**Module 1**

**Topic 1: Course Introduction**

**1. Introduction to Web Content**

* **User vs Developer**: Users view content; developers control structure & style via HTML/CSS.
* **Core Tech**:
  + **HTML**: Base structure of all web content.
  + **CSS**: Controls visual appearance and layout.

**2. HTML (HyperText Markup Language)**

* **Purpose**: Originally built to share structured documents online.
* **Modern Capabilities**:
  + Multimedia support: <video>, <audio>
  + Responsive layouts
  + Rich form inputs: type="date", type="range", etc.
  + Native spell-check, editing

**3. CSS (Cascading Style Sheets)**

* **Role**: Visual styling separated from content.
* **Features**:
  + CSS Variables: --primary-color
  + Media Queries: @media (max-width: 768px)
  + Advanced Styling:
    - box-sizing: border-box
    - Multiple backgrounds
    - Border & text shadows
  + CSS Animations & transitions

**4. Standards & Governance**

* **W3C**: Oversees HTML/CSS evolution
* **Updates**: HTML5, CSS3 introduce modern capabilities

**5. Cross-Device Compatibility**

* HTML/CSS used in:
  + Mobile devices
  + Smart TVs
  + Game consoles
  + IoT interfaces

**6. Impact**

* **Device-Agnostic**: Uniform access across platforms
* **Future-Proof**: Standards evolve with tech

**✅ Example Workflow**

<!-- HTML -->

<video controls>

<source src="video.mp4" type="video/mp4">

</video>

<!-- CSS -->

<style>

video {

width: 100%;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

@media (max-width: 480px) {

video { margin: 10px; }

}

</style>

**Topic 2: Semantic and Meta Tags**

**1. Why Semantic HTML Matters**

* **Purpose**: Describes the role/meaning of content.
* **Benefits**:
  + Accessibility: Screen readers understand better.
  + SEO: Search engines index smarter.

**2. Basic HTML Page Structure**

* <head>: Metadata, scripts, SEO tags.
* <body>: Visible content.
* Semantic Tags:
  + <header>, <main>, <footer> for layout.
  + <section>, <article> for content blocks.

**3. Navigation Structure**

* <nav>: Groups navigation links.
* Common layout:

<nav>

<ul>

<li><a href="#">Home</a></li>

</ul>

</nav>

**4. Main Content Layout**

* <article>: Self-contained unit (e.g., blog post).
* <section>: Thematic grouping with a heading.
* Nested structure supported:

<article>

<header><h1>Title</h1></header>

<section><h2>Subsection</h2></section>

</article>

**5. Footer Usage**

* Can appear globally or within components.
* Typical content:
  + Copyright
  + Contact info
  + Extra links

**6. Best Practices**

* Use semantic tags for meaning, not style.
* Improves structure, SEO, and accessibility.

**8. Semantic HTML Cheat Sheet**

**🔹 8.1 Sectioning Tags**

|  |  |
| --- | --- |
| Tag | Purpose |
| <header> | Top section (logo, nav) |
| <nav> | Navigation links |
| <main> | Main page content |
| <footer> | Bottom section, extra info |
| <aside> | Secondary content (ads, sidebars) |
| <article> | Self-contained item (blog, comment) |
| <section> | Thematic groupings with headings |
| <details> | Expandable content area |
| <summary> | Summary for <details> |
| <h1>-<h6> | Headings (h1 = most important) |

**🔹 2. Content Tags**

|  |  |
| --- | --- |
| Tag | Purpose |
| <h1> to <h6> | Define headings from most to least important. |
| <p> | Defines a paragraph. |
| <ul> | Unordered (bulleted) list. |
| <ol> | Ordered (numbered) list. |
| <li> | List item in <ul> or <ol>. |
| <dl> | Description list (used with <dt> and <dd>). |
| <dt> | Term in a description list. |
| <dd> | Description of the term in a description list. |
| <figure> | Wraps media elements (images, charts) with a caption. |
| <figcaption> | Caption for a <figure>. |
| <blockquote> | Defines a block quote. |
| <pre> | Displays preformatted text in a monospace font. |
| <hr> | Inserts a horizontal rule. |
| <menu> | Semantic alternative to <ul>. |

**🔹 8.3 Inline Tags**

|  |  |
| --- | --- |
| Tag | Purpose |
| <a> | Link to another document |
| <abbr> | Abbreviation/acronym |
| <b> | Bold text |
| <br> | Line break |
| <cite> | Work citation (italicized) |
| <code> | Inline code block |
| <em> | Emphasized text |
| <i> | Italic (idioms/technical terms) |
| <mark> | Highlighted/marked text |
| <q> | Short inline quote |
| <s> | Strikethrough |
| <samp> | Sample output from program |
| <small> | Smaller text (e.g., legal notice) |
| <span> | Generic inline container for styling |
| <strong> | Bold with importance |
| <sub> | Subscript text |
| <sup> | Superscript text |

**🔹 4. Multimedia Tags**

|  |  |
| --- | --- |
| Tag | Purpose |
| <video> | Embeds video content. |
| <source> | Specifies media source file for <video> or <audio>. |
| <audio> | Embeds audio content. |

**🔹 5. Meta and Structure Tags**

|  |  |
| --- | --- |
| Tag | Purpose |
| <html> | Root element of an HTML page. |
| <head> | Contains metadata and links (e.g., title, CSS). |
| <body> | Contains visible page content. |

**Introduction to Search Engines and SEO**

* Search engines return results based on relevance to search queries.
* SEO (Search Engine Optimization) improves a website’s visibility and ranking.
* SEO focuses on optimizing website content, semantics, and delivery.

**How Search Engines Analyze Web Pages**

* Search engines scan HTML documents and media content.
* They follow internal and external links to analyze the entire site.
* Rankings vary per keyword and are based on proprietary algorithms.
* Following best practices can positively influence search rankings.

**Role of Meta Tags in SEO**

* Meta tags provide metadata (information about the page) to search engines.
* Located inside the <head> element and are not visible in the browser.
* No closing tag is required for <meta> elements.

**Important Meta Tag Attributes**

* **name**: Describes the type of metadata.
* **content**: Holds the value of the metadata.

