting Websites:** The most common use of web servers is to host websites, making them accessible on the internet.
- **Web Applications:** Web servers provide the infrastructure for hosting web applications, enabling users to interact with software through a web interface.
- **File Sharing:** Some web servers are used for file sharing and collaboration, allowing users to upload and download files securely.
- **Content Delivery:** Content delivery networks (CDNs) use web servers to distribute content like images and videos to users worldwide, reducing load times.
- **API Hosting:** Web servers are used to host APIs (Application Programming Interfaces) that allow applications to communicate and exchange data over the internet.

CLIENT AND SERVER AND ITS COMMUNICATION

- **Client and Server** are two key components of a network communication model, particularly in computing and web applications.

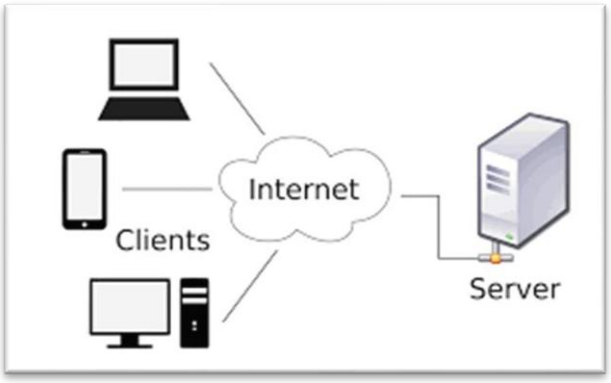
Client

- A **client** is a device or software that requests services or resources from a **server**. It initiates communication by sending requests and waits for responses.
- **Examples:**
    - A web browser (Chrome, Firefox) acting as a client when accessing a website.
    - A mobile app fetching data from a cloud server.
    - A desktop email client retrieving emails from a mail server.

Server

- A **server** is a system that provides services, data, or resources to clients. It processes incoming client requests and responds accordingly.
- **Examples:**
    - A **web server** (e.g., Apache, Nginx) hosting websites.
    - A **database server** (e.g., MySQL, PostgreSQL) storing and managing data.
    - A **file server** providing shared access to files over a network.

Client-Server Communication



1. **Client sends a request** (e.g., HTTP request to a web server).
2. **Server processes the request** and retrieves the necessary information.
3. **Server sends a response** back to the client.
4. **Client receives and processes the response** (e.g., rendering a web page).

INTERNET

- The Internet is a global network that connects billions of computers across the world with each other and to the World Wide Web. It uses a standard internet protocol suite (TCP/IP) to connect billions of computer users worldwide.
- It is set up by using cables such as optical fibers and other wireless and networking technologies. At present, the internet is the fastest means of sending or exchanging information and data between computers across the world.
- It is believed that the Internet was developed by "Defense Advanced Projects Agency" (DARPA) department of the United States. And, it was first connected in 1969.

• Key Characteristics of the Internet

- **Global Network:** A massive interconnection of computers and servers across the world.
- **Uses TCP/IP Protocol:** Data is transmitted using the **Transmission Control Protocol (TCP)** and **Internet Protocol (IP)**.
- **Decentralized Structure:** No single entity owns the internet; it is managed by different organizations and service providers.
- **Allows Various Services:** Supports services like the **World Wide Web (WWW), email, file sharing, video streaming, and online gaming**.
- **Relies on ISPs:** Internet Service Providers (ISPs) provide access to users.

• How Does the Internet Work?

- **Data Transmission:** Data is broken into small packets and sent through different paths.
  IP Addressing: Each device has a unique IP address for identification.
  DNS (Domain Name System): Converts human-readable domain names (e.g.,google.com) into IP addresses.
  Web Browsing: A browser sends a request to a web server, which responds with the requested webpage.

Basic Internet Protocols

- Protocols are rules that govern how data is transmitted over a network.

- Internet protocols are a set of rules that define how data is transmitted and received over the Internet. These protocols ensure seamless communication between devices across networks.

• Types of Internet Protocols

1. Communication Protocols

- These protocols define how data is transmitted over the internet.
- **TCP (Transmission Control Protocol):** Ensures reliable data transfer by breaking data into packets and reassembling them at the destination.
- **IP (Internet Protocol):** Assigns unique IP addresses to devices and routes data between them.
- **UDP (User Datagram Protocol):** Faster but less reliable than TCP; used for streaming and online gaming.

2. Web Protocols

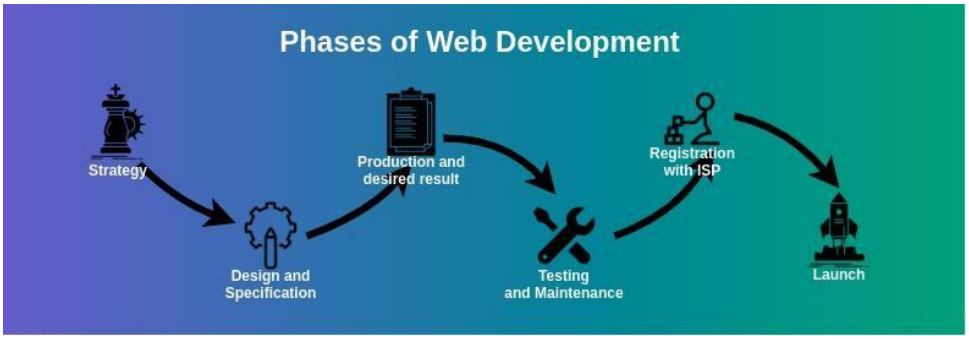
- These protocols help in browsing and accessing web content.
- **HTTP (Hypertext Transfer Protocol):** Used for transferring web pages and other resources.
- **HTTPS (HTTP Secure):** Secure version of HTTP with encryption (SSL/TLS).
- **3. File Transfer Protocols**
- These protocols manage file uploads and downloads.
- **FTP (File Transfer Protocol):** Transfers files between computers.
- **SFTP (Secure File Transfer Protocol):** Secure version of FTP with encryption.
- **4. Network Security Protocols**
- These protocols ensure data security and encryption.
- **SSL/TLS (Secure Sockets Layer / Transport Layer Security):** Encrypts data for secure communication over the web.
- **IPSec (Internet Protocol Security):** Secures IP communications through encryption and authentication.
- **6. Address Resolution Protocols**
- **DNS (Domain Name System):** Converts domain names (e.g., google.com) into IP addresses.
- **DHCP (Dynamic Host Configuration Protocol):** Automatically assigns IP addresses to devices on a network.
- **ARP (Address Resolution Protocol):** Maps IP addresses to MAC (Media Access Control) addresses.

- Example: When you type https://www.google.com, your browser uses **DNS** to find the IP address of Google's server, then establishes a **TCP/IP** connection, and retrieves data using **HTTP/HTTPS**.

Protocol	Description	Purpose
HTTP (HyperText Transfer Protocol)	Transfers web pages	Used to load websites
HTTPS (Secure HTTP)	Secure version of HTTP with encryption	Ensures secure data transfer
FTP (File Transfer Protocol)	Transfers files between computers	Uploading/downloading files
TCP/IP (Transmission Control Protocol/Internet Protocol)	Fundamental protocol for internet communication	Ensures reliable data transmission
SMTP (Simple Mail Transfer Protocol)	Sends emails	Used for outgoing emails
POP3 (Post Office Protocol 3)	Retrieves emails from the server	Downloads emails to local devices
IMAP (Internet Message Access Protocol)	Synchronizes emails across devices	Used for managing emails on multiple devices
DNS (Domain Name System)	Converts domain names to IP addresses	Helps browsers locate websites
DHCP (Dynamic Host Configuration Protocol)	Assigns IP addresses dynamically	Helps devices connect to networks
SSH (Secure Shell Protocol)	Secure remote access to servers	Used by developers and administrators

WEB DEVELOPMENT STRATEGIES

Web Development refers to a term that includes all the processes involved in developing a web project or website. It contains the various phases such as planning, designing, testing, and launching of the web project. The web development process requires a team of experts responsible for implementing the different tasks needed to create a website.



- The various stages that are needed to develop a web project in web development are as follows:
- **Strategy:** The first step in the web development process for a developer is to make a strategy for developing a web page or website. In the strategy phase, the web developer has to do the following:

1. Deciding goals and objectives
2. Developing team
3. Make the appropriate analysis associated with the problem and review the analysis.
4. Formulate a list of tasks
5. Proposal of a project to the web team for developing.

1. **Planning and Requirement Analysis**

   Define Goals – Identify the purpose of the website (e.g., e-commerce, blog, portfolio, SaaS).
   Target Audience – Understand user demographics and preferences.
   Technology Stack – Choose the right front-end, back-end, database, and hosting solutions (e.g., MERN, MEAN, LAMP).
2. **User Experience (UX) and Design**

   Ensure the website works across all devices (mobile, tablet, desktop).
   Use clean layouts, intuitive navigation, and fast-loading elements.
   Ensure the site is usable by people with disabilities.
   Use uniform colors, fonts, and visuals to create brand identity.
3. **Front-End Development Strategies**

   Choose a Modern Framework

   Optimize Performance

   Use proper HTML structure, meta tags, and alt text.
   Use CSS Frameworks
4. **Back-End Development Strategies**

   Select a Suitable Language – Node.js, Python (Django/Flask), PHP, or Ruby on Rails.

   Enable smooth data exchange between front-end and back-end.

   Database Optimization – Choose SQL (MySQL, PostgreSQL) or NoSQL (MongoDB, Firebase) based on project needs.

   Implement Authentication –use session-based authentication for security.
5. **Performance and Security**

   SSL/TLS Encryption – Enable HTTPS for secure connections.

   Prevent Security Threats – Protect against SQL injection, XSS, and CSRF attacks.

   Regular Updates – Keep software and libraries updated to fix vulnerabilities.
6. **Testing and Debugging**
7. **Deployment and Maintenance- (Use Cloud Hosting** – AWS, Google Cloud or Netlify for scalability.)
8. **Marketing and SEO** (Search Engine Optimization)**Optimization**

   SEO Best Practices – Optimize meta descriptions, headings, and keywords.

   Social Media Integration – Add social sharing buttons and metadata.

HTML Basics (HyperText Markup Language)

HTML (HyperText Markup Language) is the standard language used to create and structure web pages. It consists of **elements** that define the structure and content of a webpage.

1. Basic Structure of an HTML Document

Every HTML document has a standard structure:

```
<!DOCTYPE html>
<html>
<head>
    <title>My First Webpage</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <p>This is a simple paragraph in HTML.</p>
</body>
</html>
```

HTML uses **tags** enclosed in angle brackets (< >). Tags usually come in **pairs** (<tag>...</tag>) with an **opening tag** and a **closing tag** (</tag>).

Common HTML Tags:

Tag	Description	Example
`<h1>` to `<h6>`	Headings (largest to smallest)	`<h1>Heading 1</h1>`
`<p>`	Paragraph	`<p>This is a paragraph.</p>`
`<a>`	Hyperlink	`<a href="https://example.com">Click here</a>`
`<img>`	Image	`<img src="image.jpg" alt="My Image">`
`<ul>` / `<ol>`	Unordered/Ordered List	`<ul><li>Item 1</li></ul>`
`<table>`	Table	`<table><tr><td>Data</td></tr></table>`
`<div>`	Division (for layout)	`<div>This is a section</div>`

Attributes in HTML

- Attributes provide additional information about elements. They are written inside the opening tag.
- Example:

```
<a href="https://example.com" target="_blank">Visit Example</a>
```

  - `href="https://example.com"` → Specifies the link.
  - `target="_blank"` → Opens the link in a new tab.

- HTML Elements

- An HTML element is defined by a start tag, some content, and an end tag.
- The HTML **element** is everything from the start tag to the end tag:

<tagname> Content goes here... </tagname>

- Examples of some HTML elements:

<h1> My First Heading </h1>

 My first paragraph.

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
	My first paragraph.	
 	none	*none*

Note: Some HTML elements have no content (like the
 element). These elements are called empty elements. Empty elements do not have an end tag!

