Unit II: Hyper Text Markup Language

ℹ️ 2.1 Introduction to HTML; Elements of HTML Document; HTML Elements and HTML Attributes: Headings, Paragraph, Division, Comments in HTML

Introduction to HTML

📌 What is HTML?

HTML (HyperText Markup Language) is the standard language used to create and structure web pages.

- **HyperText:** Refers to text that contains links (hyperlinks) to other web pages.
- **Markup Language:** Uses tags (like `<p>`, `<h1>`, `<a>`) to define elements on a webpage.

🏗️ Basic Structure of an HTML Document:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Webpage</title>
  </head>
  <body>
    <h1>Welcome to My Page!</h1>
    <p>This is a paragraph of text.</p>
  </body>
</html>
```

🔍 Explanation:

Tag	Description
`<!DOCTYPE html>`	Declares the document type (HTML5)
`<html>`	Root element of the webpage
`<head>`	Contains metadata (info about the page: title, links, styles)
`<title>`	Sets the title of the page (appears in browser tab)
`<body>`	Contains all the content (text, images, links) shown on the page
`<h1>`	Heading level 1 (largest heading)
`<p>`	Paragraph element

🧩 HTML Elements & Attributes:

🔷 HTML Element = Start Tag + Content + End Tag

Example:

```
<p>This is a paragraph.</p>
```

🔷 HTML Attributes = Extra information about an element (always in start tag)

Example:

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

Attribute	Description
`href`	Specifies URL for the link
`src`	Specifies source (used in images, videos)
`alt`	Alternative text for images
`id`	Unique identifier for an element
`class`	Categorizes elements for styling (CSS/JS usage)

🏷️ Commonly Used Elements:

Element	Purpose
`<h1>` to `<h6>`	Headings (h1 = largest, h6 = smallest)
`<p>`	Paragraph of text
`<div>`	Division/section container
`<span>`	Inline container, used for styling specific parts
` `	Line break (self-closing)
`<!-- -->`	Comment in HTML (ignored by browser)

📚 Quick Recap Table:

Term	Meaning
HTML	HyperText Markup Language
Element	Defined by a start tag, content, and end tag
Attribute	Provides additional info (placed inside start tag)
Metadata	Info about the page (inside `<head>`)
Container Tags	Tags with both opening and closing tags (`<p> ... </p>`)
Empty Tags	Self-closing tags (` `, `<img>`)

📝 Practice Questions:

1. **Write the basic HTML structure for a webpage that has a title "My Website" and a heading "Hello World!".**
2. **What is the purpose of the `alt` attribute in an `<img>` tag?**
3. **Explain the difference between a `<div>` and a `<span>` element.**

🌟 2.2 HTML Formatting Elements

📌 What are HTML Formatting Elements?

Formatting elements are used to define the appearance and importance of text on a web page, like making text **bold**, *italic*, smaller, superscript, etc.

🏷️ Common Formatting Tags:

Tag	Purpose	Example Output
`<b>`	Makes text **bold**	**Bold Text**
`<strong>`	Strong importance, usually shown as bold	**Strong Text (semantically strong)**
`<i>`	Makes text *italic*	*Italic Text*
`<em>`	Emphasizes text, usually italic	*Emphasized Text*
`<small>`	Makes text smaller	Smaller Text
`<sup>`	Superscript text	X2 → X^2
`<sub>`	Subscript text	H2O → H_2O
`<pre>`	Preserves whitespace and line breaks	Displays preformatted text
` `	Inserts a line break	Breaks to the next line
`<tt>`	(Deprecated) Monospaced (teletype) text	Like typewriter text
`<span>`	Inline container, used with CSS	No visible effect without styling
`<div>`	Block container, used for grouping	No visible effect without styling

📝 Example Usage:

```
<p>This is <b>bold</b> and <i>italic</i> text.</p>
<p>Water formula: H<sub>2</sub>O</p>
<p>Square: x<sup>2</sup></p>

<pre>
```

```
  This    text    preserves
  whitespace and line breaks
</pre>

<p>This is <span style="color:blue;">blue</span> text.</p>
```

🚩 Key Difference Table:

Tag	Visual Effect	Semantic Meaning (Importance)
`<b>`	Bold	No importance
`<strong>`	Bold	Strong importance
`<i>`	Italic	No importance
`<em>`	Italic	Emphasis

🌟 Practice Problem:

1. Write an HTML snippet to display:

```
E = mc^2
```

- where **E = mc²**, the "2" is superscript.

1. Write a paragraph where the word **Important** is bold and emphasized.

📚 Quick Recap Table:

Tag	Function
`<b>`	Bold text (no extra importance)
`<strong>`	Bold with strong importance
`<i>`	Italic text (no extra importance)
`<em>`	Italic with emphasis
`<small>`	Small text
`<sup>`	Superscript
`<sub>`	Subscript
`<pre>`	Preformatted text
` `	Line break
`<span>`	Inline container (styled via CSS)
`<div>`	Block-level container (styled via CSS)

📌 2.3 - Image Element, Anchors, Lists (Ordered, Unordered, Definition)

🌟 Overview

Element	Purpose
`<img>`	To display images in a web page.
`<a>`	To create hyperlinks (anchors) to other pages or sections.
`<ul>`	To create an unordered list (bullets).
`<ol>`	To create an ordered list (numbers/letters).
`<dl>`	To create a definition list (terms and descriptions).

1️⃣ Image Element: `<img>`

📄 Syntax:

```
<img src="image.jpg" alt="Description" width="300" height="200">
```

Attribute	Purpose
`src`	Source path of the image (URL or local path).
`alt`	Alternative text if image cannot load (important for accessibility & SEO).
`width`	Width of image (in pixels or %).
`height`	Height of image (in pixels or %).

✅ Example:

```
<img src="sunset.jpg" alt="Beautiful Sunset" width="400" height="300">
```

2️⃣ Anchor Element: `<a>`

📄 Syntax:

```
<a href="url">Link Text</a>
```

Attribute	Purpose
`href`	Hyperlink Reference (URL or section id).
`target`	Specifies where to open the linked document (`_blank` , etc.).

✅ Examples:

```
<!-- External link -->
<a href="https://www.google.com" target="_blank">Visit Google</a>

<!-- Internal page link -->
<a href="about.html">About Us</a>

<!-- Jump to section -->
<a href="#section1">Go to Section 1</a>
```

3️⃣ Lists

a) Unordered List `<ul>`

Displays items with bullet points.

```
<ul>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ul>
```

---

b) Ordered List `<ol>`

Displays items with numbers or letters.

```
<ol>
  <li>First Step</li>
  <li>Second Step</li>
  <li>Third Step</li>
</ol>
```

---

c) Definition List `<dl>`

Used for terms and definitions.