**Common Meta Tags and Their Purpose**

|  |  |
| --- | --- |
| Meta Tag | Purpose |
| <meta name="author" ...> | Specifies the page's author or organization. |
| <meta name="description" ...> | Describes page content; often shown in search results. |
| <meta name="keywords" ...> | Previously used for search keywords; now largely ignored or flagged. |
| <meta name="robots" ...> | Controls how search bots index/follow the page and its links. |
| <meta name="viewport" ...> | Helps with responsive design and mobile optimization. |

**Robots Meta Tag Values**

* index: Allow indexing of the page.
* follow: Allow bots to follow links on the page.
* noindex: Do not index the page (may be ignored by some bots).
* nofollow: Do not follow links (may also be ignored by some bots).

**Importance of Viewport Meta Tag**

* Defines how content appears on mobile devices.
* Prevents desktop-style scaling on mobile.
* Example:  
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
* Essential for responsive design and influences SEO ranking.

**Metadata Cheat Sheet: Meta Tag Structure**

**✅ Basic Meta Tags for SEO**

* **Author**: <meta name="author" content="Name">
* **Title**: <meta name="title" content="Page Title">
* **Description**: <meta name="description" content="Short description">
* **Language**: <meta name="language" content="English">
* **Robots**: <meta name="robots" content="index,follow">
* **Google Specific**:
  + No sitelinks box: <meta name="google">
  + No translate: <meta name="googlebot" content="notranslate">
* **Revised**: <meta name="revised" content="Date and Time">
* **Rating**: <meta name="rating" content="safe for kids">
* **Copyright**: <meta name="copyright" content="Copyright 2022"

**HTTP-Equiv Meta Tags**

|  |  |
| --- | --- |
| Tag | Function |
| content-type | Sets the document format. |
| default-style | Specifies styling document. |
| refresh | Refreshes the page after a time interval. |
| Content-language | Specifies the page language. |
| Cache-Control | Instructs browser caching behavior. |

**Responsive Design / Mobile Meta Tags**

|  |  |
| --- | --- |
| Tag | Description |
| format-detection | Makes phone numbers clickable for calls. |
| HandheldFriendly | Declares mobile-friendly page. |
| viewport | Defines display area for responsive layout. |

**HTML Document Overview (index.html)**

* Follows standard **semantic HTML structure**:
  + <!DOCTYPE html> declaration
  + <html> root element containing:
    - <head>: metadata and stylesheet links
    - <body>: visible page content

**Head Element Details**

* Includes:
  + <title> tag (should be updated per project)
  + <meta> tags for SEO (description, author, charset, viewport)
  + Commented-out **Open Graph Protocol** tags for social media sharing
  + Commented-out **favicon** setup
  + A link to styles.css is already included

**Body Element Structure**

* Contains **semantic layout elements**:
  + <header>: for site title or logo
  + <nav>: for navigation menus
  + <main>: for main content
  + <footer>: for copyright

**Script Integration**

* A <script> tag is already included at the end of the body.
* This tag links to script.js in the JS folder.

**Template Reusability**

* The template is reusable for future web development projects.
* Saves time and ensures consistent structure.

**Web Layout Design Types**

|  |  |  |
| --- | --- | --- |
| Layout Type | Description | Common Usage |
| Top Navbar Layout | Horizontal bar at the top with essential links (e.g., Home, Products, Contact). | Site-wide navigation for consistency. |
| Carousel Layout | Rotating banner displaying featured items, offers, or promotions. | E-commerce, landing pages, product highlights. |
| Blog Layout | Multiple content blocks with varying sizes; may include summaries linking to full articles. | News sites, personal or corporate blogs. |
| Dashboard Layout | Sidebar navigation + main content area with data, forms, and charts. | Admin panels, business/enterprise apps. |
| More Layouts | Additional layout styles available via frameworks like Bootstrap. | Custom needs, inspiration, best practices. |

**Role of Social Networks in Sharing Content**

* Social networks allow sharing links with preview snippets.
* Previews provide users with information about the webpage before clicking.

**Importance of Meta Tags in Web Previews**

* Meta tags control what is shown in link previews on social media.
* These are **not** the same as traditional SEO meta tags (used for search engines).

**Introduction to Open Graph Protocol (OGP)**

* Created by Facebook in 2010 to improve link-sharing experience.
* Defines how metadata should be embedded for social networks.
* Allows webpages to describe themselves through structured metadata.

**Open Graph Metadata Basics**

* OGP meta tags are placed inside the <head> element of HTML.
* Uses property instead of name to define metadata name.
* Uses content to assign the value of the metadata.

**Required Open Graph Properties**

|  |  |
| --- | --- |
| Property | Purpose |
| og:title | Title of the page shown in the preview. |
| og:type | Type of content (e.g., website, article, video, music). |
| og:url | Canonical URL to be used for the page. |
| og:image | Image URL to display in the link preview. |

**✨ Optional Open Graph Properties**

|  |  |
| --- | --- |
| Property | Purpose |
| og:description | A short description of the page content. |
| og:locale | Specifies language and territory (e.g., en\_US for English - United States). |
| og:site\_name | Defines the name of the site the content belongs to. |

**✅ Benefits of Using Open Graph Metadata**

* Enhances link previews, increasing user engagement and click-through rate.
* Provides a consistent and professional appearance for shared content.
* Enables fine control over what users see before visiting the website

**Introduction to Social Media Cards**

* Before Open Graph Protocol (OGP), social media platforms relied on **internal heuristics** to guess a page's title, description, and preview image.
* This resulted in **inaccurate or poorly scaled previews**.
* **Meta tags** now give content creators more control over how their content appears when shared.

**Evolution and Adoption of OGP**

* Introduced by **Facebook**, OGP has now been widely adopted by other **Meta platforms** and **social media giants**.
* Platforms may use their **own meta tag prefixes**, replacing the default og: tag format.

**Need for Social Media Cards**

* In the attention economy, **visual content** is crucial to attract users.
* A **clear title and relevant image** can significantly impact user engagement.
* SM cards can be:
  + **Generic** (site-wide content)
  + **Page-specific** (customized for particular pages)

**The ‘type’ OGP Tag**

* Defines **the nature of the shared content** (e.g., article, video.movie, music.song).
* Enables **rich metadata**: For example, a music post can include song name, album, duration.
* Helps **capture attention instantly**—essential for user engagement on fast-scrolling platforms.