Nested HTML Elements

- HTML elements can be nested (this means that elements can contain other elements).
- All HTML documents consist of nested HTML elements.
- The following example contains four HTML elements (<html>, <body>, <h1> and):

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

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

- The <html> element is the root element and it defines the whole HTML document.
- It has a start tag <html> and an end tag </html>.
- Then, inside the <html> element there is a <body> element:

- The <body> element defines the document's body.
- It has a start tag <body> and an end tag </body>.
- Then, inside the <body> element there are two other elements: <h1> and :
- The <h1> element defines a heading.
- It has a start tag <h1> and an end tag </h1>:
- The element defines a paragraph.
- It has a start tag and an end tag :

Empty HTML Elements

- HTML elements with no content are called empty elements.
- The
 tag defines a line break, and is an empty element without a closing tag:

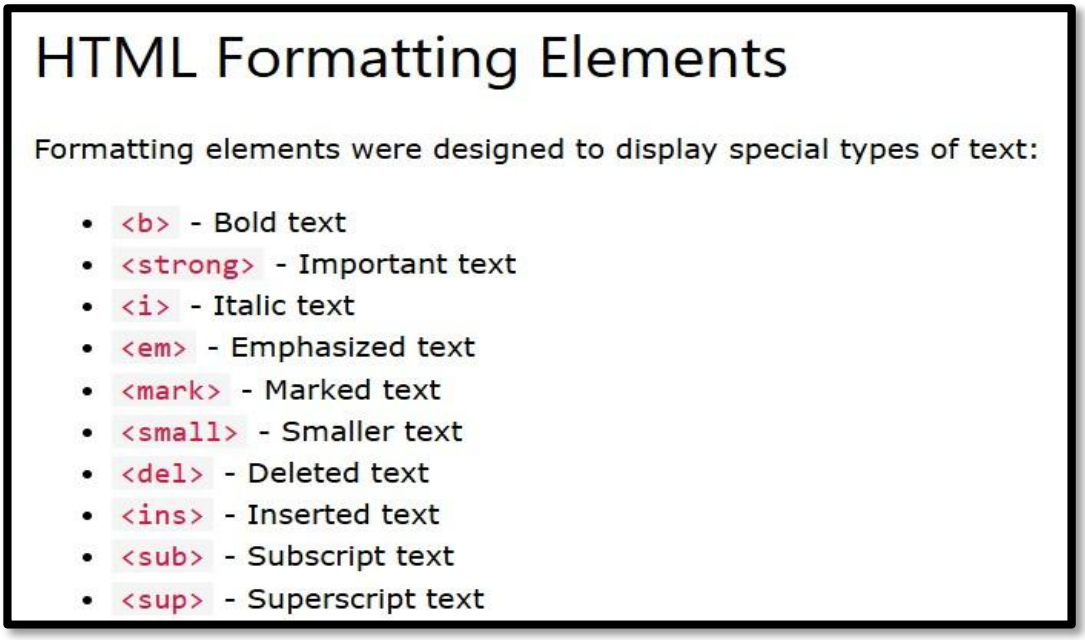
```
<p>This is a <br> paragraph with a line break.</p>
```

HTML is Not Case Sensitive

- HTML tags are not case sensitive: <P> means the same as

HTML Text Formatting

HTML contains several elements for defining text with a special meaning.



Example:

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

<p>This text is normal.</p>

<p><b>This text is bold.</b></p>

</body>
</html>
```

This text is normal.

This text is bold.

HTML Comments

- HTML comments are not displayed in the browser, but they can help document your HTML source code.

HTML Comment Tag

- You can add comments to your HTML source by using the following syntax:

  <!-- Write your comments here -->

Notice that there is an exclamation point (!) in the start tag, but not in the end tag.

Example:-

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

<!-- This is a comment -->
<p>This is a paragraph.</p>
<!-- Comments are not displayed in the browser -->

</body>
</html>
```

HTML Marquee tag

- Marquee is one of the important tags introduced in HTML to support such scrollable texts and images within a web page. In this tutorial, you will learn about the Marquee tag and its different attributes for developing a well-groomed static website.

- The <marquee> tag is a container tag of HTML that is implemented for creating scrollable text or images within a web page from either left to right or vice versa, or top to bottom or vice versa. But

this tag has been deprecated in the new version of HTML, i.e., HTML 5.

The different attributes of `<marquee>` tag are:

Attribute	Description
width	provides the width or breadth of a marquee. For example `width="10"` or `width="20%"`
height	provides the height or length of a marquee. For example `height="20"` or `height="30%"`
direction	provides the direction or way in which your marquee will allow you to scroll. The value of this attribute can be: left, right, up, or down
scrolldelay	provides a feature whose value will be used for delaying each jump.
scrollamount	provides value for speeding the marquee feature
behavior	provides the scrolling type in a marquee. That scrolling can be like sliding, scrolling, or alternate
loop	provides how many times the marquee will loop
bgcolor	provides a background color where the value will be either the name of the color or the hexadecimal color code.
vspace	provides a vertical space, and its value can be like `vspace="20"` or `vspace="30%"`
hspace	provides a horizontal space, and its value can be like `hspace="20"` or `hspace="30%"`

HTML LISTS

- HTML provides three different techniques to specify information in the form of lists. It is to be noted that every list needs to have at least one multiple data element within it. The various forms of HTML list tags are:

HTML Tag	Description
`<ol>`	HTML `<ol>` tag is abbreviated as an Ordered List, which is used for numbering the lists on a web page.
`<ul>`	HTML UL tag is abbreviated as an Unordered List, used to list your items via bullets and circles.
`<dl>`	HTML dl tag is abbreviated as a Definition List, which is used to arrange your data items like how items remain placed in any dictionary.

The type Attribute

You can use the type attribute in these above listing tags. This type of attribute will eventually help you specify your customized type from the types pre-specified by HTML.

For Ordered List, the type can have five values:

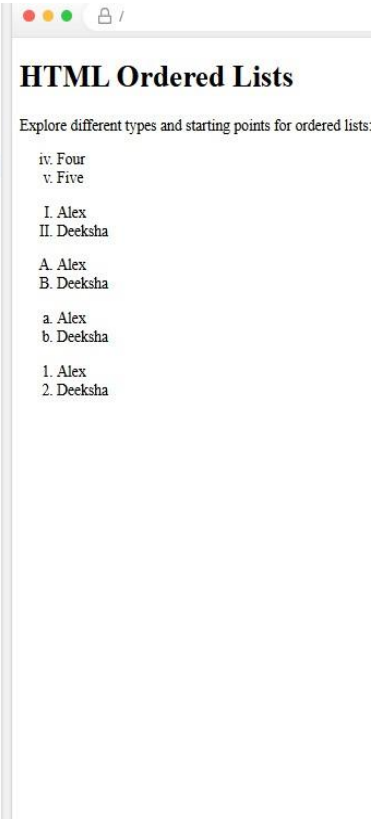
- `<ol type = "1">` - Numbers, which is the default type of Ordered List
- `<ol type = "i">` - Numerals (roman) with lower caps
- `<ol type = "I">` - Numerals (roman) with upper caps
- `<ol type = "a">` - Numbering will be done in the form of Lower-case Letters
- `<ol type = "A">` - Numbering will be done in the form of upper-case Letters

Similarly, for Unordered lists also, there are three symbols that will act as bullets for your lists.

These three values can be used for type attribute of `<ul>`

- `<ul type = "square">` - Bulleting using a filled square.
- `<ul type = "disc">` - Bulleting using a filled circle.
- `<ul type = "circle">` - Bulleting using an empty circle.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML Ordered Lists</title>
</head>
<body>
    <h1>HTML Ordered Lists</h1>

    <p>Explore different types and starting points for ordered lists:</p>
    <ol type="i" start="4">
        <li> Four </li>
        <li> Five </li>
    </ol>

    <ol type="I" start="1">
        <li> Alex </li>
        <li> Deeksha </li>
   </ol>

    <ol type="A" start="1">
        <li> Alex </li>
        <li> Deeksha </li>
    </ol>

    <ol type="a" start="1">
        <li> Alex </li>
        <li> Deeksha </li>
    </ol>

    <ol type="1" start="1">
        <li> Alex </li>
        <li> Deeksha </li>
      </ol>
</body>
</html>
```



HTML Tables

- HTML tables allow you to organize and display data in a structured format using rows and columns.

Basic Table Structure

- An HTML table is created using the <table> tag.
- Rows are defined with <tr>
- Table cells are defined using <td> (data cell) or <th> (header cell).

```
<table border="1">
    <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Country</th>
    </tr>
    <tr>
        <td>Alice</td>
        <td>25</td>
        <td>USA</td>
    </tr>
    <tr>
        <td>Bob</td>
        <td>30</td>
        <td>UK</td>
    </tr>
</table>
```

Result:

Name	Age	Country
Alice	25	USA
Bob	30	UK

HTML IMAGES

- Images can improve the design and the appearance of a web page.

HTML Images Syntax

- The HTML <img> tag is used to embed an image in a web page.
- Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.
- The <img> tag is empty, it contains attributes only, and does not have a closing tag.
- The <img> tag has two required attributes:
    - src - Specifies the path to the image
    - alt - Specifies an alternate text for the image

```
<!DOCTYPE html>
<html>
<body>
<h2>HTML Image</h2>>
<img src="pic_trulli.jpg" alt=Trulli" width="500" height="333">
</body>
</html>
```

HTML FRAMES

- The <frame> tag was used in HTML 4 to define one particular window (frame) within a <frameset>.

HTML FORMS

- An HTML form is used to collect user input. The user input is most often sent to a server for processing. The <form> Element
- The HTML <form> element is used to create an HTML form for user input:
- The <form> element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.
- All the different form elements are covered in this chapter: HTML Form Elements.

The <input> Element

- The HTML <input> element is the most used form element.
- An <input> element can be displayed in many ways, depending on the type attribute.

Here are some examples:

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

Text Fields

- The <input type="text"> defines a single-line input field for text input.

The <label> Element

- The <label> tag defines a label for many form elements.
- The <label> element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focuses on the input element.

- The <label> element also helps users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it toggles the radio button/checkbox.
- The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

Radio Buttons

- The <input type="radio"> defines a radio button.
- Radio buttons let a user select ONE of a limited number of choices.

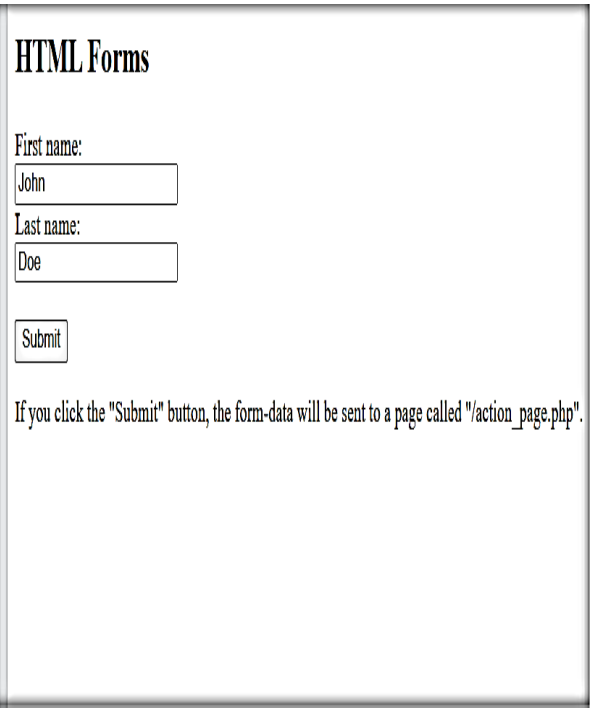
Checkboxes

- The <input type="checkbox"> defines a checkbox.
- Checkboxes let a user select ZERO or MORE options of a limited number of choices.

The Submit Button

- The <input type="submit"> defines a button for submitting the form data to a form-handler.
- The form-handler is typically a file on the server with a script for processing input data.

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

<h2>HTML Forms</h2>

<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called "/action_page.php".</p>

</body>
</html>
```



HTTP METHODS:-

1. HTTP GET

- The HTTP GET method requests data from a server without altering its state. It appends parameters to the URL, making it suitable for retrieving non-sensitive data. Commonly used for viewing content, GET is ideal for requests that don't involve data modification.
- **Example:** In the following HTML code we have created a form with text fields such as Username and City. we have also included a PHP file *getmethod.php* where our data would be sent after we click the submit button.

```
<!DOCTYPE html>
<html>

<body>
    <form action="getmethod.php" method="GET">
        Username:
          <input type="text" name="username" /> <br>
        City:
          <input type="text" name="city" /> <br>
        <input type="submit" />
    </form>
</body>

</html>
```

2. HTTP POST

- The HTTP POST method sends data from the client to the server to create or update resources, storing data in the request body. It's suitable for secure data transfer, like images or documents, with security relying on encryption (HTTPS), authentication, and validation.