```
<dl>
  <dt>HTML</dt>
  <dd>HyperText Markup Language</dd>

  <dt>CSS</dt>
```

```
  <dd>Cascading Style Sheets</dd>
</dl>
```

📝 Practice Problems:

1. Create an image element that shows a cat picture with the text "Cute Cat" if the image doesn't load.
2. Create a hyperlink that opens **Wikipedia** in a new tab.
3. Write an ordered list of three of your favorite fruits.
4. Write a definition list defining **CPU** and **RAM**.

🔥 Quick Recap Table:

Element	Syntax Example	Purpose
`<img>`	`<img src="img.jpg" alt="desc">`	Display image, provide alt description.
`<a>`	`<a href="url">Link</a>`	Hyperlink to pages, sections, sites.
`<ul>`	`<ul><li>Item</li></ul>`	Unordered bullet list.
`<ol>`	`<ol><li>Item</li></ol>`	Ordered numbered list.
`<dl>`	`<dl><dt>Term</dt><dd>Definition</dd></dl>`	Definition list with terms & descriptions.

📌 2.4 - Tables (table, tr, td, colspan, rowspan, thead, tbody, tfoot), Frames (iframe)

🌟 Overview

Tag/Attribute	Purpose
`<table>`	Defines a table
`<tr>`	Table row
`<td>`	Table data/cell
`<th>`	Table header cell
`colspan`	Merge columns
`rowspan`	Merge rows
`<thead>`	Groups header content
`<tbody>`	Groups body content
`<tfoot>`	Groups footer content
`<iframe>`	Embeds another web page within the page

Basic Outline

```
<table>
  <thead>
    <tr>
      <th> ... </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td> ... </td>
    </tr>
  </tbody>
</table>
```

1️⃣ HTML Table Basics

📄 Syntax Example:

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

Explanation:

Tag	Purpose
`<table>`	Starts the table
`<tr>`	Defines a row
`<th>`	Table heading (bold, centered)
`<td>`	Table data cell

2️⃣ Colspan and Rowspan

a) Colspan: Merge Columns

```
<table border="1">
  <tr>
    <th colspan="2">Personal Info</th>
  </tr>
  <tr>
    <td>Name</td>
    <td>John</td>
  </tr>
</table>
```

✔️ `colspan="2"` merges two columns.

---

b) Rowspan: Merge Rows

```
<table border="1">
  <tr>
    <td rowspan="2">John</td>
    <td>Math</td>
  </tr>
  <tr>
    <td>Science</td>
  </tr>
</table>
```

✔️ `rowspan="2"` merges two rows.

3️⃣ Table Sections: `<thead>`, `<tbody>`, `<tfoot>`

```
<table border="1">
  <thead>
    <tr>
      <th>Item</th>
      <th>Price</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Apple</td>
      <td>$1</td>
    </tr>
```

```
    <tr>
      <td>Banana</td>
      <td>$2</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Total</td>
      <td>$3</td>
    </tr>
  </tfoot>
</table>
```

Purpose:

- `<thead>`: For header section.
- `<tbody>`: Main body rows.
- `<tfoot>`: Summary/footer (like totals).

4️⃣ Iframe: Embedding Web Pages

📄 Syntax:

```
<iframe src="https://www.example.com" width="600" height="400"></iframe>
```

Attribute	Purpose
`src`	URL of the page to embed
`width`	Width of frame
`height`	Height of frame

✅ Example:

```
<iframe src="https://www.wikipedia.org" width="500" height="300"></iframe>
```

📝 Practice Problems:

1. Create a table with 3 rows and 2 columns showing your favorite books and their authors.
2. Use `colspan` to merge the header of two columns in a table.
3. Create a table using `<thead>`, `<tbody>`, `<tfoot>` for a product list.
4. Embed Google.com inside your webpage using `<iframe>` (just as an example, Google may block it due to security policies).

🔥 Quick Recap Table:

Element/Attribute	Purpose
`<table>`	Defines a table
`<tr>`	Table row
`<td>`	Table cell
`<th>`	Table header
`colspan`	Merge columns
`rowspan`	Merge rows
`<thead>`	Header section
`<tbody>`	Body section
`<tfoot>`	Footer section
`<iframe>`	Embeds external page

📄 2.5 HTML Forms (Form Elements, ID attributes, Class Attributes)

Forms are **crucial** for collecting user input in web applications.

🌟 1. What is an HTML Form?

An **HTML Form** is a section of a document that contains input elements like:

- Text fields
- Checkboxes
- Radio buttons
- Submit buttons

Forms are used to **send data to the server**.

🧩 2. Basic Structure of an HTML Form:

```
<form action="submit.php" method="post">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">

  <br><br>
```

```
  <label for="password">Password:</label>
  <input type="password" id="password" name="password">

  <br><br>

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

🟢 Explanation:

Attribute	Purpose
`action`	Specifies where to send form data (URL/file)
`method`	`GET` (data in URL, for insensitive information) or `POST` (hidden data, for sensitive information)
`label`	Defines a label for an input field (improves accessibility)
`input`	Creates input fields (text, password, submit, checkbox, etc.)
`name`	Important to identify data on the server side
`id`	Uniquely identifies an element (used for JS, CSS, labels)

🎯 3. Common Form Elements:

Element	Purpose	Example
`<input type="text">`	Single-line text field	`<input type="text">`
`<input type="password">`	Password field	`<input type="password">`
`<input type="checkbox">`	Checkbox	`<input type="checkbox">`
`<input type="radio">`	Radio button	`<input type="radio" name="group1">`
`<textarea>`	Multi-line text field	`<textarea></textarea>`
`<select>` + `<option>`	Drop-down menu	`<select><option value = "correspoding_val">Option1</option></select>`
`<button>`	Customizable button	`<button>Click Me</button>`
`<input type="submit">`	Submits the form	`<input type="submit" value="Submit">`
`<fieldset>` & `<legend>`	Group related fields	`<fieldset><legend>Info</legend> ... </fieldset>`

🎨 4. ID and Class Attributes:

Attribute	Purpose	Example
`id`	Uniquely identifies a single element	`<input id="email">`
`class`	Assigns a common style/behavior to multiple elements	`<input class="form-input">`

📝 5. Quick Example:

```
<form action="/submit" method="post">
  <input type="text" id="email" name="email" class="form-input" placeholder="Enter Email">
  <input type="submit" value="Submit">
</form>
```

🟢 6. Practice Questions:

1. Write an HTML form that asks for:
   - First Name (text)
   - Gender (radio buttons: Male, Female)
   - Favorite Color (drop-down with 3 colors)
   - Submit Button
2. What's the difference between `id` and `class` attributes?

Shall I wait for your answers to the practice questions before moving forward? 😊

2.6 Meta Tag, Audio, Video, Canvas, Main, Section, Article, Header, Footer, Aside, Nav, Figure Tags

📄 HTML Meta Tags

🔷 What is a Meta Tag?

- `<meta>` tags are **HTML tags** used inside the `<head>` section of an HTML document.
- They provide **metadata (data about data)** about the web page.
- It doesn't display anything on the web page but plays a crucial role in:
  - **SEO (Search Engine Optimization)**
  - **Character encoding**
  - **Responsive design**
  - **Page description**
  - **Author information**
  - **Auto-refresh settings**

🔷 Syntax:

```
<head>
  <meta attribute="value">
</head>
```

🔷 Common Attributes of `<meta>`:

Attribute	Purpose	Example Usage
`name`	Specifies the name of the metadata	`<meta name="description" content="About page">`
`content`	The actual metadata value	`<meta name="keywords" content="HTML, CSS, Web">`
`charset`	Defines the character encoding	`<meta charset="UTF-8">`
`http-equiv`	Provides HTTP header information	`<meta http-equiv="refresh" content="30">`

🔷 Important Examples:

1. **Character Encoding (UTF-8):**

```
<meta charset="UTF-8">
```

- Ensures the web page supports most characters from all languages.