**Impact of Social Media Cards on SEO**

* SM cards are part of the **modern SEO ecosystem**.
* They:
  + Help **search engine crawlers** better interpret and rank pages.
  + Provide structured metadata to **boost visibility**.
  + Improve **click-through rates** via attractive previews.
  + Assist in **tracking traffic sources** for analytics.

**Customization and Developer Support**

* Developers can add **additional OGP tags**, like og:video, for richer media content.
* Meta provides a **developer support page** for implementing and testing SM cards (refer to additional resources).

**Table: Open Graph & Social Media Card Tags**

|  |  |
| --- | --- |
| Tag / Concept | Purpose / Description |
| og:title | Sets the title shown on social media cards. |
| og:description | Gives a short summary or description of the content. |
| og:image | Defines the image shown in the preview (thumbnail). |
| og:url | Canonical URL of the web page being shared. |
| og:type | Defines the type of content (e.g., article, video, music, book, etc.). |
| music:\* | Sub-tags under music type — can include album, duration, song name, etc. |
| video:\* | Optional sub-tags when the type is video, for embedded playback on supported platforms. |
| og:site\_name | Name of the site being shared. |
| og:locale | Language and region (e.g., en\_US). |

**Topic 3: User Input and Forms**

**HTML Forms and Validation**

**Purpose of HTML Forms**

* Used to **capture user input** (e.g., during account registration or online purchases).
* Ensures that the **input data is structured and usable**.

**Importance of Validation**

* Prevents errors such as incorrect delivery addresses.
* Improves **user experience** and avoids system errors.

**✅ Types of Form Validation**

**1. Client-side Validation**

* Done using **HTML input types** or **JavaScript** in the browser.
* Provides **immediate feedback** to users.
* Prevents submission until inputs are correct.
* Example: <input type="email"> checks for a valid email format.

**2. Server-side Validation**

* Performed after form submission, on the server.
* Ensures **security** and prevents **malicious data**.
* Can perform **complex checks** (e.g., database queries, business rules).

**⚙️ Best Practice**

* Use **both client-side and server-side** validation for:
  + Instant feedback (client-side).
  + Data security and integrity (server-side).

**✅ Common HTML Input Types & Their Use Cases**

**🔘 Button**

* Used to trigger actions:  
  <input type="button"> or <button>Click me</button>

**🔘 Checkbox**

* Allows **multiple selections**:  
  <input type="checkbox">

**🔘 Radio**

* Allows **one selection** from a group:  
  <input type="radio" name="theme">

**📤 Submit**

* Submits the form to a server:  
  <input type="submit">

**✍️ Text**

* Basic single-line input:  
  <input type="text">

**🔐 Password**

* Obscured input for sensitive data:  
  <input type="password">

**📅 Date**

* For date values (year, month, day):  
  <input type="date">

**🕓 Datetime-local**

* Includes date and time (no time zone):  
  <input type="datetime-local">

**📧 Email**

* For validating email format:  
  <input type="email">

**📁 File**

* Uploads files from the device:  
  <input type="file" multiple>

**🕶️ Hidden**

* Invisible field, still submitted:  
  <input type="hidden">

**🖼️ Image**

* Image-based submit button:  
  <input type="image" src="submit.png">

**🔢 Number**

* Numeric input with constraints:  
  <input type="number" min="1" max="10">

**🎚️ Range**

* Slider for number range:  
  <input type="range" min="0" max="100">

**🔄 Reset**

* Resets form to initial values:  
  <input type="reset">

**🔍 Search**

* Styled text field for search queries:  
  <input type="search">

**⏰ Time**

* Time input (hours and minutes):  
  <input type="time">

**📞 Tel**

* For telephone number inputs:  
  <input type="tel" pattern="\d{10}">

**Form Validation Best Practices**

* Use HTML input types for **basic validation**.
* Use JavaScript for **custom client-side logic**.
* Use backend logic for **secure server-side validation**.
* Always include required for mandatory fields:  
  <input type="text" required>

**Client-Side Form Validation in HTML and CSS**

**✅ Importance of Client-Side Validation**

* Prevents unnecessary HTTP requests to the server.
* Enhances user experience by providing immediate feedback.
* Reduces server resource usage by catching errors early.

**✅ Common Reasons for Form Errors**

* Missing required fields.
* Input values too short or too long.
* Incorrect data formats.

**✅ Key HTML Validation Attributes**

* required: Ensures the field is not left empty.
* minlength: Sets minimum number of characters allowed.
* maxlength: Sets maximum number of characters allowed.

<input type="text" required minlength="3" maxlength="12">

**✅ Using CSS to Highlight Errors**

* Use :invalid pseudo-class to style invalid inputs.
* Combine with :focus to highlight only when the user is interacting.

input:invalid:focus {

border: 2px solid red;

}

**✅ Benefits**

* Provides real-time feedback.
* Prevents form submission until all validations pass.
* Improves form usability and accessibility.

**✅ Creating a Table Booking Form with Radio Buttons**

**Use Case: Little Lemon Restaurant Booking System**

* Different table sizes (2, 4, 6 person).
* Choice of seating location (Indoors or Outdoors).

**Why Use Radio Buttons?**

* Ideal for mutually exclusive choices.
* Groups options where only one can be selected.

**HTML Setup Structure**

* Use <fieldset> to group related options.
* Use <legend> (optional) for accessibility.
* Set name attribute same for grouped radio buttons.

<fieldset id="size">

<input type="radio" name="size" value="2"> Two-person table

<input type="radio" name="size" value="4" checked> Four-person table

<input type="radio" name="size" value="6"> Six-person table

</fieldset>

**Mobile Accessibility Tip**

* Wrap each radio button and label text in a <label> tag to make them easier to click.

<label><input type="radio" name="location" value="indoors" checked> Indoors</label>

<label><input type="radio" name="location" value="outdoors"> Outdoors</label>

**Testing Behavior**

* Only one option per group can be selected.
* Improved user experience on both desktop and mobile devices.