- **Example:** In the following HTML code we have created a form with text field as Username and Area of study. we have also included a PHP file postmethod.php, where our data would be sent after we click the submit button.

```
<!DOCTYPE html>
<html>

<body>
    <form action="postmethod.php" method="post">
        Username:
          <input type="text" name="username" /> <br>
        Area of Study:
          <input type="text" name="area" /> <br>

        <input type="submit" />
    </form>
</body>

</html>
```

Difference between HTTP GET and HTTP POST

HTTP GET	HTTP POST
In GET method we can not send large amount of data rather limited data of some number of characters is sent because the request parameter is appended into the URL.	In POST method large amount of data can be sent because the request parameter is appended into the body.
GET request is comparatively better than Post so it is used more than the Post request.	POST request is comparatively less better than Get method, so it is used less than the Get request.
GET requests are only used to request data (not modify)	POST requests can be used to create and modify data.
GET request is comparatively less secure because the data is exposed in the URL bar.	POST request is comparatively more secure because the data is not exposed in the URL bar.
Request made through GET method are stored in Browser history.	Request made through POST method is not stored in Browser history.
GET method request can be saved as bookmark in browser.	POST method request can not be saved as bookmark in browser.
Request made through GET method are stored in cache memory of Browser.	Request made through POST method are not stored in cache memory of Browser.
In GET method only ASCII characters are allowed.	In POST method all types of data is allowed.

HTML Links

- **HTML Links**, also known as **hyperlinks**, are defined by the **<a>** tag in HTML, which stands for “anchor.” These links are essential for navigating between web pages and directing users to different sites, documents, or sections within the same page.
- The basic attributes of the <a> tag include **href**, **title**, and **target**, among others.

Basic Syntax of an HTML Link:

```
<a href="https://www.example.com">Visit Example</a>
```

- **Note:** A hyperlink can be represented by an image or any other HTML element, not just text.

1.1. Basic Text Link

```html
<a href="https://www.google.com">Visit Google</a>
```

- Clicking this link opens Google.

1.2. Open Link in a New Tab

```html
<a href="https://www.google.com" target="_blank">Open Google in New Tab</a>
```

- The `target="_blank"` attribute opens the link in a new tab.

1.3. Email Link

```html
<a href="mailto:example@example.com">Send Email</a>
```

- Clicking this opens the default email client.

1.4. Download Link

```html
<a href="file.pdf" download>Download PDF</a>
```

- The `download` attribute allows users to download the file instead of opening it.

HTML Media Elements

2.1. Images (`<img>`)

```html
<img src="image.jpg" alt="Description" width="300">
```

- The `alt` attribute improves accessibility.

2.2. Audio (`<audio>`)

```html
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    Your browser does not support audio.
</audio>
```

- The `controls` attribute adds play/pause buttons.

2.3. Video (`<video>`)

```html
<video width="400" controls>
    <source src="video.mp4" type="video/mp4">
    Your browser does not support video.
</video>
```

- Similar to `<audio>`, the `<video>` tag allows embedding video files.

HTML5 Graphics

- HTML5 introduced two powerful elements for graphics: **Canvas** and **SVG**.
- **Canvas (<canvas>)**
    - Used for **dynamic** graphics (like animations and games).
    - Requires **JavaScript** for drawing.
- **Scalable Vector Graphics (SVG)**

- **XML-based** graphics (better for **static images** like icons and logos).
- Scales without losing quality.

EXAMPLE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>HTML5 Graphics: Canvas & SVG</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         text-align: center;
         margin: 20px;
      }
      canvas {
         border: 2px solid black;
         margin-top: 10px;
      }
      svg {
         border: 2px solid black;
         margin-top: 10px;
      }
   </style>
```

```
</head>
<body>

    <h1>HTML5 Graphics: Canvas & SVG</h1>

    <!-- Canvas Example -->
    <h2>Canvas Drawing</h2>
    <canvas id="myCanvas" width="400" height="200"></canvas>

    <script>
        // Get the canvas element
        var canvas = document.getElementById("myCanvas");
        var ctx = canvas.getContext("2d");

        // Draw a blue rectangle
        ctx.fillStyle = "blue";
        ctx.fillRect(50, 50, 100, 100);

        // Draw a red circle
        ctx.beginPath();
        ctx.arc(250, 100, 50, 0, Math.PI * 2);
        ctx.fillStyle = "red";
        ctx.fill();
        ctx.stroke();
```

```
        // Draw a line
        ctx.beginPath();
        ctx.moveTo(20, 180);
        ctx.lineTo(380, 180);
        ctx.strokeStyle = "green";
        ctx.lineWidth = 3;
        ctx.stroke();
    </script>

    <!-- SVG Example -->
    <h2>SVG Graphics</h2>
    <svg width="400" height="200">
        <!-- Draw a rectangle -->
        <rect x="50" y="50" width="100" height="100" fill="blue" stroke="black" stroke-width="2"/>
        <!-- Draw a circle -->
        <circle cx="250" cy="100" r="50" fill="red" stroke="black" stroke-width="2"/>

        <!-- Draw a line -->
        <line x1="20" y1="180" x2="380" y2="180" stroke="green" stroke-width="3"/>

        <!-- Add text -->
        <text x="150" y="30" fill="black" font-size="18">SVG Example</text>
    </svg>
```

```
</body>

</html>
```

Key Differences Between Canvas & SVG

Feature	Canvas	SVG
Type	Raster-based	Vector-based
Requires JavaScript?	✅ Yes	❌ No
Scalability	❌ Loses quality when resized	✅ Scales without loss
Interactivity	✅ Can be manipulated dynamically	✅ Supports CSS & events
Use Case	Animations, games, complex graphics	Logos, charts, static graphics

XHTML: Syntax and Semantics

Introduction to XHTML

- XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage.
- It is the next step in the evolution of the internet.
- The XHTML 1.0 is the first document type in the XHTML family.
- XHTML was developed by World Wide Web Consortium (W3C) to help web developers make the transition from HTML to XML.
- By migrating to XHTML today, web developers can enter the XML world with all of its benefits, while still remaining confident in the backward and future compatibility of the content.
- Designed to improve structure, consistency, and compatibility with XML-based technologies.
- Ensures better document validation and accessibility

Syntax of XHTML

- XHTML syntax is very similar to HTML syntax and almost all the valid HTML elements are valid in XHTML as well. But when you write an XHTML document, you need to pay a bit extra attention to make your HTML document compliant to XHTML.
- XHTML is case sensitive markup language. All the XHTML tags and attributes need to be written in lower case only.

Basic Document Structure

An XHTML document must have:

- A **DOCTYPE declaration** (defines the document type)
- An **XML namespace declaration** (`xmlns="http://www.w3.org/1999/xhtml"`)
- A properly structured **`<html>`** root element

Important points to remember while writing a new XHTML document or converting existing HTML document into XHTML document

- Write a DOCTYPE declaration at the start of the XHTML document.
- Write all XHTML tags and attributes in lower case only.
- Close all XHTML tags properly.
- Nest all the tags properly.
- Quote all the attribute values.
- Forbid Attribute minimization.
- Replace the **name** attribute with the **id** attribute.
- Deprecate the **language** attribute of the script tag.

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
    <title>XHTML Example</title>
</head>
<body>
    <p>Hello, World!</p>
</body>
</html>
```

XHTML Syntax Rules

1. **Lowercase Tags:** All elements must be written in lowercase (`<p>` instead of `<P>`).
2. **Proper Nesting:** Elements must be properly nested (`<b><i>Text</i></b>` is correct, `<b><i>Text</b></i>` is incorrect).
3. **Closed Tags:** All elements must be explicitly closed (`<br />` instead of `<br>`).
4. **Attribute Quotation:** Attribute values must be enclosed in double or single quotes (`alt="Example"`).
5. **No Attribute Minimization:** Boolean attributes must have explicit values (`checked="checked"` instead of just `checked`).

Advantages of XHTML

1. **XML Compliance** – XHTML follows XML standards, making it easy to view, edit, and validate.
2. **Better Browser Compatibility** – Works efficiently in both old and new browsers.
3. **Supports Scripts & Applets** – Integrates with HTML and XML DOM-based applications.
4. **Consistent & Structured** – Ensures well-structured webpages for better parsing.
5. **Easy Maintenance** – Simplifies editing, conversion, and formatting.
6. **W3C Standard Compliance** – Enhances browser compatibility and accurate rendering.
7. **HTML & XML Combined** – Merges HTML's flexibility with XML's strictness.
8. **Quality Certification** – Meets W3C standards for high-quality webpages

. Key XHTML Syntax Differences from HTML

Feature	HTML	XHTML
Closing Tags	Optional for some elements	Required for all elements
Self-closing Tags	` ` `<img>`	` ` `<img />`
Attribute Quotes	Optional	Required (`alt="text"`)
Lowercase Tags	Optional	Required (`<h1>` , `<p>`)
Proper Nesting	Not Strict	Strict (`<b><i>Text</i></b>`)

Benefits of XHTML Syntax

- Ensures **well-structured** and **error-free** documents.
- Improves **compatibility** with future web technologies.
- Works seamlessly with **XML tools and parsers**.

Semantics in XHTML

Importance of Semantics

- **Semantics** refers to the meaning of elements and their correct use in document structure.
- Improves readability, accessibility, and SEO (Search Engine Optimization).
- Helps screen readers and assistive technologies interpret content correctly.

Meaningful Element Usage

1. **Headings (`<h1>` - `<h6>`)**: Define document hierarchy and structure.
2. **Paragraphs (`<p>`)**: Represent blocks of text.
3. **Lists (`<ul>`, `<ol>`, `<li>`)**: Organize related items.
4. **Tables (`<table>`, `<tr>`, `<td>`)**: Display tabular data, not for layout.
5. **Forms (`<form>`, `<input>`, `<label>`)**: Collect user input in a structured way.

Separation of Content and Presentation

- XHTML encourages the use of **CSS** for styling instead of presentational elements like `<font>` or `<center>`.
- Structure should be handled using **semantic HTML elements** rather than relying on tables for layout.

Accessibility and Compatibility

- Semantic elements improve web accessibility for screen readers and search engines.
- XHTML is compatible with other XML-based technologies like **SVG**, **XSLT**, and **MathML**.

XML: DTD, XML schemes, presenting and using XML DOM, XSL and XSLT Transformation

XML (eXtensible Markup Language) is a versatile format used for representing structured data. It is widely used in web services, configuration files, data storage, and communication. In this lecture, we will cover the following topics related to XML:

1. **DTD (Document Type Definition)**
2. **XML Schemas**
3. **Using XML DOM (Document Object Model)**
4. **XSL (eXtensible Stylesheet Language) and XSLT (eXtensible Stylesheet Language Transformations)**

---

1. DTD (Document Type Definition)

A **Document Type Definition (DTD)** defines the structure and the legal elements and attributes in an XML document. It serves as a blueprint or rulebook that specifies what elements are allowed and how they should be organized.

Key Points:

- DTD can be defined **inline** within the XML document or **externally** as a separate file.
- DTD defines **elements**, **attributes**, **entities**, and **notations**.