1. **Page Description (SEO Friendly):**

```
<meta name="description" content="Learn Web Technology Basics">
```

- Search engines display this in search results.

1. **Keywords (Less used today but still relevant):**

```
<meta name="keywords" content="HTML, CSS, JavaScript, Web Design">
```

1. **Author Info:**

```
<meta name="author" content="John Doe">
```

1. **Viewport Settings (Responsive Design):**

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- Ensures proper scaling on mobile devices.

1. **Automatic Page Refresh:**

```
<meta http-equiv="refresh" content="10">
```

- Refreshes the page every 10 seconds.

🔷 Why are Meta Tags Important?

- Improve **Search Engine Rankings (SEO)**
- Control **browser behavior**
- Ensure **responsive web design**
- Help social media platforms generate previews when sharing the page.

🔷 Practice Problem:

Write a meta tag that sets:

1. Author to "Alice Smith"
2. Description to "Best Web Tutorials"
3. Auto-refresh every 15 seconds.

✅ **Answer:**

```
<meta name="author" content="Alice Smith">
<meta name="description" content="Best Web Tutorials">
<meta http-equiv="refresh" content="15">
```

🔷 Recap Table:

Meta Tag Example	Purpose
`<meta charset="UTF-8">`	Sets character encoding
`<meta name="author" content="John">`	Defines author
`<meta name="viewport" content="width=device-width, initial-scale=1.0">`	Responsive scaling
`<meta http-equiv="refresh" content="10">`	Auto-refresh page

🎵 HTML `<audio>` Element

The `<audio>` element is used to embed sound content in documents. It may contain one or more audio sources, represented using the `src` attribute or the `<source>` element: the browser will choose the most suitable one. It can also be the destination for streamed media, using a MediaStream.

MDN Web Docs

🛠️ Attributes of `<audio>`

Here are some common attributes associated with the `<audio>` element:

Attribute	Description

`src`	Specifies the URL of the audio file.
`controls`	Displays audio controls like play, pause, and volume.
`autoplay`	Automatically starts playing the audio when the page loads.
`loop`	Plays the audio repeatedly (loops) after it ends.
`muted`	Mutes the audio by default.
`preload`	Hints how the browser should load the audio: `none` - No data is preloaded. `metadata` - Only metadata is preloaded. `auto` - The entire audio is preloaded.

📝 Example of Using `<audio>`

Here's how you can embed an audio file into your HTML document:

```
html
CopyEdit
<audio controls>
  <source src="audio-file.mp3" type="audio/mpeg">
  <source src="audio-file.ogg" type="audio/ogg">
  Your browser does not support the audio element.
</audio>
```

In this example:

- The `<audio>` element includes the `controls` attribute to provide play, pause, and volume controls.
- Two `<source>` elements are provided for different audio formats (`.mp3` and `.ogg`). The browser will use the first format it supports.
- The text "Your browser does not support the audio element." serves as a fallback for browsers that do not support the `<audio>` element.

🎯 Practice Problem

Task: Embed an audio file named `song.mp3` into a webpage. Ensure that:

- The audio plays automatically when the page loads.
- The audio loops indefinitely.
- Default audio controls are displayed.

Solution:

```
html
CopyEdit
<audio src="song.mp3" controls autoplay loop>
  Your browser does not support the audio element.
</audio>
```

🔄 Recap

- The `<audio>` element allows embedding of audio content in HTML documents.
- Common attributes include `src`, `controls`, `autoplay`, `loop`, `muted`, and `preload`.
- Multiple `<source>` elements can be used to provide different audio formats for broader browser compatibility.

You're right! Let's get back to the detailed, tutor-style explanation I've been using, matching exactly how we've been doing it for your exam prep—not pulling from the web.

📽️ HTML `<video>` Element — Full Explanation

✅ Purpose:

The `<video>` element allows embedding video files directly into a webpage, without needing external plugins (like Flash).

📄 Basic Syntax:

```
<video src="movie.mp4" controls width="500">
  Your browser does not support the video tag.
</video>
```

🔑 Key Attributes:

Attribute	Purpose	Example Usage
`src`	Specifies the path to the video file	`src="video.mp4"`
`controls`	Displays default video controls (play, pause, volume, etc.)	`controls`
`autoplay`	Starts playing automatically when page loads	`autoplay`
`loop`	Restarts the video after it ends	`loop`
`muted`	Starts video with no sound	`muted`
`poster`	Displays an image before video starts (preview)	`poster="thumbnail.jpg"`
`width`	Sets the width of the video player	`width="600"`
`height`	Sets the height of the video player	`height="400"`

📦 Multiple Source Example (for cross-browser support):

```
<video controls width="500">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
```

```
  Your browser does not support the video tag.
</video>
```

Browsers will pick the first supported format.

📝 Practical Example:

```
<video width="600" controls poster="preview.jpg" autoplay loop muted>
  <source src="sample.mp4" type="video/mp4">
  <source src="sample.ogg" type="video/ogg">
  Your browser does not support HTML video.
</video>
```

🧠 Why Use `<video>`:

1. **Direct embedding**: No extra plugins like Flash.
2. **Cross-device compatibility**: Mobile & desktop-friendly.
3. **Customizable controls & behavior**.

📋 Recap Table:

Feature	Description
`<video>` tag	Embeds video content
`controls`	Adds play/pause buttons
`autoplay`	Plays video on page load
`loop`	Restarts video automatically
`muted`	Starts video with sound off
`poster`	Shows an image preview before video plays
`<source>`	Provides multiple file formats for compatibility
Fallback text	Displays message if video isn't supported

🎯 Practice Task:

Write an HTML code snippet to:

- Embed a video named `nature.mp4`
- Display a preview image `nature.jpg`
- Autoplay the video, loop it, and mute it by default
- Include fallback text

🎨 HTML `<canvas>` Element

The `<canvas>` element provides a draw-able region defined in HTML code with height and width attributes. JavaScript can access this area through a full set of drawing functions, enabling the creation of graphs, animations, games, and image compositions.

Wikipedia

🛠️ Attributes of `<canvas>`

Here are the primary attributes associated with the `<canvas>` element:

Attribute	Description
`width`	Specifies the width of the canvas in pixels. Default is 300 pixels.
`height`	Specifies the height of the canvas in pixels. Default is 150 pixels.

Note: If the `width` and `height` attributes are not specified, the canvas will default to 300 pixels wide and 150 pixels high. It's recommended to set these attributes to match the desired drawing surface size to avoid unexpected scaling.

Wikipedia

📝 Basic Usage Example

To utilize the `<canvas>` element, follow these steps:

1. **Define the Canvas in HTML:**

```
<canvas id="myCanvas" width="500" height="400">
  Your browser does not support the canvas element.
</canvas>
```

In this example, a canvas with an ID of "myCanvas" is created, with a width of 500 pixels and a height of 400 pixels. The fallback text "Your browser does not support the canvas element." will display for browsers that do not support the `<canvas>` element.