**Cheat Sheet: Interactive Form Elements**

**Why Use HTML Form Validation?**

* Ensures correct data shape before submission.
* Helps users fix mistakes instantly.

**Essential Validation Attributes**

|  |  |
| --- | --- |
| Attribute | Description |
| required | Makes the field mandatory. |
| minlength | Sets the minimum input length. |
| maxlength | Sets the maximum input length. |
| pattern | Validates data against a regular expression. |
| type | Enforces specific data format (email, URL, etc). |

<input type="email" required>

<input type="text" minlength="3" maxlength="10">

<input type="url" pattern="https?://.+">

**What Happens When You Submit a Form?**

* When a form is submitted (e.g., placing an order), the browser sends a request to a web server.
* This process follows the **HTTP request-response cycle**.
* Data in forms is submitted to the server using either the **GET** or **POST** method.

**The Role of HTTP Methods in Form Submission**

**✅ GET Method**

* Appends form data to the **URL**.
* Useful for simple, non-sensitive queries.
* **Drawbacks:**
  + **URL Length Limitation:** Limited to ~2000 characters in most browsers.
  + **Server Constraints:** Some web servers cap URL length (~4096 characters).
  + **Security Risk:** Data is visible in browser history and server logs—unsafe for sensitive info.

**✅ POST Method**

* Sends form data in the **HTTP request body**.
* More secure than GET, especially when paired with **HTTPS**.
* Ideal for sending **sensitive data** (e.g., login credentials, payments).
* Keeps data hidden from URLs and browser history.

**Securing Data Transmission**

* While POST is more secure, data can still be intercepted without encryption.
* Use **HTTPS** to ensure end-to-end encryption between browser and server.

**Understanding the Form Element**

**✅ <form> Tag Basics**

* Wraps input elements to collect user data.
* Key attributes:
  + **action** – URL where form data is sent.
  + **method** – HTTP method used (GET or POST).

**action Attribute Explained**

* **Specifies the destination** (URL or path) for form submission.
* Can use:
  + **Full URL:** https://meta.com
  + **Absolute path:** /login
  + **Relative path:** login

**Path Behavior**

* **Absolute path (/login)**: Uses base domain (e.g., https://meta.com/login)
* **Relative path (login)**: Appends to current path (e.g., https://meta.com/about/login)

**method Attribute Explained**

* Defines how form data is transmitted.
* Options:
  + <form method="get"> – Data in URL
  + <form method="post"> – Data in request body
* If omitted, defaults to **GET**.

**Form Submission Flow**

1. User fills form and clicks submit.
2. Browser creates an HTTP request (GET or POST).
3. Web server processes data.
4. Server returns a response:
   * Success → redirect or confirmation page.
   * Error → handled with validations or error pages.

**Beyond HTML Forms: JavaScript and JSON**

* Developers can also submit data using **JavaScript**, bypassing forms.
* Common with front-end frameworks (e.g., React, Vue).
* Data often sent in **JSON format** via HTTP requests (using fetch() or XMLHttpRequest).
* This allows **dynamic, asynchronous** interactions (AJAX).

**Importance of Consistent User Experience in Front-End Development**

* Ensuring smooth and accessible HTML form interaction is key to good user experience.
* Challenges arise due to differences in browser technologies and operating systems.

**Browser Inconsistencies in HTML Form Rendering**

* HTML forms render differently across browsers like Chrome, Firefox, Safari, and Microsoft Edge.
* Variations also occur between operating systems (e.g., Windows vs. Mac).

**Common Form Element Inconsistencies**

**Input Elements:**

* Same HTML <input> type renders differently in:
  + Height, width, and corner radius.
  + Safari uses thinner borders and different sizing.

**Checkbox Elements:**

* Differences in:
  + Checked/unchecked state appearance.
  + Colors (gray in Edge, varying shades of blue in others).
  + Checkbox size and border styles (Safari uses smaller size).

**Solving Cross-Browser Form Styling Issues**

**Use of CSS for Styling:**

* CSS can ensure consistent visual appearance.
* Use element selectors and attribute selectors to target form elements.

**Example:**

input[type="text"] {

font-size: 16px;

width: 100%;

height: 40px;

border: 1px solid #ccc;

}

**CSS Frameworks to Simplify Form Styling**

* **Bootstrap**, **Tailwind CSS**, and **Foundation** provide ready-made, cross-browser-consistent form styles.
* These libraries reduce development effort while ensuring UI consistency.

**1. <form> Element**

* **Purpose**: Defines an HTML form for user input.
* **Usage**: Contains all input elements like text fields, checkboxes, and buttons.

**2. <input> Element**

* **Purpose**: Creates interactive controls to accept user data.
* **Common type attributes**:
  + text, password, email, number, checkbox, radio, submit, url, date, etc.
* **Example**:

<input type="text" id="uname" name="username">

<input type="password" id="pwd" name="pwd">

* **Note**: Type "password" hides the entered text.

**3. <label> Element**

* **Purpose**: Provides a caption for form controls.
* **Key Attribute**: for — should match the id of the associated input element.
* **Example**:

<label for="uname">Username:</label>

**4. <select> Element**

* **Purpose**: Defines a drop-down list.
* **Attributes**:
  + form, name, multiple, required, size
* **Uses** <option> elements to list choices.

**5. <option> Element**

* **Purpose**: Defines individual options inside <select> or <datalist>.
* **Attributes**:
  + value: Specifies the value submitted.
  + selected: Pre-selects an option.

**6. <textarea> Element**

* **Purpose**: Allows multi-line text input.
* **Attributes**:
  + rows, cols, maxlength, minlength, readonly, form
* **Example**:

<textarea rows="10" cols="30" maxlength="200"></textarea>

**7. <button> Element**

* **Purpose**: Creates a clickable button.
* **Common Attribute**:
  + onclick: Defines the action on click.
* **Example**:

<button onclick="alert('You just clicked!')">Click Me!</button>

**8. <fieldset> Element**

* **Purpose**: Groups related form controls.
* **Use Case**: Separate sections like personal info and education.