Example of DTD:

```
<!DOCTYPE note [
    <!ELEMENT note (to, from, heading, body)>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>
]>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

DTD Types:

- **Internal DTD**: Defined inside the XML document.
- **External DTD**: Referenced from an external file.

Example of External DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Tove</to>
```

```
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

---

2. XML Schemas

An **XML Schema** (XSD - XML Schema Definition) provides a more robust and feature-rich alternative to DTDs. It defines the structure, content, and data types of XML documents in a more precise and machine-readable way.

Key Features of XML Schema:

- Allows specifying **data types** (e.g., string, integer, date).
- Supports **namespaces**, making it easier to use XML in different contexts.
- **Better validation**: Can enforce more strict rules than DTD.

Example of XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example of an XML Document Validated by the Above Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Key Differences between DTD and XML Schema:

- XML Schema supports **data types**; DTD does not.
- XML Schema supports **namespaces**; DTD does not.
- XML Schema allows **more complex data structures**, such as sequences, choices, and restrictions.

---

3. Using XML DOM (Document Object Model)

The **Document Object Model (DOM)** is a programming interface for XML documents. It represents the document as a tree structure, where each node represents an element, attribute, or piece of data.

DOM Tree:

- **Document**: Root of the tree.
- **Element**: Represents tags in the XML.
- **Attribute**: Represents the attributes of an element.
- **Text**: Represents the text content inside an element.

Key Operations with DOM:

- **Loading XML** into memory.
- **Navigating** through the document.
- **Manipulating** elements (add, delete, modify).
- **Saving** the modified document back to XML.

Example (JavaScript):

```
let xmlString =
`<note><to>Tove</to><from>Jani</from><heading>Reminder</heading><body>Don't
forget me this weekend!</body></note>`;
let parser = new DOMParser();
let xmlDoc = parser.parseFromString(xmlString, "text/xml");

// Accessing elements
let toElement = xmlDoc.getElementsByTagName("to")[0];
console.log(toElement.textContent);  // Output: Tove
```

Benefits of Using DOM:

- Works on the entire XML document.
- Easy to traverse and manipulate.
- Provides a standard interface for all programming languages (e.g., JavaScript, Python).

---

4. XSL and XSLT Transformation

XSL (eXtensible Stylesheet Language) is a language used for transforming and rendering XML documents into different formats (e.g., HTML, PDF, plain text).

XSL consists of:

1. **XSLT (eXtensible Stylesheet Language Transformations)**: A language for transforming XML documents into other XML documents or formats.
2. **XPath**: A language used for navigating through elements and attributes in an XML document.

Key Concepts of XSLT:

- **Template Rules**: Define how elements in the source XML should be transformed.
- **XPath Expressions**: Used to select parts of the XML document to apply the transformation.

Example of XSLT Transformation:

XML Source:

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSLT Stylesheet:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/note">
    <html>
      <body>
        <h2>Note</h2>
        <p><strong>To:</strong> <xsl:value-of select="to"/></p>
        <p><strong>From:</strong> <xsl:value-of select="from"/></p>
        <p><strong>Heading:</strong> <xsl:value-of select="heading"/></p>
        <p><strong>Body:</strong> <xsl:value-of select="body"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Resulting HTML:

```
<html>
  <body>
    <h2>Note</h2>
    <p><strong>To:</strong> Tove</p>
    <p><strong>From:</strong> Jani</p>
    <p><strong>Heading:</strong> Reminder</p>
    <p><strong>Body:</strong> Don't forget me this weekend!</p>
  </body>
</html>
```

XSLT Process:

- **Input**: XML Document.
- **XSLT Stylesheet**: Defines how to transform the XML.
- **Output**: Transformed document (HTML, plain text, etc.).

Introduction to CSS

CSS (Cascading Style Sheets) is a style sheet language used to describe the presentation of a document written in HTML. It controls a webpage's layout, colors, fonts, and overall appearance.

Advantages of CSS

- **Separation of Content and Design**: HTML structures the content, while CSS handles the styling.
- **Reusability**: One CSS file can style multiple pages.
- **Improved Website Performance**: Reduces the need for inline styles, leading to faster load times.
- **Responsive Design**: Helps make web pages look good on different devices.

CSS Syntax

CSS follows a rule-based syntax:

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

Example:

```
h1 {
    color: blue;
}
```

This rule sets all `<h1>` elements to have a blue color.

CSS Color

Colors in CSS can be defined using different formats:

- **Named colors**: `red`, `blue`, `green`.
- **Hex codes**: `#ff0000` (red), `#00ff00` (green).
- **RGB values**: `rgb(255, 0, 0)`.
- **RGBA values**: `rgba(255, 0, 0, 0.5)` (adds transparency).
- **HSL values**: `hsl(0, 100%, 50%)`.

Example:

```
p {
    color: #ff6600;
}
```

CSS Background

CSS allows setting background properties for elements:

- **Background color**: `background-color: yellow;`
- **Background image**: `background-image: url('image.jpg');`
- **Background size**: `background-size: cover;`
- **Background position**: `background-position: center;`
- **Background repeat**: `background-repeat: no-repeat;`

Example:

```
body {
    background-color: lightgray;
    background-image: url('background.jpg');
    background-size: cover;
    background-repeat: no-repeat;
}
```

CSS Fonts

CSS provides properties to style fonts:

- **Font family**: `font-family: Arial, sans-serif;`
- **Font size**: `font-size: 16px;`
- **Font weight**: `font-weight: bold;`
- **Font style**: `font-style: italic;`
- **Text alignment**: `text-align: center;`

Example:

```
h1 {
    font-family: 'Courier New', monospace;
    font-size: 24px;
    font-weight: bold;
}
```

CSS Images

CSS allows styling images using various properties:

- **Image width and height**: `width: 200px; height: auto;`
- **Border radius**: `border-radius: 50%;` (makes images circular)
- **Opacity**: `opacity: 0.5;`
- **Box shadow**: `box-shadow: 5px 5px 10px gray;`

Example:

```
img {
    width: 300px;
    border-radius: 10px;
    box-shadow: 5px 5px 10px rgba(0, 0, 0, 0.5);
}
```

CSS Links

Links (`<a>` elements) can be styled using different states:

- **Normal link**: `a { color: blue; }`
- **Visited link**: `a:visited { color: purple; }`
- **Hover effect**: `a:hover { color: red; text-decoration: underline; }`
- **Active state**: `a:active { color: green; }`

Example:

```
a {
    color: blue;
    text-decoration: none;
}

a:hover {
    color: red;
    text-decoration: underline;
}
```

Try this activity:

Apply different CSS properties and code the styles.css for the HTML code given below.

- Change the background color of the page.
- Set a font style and size for the `<h1>` and `<p>` elements.
- Add a border radius to the image.
- Change the color of the link and add a hover effect.

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First Styled Page</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a paragraph of text that needs styling.</p>
  <a href="#">Click here to learn more</a>
  <img src="example.jpg" alt="Sample Image">
</body>
</html>
```

CSS: Tables, Margins, and Lists

1. CSS Tables

Tables in HTML (`<table>`) can be styled using CSS to control appearance, spacing, and layout.

1.1 Table Properties

Property	Description
`border`	Sets the border around table elements.
`border-collapse`	Defines whether table borders collapse into a single border or remain separate.
`width` / `height`	Sets dimensions of the table.
`padding`	Adds space inside table cells.
`text-align`	Aligns text within table cells.
`vertical-align`	Aligns content within a table cell vertically.

1.2 Example: Basic Table Styling

```
table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    border: 1px solid black;
    padding: 10px;
    text-align: center;
}

th {
    background-color: #f2f2f2;
}
```

---

2. CSS Margins

Margins control the spacing around elements, affecting their positioning relative to other elements.

2.1 Margin Properties

Property	Description
`margin`	Sets margin on all sides of an element.
`margin-top` / `margin-bottom`	Controls margin above and below an element.

Property	Description
`margin-left` / `margin-right`	Controls margin on the left and right.
`auto`	Centers an element horizontally if width is defined.

2.2 Example: Setting Margins

```
.box {
    width: 200px;
    height: 100px;
    margin: 20px auto; /* 20px margin on top/bottom, auto centers
horizontally */
    background-color: lightblue;
}
```

---

3. CSS Lists

Lists (`<ul>`, `<ol>`, `<li>`) can be styled for appearance, spacing, and layout.

3.1 List Properties

Property	Description
`list-style-type`	Defines the type of bullet (e.g., circle, square, none).
`list-style-position`	Controls if bullets appear inside or outside the list item.
`list-style-image`	Uses a custom image as a bullet.
`padding`	Controls spacing between list items and their content.

3.2 Example: Styling Lists

```
ul {
    list-style-type: square;
    padding-left: 20px;
}

ol {
    list-style-type: decimal;
}

ul.no-bullets {
    list-style-type: none;
}
```

3.3 Custom List Example

```
ul.custom {
    list-style-image: url('bullet.png');
}
```

Summary

- **Tables**: Use `border-collapse`, `padding`, and `text-align` for structure.
- **Margins**: Control spacing with `margin` and use `auto` for centering.
- **Lists**: Customize bullet points with `list-style-type` and images.

CSS: Border, Padding, Scroll, and CSS Classes

1. CSS Border

Borders define the outline of an element and can be styled with different properties.

1.1 Border Properties

Property	Description	Example
`border`	Sets the border width, style, and color.	`border: 2px solid black;`
`border-width`	Defines the thickness of the border.	`border-width: 5px;`
`border-style`	Specifies the style (solid, dashed, dotted, etc.).	`border-style: dashed;`
`border-color`	Sets the color of the border.	`border-color: red;`
`border-radius`	Rounds the corners of an element.	`border-radius: 10px;`

1.2 Example: Border Styling

```
.box {
    border: 3px solid blue;
    border-radius: 10px;
    padding: 10px;
}
```

2. CSS Padding

Padding is the space between the content and the border of an element.

2.1 Padding Properties

Property	Description	Example
`padding`	Sets padding on all sides.	`padding: 20px;`
`padding-top`	Adds padding on the top.	`padding-top: 10px;`
`padding-right`	Adds padding on the right.	`padding-right: 15px;`
`padding-bottom`	Adds padding at the bottom.	`padding-bottom: 20px;`
`padding-left`	Adds padding on the left.	`padding-left: 5px;`

2.2 Example: Padding in Use

```
.content-box {
    border: 2px solid black;
    padding: 20px;
    background-color: lightgray;
}
```

3. CSS Scroll (Overflow Handling)

Scroll behavior is controlled using the `overflow` property.

3.1 Scroll Properties

Property	Description	Example
`overflow`	Controls scrolling behavior.	`overflow: auto;`
`overflow-x`	Sets horizontal scrolling.	`overflow-x: scroll;`
`overflow-y`	Sets vertical scrolling.	`overflow-y: hidden;`
`scroll-behavior`	Defines smooth scrolling.	`scroll-behavior: smooth;`

3.2 Example: Scrollable Box

```
.scroll-box {
    width: 300px;
    height: 100px;
    border: 1px solid black;
    overflow-y: scroll;
}
<div class="scroll-box">
    <p>This is a long text that will require scrolling to view
completely.</p>
</div>
```

4. CSS Class

A CSS class is used to style multiple elements with the same properties.

4.1 Using CSS Classes

- Defined with a `.` (dot) before the class name.
- Applied using the `class` attribute in HTML.

4.2 Example: Defining and Using a Class

```
.highlight {
    color: white;
    background-color: blue;
    padding: 10px;
    border-radius: 5px;
}
<p class="highlight">This is highlighted text.</p>
```

Summary

- **Borders** outline elements and can be styled in different ways.
- **Padding** creates space inside an element between content and border.
- **Scroll properties** control how elements handle overflow.
- **CSS Classes** allow reuse of styling across multiple elements.

L12: CSS3 Border Image, Round Corner, Text Shadow, and Layers
(CSS3 Advanced Styling Techniques)

1. Border Image

CSS3 introduced the `border-image` property, allowing developers to use images as borders instead of solid colors or gradients.

1.1 Properties of `border-image`

The `border-image` property is a shorthand for setting:

- `border-image-source` – Specifies the image to use as a border.
- `border-image-slice` – Defines how to slice the image.
- `border-image-width` – Specifies the width of the border.
- `border-image-outset` – Defines how far the border image extends beyond the border box.
- `border-image-repeat` – Specifies how the border images are repeated (stretch, repeat, round, space).

1.2 Syntax

```
element {
    border: 10px solid transparent; /* Required for border-image */
    border-image-source: url('border.png');
    border-image-slice: 30; /* Cut 30px from each side */
    border-image-width: 10px;
    border-image-outset: 5px;
    border-image-repeat: round;
}
```

1.3 Example

```
.box {
    border: 20px solid transparent;
    border-image: url('border.png') 30 round;
}
```

2. CSS3 Rounded Corners

- CSS3 introduced `border-radius`, which allows for smooth, rounded corners.
- Can be used with images to create circular profile pictures.
- `border-radius` does not affect the box model; only appearance changes.

2.1 Properties of `border-radius`

- `border-radius` – Sets the radius of all four corners.
- Individual properties:
    - `border-top-left-radius`
    - `border-top-right-radius`
    - `border-bottom-left-radius`
    - `border-bottom-right-radius`

2.2 Syntax

```
element {
    border-radius: 10px; /* Uniformly rounded corners */
}
```

2.3 Example

```
.box {
    width: 200px;
    height: 200px;
    background: blue;
    border-radius: 15px;
}
```

2.4 Elliptical Borders

```
.box {
    border-radius: 50px 25px; /* Horizontal and vertical radii */
}
```

2.5 Perfect Circle

```
.circle {
    width: 100px;
    height: 100px;
    background: red;
    border-radius: 50%;
}
```

3. CSS3 Text Shadow

- The `text-shadow` property adds shadow effects to text.
- Can be used to create glowing, embossed, or neon text effects.
- Multiple shadows can be stacked for more complex effects.

3.1 Syntax

```
element {
    text-shadow: h-offset v-offset blur-radius color;
}
```

- `h-offset` – Horizontal shadow position (positive = right, negative = left).
- `v-offset` – Vertical shadow position (positive = down, negative = up).
- `blur-radius` – Defines how much the shadow should blur (optional).
- `color` – Specifies the shadow color.