1. **Access and Draw on the Canvas Using JavaScript:**

```
<script>
  var canvas = document.getElementById('myCanvas');
  var context = canvas.getContext('2d');

  // Set fill color to blue
  context.fillStyle = 'blue';

  // Draw a rectangle
  context.fillRect(50, 50, 150, 100);
```

```
</script>
```

In this script:

- The canvas is accessed via its ID using `getElementById`.
- The `getContext('2d')` method obtains the 2D rendering context, which provides methods and properties for drawing.
- The `fillStyle` property sets the color used to fill shapes.
- The `fillRect(x, y, width, height)` method draws a filled rectangle starting at (50, 50) with a width of 150 pixels and a height of 100 pixels.

🎯 Practice Problem

Task: Create a canvas element that displays a red circle with a radius of 75 pixels, centered in the canvas.

Solution:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Canvas Circle Example</title>
</head>
<body>
  <canvas id="circleCanvas" width="400" height="400">
    Your browser does not support the canvas element.
  </canvas>

  <script>
    var canvas = document.getElementById('circleCanvas');
    var context = canvas.getContext('2d');

    // Set fill color to red
    context.fillStyle = 'red';

    // Draw a circle
    context.beginPath();
    context.arc(200, 200, 75, 0, 2 * Math.PI);
    context.fill();
  </script>
</body>
```

```
</html>
```

In this example:

- A canvas with an ID of "circleCanvas" is created, with both width and height set to 400 pixels.
- The 2D rendering context is obtained.
- The `fillStyle` property sets the fill color to red.
- The `beginPath()` method starts a new path.
- The `arc(x, y, radius, startAngle, endAngle)` method creates an arc/curve. Here, it draws a circle centered at (200, 200) with a radius of 75 pixels.
- The `fill()` method fills the current drawing (the circle) with the specified fill color.

🔄 Recap

- The `<canvas>` element is used to draw graphics via scripting, typically JavaScript.
- It requires the `width` and `height` attributes to define the size of the drawable area.
- JavaScript provides the methods to draw shapes, text, images, and more on the canvas.

HTML `<main>` Element

🌟 What is `<main>`?

The `<main>` tag is a **semantic HTML5 element** used to wrap the **main content** of the web page. The content inside `<main>` is unique to that particular page and is **directly related to the central topic or purpose of the document**.

Key Properties:

- There should **only be one** `<main>` **element** per page.
- It should **not contain** content that repeats across pages like headers, footers, sidebars, or navigation links.
- Improves **SEO** and **accessibility** (screen readers can jump directly to the main content).

🧩 Syntax:

```
<main>
  <!-- Main content of the page -->
</main>
```

🎯 Where is it used?

The `<main>` element is typically placed **inside the** `<body>`, but **outside of** `<header>`, `<footer>`, `<nav>`, `<aside>` elements.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Post</title>
</head>
<body>

<header>
  <h1>My Blog</h1>
  <nav>
    <a href="#">Home</a> |
    <a href="#">About</a>
  </nav>
</header>

<main>
  <article>
    <h2>What is HTML?</h2>
    <p>HTML stands for HyperText Markup Language...</p>
  </article>
</main>

<footer>
  <p>&copy; 2025 My Blog</p>
</footer>

</body>
</html>
```

What's inside `<main>` here?

- The **blog post article**, which is the central content.

🚀 Why Use `<main>`?

Benefit	Explanation
Semantic clarity	Clearly identifies the primary content.

Accessibility	Screen readers can skip directly to main content.
SEO Friendly	Helps search engines understand the key content of the page.
Cleaner structure	Separates reusable elements (header, nav, footer) from unique content.

❌ Common Mistakes:

Mistake	Why it's wrong
Using multiple `<main>` tags	Only one `<main>` per page is valid.
Wrapping `<main>` inside `<header>`, `<footer>`, or `<nav>`	Should be separate and not part of repeated elements.
Using `<main>` for sidebar content	Sidebar belongs in `<aside>`, not `<main>`.

💡 Practice Problem:

Problem:

Create an HTML page for a company that includes:

1. A header with the company name.
2. Navigation links.
3. A `<main>` section with a product description.
4. A footer with copyright.

Solution:

```
<!DOCTYPE html>
<html>
<head>
  <title>ABC Company</title>
</head>
<body>

<header>
  <h1>ABC Company</h1>
  <nav>
    <a href="#">Home</a> |
    <a href="#">Products</a> |
    <a href="#">Contact</a>
  </nav>
</header>

<main>
  <h2>Our Latest Product</h2>
  <p>Introducing our latest gadget that simplifies your life...</p>
</main>
```

```
<footer>
  <p>&copy; 2025 ABC Company</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<main>`	Wraps the unique, central content of the page
Allowed occurrences per page	Only **one** `<main>` **element per page**
Should NOT contain	Header, Footer, Nav, Sidebar content
Improves	Accessibility, SEO, Semantic clarity
Placement	Inside `<body>`, outside `<header>`, `<footer>`, `<nav>`, `<aside>`

HTML `<section>` Element

🌟 What is `<section>`?

The `<section>` element is a **semantic HTML5 tag** used to define distinct sections within a document. Each section typically groups related content under a common theme, often accompanied by a heading.

Key Characteristics:

- Represents a **thematic grouping** of content.
- Typically includes a **heading** (`<h1>` to `<h6>`).
- Enhances **semantic structure**, aiding both accessibility and SEO.

🧩 Syntax:

```
<section>
  <!-- Thematic content goes here -->
</section>
```

🎯 When to Use `<section>`?

Use the `<section>` element to organize content into **thematically related groups**. Examples include:

- Chapters in a book.
- Sections of a report.

- Grouped articles or news items.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Company Overview</title>
</head>
<body>

<header>
  <h1>Our Company</h1>
</header>

<main>
  <section>
    <h2>About Us</h2>
    <p>We are a leading firm in the industry...</p>
  </section>

  <section>
    <h2>Services</h2>
    <p>We offer a range of services including...</p>
  </section>

  <section>
    <h2>Contact Information</h2>
    <p>Reach out to us at...</p>
  </section>
</main>

<footer>
  <p>&copy; 2025 Our Company</p>
</footer>

</body>
</html>
```

Explanation:

- Each `<section>` groups content under a specific theme with a corresponding heading.

🚀 Benefits of Using `<section>`:

Benefit	Explanation
Semantic Structure	Clarifies the document's organization.
Accessibility	Assists screen readers in navigating content.
SEO Enhancement	Helps search engines understand content hierarchy.
Maintainability	Simplifies content management and styling.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Using `<section>` without a heading	Sections should have a heading to define their purpose.
Overusing `<section>` for non-thematic content	Use `<div>` for generic grouping without thematic meaning.
Nesting `<section>` elements excessively	Can lead to overly complex structures; use judiciously.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. A header with the title "My Portfolio".
2. A `<main>` section containing:
    - An "About Me" section with a brief introduction.
    - A "Projects" section listing two projects with descriptions.
    - A "Contact" section with an email address.