**9. <legend> Element**

* **Purpose**: Adds a caption/title to a <fieldset> group.
* **Example**:

<legend>Personal Info</legend>

**10. <datalist> Element**

* **Purpose**: Provides pre-defined options for an <input> field.
* **Difference from <select>**: Allows free text input.
* **Example**:

<input list="flowers">

<datalist id="flowers">

<option value="Rose">

<option value="Tulip">

</datalist>

**11. <output> Element**

* **Purpose**: Displays calculation results or dynamic content (e.g., from JavaScript).

**12. <optgroup> Element**

* **Purpose**: Groups related <option> elements within a <select> list.
* **Attribute**: label defines the group name.

**Topic 4: Media Elements**

**Introduction to Media Embedding in HTML**

* Streaming media (audio/video) is now integral to web experiences.
* HTML provides dedicated tags for embedding media content.

**Embedding Video with <video> Tag**

**Supported Video File Types**

* Most browsers support:
  + .mp4 (MPEG-4)
  + .webm
  + .ogg

**Basic Syntax**

<video width="320" height="240" controls>

<source src="dance.mp4" type="video/mp4">

</video>

**Key Elements**

* <video>: Main container for video content.
* <source>: Specifies the video file and its type.
  + src: Path to the video file.
  + type: MIME type (e.g., video/mp4).

**Fallback Mechanism**

* Browsers select the **first supported format** listed.
* Use multiple <source> elements to ensure compatibility.

**Customization Attributes**

* width and height: Set player dimensions.
* controls: Adds play, pause, volume, etc.
* More attributes available for advanced customization (covered later).

**Embedding Audio with <audio> Tag**

**Supported Audio File Types**

* Most browsers support:
  + .mp3 (MPEG-1 Audio Layer 3)
  + .wav
  + .ogg

**Basic Syntax**

<audio controls>

<source src="audiofile.mp3" type="audio/mpeg">

</audio>

**Key Elements**

* <audio>: Main container for audio content.
* <source>: Specifies audio file and MIME type.
  + Example MIME type for MP3: audio/mpeg.

**Customization Attributes**

* controls: Enables audio control buttons (play, pause, volume).

**Introduction to the <img> Tag**

* The <img> tag is used to embed images into a webpage.
* Images enhance visual appeal and user engagement.

**Basic Image Embedding**

**Syntax**

<img src="photo.png">

* src: Specifies the path or URL of the image file.

**Image Size Customization**

**Setting Dimensions**

<img src="photo.png" width="640" height="480">

* width and height: Control the displayed size in pixels.
* Browsers **scale images automatically** if set dimensions differ from original.

**Example: Scaling by 50%**

<img src="photo.png" width="320" height="240">

**Auto-Aspect Ratio (Scaling Height Automatically)**

<img src="photo.png" width="320">

* Only setting width retains aspect ratio by auto-calculating height.

**Handling Image Load Errors**

**Using alt Attribute**

<img src="photo.png" width="320" alt="My Profile Photo">

* alt: Provides alternative text if the image fails to load.
* Essential for **accessibility** and **SEO**.

**Enhancing Accessibility with <figure> and <figcaption>**

**Example**

<figure>

<img src="photo.png" width="320" alt="My Profile Photo">

<figcaption>A photo of myself on a beach in 2015</figcaption>

</figure>

* <figure>: Groups the image with a caption.
* <figcaption>: Describes the image, improves accessibility for screen readers.

**Supported Image File Formats in Browsers**

* .APNG – Animated PNG
* .AVIF – AV1 Image Format
* .GIF – Graphics Interchange Format
* .JPEG / .JPG – Common photo format
* .PNG – Transparent/High-quality images
* .SVG – Vector graphics (scalable)
* .WEBP – Web-optimized image format

**Introduction to <iframe>**

* <iframe> is an HTML element used to embed another webpage within a current webpage.
* Common uses: maps, videos, advertisements, social media posts.
* Embeds an entire browsing instance (HTML, CSS, JS) from another site.

**Basic Usage of <iframe>**

* Defined using <iframe> tag.
* **src attribute**: Specifies the URL of the page to be embedded.

<iframe src="https://example.com"></iframe>

* **width and height attributes**: Define the size of the iframe.

**Security Concerns & Solutions**

* Iframes can run external JS and may pose **security risks**.
* Always **trust the source** of embedded content.

**Security Attributes**

**1. allow**

* Controls which browser features are available inside the iframe.
* Example:

allow="camera; microphone"

* Other examples:
  + fullscreen – Allow fullscreen mode.
  + geolocation – Access user’s location.

**2. sandbox**

* Restricts iframe behavior for added security.
* Empty value applies **all restrictions**.
* Specific permissions can be selectively enabled:
  + allow-downloads
  + allow-forms
  + allow-modals
  + allow-orientation-lock
  + allow-popups
  + allow-presentation
  + allow-scripts

**Example** – Allow only forms and scripts:

<iframe src="my\_iframe.html" sandbox="allow-forms allow-scripts"></iframe>

**Other Important Attributes**

**name**

* Assigns a name to the iframe.

<iframe name="MyFrame"></iframe>

**referrerpolicy**

* Controls the HTTP referrer information sent.
  + no-referrer, origin, strict-origin, etc.

**srcdoc**

* Embeds inline HTML directly instead of a source URL.

<iframe srcdoc="<p>My inline html</p>"></iframe>

**loading**

* Controls when the iframe should load.
  + eager: load immediately.
  + lazy: load only when visible.

**title**

* Provides a description of the iframe’s content for screen readers (accessibility).

<iframe title="Location map"></iframe>

**✅Best Practices for Using <iframe>**

* Use sandbox and allow attributes to minimize security risks.
* Use title for accessibility.
* Prefer loading="lazy" for better performance on long pages.
* Use srcdoc for embedding small pieces of static HTML without making a network request.

**Introduction to Web Graphics Programming**

* Graphics programming in browsers powers animations and video games.
* Technologies underpinning web-based graphics include: **GIF**, **WebP**, **2D Canvas**, and **WebGL**.

**Web Graphics Technologies Overview**

**1. GIF (Graphics Interchange Format)**

* Supports animated images without interactivity.
* Commonly used for embedding simple animated content.

**2. WebP Format**

* Modern alternative to GIF.
* Supports both static and animated images.