3.2 Example

```
h1 {
    text-shadow: 3px 3px 5px rgba(0, 0, 0, 0.5);
}
```

3.3 Multiple Shadows

```
h1 {
    text-shadow: 2px 2px 5px red, -2px -2px 5px blue;
}
```

4. CSS3 Layers (Z-Index and Stacking Context)

Layers control how elements are displayed relative to each other.

4.1 The `z-index` Property

Defines the stack order of elements.

Syntax

```
element {
    position: relative; /* Or absolute, fixed, sticky */
    z-index: value;
}
```

- `z-index: auto` (default) – Follows the natural document flow.
- Higher values appear on top of lower values.

Example

```
.box1 {
    position: absolute;
    z-index: 1;
    background: red;
}
.box2 {
    position: absolute;
    z-index: 2;
    background: blue;
}
```

4.2 Stacking Context

- Created when an element has `z-index` other than `auto` and a positioned parent.
- Nested elements can create new stacking contexts.

4.3 CSS Layering Example

```
.parent {
    position: relative;
}
.child1 {
    position: absolute;
    z-index: 5;
}
.child2 {
    position: absolute;
    z-index: 10; /* Will appear on top */
}
```

- `z-index` only works on elements with `position: relative | absolute | fixed | sticky`.
- Elements with higher `z-index` will be placed above lower ones.

CSS3 properties enhance UI design, improve user experience, and allow creative effects.

Feature	Description
`border-image`	Allows using an image as a border.
`border-radius`	Rounds the corners of an element.
`text-shadow`	Adds shadow effects to text.
`z-index`	Controls stacking order of elements.

Suggested activity:

"Design a stylish profile card using CSS3 properties like border-image, border-radius, text-shadow, and z-index to enhance its visual appeal."

Activity Instructions:

1. **Create a profile card** using `div` elements.
2. Use **border-radius** to give the card rounded corners.
3. Apply a **border-image** to create a unique border style.
4. Use **text-shadow** to enhance the text appearance.
5. Utilize **z-index** to position elements creatively within the card.
6. **Try different `border-image` styles** using various images.
7. **Experiment with multiple `text-shadow` layers** for a glowing effect.
8. **Modify the `z-index`** to place text over the image creatively.
9. **Enhance the button with gradient effects and hover animations**

Sample code for modification:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>CSS3 Profile Card</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         display: flex;
         justify-content: center;
         align-items: center;
         height: 100vh;
         background: #f4f4f4;
      }

      .card {
         width: 200px;
         height: 200px;
         background: white;
         border-radius: 15px;
         text-align: center;
         padding: 20px;
         position: relative;
         border: 10px solid transparent;
         border-image: url('border.png') 30 round; /* Use a border image */
         box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.2);
      }
      .profile-img {
         width: 100px;
         height: 100px;
```

```
            border-radius: 50%;
            border: 4px solid white;
            position: absolute;
            top: -50px;
            left: 50%;
            transform: translateX(-50%);
            z-index: 2;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
        }
        .name {
            margin-top: 60px;
            font-size: 22px;
            font-weight: bold;
            text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
        }

        .description {
            font-size: 14px;
            color: #555;
            margin-top: 10px;
        }
        .btn {
            display: inline-block;
            margin-top: 20px;
            padding: 10px 20px;
            background: #007BFF;
            color: white;
            text-decoration: none;
            border-radius: 10px;
            box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);
            transition: background 0.3s ease;
        }
        .btn:hover {
            background: #0056b3;
        }
    </style>
</head>
<body>

    <div class="card">
        <img src="profile.jpg" alt="Profile Picture" class="profile-img">
        <div class="name">Prabaharan S</div>
        <div class="description">Web Developer | UI Designer</div>
        <a href="#" class="btn">Follow</a>
    </div>

</body>
</html>
```

Lecture 13: Bootstrap 3 - Making Grids for Mobile-First Responsive Web Applications

1. Introduction to Bootstrap 3

Bootstrap 3 is a **front-end framework** that helps developers create **responsive, mobile-first web applications** efficiently. It includes **a flexible grid system, responsive utilities, components, and JavaScript plugins** to design web pages that adapt to different screen sizes.

Why Bootstrap 3?

✅ **Mobile-First Approach** – Designed with small screens in mind first, then scales up for larger screens.
✅ **Grid System** – A powerful and flexible layout system.
✅ **Predefined CSS Classes** – Simplifies responsive design without custom CSS.
✅ **Cross-Browser Compatibility** – Works on all modern browsers.

2. Bootstrap 3 Grid System Overview

What is the Grid System?

The **Bootstrap 3 grid system** uses a **12-column layout** that allows easy arrangement of content in rows and columns. It ensures a responsive and flexible design across devices.

Grid Classes:

Class	Screen Size	Min Width
`.col-xs-`	Extra Small	<768px (Phones)
`.col-sm-`	Small	≥768px (Tablets)
`.col-md-`	Medium	≥992px (Laptops)
`.col-lg-`	Large	≥1200px (Desktops)

📌 **Mobile-First**: Bootstrap 3 starts with `col-xs-*` for mobile devices and progressively enhances for larger screens.

3. Creating a Responsive Grid Layout

3.1 Basic Grid Structure

A Bootstrap grid layout consists of:
✅ **Container** (`.container` or `.container-fluid`) – Wraps the grid.
✅ **Row** (`.row`) – Divides the layout horizontally.
✅ **Columns** (`.col-*`) – Defines content width in 12-column grid.

Example: Simple Responsive Grid

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 3 Grid Example</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
</head>
<body>

<div class="container">
    <div class="row">
        <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
            <div class="well">Column 1</div>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
            <div class="well">Column 2</div>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
            <div class="well">Column 3</div>
        </div>
        <div class="col-xs-12 col-sm-6 col-md-12 col-lg-3">
            <div class="well">Column 4</div>
        </div>
    </div>
</div>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</body>
</html>
```

Explanation:

✅ `.col-xs-12` – Columns take full width on extra small screens.
✅ `.col-sm-6` – Two columns per row on small screens.
✅ `.col-md-4` – Three columns per row on medium screens.
✅ `.col-lg-3` – Four columns per row on large screens.

📌 **Result:** Layout adjusts dynamically across different devices.

4. Grid Nesting (Grids Inside Grids)

Bootstrap allows **nesting grids**, meaning you can place a row inside a column.

Example: Nested Grid

```
<div class="container">
    <div class="row">
        <div class="col-md-6">
            <div class="well">Parent Column</div>
            <div class="row">
                <div class="col-xs-6">
                    <div class="well">Nested Column 1</div>
                </div>
                <div class="col-xs-6">
                    <div class="well">Nested Column 2</div>
                </div>
            </div>
        </div>
    </div>
</div>
```

📌 **Result:** The **nested columns** divide only inside their parent column.

5. Grid Offsets & Push/Pull for Alignment

Offsets and push/pull help adjust column alignment.

Example: Offsetting Columns

```
<div class="container">
    <div class="row">
        <div class="col-md-4 col-md-offset-2">
            <div class="well">Offset Column</div>
        </div>
    </div>
</div>
```

📌 **Result:** Moves the column right by **2 grid spaces**.

Example: Column Push & Pull

```
<div class="container">
    <div class="row">
        <div class="col-md-4 col-md-push-8">
            <div class="well">Pushed Column</div>
        </div>
        <div class="col-md-8 col-md-pull-4">
            <div class="well">Pulled Column</div>
        </div>
    </div>
</div>
```

📌 **Result:** Changes column order while maintaining responsiveness.

6. Responsive Utilities & Hidden Classes

Bootstrap provides classes to show/hide content based on screen sizes.

Class	Hidden On
`.hidden-xs`	Extra small screens (Phones)
`.hidden-sm`	Small screens (Tablets)
`.hidden-md`	Medium screens (Laptops)
`.hidden-lg`	Large screens (Desktops)

Example: Show/Hide Elements

```
<p class="hidden-xs">This text is hidden on extra-small screens.</p>
<p class="hidden-lg">This text is hidden on large screens.</p>
```

📌 **Result:** Content appears/disappears based on screen width.

7. Summary of Key Concepts

Feature	Description
12-Column Grid	Divides layout into a flexible 12-column structure.
Mobile-First Design	Uses `.col-xs-*` as default and scales up.
Nesting	Allows grids inside grids for advanced layouts.
Offset & Push/Pull	Adjusts positioning and order of columns.
Responsive Utilities	Show/hide content for different screen sizes.

---

8. Summary

- Bootstrap 3 **simplifies responsive web design** using its **grid system**.
- It provides **predefined CSS classes** to handle different screen sizes.
- The **mobile-first approach** ensures a seamless experience across devices.

📌 Homework/Practice

1. **Create a Bootstrap 3 webpage** with a **3-column layout** that adjusts to different screen sizes.
2. **Experiment with offsets, push/pull, and hidden classes** to see their effects.
3. **Convert an existing non-responsive layout** into a Bootstrap 3 responsive grid system.

Activity: Build a Responsive Portfolio Layout Using Bootstrap 3

🔹 Problem Statement:

"Create a simple portfolio webpage using Bootstrap 3's grid system. The layout should adjust to different screen sizes: full-width on mobile, two columns on tablets, and three columns on desktops."

🔹 Activity Instructions:

1. Use **Bootstrap 3** to create a **responsive portfolio** with at least **6 project thumbnails**.
2. Follow this grid behavior:
   - **Extra-small screens (`xs`)** → Each project takes **full width** (1 per row).
   - **Small screens (`sm`)** → Two projects per row (6 columns each).
   - **Medium screens (`md`) and larger** → Three projects per row (4 columns each).
3. Add **Bootstrap's `.container`, `.row`, and `.col-*` classes** to structure the layout.
4. Each project should be inside a **Bootstrap card or a simple div with styling**.

🔹 Expected Solution (Bootstrap 3 Code)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Character encoding set to UTF-8 for universal character support -->
    <meta charset="UTF-8">

    <!-- Viewport meta tag for responsive design on mobile devices -->
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Title displayed in the browser tab -->
    <title>Bootstrap 3 Responsive Portfolio</title>

    <!-- Linking to Bootstrap 3 CSS for responsive grid and styling -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

    <!-- Custom CSS to style the portfolio items -->
    <style>
        .portfolio-item {
            background-color: #f8f9fa;  /* Light background for contrast */
            padding: 15px;              /* Spacing inside each box */
            margin-bottom: 15px;        /* Spacing below each item */
            text-align: center;         /* Center-align text inside item */
```

```
            border: 1px solid #ddd;     /* Light border around the box */
        }
    </style>
</head>
<body>

<!-- Main container to center content and provide horizontal padding -->
<div class="container">
    <!-- Centered heading for the portfolio -->
    <h2 class="text-center">My Portfolio</h2>

    <!-- Bootstrap row to group the columns (projects) -->
    <div class="row">
        <!-- Responsive column: full width on extra small, half on small, one-third on medium+
-->
        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 1</div>
        </div>

        <!-- Repeat of the structure for each portfolio item -->
        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 2</div>
        </div>

        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 3</div>
        </div>

        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 4</div>
        </div>

        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 5</div>
        </div>

        <div class="col-xs-12 col-sm-6 col-md-4">
            <div class="portfolio-item">Project 6</div>
        </div>
    </div> <!-- End of row -->
</div> <!-- End of container -->