Solution:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Portfolio</title>
</head>
<body>

<header>
  <h1>My Portfolio</h1>
</header>

<main>
  <section>
    <h2>About Me</h2>
    <p>I am a web developer with a passion for creating interactive applications...</p>
  </section>
```

```
  <section>
    <h2>Projects</h2>
    <article>
      <h3>Project One</h3>
      <p>A web application that allows users to track their tasks...</p>
    </article>
    <article>
      <h3>Project Two</h3>
      <p>An e-commerce platform for local artisans...</p>
    </article>
  </section>

  <section>
    <h2>Contact</h2>
    <p>Email: me@example.com</p>
  </section>
</main>

<footer>
  <p>&copy; 2025 My Portfolio</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<section>`	Groups related thematic content within a document.
Typical Content	Includes a heading and related content.
Difference from `<div>`	`<section>` adds semantic meaning; `<div>` is purely structural.
Placement	Within `<main>`, `<article>`, or other structural elements.
Best Practices	Always include a heading; use for distinct thematic groups.

HTML `<article>` Element

🌟 What is `<article>`?

The `<article>` element is a semantic HTML5 tag that represents a self-contained piece of content intended to be independently distributable or reusable. This means the content within an `<article>` should make sense on its own and be suitable for distribution, such as syndication.

Key Characteristics:

- **Self-Contained:** The content is complete and understandable without additional context.
- **Independently Distributable:** Suitable for sharing or republishing elsewhere.
- **Semantic Meaning:** Enhances the document's structure, aiding accessibility and SEO.

🧩 Syntax:

```
<article>
  <!-- Self-contained content goes here -->
</article>
```

🎯 When to Use `<article>`?

Use the `<article>` element for content that forms an independent part of a document or site. Examples include:

- **Blog Posts:** Each post can be an `<article>`.
- **News Stories:** Individual news items.
- **Forum Posts:** Each discussion entry.
- **User Comments:** Standalone comments or reviews.
- **Product Cards:** Individual product descriptions.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Blog Page</title>
</head>
<body>

<header>
  <h1>My Tech Blog</h1>
</header>

<main>
  <article>
    <h2>The Rise of Quantum Computing</h2>
    <p>Quantum computing is set to revolutionize technology...</p>
  </article>

  <article>
    <h2>Understanding Artificial Intelligence</h2>
```

```
    <p>Artificial Intelligence (AI) has become integral to modern applications...</p>
  </article>
</main>

<footer>
  <p>&copy; 2025 My Tech Blog</p>
</footer>

</body>
</html>
```

Explanation:

- Each `<article>` contains a distinct topic with its own heading and content, making them independently understandable.

🚀 Benefits of Using `<article>`:

Benefit	Explanation
Semantic Structure	Clarifies the document's organization.
Accessibility	Assists screen readers in navigating content.
SEO Enhancement	Helps search engines understand content hierarchy.
Reusability	Content can be syndicated or republished easily.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Using `<article>` for non-independent content	`<article>` is meant for self-contained content.
Omitting headings within `<article>`	Headings provide context and improve accessibility.
Over-nesting `<article>` elements	Can lead to confusing structure; use nesting appropriately.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. A header with the title "News Portal".
2. A `<main>` section containing:
   - An article titled "Tech Innovations 2025" with a brief description.
   - An article titled "Health Advances in the Modern Age" with a brief description.

Solution:

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

```
<head>
  <title>News Portal</title>
</head>
<body>

<header>
  <h1>News Portal</h1>
</header>

<main>
  <article>
    <h2>Tech Innovations 2025</h2>
    <p>The year 2025 brings groundbreaking technological advancements...</p>
  </article>

  <article>
    <h2>Health Advances in the Modern Age</h2>
    <p>Modern medicine has achieved remarkable progress in recent years...</p>
  </article>
</main>

<footer>
  <p>&copy; 2025 News Portal</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<article>`	Represents self-contained, independent content.
Typical Content	Blog posts, news articles, forum posts, user comments.
Difference from `<section>`	`<article>` is for independent content; `<section>` groups related content without implying independence.
Placement	Within `<main>`, `<section>`, or other structural elements.
Best Practices	Ensure content is self-contained; include appropriate headings.

HTML `<header>` Element

🌟 What is `<header>`?

The `<header>` element is a semantic HTML5 tag used to define introductory or navigational content for a section or the entire webpage. It typically contains:

- **Headings (`<h1>` – `<h6>`):** Titles or subtitles of the content.
- **Navigation Menus:** Links to other parts of the website.
- **Logos or Branding Elements:** Visual identifiers of the site or company.
- **Search Forms:** Input fields for site-wide searches.
- **Author Information:** Details about the content's creator.

🧩 Syntax:

```
<header>
  <!-- Introductory or navigational content goes here -->
</header>
```

🎯 When to Use `<header>`?

Use the `<header>` element to group introductory content or navigational links for a section or the entire page. Examples include:

- **Page Header:** Contains the site's main navigation, logo, and search bar.
- **Section Header:** Introduces a specific section within the page, like a blog post or a product description.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page</title>
</head>
<body>

<header>
  <h1>My Website</h1>
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
</header>
```

```
<main>
  <section>
    <header>
      <h2>About Us</h2>
    </header>
    <p>Welcome to our company. We specialize in...</p>
  </section>
</main>

<footer>
  <p>&copy; 2025 My Website</p>
</footer>

</body>
</html>
```

Explanation:

- The **page-level** `<header>` includes the site's title and a navigation menu.
- The **section-level** `<header>` introduces the "About Us" section with a heading.

🚀 Benefits of Using `<header>`:

Benefit	Explanation
Semantic Structure	Clarifies the document's organization, aiding both users and search engines.
Accessibility	Helps assistive technologies navigate and interpret the page structure effectively.
SEO Enhancement	Search engines better understand the content hierarchy, potentially improving rankings.
Consistent Styling	Allows for uniform styling of header sections across the site using CSS.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Using multiple `<header>` elements within a single section	Each section should have only one `<header>` to maintain clear structure.
Placing `<header>` inside `<footer>`	Semantically incorrect as headers introduce content, while footers conclude it.
Omitting `<header>` for main navigation	Using `<header>` for main navigation enhances clarity and accessibility.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. A header containing:

- The website's logo.
- A navigation menu with links to "Home," "Services," and "Contact."

2. A main section with:
   - An article titled "Our Services" with a brief description.
3. A footer with:
   - Contact information.

Solution:

```
<!DOCTYPE html>
<html>
<head>
 <title>Company Services</title>
</head>
<body>

<header>
 <img src="logo.png" alt="Company Logo">
 <nav>
  <ul>
   <li><a href="#">Home</a></li>
   <li><a href="#">Services</a></li>
   <li><a href="#">Contact</a></li>
  </ul>
 </nav>
</header>

<main>
 <article>
  <header>
   <h2>Our Services</h2>
  </header>
  <p>We offer a wide range of services to meet your needs...</p>
 </article>
</main>

<footer>
 <p>Contact us at: info@company.com</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<header>`	Defines introductory or navigational content for a section or page.
Typical Content	Headings, navigation menus, logos, search forms, author info.
Difference from `<head>`	`<header>` is for visible content; `<head>` contains metadata and links to scripts/styles.
Placement	Can be used within `<body>` and various sectioning elements like `<article>`, `<section>`, etc.
Best Practices	Use `<header>` to group introductory content; avoid nesting `<header>` within `<footer>`.