**3. 2D Canvas**

* HTML5 element used for rendering 2D graphics in browsers.
* Popular for animations and 2D video games.
* Graphics are controlled via **JavaScript**.

**4. WebGL (Web Graphics Library)**

* Enables advanced **2D and 3D rendering**.
* Utilizes the **GPU (Graphics Processing Unit)** for performance-intensive graphics.
* Supports effects like **lighting simulation** and rendering **complex 3D worlds**.
* More complex than 2D Canvas and requires deeper understanding.

**GPU Usage in Web Graphics**

* **WebGL** always uses the GPU.
* **2D Canvas** may or may not use GPU; behavior is browser-dependent.

**Basic 2D Canvas Animation: Bouncing Ball Example**

**HTML Setup (index.html)**

* Include <canvas> element with:
  + width="640"
  + height="480"
  + id="my\_canvas"
* Link JavaScript file:  
  <script src="animation.js"></script>

**JavaScript Animation (animation.js)**

* **Canvas Initialization**:
  + Access canvas: document.getElementById("my\_canvas")
  + Get 2D context: canvas.getContext("2d")
* **Variables Setup**:
  + Define position, speed, movement, and radius of the ball.
* **Core Functions**:
  + moveBall(): Updates position and reverses direction at canvas edges.
  + drawBall(): Clears canvas and redraws the ball using:
    - clearRect, beginPath, arc, fill
* **Animation Loop**:
  + animate() function:
    - Calls moveBall() and drawBall()
    - Uses window.requestAnimationFrame(animate) to keep updating the frame.
  + Start animation loop with initial call to requestAnimationFrame.

**Final Notes**

* This is a simple example of 2D canvas capabilities.
* Extensive possibilities for animations and game development.
* Multiple JavaScript **libraries** are available to simplify canvas-based development (listed in additional resources).

**Module 2: CSS**

**Topic 1: Creating layouts**

**✅ Introduction to CSS**

* CSS = Cascading Style Sheets.
* Enhances **visual appearance** of web pages.
* Enabled by browsers early for **design creativity** and layout control.

**✅ Evolution and Role of CSS in Web Development**

* CSS evolved with browser capabilities.
* Supports **responsive design** and **modern layout systems**.
* Introduced layout tools like:
  + **Flexbox**
  + **Grid**
  + **Box models**

**✅ CSS Capabilities**

* Customizes:
  + Fonts
  + Colors
  + Sizes
  + Layouts
  + Styling formats
* Targets the **viewport** (visible area in the browser) for better responsiveness.

**✅ CSS Layout Fundamentals**

* Objective: Create well-structured, visually appealing pages.
* Layouts organize content into **sections** for readability and structure.
* **Box models** used to allocate rectangular regions to HTML elements.

**✅ Display Property in CSS**

* Controls **how elements are displayed** (inline vs. block).
* Key in managing layouts.
* Example usage:

#sample {

display: block;

}

**✅ Flexbox Layout**

* Short for **Flexible Box Layout**.
* Designed for **one-dimensional layouts** (row or column).
* Enables:
  + Element flexibility
  + Responsive alignment
  + Simplified item arrangement
* Sample code:

#sample {

display: flex;

}

**✅ Grid Layout**

* Supports **two-dimensional layouts** (rows & columns).
* Ideal for **complex page structures**.
* Offers precise **element placement** across grid lines.
* Sample code:

#sample {

display: grid;

}

**✅ Flexbox vs Grid: When to Use**

* **Flexbox**:
  + Best for **simple, single-axis** alignment.
  + Suitable for **smaller UI components**.
* **Grid**:
  + Ideal for **large-scale layouts**.
  + Preferred when both row and column control is needed.

**✅ Creative Use of CSS Layouts**

* It’s common to **mix layouts** (Flexbox + Grid) in modern design.
* Rules are **standardized**, yet flexible for:
  + **Creative expression**
  + **Performance optimization**
  + **Aesthetic customization**

✅ **Understanding Flexbox**

* Flexbox is a container model similar to div and box, but offers better scalability and dynamic control.
* It allows items inside to **grow, shrink, and align**, giving the programmer better control over content layout.

✅ **Core Characteristics**

* **Single-dimensional** layout: align items in **row** (default) or **column**.
* Uses two axes:
  + **Main axis**: row by default (horizontal).
  + **Cross axis**: perpendicular to main (vertical by default).
* With flex-direction, the main axis can be changed to vertical.

✅ **Basic Flexbox Structure**

<div class="flex-container">

<div class="box box1">One..</div>

<div class="box box2">Two..</div>

...

</div>

✅ **Box Styling (CSS)**

.box {

background-color: aquamarine;

border-radius: 5px;

margin: 2px;

padding: 10px;

}

✅ **Converting to Flexbox**

.flex-container {

display: flex;

}

* Items now align left to right in a single row.

✅ **Main Alignment Properties**

* justify-content: Aligns items on the **main axis**.
* align-items: Aligns items on the **cross axis**.
* align-self: Applies to **individual items** on the cross axis.
* align-content: Manages spacing of multiple lines.

✅ **Using justify-content**

.flex-container {

display: flex;

justify-content: center;

}

* Centers items along the main axis.

✅ **Using flex-wrap**

.flex-container {

display: flex;

justify-content: center;

flex-wrap: wrap;

}

* Allows items to wrap within the viewport.

✅ **Using flex-direction**

.flex-container {

display: flex;

flex-direction: column;

flex-wrap: wrap;

}

* Changes layout from row to column.

✅ **Using align-items**

.flex-container {

display: flex;

align-items: flex-end;

}

* Aligns all items to the **end** of the cross axis (right side if in row layout).

✅ **Using align-self**

.box3 {

background-color: blanchedalmond;

align-self: center;

}

* Overrides align-items for this specific item only.

✅ **Using gap**

.flex-container {

gap: 10px;

}

* Adds spacing between flex items. Also supports row-gap and column-gap.

✅ **Using flex Shorthand**

.box3 {

flex: 1 1 auto;

}

* flex-grow: 1 → allows it to grow and take available space.
* flex-shrink: 1 → allows it to shrink if needed.
* flex-basis: auto → sets initial size.

✅ **Effect of flex Values**

* If box3 has flex: 1 1 auto, it will expand to fill available space.
* If set to a higher number (e.g. 1.5), it will take up more space than others.