<!-- jQuery library required for Bootstrap's JavaScript plugins -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<!-- Bootstrap 3 JavaScript for responsive behavior and components -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</body>
</html>
```

🔹 Explanation of the Solution

✅ `<div class="container">` – Wraps the content to provide padding and centering.
✅ `<div class="row">` – Defines a horizontal layout section.
✅ `<div class="col-xs-12 col-sm-6 col-md-4">` – Adjusts the number of columns:

- **xs (extra-small screens, <768px)** → Full width (1 project per row).
- **sm (small screens, ≥768px)** → Two projects per row.
- **md (medium screens, ≥992px)** → Three projects per row.
  ✅ `.portfolio-item` **class** – Adds basic styling to project boxes.

🔹 Expected Output (Layout Adjustments Based on Screen Size)

Screen Size	Layout
Phones (xs)	1 column (Full width for each project)
Tablets (sm)	2 columns per row
Laptops (md) & Desktops (lg)	3 columns per row

🔹 Bonus Challenge (Optional Enhancements)

💡 **Add images to each project** instead of plain text.
💡 **Use Bootstrap's card component** instead of simple divs.
💡 **Apply hover effects with CSS** to highlight projects.
💡 **Use Bootstrap's navbar** to create a header section.

🎯 Learning Outcomes

✓ Understanding **Bootstrap's 12-column grid system**.
✓ Building **mobile-first responsive layouts**.
✓ Using **Bootstrap's `.container`, `.row`, and `.col-*` classes** effectively.
✓ Applying **basic styling** with Bootstrap utilities.

This activity provides **hands-on experience** with Bootstrap 3's grid system, reinforcing the concept of **mobile-first design**! 🚀

Here are **detailed lecture notes** on **Bootstrap 3 – Designing a Navigation Bar in Responsive Applications**, including theory and sample code.

Lecture Notes on Bootstrap 3 – Designing Navigation Bar in Responsive Applications

1. Introduction to Bootstrap 3

What is Bootstrap 3?

- Bootstrap 3 is a **front-end framework** for developing **responsive** and **mobile-first** websites.
- It provides **predefined CSS and JavaScript components**, making web development faster.
- It follows a **grid-based** layout system.

Why Use Bootstrap for Navigation Bars?

- **Pre-styled components** make navbar creation quick and easy.
- **Responsiveness** ensures it adapts to different screen sizes.
- **Collapsible menu** for mobile-friendly navigation.
- **Customizable** through classes and additional styling.

---

2. Bootstrap 3 Navigation Bar (Navbar)

Navbar Features

- The `<nav>` element contains navigation links.
- The **`navbar` class** applies Bootstrap's built-in navbar styles.
- It includes:
  - **Brand Name/Logo**
  - **Navigation Links**
  - **Dropdown Menus**
  - **Search Bar (Optional)**
  - **Hamburger Toggle Button (for small screens)**

Bootstrap 3 Navbar Classes

Class Name	Description
`navbar`	Defines a navigation bar.
`navbar-default`	Applies default Bootstrap styling.

Class Name	Description
`navbar-inverse`	Applies a dark-themed navbar.
`navbar-header`	Contains the brand/logo and toggle button.
`navbar-brand`	Defines the brand name or logo.
`navbar-nav`	Defines the navigation links.
`navbar-toggle`	Creates a collapsible button for mobile views.
`collapse navbar-collapse`	Enables a collapsible menu.

---

3. Building a Basic Responsive Navbar in Bootstrap 3

Step 1: Include Bootstrap 3

Bootstrap 3 can be added using a **CDN** (Content Delivery Network):

```
<!-- Bootstrap 3 CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css
">

<!-- jQuery and Bootstrap JavaScript -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></sc
ript>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"><
/script>
```

Step 2: Create a Basic Responsive Navbar

Here's a simple **responsive** navigation bar using Bootstrap 3:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 3 Navbar</title>

    <!-- Bootstrap 3 CDN -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css
">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></sc
ript>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"><
/script>
</head>
<body>

<!-- Navigation Bar -->
```

```
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <!-- Brand and Toggle for Mobile View -->
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#myNavbar">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">MyBrand</a>
        </div>

        <!-- Navigation Links -->
        <div class="collapse navbar-collapse" id="myNavbar">
            <ul class="nav navbar-nav">
                <li class="active"><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </div>
    </div>
</nav>

<div class="container">
    <h2>Welcome to My Website</h2>
    <p>This is a basic responsive navigation bar using Bootstrap 3.</p>
</div>

</body>
</html>
```

Step 3: Explanation of the Code

1. **Bootstrap and jQuery Included**:
    - The Bootstrap **CSS** and **JavaScript** are loaded via **CDN**.
    - jQuery is required for Bootstrap's JavaScript components.
2. **Navbar Structure**:
    - The `<nav>` element contains the navigation bar.
    - The **class `navbar navbar-default`** applies Bootstrap styling.
3. **Brand and Toggle Button**:
    - The `<div class="navbar-header">` contains:
        - **Brand Name (`navbar-brand`)**
        - **Toggle Button (`navbar-toggle`)** – Used for smaller screens.
4. **Navigation Links (`ul.navbar-nav`)**:
    - Contains `li` elements for different sections of the website.
5. **Collapsible Menu (`collapse navbar-collapse`)**:
    - Allows the navbar to collapse on small screens.

---

4. Adding a Dropdown Menu

Bootstrap 3 also supports **dropdown menus** in the navbar.

Sample Navbar with Dropdown

```
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#myNavbar">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">MyBrand</a>
        </div>

        <div class="collapse navbar-collapse" id="myNavbar">
            <ul class="nav navbar-nav">
                <li class="active"><a href="#">Home</a></li>
                <li><a href="#">About</a></li>

                <!-- Dropdown -->
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown"
href="#">Services <span class="caret"></span></a>
                    <ul class="dropdown-menu">
                        <li><a href="#">Web Design</a></li>
                        <li><a href="#">SEO</a></li>
                        <li><a href="#">Marketing</a></li>
                    </ul>
                </li>

                <li><a href="#">Contact</a></li>
            </ul>
        </div>
    </div>
</nav>
```

Explanation of Dropdown Menu

- **`dropdown` class**: Defines a dropdown menu.
- **`dropdown-toggle`**: Makes the link clickable for expanding the menu.
- **`dropdown-menu`**: Defines the list of dropdown items.

---

5. Customizing the Navbar

If you want to change the navbar color:

```
.navbar-custom {
    background-color: #007bff;
    border-color: #0056b3;
}

.navbar-custom .navbar-brand,
```

```
.navbar-custom .nav > li > a {
    color: white;
}
```

Apply it by replacing `navbar-default` with `navbar-custom` in the `<nav>` element.

---

6. Summary

Feature	Bootstrap 3 Class
Basic Navbar	`navbar navbar-default`
Dark Navbar	`navbar navbar-inverse`
Responsive Toggle Button	`navbar-toggle`
Navigation Links	`nav navbar-nav`
Dropdown Menu	`dropdown-menu`
Collapsible Menu	`collapse navbar-collapse`

---

7. Conclusion

- Bootstrap 3 simplifies **responsive navbar design**.
- It supports **collapsible menus** and **dropdowns**.
- The **grid system** makes navigation adaptable to mobile devices.
- Custom styles can be added for a unique design.

Here's a table summarizing **Bootstrap 3 Navbar Classes** with descriptions and examples:

Bootstrap 3 Navbar Classes Table

Class Name	Description	Example Code
`.navbar`	Defines the main navigation bar container.	`<nav class="navbar"></nav>`
`.navbar-default`	Applies a light-colored default Bootstrap navbar.	`<nav class="navbar navbar-default"></nav>`
`.navbar-inverse`	Applies a dark-colored Bootstrap navbar.	`<nav class="navbar navbar-inverse"></nav>`
`.navbar-header`	Groups the **brand name/logo** and **toggle button** for mobile views.	`<div class="navbar-header"></div>`
`.navbar-brand`	Styles the **brand name or logo** inside the navbar.	`<a class="navbar-brand" href="#">Brand</a>`
`.navbar-nav`	Defines a **list of navigation links** inside the navbar.	`<ul class="nav navbar-nav"><li><a href="#">Home</a></li></ul>`
`.navbar-toggle`	Creates a **hamburger menu button** for small screens.	`<button class="navbar-toggle" data-toggle="collapse" data-target="#menu"></button>`
`.collapse .navbar-collapse`	Allows the navbar to **collapse** on small screens.	`<div class="collapse navbar-collapse" id="menu"></div>`
`.active`	Highlights the **active navigation link**.	`<li class="active"><a href="#">Home</a></li>`
`.navbar-fixed-top`	Fixes the navbar **at the top** of the page.	`<nav class="navbar navbar-default navbar-fixed-top"></nav>`
`.navbar-fixed-bottom`	Fixes the navbar **at the bottom** of the page.	`<nav class="navbar navbar-default navbar-fixed-bottom"></nav>`

`.navbar-right`	Aligns **navigation links to the right** inside the navbar.	`<ul class="nav navbar-nav navbar-right"></ul>`
`.navbar-form`	Adds an **inline search form** inside the navbar.	`<form class="navbar-form navbar-left"><input type="text" class="form-control"></form>`
`.dropdown`	Creates a **dropdown menu** inside the navbar.	`<li class="dropdown"><a href="#" class="dropdown-toggle" data-toggle="dropdown">Menu</a></li>`
`.dropdown-menu`	Defines the **dropdown menu items**.	`<ul class="dropdown-menu"><li><a href="#">Option 1</a></li></ul>`

Conclusion

- These **Bootstrap 3 navbar classes** help create **responsive** and **customized navigation bars**.
- You can combine multiple classes to **enhance functionality** (e.g., `.navbar-default navbar-fixed-top`).
- **Dropdowns, search bars, and collapsible menus** are easy to integrate.

Bootstrap 3 Dropdown Menu – Classes & Examples

Class Name	Description	Example Code
`.dropdown`	Defines a **dropdown menu** inside the navbar.	`<li class="dropdown"></li>`
`.dropdown-toggle`	Makes the menu link **clickable** for expanding the dropdown.	`<a class="dropdown-toggle" data-toggle="dropdown" href="#">Menu</a>`
`.dropdown-menu`	Defines the **list of dropdown items** inside the dropdown.	`<ul class="dropdown-menu"><li><a href="#">Item 1</a></li></ul>`

`.caret`	Adds a **small downward arrow** next to the dropdown link.	`<a class="dropdown-toggle" data-toggle="dropdown" href="#">Menu <span class="caret"></span></a>`
`.divider`	Adds a **horizontal separator** between dropdown items.	`<li class="divider"></li>`
`.dropdown-header`	Adds a **non-clickable header** inside the dropdown menu.	`<li class="dropdown-header">Section 1</li>`
`.disabled`	Disables a dropdown item (makes it unclickable).	`<li class="disabled"><a href="#">Disabled Item</a></li>`

Example: Simple Dropdown Menu

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 3 Dropdown Menu</title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css
">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></sc
ript>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"><
/script>
</head>
<body>

<nav class="navbar navbar-default">
    <div class="container-fluid">
        <div class="navbar-header">
            <a class="navbar-brand" href="#">MyWebsite</a>
        </div>
        <div class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li class="active"><a href="#">Home</a></li>
                <li class="dropdown">
                    <a class="dropdown-toggle" data-toggle="dropdown"
href="#">Services <span class="caret"></span></a>
                    <ul class="dropdown-menu">
                        <li><a href="#">Web Design</a></li>
                        <li><a href="#">SEO</a></li>
                        <li class="divider"></li>
                        <li class="dropdown-header">More Services</li>
                        <li><a href="#">Marketing</a></li>
                        <li class="disabled"><a href="#">Coming
Soon</a></li>
                    </ul>
                </li>
                <li><a href="#">Contact</a></li>
```

```
            </ul>
        </div>
    </div>
</nav>

</body>
</html>
```

Key Features of Dropdown Menus in Bootstrap 3

✅ **Fully Responsive** – Works on all screen sizes.
✅ **Customizable** – Supports headers, dividers, and disabled items.
✅ **Easy to Implement** – Uses simple Bootstrap classes.

JavaScript: Introduction and Object Model

1. Introduction to JavaScript

1.1 What is JavaScript?

- JavaScript (JS) is a **high-level, interpreted programming language** used primarily for making web pages interactive.
- It is an **event-driven, functional, and object-oriented language**.
- Runs in the browser (client-side) but can also be used server-side (Node.js).

1.2 Features of JavaScript

- **Lightweight & Fast**: Runs in the browser without compilation.
- **Cross-platform**: Works on all modern browsers.
- **Dynamic Typing**: No need to declare variable types (`var`, `let`, `const`).
- **Prototype-based Object Model**: Objects inherit directly from other objects.
- **Event-driven & Asynchronous**: Supports event handling and asynchronous programming with Promises and async/await.
- **Interpreted Language**: No need for compilation, executed line by line.

1.3 JavaScript in Web Development

- **Client-Side**: DOM Manipulation, form validation, animations, user interactions.
- **Server-Side**: Backend development using Node.js.
- **Full-Stack Development**: Used with frameworks like React, Angular, and Vue.

1.4 How JavaScript Works in the Browser

1. JavaScript is embedded in HTML (`<script>` tag) or included as an external file (`.js`).
2. The **JavaScript Engine** (like V8 in Chrome, SpiderMonkey in Firefox) interprets and executes the code.
3. Uses the **Document Object Model (DOM)** to interact with web pages.

---

2. JavaScript Object Model

2.1 Objects in JavaScript

- **Everything in JavaScript is an object** or can be treated as an object.
- Objects store properties (key-value pairs) and methods (functions).
-

Example of an Object