HTML `<footer>` Element

🌟 What is `<footer>`?

The `<footer>` element is a semantic HTML5 tag that designates a footer for its nearest ancestor sectioning content or sectioning root elements. Typically, a `<footer>` contains information about the author, copyright details, contact information, or links to related documents.

🧩 Syntax:

```
<footer>
  <!-- Footer content goes here -->
</footer>
```

🎯 *When to Use `<footer>`?*

Use the `<footer>` element to group footer content for a section or the entire page. Examples inclue:

- *Page Footer:* Contains site-wide information such as copyright details or contact information.
- *Section Footer:* Provides additional information related to a specific section, like an article's author bio or related links.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page</title>
</head>
<body>

<header>
  <h1>My Website</h1>
```

```
    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <article>
      <header>
        <h2>Article Title</h2>
      </header>
      <p>Article content goes here...</p>
      <footer>
        <p>Written by John Doe</p>
      </footer>
    </article>
  </main>

  <footer>
    <p>&copy; 2025 My Website</p>
  </footer>

</body>
</html>
```

Explanation:

- The **page-level** `<footer>` includes the site's copyright information.
- The **article-level** `<footer>` provides the author's name.

🚀 Benefits of Using `<footer>`:

Benefit	Explanation
Semantic Structure	Clarifies the document's organization, aiding both users and search engines.
Accessibility	Assists assistive technologies in navigating and interpreting the page structure effectively.
SEO Enhancement	Helps search engines understand the content hierarchy, potentially improving rankings.
Consistent Styling	Allows for uniform styling of footer sections across the site using CSS.

❌ Common Mistakes:

Mistake	Why It's Incorrect

Using multiple `<footer>` elements within a single section	Each section should have only one `<footer>` to maintain clear structure.
Placing `<footer>` inside `<header>`	Semantically incorrect as headers introduce content, while footers conclude it.
Omitting `<footer>` for main site information	Using `<footer>` for site-wide information enhances clarity and accessibility.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. A header containing:
    - The website's logo.
    - A navigation menu with links to "Home," "Services," and "Contact."
2. A main section with:
    - An article titled "Our Services" with a brief description.
    - A footer within the article containing the author's name.
3. A page footer with:
    - Copyright information.

Solution:

```
<!DOCTYPE html>
<html>
<head>
 <title>Company Services</title>
</head>
<body>

<header>
 <img src="logo.png" alt="Company Logo">
 <nav>
  <ul>
   <li><a href="#">Home</a></li>
   <li><a href="#">Services</a></li>
   <li><a href="#">Contact</a></li>
  </ul>
 </nav>
</header>

<main>
 <article>
```

```
    <header>
      <h2>Our Services</h2>
    </header>
    <p>We offer a wide range of services to meet your needs...</p>
    <footer>
      <p>Written by Jane Smith</p>
    </footer>
  </article>
</main>

<footer>
  <p>&copy; 2025 Company Name. All rights reserved.</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<footer>`	Defines footer content for a section or entire page.
Typical Content	Authorship information, copyright details, contact information, related links.
Difference from `<header>`	`<footer>` concludes or summarizes content; `<header>` introduces content.
Placement	Can be used within `<body>` and various sectioning elements like `<article>`, `<section>`, etc.
Best Practices	Use `<footer>` to group footer content; avoid nesting `<footer>` within `<header>`.

HTML `<aside>` Element

🌟 What is `<aside>`?

The `<aside>` element is a semantic HTML5 tag used to represent content that is tangentially related to the main content of a webpage. This content is often presented as sidebars, pull quotes, or additional information that complements the primary content without being essential to its understanding.

🧩 Syntax:

```
<aside>
  <!-- Tangentially related content goes here -->
</aside>
```

🎯 *When to Use `<aside>`?*

The `<aside>` element is appropriate for content that enhances or provides additional context to the main content but isn't crucial for its comprehension. Common uses include:

- *Sidebars:* Containing related links, author bios, or advertisement.
- *Pull Quotes:* Highlighting notable excerpts from the main content.
- *Additional Information:* Providing glossaries, bibliographies, or related resources.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Understanding the Solar System</title>
</head>
<body>

<article>
  <h1>The Solar System</h1>
  <p>The Solar System consists of the Sun and the objects that orbit it...</p>
</article>

<aside>
  <h2>Did You Know?</h2>
  <p>Jupiter is the largest planet in our Solar System.</p>
</aside>

</body>
</html>
```

Explanation:

- The `<article>` contains the main content about the Solar System.
- The `<aside>` provides additional, interesting facts related to the Solar System but not essential to the main article.

🚀 Benefits of Using `<aside>`:

Benefit	Explanation
Semantic Structure	Enhances the document's structure, making it clearer for both users and search engines.
Accessibility	Assists assistive technologies in distinguishing supplementary content from main content.
SEO Enhancement	Helps search engines understand the hierarchy and relevance of content, potentially improving rankings.

Consistent Styling	Allows for uniform styling of supplementary content across the site using CSS.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Using `<aside>` for unrelated content	The `<aside>` should contain content related to the main content; unrelated content should be placed elsewhere.
Overusing `<aside>` elements	Excessive use can clutter the page and distract from the main content.
Placing essential content within `<aside>`	Crucial information should be in the main content area, not within an `<aside>`.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. An article titled "The Benefits of Exercise" with a brief description.
2. An aside section containing:
   - A list of related articles.
   - A notable quote about exercise.

Solution:

```
<!DOCTYPE html>
<html>
<head>
  <title>The Benefits of Exercise</title>
</head>
<body>

<article>
  <h1>The Benefits of Exercise</h1>
  <p>Engaging in regular physical activity is essential for maintaining good health...</p>
</article>

<aside>
  <h2>Related Articles</h2>
  <ul>
    <li><a href="#">10 Tips for Starting a Workout Routine</a></li>
    <li><a href="#">How Exercise Improves Mental Health</a></li>
    <li><a href="#">The Importance of Stretching Before Exercise</a></li>
  </ul>
  <blockquote>
    "Exercise is medicine." — Hippocrates
  </blockquote>
```

```
</aside>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<aside>`	Represents content tangentially related to the main content.
Typical Content	Sidebars, pull quotes, related links, advertisements, author information.
Difference from `<section>`	`<section>` groups related content; `<aside>` contains supplementary content related to the main content.
Placement	Can be used within `<body>`, `<article>`, `<section>`, or other sectioning elements.
Best Practices	Use `<aside>` for content that enhances the main content without being essential to its understanding.

HTML `<nav>` Element

🌟 What is `<nav>`?

The `<nav>` element is a semantic HTML5 tag that represents a section of a webpage dedicated to navigation links. These links facilitate users in navigating through the website or within different sections of a single page.