**✅ CSS Units of Measurement**

* A web page has **2D dimensions**: width and height (x and y axes).
* CSS allows expressing sizes with **various units** for flexibility and responsiveness.
* Units are categorized as:
  + **Absolute Units**
  + **Relative Units**

**✅ Absolute Units**

* **Definition**: Fixed-size units that remain constant across devices.
* **Use Case**: Best for **printing** or **static layouts** with known dimensions.
* **Limitation**: Not suitable for responsive design across different device sizes.

**📌 Common Absolute Units**

|  |  |  |
| --- | --- | --- |
| Unit | Name | Comparison |
| Q | Quarter-millimeters | 1Q = 1/40th of 1cm |
| mm | Millimeters | 1mm = 1/10th of 1cm |
| cm | Centimeters | 1cm = 37.8px = 25.2/64in |
| in | Inches | 1in = 2.54cm = 96px |
| pc | Picas | 1pc = 1/6th of 1in |
| pt | Points | 1pt = 1/72nd of 1in |
| px | Pixels | 1px = 1/96th of 1in |

* **Most commonly used**: px, cm

**✅ Relative Units**

* **Definition**: Units defined **in relation to parent element, root element, or viewport**.
* **Use Case**: Preferred for **responsive design** and modern web development.

**📌 Common Relative Units**

|  |  |
| --- | --- |
| Unit | Description |
| em | Relative to the font size of the parent element |
| ex | Height of the font’s lowercase ‘x’ |
| ch | Width of the ‘0’ character |
| rem | Relative to the font size of the root element (<html>) |
| lh | Relative to the line height of the parent element |
| rlh | Relative to the line height of the root element |
| vw | 1% of the viewport width |
| vh | 1% of the viewport height |
| vmin | 1% of the smaller viewport dimension |
| vmax | 1% of the larger viewport dimension |
| % | Percentage of the parent element's value |

* **Common usage**: em, rem, vh, vw, %
* **Recommendation**: Use based on context—e.g., vw/vh for layouts, em/rem for font sizing.

**✅ Unit Choice Depends on Property**

* Different CSS properties accept different types of values.
* **Example**:
  + Color properties accept values like: #hex, rgb(), rgba(), hsl(), hsla().
* **Tip**: Understand the context and property type to choose the most suitable unit.

**✅ Introduction to Flexbox Usage**

* Flexbox is ideal for simple layouts and components on a web page.
* Common design elements using Flexbox include:
  + Search bars
  + Navigation bars
  + Image galleries

**✅ Practical Use Cases of Flexbox**

**🔸 1. Creating a Search Bar**

* Flex ties elements like the search icon, input box, and submit button.
* Steps to implement:
  + Link CSS in the <head> of the HTML file.
  + Inside <body>, use a container <div> with:
    - Search icon
    - Search box
    - Submit button
* Key CSS properties:
  + display: inline-flex; — makes the container act like an inline element.
  + overflow — clips overflowing content (e.g., long text in input).
* Result:
  + Search bar layout becomes more structured.
  + Layout does **not** change with screen size.

**🔸 2. Building a Navigation Bar**

* Navigation menus are a frequent use case for Flexbox.
* Structure:
  + Use an unordered list <ul> with list items <li>.
* Key CSS strategies:
  + Use \* (universal selector) to reset browser-specific defaults.
  + Use flex-flow — sets direction and wrapping behavior.
  + justify-content: stretch; — aligns items along the main axis.
* Responsive behavior:
  + Items stack vertically on small screens.
  + Items appear horizontally on wider screens.

**🔸 3. Designing a Responsive Image Gallery**

* Gallery includes a container <div> with multiple <img> tags.
* CSS Implementation:
  + Reset margin, padding, and borders using universal selector.
  + display: flex; for the container.
  + flex-wrap: wrap; — allows images to flow onto new lines.
  + justify-content: space-between; — distributes images evenly.
* Outcome:
  + Images stack on smaller windows.
  + Images spread out responsively on larger screens.

**✅ Key Flexbox Properties Used**

* display: flex / inline-flex — establishes a flex context.
* flex-wrap — controls whether items stay in a single line or wrap.
* justify-content — aligns items horizontally.
* flex-flow — shorthand for flex-direction and flex-wrap.

**✅ Introduction to Flexbox in Data Visualization**

* Flexbox offers versatility beyond layout control—it can be used for creating bar charts.
* Bar charts help visualize data for easier understanding and comparison.
* Flexbox enables creation of both **vertical and horizontal bar charts**, also known as **flex charts**.

**✅ Key Flex Properties for Charts**

* The flex property is critical and consists of three sub-properties:  
  • **Flex Grow** – defines how much an item grows relative to others.  
  • **Flex Shrink** – defines how much an item shrinks relative to others.  
  • **Flex Basis** – defines the initial size before growing/shrinking.
* **Shorthand Syntax:**  
  • Example: flex: 0 1 auto – default values for grow, shrink, and basis.

**✅ Importance of CSS Grid Layouts**

* CSS Grids help design user-friendly, organized web layouts.  
  ▪️ Provide visual cues and structure to users.  
  ▪️ Make content easier to understand and navigate.

**✅ What is a CSS Grid?**

* A two-dimensional layout system for the web.  
  ▪️ Allows arrangement in both rows and columns.  
  ▪️ More suitable for large-scale layouts compared to Flexbox or tables.  
  ▪️ Responsive and browser-compatible.

**✅ Basic Grid Terminology**

* **Columns**: Vertical tracks.
* **Rows**: Horizontal tracks.
* **Gutters (Gaps)**: Spaces between columns and rows.
* **Cell**: The intersection of a row and column.

**✅ Creating a Basic Grid Layout**

* Start with an index.html displaying unstyled content (A to E).  
  ▪️ Initial layout appears vertical without grid styling.

**✅ Applying CSS Grid**

* Set display: grid on the container class.  
  ▪️ Grid defined using grid-template-columns and grid-template-rows.  
  ▪️ Example: 3 columns and 2 rows.  
  ▪️ Before grid: Items take unnecessary space and appear unstructured.