```
let person = {
    name: "John",
    age: 30,
    greet: function() {
        console.log("Hello, my name is " + this.name);
    }
};
person.greet();  // Output: Hello, my name is John
```

2.2 Object Properties and Methods

- **Properties**: Hold values (`name: "John"`, `age: 30`).
- **Methods**: Functions inside objects (`greet: function() {...}`).
- **Accessing Properties**:
    - Dot notation: `person.name`
    - Bracket notation: `person["age"]`
- **Modifying Properties**: `person.age = 35;`

2.3 Creating Objects

1. Object Literal (Most Common)

```
let car = { brand: "Toyota", model: "Camry", year: 2022 };
```

2. Using `new Object()` (Less Common)

```
let car = new Object();
car.brand = "Toyota";
car.model = "Camry";
car.year = 2022;
```

3. Constructor Function

```
function Car(brand, model, year) {
    this.brand = brand;
    this.model = model;
    this.year = year;
}
let myCar = new Car("Toyota", "Camry", 2022);
```

4. Using `Object.create()`

```
let personProto = {
    greet: function() {
        console.log("Hello, " + this.name);
    }
};
let john = Object.create(personProto);
john.name = "John";
john.greet();
```

2.4 Prototype and Inheritance

- **JavaScript uses prototypal inheritance**, meaning objects inherit properties and methods from prototypes.
- Every object in JavaScript has a prototype.
- Example:

```
function Animal(name) {
    this.name = name;
}
Animal.prototype.speak = function() {
    console.log(this.name + " makes a sound.");
};
let dog = new Animal("Dog");
dog.speak(); // Output: Dog makes a sound.
```

2.5 Built-in Objects

JavaScript provides several built-in objects like:

- `Math`: `Math.random()`, `Math.PI`
- `Date`: `new Date()`
- `String`: `"Hello".toUpperCase()`
- `Array`: `[1, 2, 3].length`

2.6 Object Methods (`Object.keys()`, `Object.values()`)

```
let obj = { a: 1, b: 2, c: 3 };
console.log(Object.keys(obj));   // ["a", "b", "c"]
console.log(Object.values(obj)); // [1, 2, 3]
```

Summary

- **JavaScript is a dynamic, interpreted language used mainly for web development**.
- **Everything in JavaScript revolves around objects**.
- Objects can be created using **literals, constructors, or `Object.create()`**.
- JavaScript follows **prototypal inheritance**, where objects inherit from other objects.
- Understanding the **object model** is key to mastering JavaScript.

JavaScript: Handling Forms & Statements

I. Handling Forms in JavaScript

Forms are essential in web development for gathering user input. JavaScript allows you to validate, manipulate, and interact with form elements to provide a dynamic user experience.

1. Accessing Form Elements

In JavaScript, you can access form elements in multiple ways, using the `document` object, typically through `getElementById()`, `getElementsByName()`, `getElementsByTagName()`, or `querySelector()`.

Syntax:

```
let element = document.getElementById('elementID');
let element = document.forms['formName'].elementName;
let element = document.querySelector('form input[name="username"]');
```

Example:

```
<form id="userForm">
  <input type="text" name="username" id="username">
  <input type="password" name="password" id="password">
  <input type="submit" value="Submit">
</form>

<script>
  let form = document.getElementById('userForm');
  let username = form.username.value;
  let password = form.password.value;

  console.log(username, password); // Logs the values when form is
submitted
</script>
```

2. Handling Form Submission

You can handle form submission with the `onsubmit` event. You can use this event to validate the form, send the form data to a server, or prevent the form from submitting.

Syntax:

```
<form id="formId" onsubmit="return functionName();">
```

Example (Simple form validation):

```
<form id="myForm" onsubmit="return validateForm();">
  <input type="text" id="name" name="name">
  <input type="submit" value="Submit">
</form>

<script>
  function validateForm() {
    let name = document.getElementById('name').value;
```

```
    if (name == "") {
      alert("Name must be filled out");
      return false; // Prevent form submission if validation fails
    }
    return true; // Allow form submission if validation passes
  }
</script>
```

3. Event Listeners for Form Input Fields

Instead of using inline event handlers (like `onsubmit` or `onchange`), you can use `addEventListener()` to attach event listeners for better separation of JavaScript and HTML.

Example (Listening for input changes):

```
<form id="signupForm">
  <input type="text" id="email" name="email" placeholder="Enter email">
  <input type="submit" value="Submit">
</form>

<script>
  document.getElementById('email').addEventListener('change', function() {
    let email = this.value;
    console.log("Email Changed:", email);
  });
</script>
```

II. JavaScript Statements

In JavaScript, statements are used to perform operations and define the flow of control in your program. JavaScript supports several types of statements: expressions, conditionals, loops, and more.

1. Declaration Statements

- **Variable Declaration:** Variables can be declared using `var`, `let`, or `const` in JavaScript.

  Syntax:

  ```
  let x = 10;
  const pi = 3.14;
  var name = 'John';
  ```

2. Conditional Statements

Conditional statements in JavaScript control the flow of execution based on conditions.

- **If Statement:**

  Syntax:

  ```
  if (condition) {
    // Code block if condition is true
  }
  ```

Example:

```
let age = 18;
if (age >= 18) {
  console.log("You are an adult.");
}
```

- **If-Else Statement:**

  Syntax:

```
if (condition) {
  // Code block if condition is true
} else {
  // Code block if condition is false
}
```

  Example:

```
let age = 16;
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

- **Else If Statement:**

  Syntax:

```
if (condition1) {
  // Code block if condition1 is true
} else if (condition2) {
  // Code block if condition2 is true
} else {
  // Code block if neither condition is true
}
```

  Example:

```
let score = 75;
if (score >= 90) {
  console.log("Grade A");
} else if (score >= 70) {
  console.log("Grade B");
} else {
  console.log("Grade C");
}
```

3. Looping Statements

JavaScript provides several loop structures to iterate over a block of code.

- **For Loop:**

  Syntax:

  ```
  for (let i = 0; i < length; i++) {
    // Code block to be executed
  }
  ```

  Example:

  ```
  for (let i = 0; i < 5; i++) {
    console.log(i); // Logs 0, 1, 2, 3, 4
  }
  ```

- **While Loop:**

  Syntax:

  ```
  while (condition) {
    // Code block to be executed
  }
  ```

  Example:

  ```
  let i = 0;
  while (i < 5) {
    console.log(i); // Logs 0, 1, 2, 3, 4
    i++;
  }
  ```

- **Do-While Loop:**

  Syntax:

  ```
  do {
    // Code block to be executed
  } while (condition);
  ```

  Example:

  ```
  let i = 0;
  do {
    console.log(i); // Logs 0, 1, 2, 3, 4
    i++;
  } while (i < 5);
  ```

4. Switch Statement

The `switch` statement is used to perform different actions based on different conditions. It is often used as an alternative to multiple `if`/`else` statements.

Syntax:

```
switch (expression) {
  case value1:
    // Block of code if expression == value1
    break;
  case value2:
    // Block of code if expression == value2
    break;
  default:
    // Block of code if none of the cases match
}
```

Example:

```
let day = 2;
switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  default:
    console.log("Invalid day");
}
```

5. Break & Continue Statements

- **Break:** The `break` statement is used to exit a loop or `switch` statement prematurely.
- **Continue:** The `continue` statement skips the current iteration of a loop and proceeds with the next iteration.

Example (Break in a loop):

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Exits the loop when i equals 5
  }
  console.log(i);
}
```

Example (Continue in a loop):

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    continue; // Skips this iteration when i equals 5
  }
  console.log(i);
}
```

6. Try-Catch-Finally Statements

JavaScript uses `try`, `catch`, and `finally` blocks for handling errors.

- **Try:** The `try` block contains code that might throw an error.
- **Catch:** The `catch` block contains code to handle any errors that are thrown.
- **Finally:** The `finally` block contains code that is executed regardless of whether an error occurred.

Example:

```
try {
  let result = 10 / 0;
  if (result === Infinity) throw "Division by zero is not allowed";
} catch (error) {
  console.log("Error:", error);
} finally {
  console.log("Execution completed.");
}
```

JavaScript Lecture Notes: Functions and Objects

1. Functions in JavaScript

What is a Function?

A function is a block of reusable code that performs a specific task.

Syntax:

```
function functionName(parameters) {
  // code to be executed
}
```

Example:

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
greet("Alice"); // Output: Hello, Alice!
```

Types of Functions

1. Function Declaration:

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

2. Function Expression:

```
const subtract = function(a, b) {
  return a - b;
};
```

3. Arrow Function:

```
const multiply = (a, b) => a * b;
```

JavaScript Lecture Notes: Functions and Objects

4. Anonymous Function:

```
setTimeout(function () {
  console.log("Executed after 1 second");
}, 1000);
```

Function Parameters and Return

Parameters are in the function definition; arguments are the values passed.

Example:

```
function sayHi(name = "Guest") {
  console.log("Hi, " + name);
}
sayHi("Bob"); // Hi, Bob
sayHi(); // Hi, Guest
```

Return Statement:

```
function square(num) {
  return num * num;
}
let result = square(4); // 16
```

Callback Functions

Functions can be passed as arguments:

```
function process(callback) {
  callback("Data processed");
}
process(function (message) {
  console.log(message);
});
```

JavaScript Lecture Notes: Functions and Objects

2. Objects in JavaScript

What is an Object?

An object is a collection of key-value pairs (properties and methods).

Syntax:

```
let person = {
  name: "Alice",
  age: 25,
  greet: function() {
    console.log("Hello!");
  }
};
```

Accessing Properties:

```
person.name;       // Dot notation
person["age"];     // Bracket notation
```

Modifying Objects

Add/Modify:

```
person.city = "New York";
person.age = 26;
```

Delete:

```
delete person.city;
```

Methods and `this` Keyword

Methods are functions in objects.

```
let user = {
```

```
  name: "John",
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
user.greet(); // Hello, John
```

Object Constructors and Prototypes

Constructor:

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
let p1 = new Person("Alice", 25);
```

Object.create():

```
let animal = {
  speak: function() {
    console.log("Animal sound");
  }
};
let dog = Object.create(animal);
dog.speak(); // Animal sound
```

Looping and JSON

Looping:

```
let car = {
  brand: "Toyota",
  model: "Corolla"
```

JavaScript Lecture Notes: Functions and Objects

```
};
for (let key in car) {
  console.log(key + ": " + car[key]);
}
```

JSON:

```
let jsonString = '{"name":"John", "age":30}';
let obj = JSON.parse(jsonString);
let str = JSON.stringify(obj);
```

JavaScript: DOM & Event Handling

---

1. Introduction to DOM

What is the DOM?

The **Document Object Model (DOM)** is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as a **tree of nodes**, where each node is an object representing a part of the page.

Why DOM?

- It allows dynamic access and manipulation of content.
- JavaScript can change HTML elements, attributes, styles, and more through the DOM.

Example HTML Structure:

```
<!DOCTYPE html>
<html>
  <head><title>DOM Example</title></head>
  <body>
    <h1 id="heading">Hello DOM</h1>
    <p class="text">Welcome to JavaScript</p>
    <button id="btn">Click Me</button>
  </body>
</html>
```

---

2. Accessing DOM Elements

Common Methods:

- `document.getElementById("id")` – returns the element with the specified ID.
- `document.getElementsByClassName("class")` – returns a collection of all elements with the given class name.
- `document.getElementsByTagName("tag")` – returns a collection of all elements with the specified tag.
- `document.querySelector("selector")` – returns the first element matching a CSS selector.
- `document.querySelectorAll("selector")` – returns all elements matching a CSS selector.

Example:

```
let heading = document.getElementById("heading");
```

```
let text = document.querySelector(".text");
let allParagraphs = document.getElementsByTagName("p");
```

---

3. Changing Content and Style

Changing Text and HTML:

```
heading.textContent = "New Heading"; // changes only text
text.innerHTML = "<strong>Changed!</strong>"; // allows HTML
```

Changing Style:

```
heading.style.color = "blue";
text.style.fontSize = "20px";
```

---

4. Introduction to Events

What is an Event?

An **event** is a signal that something has happened on the page (e.g., a user clicks a button, hovers over an element, presses a key, etc.).

Common Events:

- `click` – when an element is clicked
- `mouseover` – when the mouse pointer is over an element
- `mouseout` – when the pointer leaves the element
- `keydown`, `keyup` – when a key is pressed or released
- `submit` – when a form is submitted
- `load` – when the page is fully loaded

---

5. Event Handling

Ways to Handle Events:

1. **Inline (HTML)** – not recommended

```
<button onclick="alert('Clicked')">Click</button>
```

2. **DOM Property Method**

```
button.onclick = function() {
  alert("Clicked!");
};
```

3. **addEventListener() Method** – recommended

```
button.addEventListener("click", function() {
  alert("Clicked!");
});
```

Why use addEventListener?