🧩 Syntax:

```
<nav>
  <!-- Navigation links go here -->
</nav>
```

🎯 *When to Use `<nav>`?*

The `<nav>` element is intended for major blocks of navigation links. It's important to note that not all groups of links on a page need to be enclosed within a `<nav>` element. For instance, links within a footer that point to terms of service or a copyright page don't necessarily require a `<nav>` wrapper.

Common uses include:

- *Primary Navigation Menus:* Main site navigation links.
- *Table of Contents:* Links to different sections within a document.
- *Index Pages:* Alphabetical or categorical lists of links.

📄 Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page</title>
</head>
<body>

<header>
  <h1>My Website</h1>
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="about.html">About</a></li>
      <li><a href="services.html">Services</a></li>
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </nav>
</header>

<main>
  <h2>Welcome to My Website</h2>
  <p>This is the main content area.</p>
</main>

<footer>
  <p>&copy; 2025 My Website</p>
</footer>

</body>
</html>
```

Explanation:

- The `<nav>` element contains the main navigation links of the website, allowing users to navigate to different pages.

🚀 Benefits of Using `<nav>`:

Benefit	Explanation
Semantic Structure	Enhances the document's structure, making it clearer for both users and search engines.
Accessibility	Assists assistive technologies in identifying navigation sections, improving the browsing experience for users with disabilities.

SEO Enhancement	Helps search engines understand the site's structure, potentially improving indexing and ranking.
Consistent Styling	Allows for uniform styling of navigation sections across the site using CSS.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Wrapping all links in `<nav>`	Only major navigation blocks should be enclosed in `<nav>`; minor links don't require it.
Nesting `<nav>` elements unnecessarily	Over-nesting can confuse users and assistive technologies; keep navigation structures simple.
Using `<nav>` for unrelated content	The `<nav>` element should exclusively contain navigation links, not unrelated content.

💡 Practice Problem:

Problem:

Create an HTML page with the following structure:

1. A header containing:
   - The website's title.
   - A navigation menu with links to "Home," "Blog," "Projects," and "Contact."
2. A main section with:
   - A welcome message.
3. A footer with:
   - Copyright information.

Solution:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Portfolio</title>
</head>
<body>

<header>
  <h1>My Portfolio</h1>
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="blog.html">Blog</a></li>
      <li><a href="projects.html">Projects</a></li>
```

```
      <li><a href="contact.html">Contact</a></li>
    </ul>
  </nav>
</header>

<main>
  <h2>Welcome!</h2>
  <p>Thank you for visiting my portfolio website.</p>
</main>

<footer>
  <p>&copy; 2025 My Portfolio</p>
</footer>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of `<nav>`	Defines a section containing navigation links.
Typical Content	Major navigation links like menus, tables of contents, or indexes.
Difference from `<div>`	`<nav>` is semantic and specifically for navigation links; `<div>` is a generic container without semantic meaning.
Placement	Typically within `<header>`, but can be placed anywhere appropriate within the body.
Best Practices	Use for major navigation blocks; avoid wrapping all links or unrelated content in `<nav>`.

HTML `<figure>` Element

🌟 What is `<figure>`?

The `<figure>` HTML element represents self-contained content, such as illustrations, diagrams, photos, code listings, or other media. This content is typically referenced as a single unit and can include an optional caption using the `<figcaption>` element.

🧩 Syntax:

```
html
CopyEdit
<figure>
  <!-- Self-contained content goes here -->
  <figcaption>Optional caption describing the content</figcaption>
```

```
</figure>
```

🎯 When to Use `<figure>`?

The `<figure>` element is ideal for content that is self-contained and related to the main flow of the document but can be moved or removed without affecting the document's meaning. Common uses include:

- **Images with Captions:** Photographs or illustrations accompanied by descriptions.
- **Diagrams and Charts:** Visual representations of data or processes.
- **Code Snippets:** Blocks of code examples.
- **Tables:** Data tables with explanatory captions.

Using `<figure>` helps group the media content and its caption as a single unit, enhancing semantic meaning and accessibility.

📄 Example:

```
html
CopyEdit
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page with Figure</title>
</head>
<body>

<h1>Understanding the Solar System</h1>

<figure>
  <img src="solar-system.jpg" alt="Diagram of the Solar System" width="500" height="300">
  <figcaption>Figure 1: A diagram illustrating the planets in our Solar System.</figcaption>
</figure>

<p>The Solar System consists of the Sun and the objects that orbit it, including eight planets, their moons, and other celestial bodies.</p>

</body>
</html>
```

Explanation:

- `<figure>` encapsulates the image and its caption, treating them as a single unit.

- `<img>` displays the image of the Solar System.
- `<figcaption>` provides a caption describing the image.

This structure ensures that the image and its description are semantically linked, improving accessibility and SEO.

🚀 Benefits of Using `<figure>`:

Benefit	Explanation
Semantic Structure	Clearly defines self-contained content, enhancing document structure.
Accessibility	Assists assistive technologies in understanding the relationship between media and its description.
SEO Enhancement	Provides context to search engines about the media content, potentially improving indexing.
Styling Flexibility	Allows for targeted CSS styling of figures and their captions.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Omitting `<figcaption>` for descriptive content	Without `<figcaption>`, users may lack context for the media content.
Using `<figure>` for non-self-contained content	`<figure>` should only wrap content that is self-contained and relevant independently.
Nesting block-level elements improperly within `<figure>`	Ensure that content within `<figure>` maintains proper HTML structure.

💡 Practice Problem:

Problem:

Create an HTML snippet that includes a code example using the `<figure>` and `<figcaption>` elements.

Solution:

```
html
CopyEdit
<figure>
  <pre>
<code>
function greet() {
  console.log("Hello, world!");
}
</code>
  </pre>
  <figcaption>Listing 1: A simple JavaScript function that logs a greeting message.</figcaption>
```

```
</figure>
```

📋 Recap Table:

Feature	Details
Purpose of `<figure>`	Encapsulates self-contained content, often with a caption.
Typical Content	Images, diagrams, code listings, tables, or other media.
Difference from `<img>`	`<img>` embeds images; `<figure>` groups media content with optional captions.
Placement	Can be used anywhere in the document; content should be self-contained.
Best Practices	Use `<figcaption>` to provide context; ensure content is relevant and self-contained.

HTML events are actions or occurrences that happen in the browser, which can be detected and handled using JavaScript. These events enable interactive and dynamic user experiences by allowing developers to execute specific code in response to user interactions or other changes within the document.

🧩2.7 HTML Events: Window Events, Form Element Events, Keyboard Events, Mouse Events:

Overview:

1. **Mouse Events:**
   - `click`: Triggered when an element is clicked.
   - `dblclick`: Triggered when an element is double-clicked.
   - `mousedown`: Triggered when a mouse button is pressed over an element.
   - `mouseup`: Triggered when a mouse button is released over an element.
   - `mouseover`: Triggered when the mouse pointer moves over an element.
   - `mouseout`: Triggered when the mouse pointer moves out of an element.
2. **Keyboard Events:**
   - `keydown`: Triggered when a key is pressed down.
   - `keypress`: Triggered when a key is pressed and released.
   - `keyup`: Triggered when a key is released.
3. **Form Events:**
   - `submit`: Triggered when a form is submitted.
   - `reset`: Triggered when a form is reset.

- `change`: Triggered when the value of an element changes.
- `focus`: Triggered when an element gains focus.
- `blur`: Triggered when an element loses focus.

4. **Window Events:**
   - `load`: Triggered when the whole page has loaded.
   - `resize`: Triggered when the browser window is resized.
   - `scroll`: Triggered when the user scrolls the document.

🎯 Adding Event Handlers:

Event handlers can be added to HTML elements in various ways:

1. **Inline Event Handlers:**

   Directly within the HTML tag using event attributes.