**✅ Using fr Units for Flexibility**

* fr stands for "fraction".  
  ▪️ Allows dynamic allocation of space in columns/rows.  
  ▪️ Example: 2fr 1fr divides space in a 2:1 ratio.  
  ▪️ Can be combined with pixel units.

**✅ Enhancing Grid with More Properties**

* Add grid-gap and background-color.  
  ▪️ Example: 10px grid gap improves spacing.  
  ▪️ Background helps visually identify grid structure.

**✅ Implicit Grid Properties**

* Use grid-auto-rows and grid-auto-columns to let the grid auto-resize.  
  ▪️ Example: grid-auto-rows: 100px resizes each row automatically.

**✅ Useful CSS Grid Functions**

**🔹 repeat() Function**

* Reduces redundancy in defining grid tracks.  
  ▪️ Example: repeat(3, 1fr) instead of 1fr 1fr 1fr.  
  ▪️ Improves readability and maintainability.

**🔹 minmax() Function**

* Sets the minimum and maximum size for grid tracks.  
  ▪️ Example: minmax(100px, 1fr) for adaptive sizing.  
  ▪️ Useful for responsive designs.

**✅ Grid Frameworks**

* Predefined grid structures used for layout design.

**🔹 12-Column and 16-Column Grids**

* Divide the page into fixed column tracks.  
  ▪️ Commonly used in responsive design systems.  
  ▪️ Allows precise control over layout sections.

**✅ Grid Container Properties**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Values / Syntax | Description | Example |
| display | grid | Defines a grid container | display: grid; |
| grid-template-columns | 100px 1fr 2fr | Defines column sizes | grid-template-columns: 100px 1fr 2fr; |
| grid-template-rows | auto 200px | Defines row sizes | grid-template-rows: auto 200px; |
| grid-template-areas | "header header" "main sidebar" | Named layout areas | grid-template-areas: "header header" "main sidebar"; |
| grid-gap | 10px | Sets row and column gaps | grid-gap: 10px; |
| grid-auto-flow | row | column | Controls auto item placement direction | grid-auto-flow: row; |

✅ **Grid Container Example**:

.container {

display: grid;

grid-template-columns: 200px 1fr;

grid-gap: 10px;

}

**✅ Grid Item Properties**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Values / Syntax | Description | Example |
| grid-column | 1 / 3 | Spans item across columns | grid-column: 1 / 3; |
| grid-row | 2 / 4 | Spans item across rows | grid-row: 2 / 4; |
| justify-self | center | Aligns item horizontally | justify-self: center; |
| align-self | end | Aligns item vertically | align-self: end; |

✅ **Grid Item Example**:

.item {

grid-column: 1 / 3;

justify-self: center;

}

**✅ Grid Alignment (Container)**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Values | Description | Example |
| justify-items | center | Aligns items horizontally | justify-items: center; |
| align-items | stretch | Aligns items vertically | align-items: stretch; |
| justify-content | space-between | Aligns the grid tracks horizontally | justify-content: space-between; |
| align-content | center | Aligns the grid tracks vertically | align-content: center; |

✅ **Grid Alignment Example**:

.container {

display: grid;

justify-content: space-between;

align-items: center;

}

**✅ Flexbox Container Properties**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Values | Description | Example |
| display | flex | Declares a flex container | display: flex; |
| flex-direction | row | column | Sets main axis direction | flex-direction: row; |
| flex-wrap | wrap | nowrap | Allows wrapping | flex-wrap: wrap; |
| justify-content | center | space-between | Aligns items along main axis | justify-content: center; |
| align-items | stretch | center | Aligns items along cross axis | align-items: center; |

✅ **Flexbox Container Example**:

.flexbox { display: flex;

flex-direction: row;

justify-content: space-between;

}

**✅ Flexbox Item Properties**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Values / Syntax | Description | Example |
| flex-grow | 1 | Item grows to fill space | flex-grow: 1; |
| flex-shrink | 0 | Item won’t shrink below base size | flex-shrink: 0; |
| flex-basis | 200px | Initial size before growing/shrinking | flex-basis: 200px; |
| order | 2 | Changes order of items | order: 2; |
| align-self | center | Overrides container's alignment | align-self: center; |

✅ **Flexbox Item Example**:

.item {

flex-grow: 1;

align-self: flex-end;}

**Topic 2: CSS Selectors**

**✅ Revision of Previously Learned CSS Selectors**

* **Element (Type) Selector**
  + Targets HTML elements based on their tag name.
  + Example: p { } targets all paragraph tags.
* **ID Selector**
  + Uses the unique id attribute to style a specific element.
  + Example: #header { } targets an element with id="header".
* **Class Selector**
  + Targets elements with a specific class attribute.
  + Example: .button { } applies styling to all elements with class="button".

**✅ Newly Introduced CSS Selectors**

**🔹 Attribute Selectors**

* Match elements based on specific attribute and its value.
* Syntax Variations:
  + [class] targets elements that have a class.
  + [href\*="meta"] targets all elements whose href contains the word “meta”.
  + [href="https://meta.com"] targets elements with an exact attribute value.
* **Example Use Case:**
  + You have three <a> tags with different classes or href values.
  + Use attribute selectors to style specific links or a group based on shared attributes.

**🔹 nth-of-type and nth-child Selectors**

* Used to target specific children within a parent container.
* Syntax:
  + :nth-child(n) targets the nth child of a parent, regardless of type.
  + :nth-of-type(n) targets the nth element of the same type.
* **Example Use Case:**
  + In an unordered list (<ul>), you can color the second <li> using either selector.
  + Output: The second list item appears with the applied styling.

**🔹 Universal (Star) Selector**

* Uses \* to target **all elements** in the document.
* Helpful for:
  + Resetting default browser styles.
  + Applying global styles quickly.

**🔹 Group Selectors (Selector Stacking)**

* Apply the same styles to multiple elements in one rule.
* Syntax: h1, p { } applies the same CSS rules to both h1 and p tags.
* **Benefits:**
  + Saves time.
  + Reduces repetition in CSS code.

**✅ Importance of Mastering CSS Selectors**

* Provides **precision and flexibility** in styling.
* Helps build **clean, scalable, and reusable** CSS.
* Supports development of **responsive and well-structured** web pages.