- Can add multiple handlers to a single event
- Separation of HTML and JS (cleaner code)
- Supports event bubbling and capturing

---

6. Examples of Event Handling

Click Event:

```
<button id="btn">Click Me</button>
<p id="result"></p>
let button = document.getElementById("btn");
let result = document.getElementById("result");

button.addEventListener("click", function() {
  result.textContent = "Button was clicked!";
});
```

Mouseover and Mouseout:

```
<h2 id="hoverText">Hover over me</h2>
let hoverText = document.getElementById("hoverText");

hoverText.addEventListener("mouseover", function() {
  hoverText.style.color = "red";
});
hoverText.addEventListener("mouseout", function() {
  hoverText.style.color = "black";
});
```

Keyboard Events:

```
<input type="text" id="inputBox" placeholder="Type here">
<p id="output"></p>
let input = document.getElementById("inputBox");
let output = document.getElementById("output");

input.addEventListener("keyup", function() {
  output.textContent = "You typed: " + input.value;
});
```

Form Submit Event:

```
<form id="myForm">
  <input type="text" id="name" placeholder="Enter your name">
```

```
  <button type="submit">Submit</button>
</form>
let form = document.getElementById("myForm");

form.addEventListener("submit", function(e) {
  e.preventDefault();
  let name = document.getElementById("name").value;
  alert("Form submitted by " + name);
});
```

---

7. Event Object and Event Bubbling

The Event Object

When an event occurs, an `event` object is passed to the event handler which contains details such as:

- `event.type`
- `event.target`
- `event.preventDefault()`

Event Bubbling

Events in the DOM bubble up from the target element to the root.

```
document.getElementById("child").addEventListener("click", function(event)
{
  alert("Child Clicked");
});

document.getElementById("parent").addEventListener("click", function(event)
{
  alert("Parent Clicked");
});
```

Stopping Propagation:

```
event.stopPropagation();
```

---

8. Summary

- The DOM represents the HTML document as an object tree.
- JavaScript can access and modify DOM elements using methods like `getElementById()` and `querySelector()`.
- Events allow us to interact with users (e.g., clicks, keypresses).
- `addEventListener()` is the best practice for event handling.
- Understanding the event object and event flow (bubbling) is key for advanced interactions.

JavaScript – Form Handling and Validations

📜 Table of Contents

1. Introduction
2. HTML Forms Overview
3. JavaScript and Form Handling
4. Types of Form Validations
5. Validation Techniques (with Examples)
6. Event-Driven Validation
7. HTML5 vs JavaScript Validation
8. Best Practices
9. Complete Example
10. Summary

🧠 1. Introduction

In web development, **forms** are used to collect user input. Handling and validating form data is critical to ensure correctness, security, and usability.

Form handling refers to managing form submissions, while **form validation** ensures the data entered is valid and safe to process.

📐 2. HTML Forms Overview

An HTML form is defined using the `<form>` tag and typically includes input fields like textboxes, checkboxes, radio buttons, dropdowns, etc. 📌 **Basic Form Structure:**

```
<form id="myForm" onsubmit="return validateForm()">
  <label>Username:</label>
  <input type="text" id="username" name="username" required>

  <label>Email:</label>
  <input type="email" id="email" name="email">

  <input type="submit" value="Submit">
</form>
```

💻 3. JavaScript and Form Handling

JavaScript allows us to:

- Access form elements
- Validate input values
- Prevent submission on invalid data
- Provide real-time feedback

Accessing Form Values:

```
let username = document.getElementById("username").value;
```

Preventing Form Submission:

```
document.getElementById("myForm").addEventListener("submit",
function(event) {
  event.preventDefault(); // Stops form submission
});
```

✅ 4. Types of Form Validations

Validation Type	Description
Required Field	Ensures input is not empty
Format Validation	Checks format using patterns (e.g., email)
Length Validation	Enforces input length
Number Validation	Checks for numeric values and ranges
Password Strength	Ensures password complexity
Matching Fields	Confirms fields (like password confirm)

🛠️ 5. Validation Techniques

🔹 Required Field Validation

```
function validateForm() {
  let username = document.getElementById("username").value;
  if (username == "") {
    alert("Username is required!");
    return false;
  }
}
```

🔹 Email Format Validation

```
function validateEmail() {
  let email = document.getElementById("email").value;
  let pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!pattern.test(email)) {
    alert("Invalid email format!");
    return false;
  }
}
```

🔹 Number Validation

```
function validateAge() {
  let age = document.getElementById("age").value;
  if (isNaN(age) || age < 18) {
    alert("Age must be 18 or above!");
    return false;
  }
}
```

🔹 Length Validation

```
function validatePassword() {
  let pwd = document.getElementById("password").value;
  if (pwd.length < 6) {
    alert("Password must be at least 6 characters long");
    return false;
  }
}
```

⚡ 6. Event-Driven Validation (Real-Time)

```
document.getElementById("email").addEventListener("blur", function() {
  if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(this.value)) {
    alert("Invalid email");
  }
});
```

- `blur` event triggers when the input loses focus.
- Real-time validation improves user experience.

---

7. HTML5 vs JavaScript Validation

Feature	HTML5 Validation	JavaScript Validation
Easy to implement	Yes	Requires scripting
Custom messages	Limited	Fully customizable
Flexibility	Low	High
User feedback	Basic (default browser messages)	Advanced (alerts, custom UI, etc.)

HTML5 Example:

```
<input type="email" required>
```

JavaScript Example:

```
if (!email.match(/^[^\s@]+@[^\s@]+\.[^\s@]+$/)) {
  alert("Invalid email");
}
```

📋 8. Best Practices in Form Validation

- Always **validate on both client and server sides**.
- Provide **real-time feedback** for better UX.
- Use **regular expressions** for pattern matching.
- Display **clear error messages**.
- Prevent form submission until all fields are valid.
- Sanitize and escape inputs to prevent XSS attacks.

🧪 9. Complete Example

```
<form id="registerForm" onsubmit="return validateForm()">
  <input type="text" id="username" placeholder="Username">
  <input type="email" id="email" placeholder="Email">
  <input type="number" id="age" placeholder="Age">
  <input type="submit" value="Register">
</form>

<script>
function validateForm() {
  let username = document.getElementById("username").value;
  let email = document.getElementById("email").value;
  let age = document.getElementById("age").value;

  let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  if (username == "" || email == "" || age == "") {
    alert("All fields are required!");
    return false;
  }
```

```
  if (!emailPattern.test(email)) {
    alert("Invalid email address!");
    return false;
  }

  if (isNaN(age) || age < 18) {
    alert("Age must be a number and 18 or older!");
    return false;
  }

  alert("Form submitted successfully!");
  return true;
}
</script>
```

📌 10. Summary

- JavaScript provides **powerful tools** for client-side form handling.
- Validating inputs before submission improves **data quality**, **security**, and **user experience**.
- Use **event listeners** for real-time validation.
- Combine **HTML5 attributes** with JavaScript for robust solutions.

JavaScript: Error Handling & Validation

1. ⚠️ Understanding Errors in JavaScript

💡 What is an Error?

An error is a problem that arises during the execution of a script. JavaScript provides error-handling mechanisms to catch and resolve such issues without halting execution.

🔍 Categories of Errors

Type	Description	Example
SyntaxError	Occurs when code violates JS grammar rules	`if (true {`
ReferenceError	Refers to an undefined variable	`console.log(x);` (x not declared)
TypeError	Operation on the wrong data type	`null.f()`
RangeError	Number is outside allowable range	`new Array(-1)`
URIError	Malformed URI sequence	`decodeURIComponent('%')`
EvalError	Rare; related to `eval()` misuse	Older JS versions

---

2. 🔄 The `try...catch...finally` Block

🔧 Purpose:

- Allows you to **try** code that may throw an error.
- Use `catch` to handle the error gracefully.
- `finally` executes cleanup code (optional, always runs).

📌 Syntax:

```
try {
  // Code that may throw an error
} catch (error) {
  // Handle error
} finally {
  // Code that runs regardless of error
}
```

🔎 Example:

```
function readFile() {
```

```
  try {
    // Simulate error
    let content = JSON.parse('INVALID JSON');
    console.log(content);
  } catch (err) {
    console.error("Caught error:", err.message);  // JSON.parse error
  } finally {
    console.log("Cleanup: Closing file or releasing resources");
  }
}
```

---

3. 📦 Understanding the Error Object

The error object caught by `catch` provides details.

🔍 Error Properties:

- `name`: Type of error (e.g., ReferenceError)
- `message`: Description of the error
- `stack`: Stack trace (useful for debugging)

```
try {
  throw new Error("File not found");
} catch (err) {
  console.log("Name:", err.name);       // Error
  console.log("Message:", err.message); // File not found
  console.log("Stack:", err.stack);     // Stack trace
}
```

---

4. 🧰 Throwing Custom Errors

You can **manually create and throw** errors using the `throw` keyword.

📌 Syntax:

```
throw new Error("Custom error message");
```

👮 Example: Input Validation

```
function validateAge(age) {
  if (age < 0) {
    throw new RangeError("Age cannot be negative");
  }
  return true;
}

try {
  validateAge(-5);
} catch (err) {
  console.error(`${err.name}: ${err.message}`);  // RangeError: Age cannot
be negative
```

```
}
```

---

5. ✍️ Writing Custom Error Classes (Advanced)

JavaScript supports creating your own custom error classes.

🧱 Example:

```
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}

try {
  throw new ValidationError("Invalid input!");
} catch (e) {
  console.log(e.name);    // ValidationError
  console.log(e.message); // Invalid input!
}
```

---

6. ✅ Input Validation in JavaScript

✍️ What is Validation?

Validation is the process of ensuring user input is correct, complete, and secure.

📌 Validation Layers:

- **Client-side**: Fast, responsive, but not secure alone.
- **Server-side**: Secure and necessary for real protection.

---

7. 📋 Common Client-Side Validation Checks

Type	Example
Required Field	Name/email/password not empty
Pattern Match	Valid email format
Range Check	Age between 18 and 60
Length Check	Password min 8 characters
Type Check	Numeric fields only

🧪 Basic HTML + JavaScript Example:

```
<form onsubmit="return validateForm()">
  <input id="email" type="text" placeholder="Enter email" />
  <input type="submit" value="Submit" />
</form>

<script>
function validateForm() {
  let email = document.getElementById("email").value;
  if (email === "") {
    alert("Email is required.");
    return false;
  }
  if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
    alert("Invalid email format.");
    return false;
  }
  return true;
}
</script>
```

---

8. 📚 Validation with Regular Expressions (Regex)

Regular expressions help match patterns for input.

Examples:

Purpose	Regex
Email	`/^[^\s@]+@[^\s@]+\.[^\s@]+$/`
Phone	`/^\d{10}$/`
Alphanumeric	`/^[a-zA-Z0-9]+$/`
Password (strong)	`/^(?=.*[A-Z])(?=.*\d)[A-Za-z\d]{8,}$/`

---

9. 💡 Combining Validation with Error Handling

🔀 Example:

```
function processForm(email) {
  try {
    if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
      throw new Error("Invalid email address");
    }
    console.log("Email is valid:", email);
  } catch (e) {
    console.error("Error:", e.message);
  }
}

processForm("invalid-email");  // Error: Invalid email address
```

---

10. 🔁 Validating Multiple Fields (Modular Approach)

```
function validateForm() {
  let username = document.getElementById("username").value;
  let password = document.getElementById("password").value;

  try {
    if (!username) throw new Error("Username is required");
    if (password.length < 8) throw new Error("Password must be at least 8
characters");

    console.log("Validation successful");
    return true;
  } catch (error) {
    alert(error.message);
    return false;
  }
}
```

---

11. ✅ Best Practices Summary

Principle	Description
Validate both client & server	Prevent bypassing JS with browser dev tools
Use custom errors where needed	Improve code readability and error categorization
Avoid catching everything	Don’t use catch blindly unless necessary
Log stack traces for debugging	Especially in production systems
Use HTML5 validation attributes	`required`, `min`, `type="email"`

✅ HTML5 Built-In Validation (Bonus)

```
<form>
  <input type="email" required>
  <input type="number" min="1" max="10">
  <input type="submit">
</form>
```

Browser handles basic validation **automatically** with these attributes.