```
html
CopyEdit
<button onclick="alert('Button clicked!')">Click Me</button>
```

2. **DOM Property Event Handlers:**

   Assigning a function to an event property of a DOM object.

```
html
CopyEdit
<button id="myButton">Click Me</button>

<script>
 document.getElementById('myButton').onclick = function() {
  alert('Button clicked!');
 };
</script>
```

3. **Event Listener Method:**

   Using `addEventListener` to attach event handlers.

```
<button id="myButton">Click Me</button>

<script>
 document.getElementById('myButton').addEventListener('click', function() {
  alert('Button clicked!');
```

```
  });
</script>
```

🚀 Best Practices:

- **Separation of Concerns:** Keep JavaScript separate from HTML by using external scripts or DOM methods to attach event handlers, promoting cleaner and more maintainable code.
- **Event Delegation:** Use event delegation to manage events efficiently, especially when dealing with multiple child elements.
- **Cross-Browser Compatibility:** Ensure that event handling works consistently across different browsers by using standard methods like `addEventListener`.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Using Inline Event Handlers Excessively	Leads to mixing HTML and JavaScript, making code harder to maintain.
Not Removing Event Listeners When No Longer Needed	Can cause memory leaks and unintended behavior.
Assuming Event Order	Relying on a specific order of event firing can lead to bugs, as event order is not guaranteed.

💡 Practice Problem:

Problem:

Create an HTML snippet that includes a text input field and a button. When the button is clicked, display an alert with the current value of the text input.

Solution:

```
html
CopyEdit
<!DOCTYPE html>
<html>
<head>
  <title>Event Handling Example</title>
</head>
<body>

<input type="text" id="userInput" placeholder="Enter text here">
<button id="alertButton">Show Alert</button>

<script>
```

```
  document.getElementById('alertButton').addEventListener('click', function() {
    var inputText = document.getElementById('userInput').value;
    alert('You entered: ' + inputText);
  });
</script>

</body>
</html>
```

📋 Recap Table:

Feature	Details
Purpose of HTML Events	Allow interaction and dynamic behavior in web pages by responding to user actions or browser events.
Common Event Types	Mouse, Keyboard, Form, and Window events.
Adding Event Handlers	Inline, DOM Property, and `addEventListener` methods.
Best Practices	Separate JavaScript from HTML, use event delegation, and ensure cross-browser compatibility.

HTML Form Element Events

🌟 Introduction

HTML forms are interactive elements that enable user input and data submission. To manage and respond to user interactions effectively, understanding form-related events is crucial. These events allow developers to validate data, enhance user experience, and control form behavior dynamically.

🧩 Common Form Events:

1. `submit`:
    - **Description:** Triggered when a form is submitted.
    - **Usage:** Commonly used to validate form data before sending it to the server.
    - **Example:**

```
const form = document.querySelector('form');
form.addEventListener('submit', function(event) {
  event.preventDefault(); // Prevents the default form submission
  // Validation or submission logic here
```

```
});
```

2. `reset`:
   - **Description:** Occurs when a form is reset.
   - **Usage:** Can be used to confirm the reset action or to perform additional cleanup tasks.
   - **Example:**

```
const form = document.querySelector('form');
form.addEventListener('reset', function() {
  console.log('Form has been reset.');
});
```

3. `change`:
   - **Description:** Fires when the value of an input, select, or textarea element has been modified and loses focus.
   - **Usage:** Useful for validating or responding to user input after they have moved away from the field.
   - **Example:**

```
const inputField = document.querySelector('input');
inputField.addEventListener('change', function() {
  console.log('Input value changed to:', this.value);
});
```

4. `input`:
   - **Description:** Occurs immediately after the value of an element has changed.
   - **Usage:** Ideal for real-time validation or dynamic UI updates as the user types.
   - **Example:**

```
const inputField = document.querySelector('input');
inputField.addEventListener('input', function() {
  console.log('Current input value:', this.value);
});
```

5. `focus` and `blur`:
   - **Description:** `focus` is triggered when an element gains focus; `blur` occurs when it loses focus.
   - **Usage:** Commonly used to apply or remove highlighting, display helper text, or validate input upon entering or leaving a field.
   - **Example:**

```
const inputField = document.querySelector('input');
inputField.addEventListener('focus', function() {
  console.log('Input field is focused.');
});
inputField.addEventListener('blur', function() {
  console.log('Input field lost focus.');
});
```

🚀 Best Practices:

- **Prevent Default Behavior:** Use `event.preventDefault()` to stop the default form submission, allowing for custom validation or processing.
- **Debounce Input Events:** For performance optimization, especially with the `input` event, implement debouncing to limit the rate of function execution.
- **Delegate Events:** Attach event listeners to form containers to manage events from multiple child elements efficiently.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Not Preventing Default Submission	Without `event.preventDefault()`, forms may submit before validation occurs.
Overusing `change` Event for Instant Feedback	The `change` event only fires after an element loses focus, making it less suitable for real-time validation.
Ignoring Accessibility Considerations	Not managing focus properly can lead to a poor experience for users relying on assistive technologies.

💡 Practice Problem:

Problem:

Create a form with an email input field that validates the entered email address in real-time. Display a message indicating whether the email is valid or not as the user types.

Solution:

```
html
CopyEdit
```

```
<form>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email">
  <span id="message"></span>
</form>

<script>
  const emailInput = document.getElementById('email');
  const message = document.getElementById('message');

  emailInput.addEventListener('input', function() {
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (emailPattern.test(this.value)) {
      message.textContent = 'Valid email address.';
      message.style.color = 'green';
    } else {
      message.textContent = 'Invalid email address.';
      message.style.color = 'red';
    }
  });
</script>
```

In this example, the `input` event ensures that the email address is validated as the user types, providing immediate feedback.

📋 Recap Table:

Event	Description	Common Use Case
`submit`	Triggered upon form submission.	Validating data before sending to the server.
`reset`	Occurs when a form is reset.	Confirming reset actions.
`change`	Fires when an element's value changes and loses focus.	Post-input validation.
`input`	Occurs immediately after the value of an element changes.	Real-time validation or UI updates.
`focus`	Triggered when an element gains focus.	Highlighting or displaying helper text.
`blur`	Occurs when an element loses focus.	Validating input upon leaving a field.

Understanding and effectively handling form events is essential for creating responsive and user-friendly web applications. By implementing appropriate event listeners and adhering to best practices, developers can enhance form interactions and ensure data integrity.

JavaScript Keyboard and Mouse Events

🌟 Introduction

JavaScript provides a robust event handling system that allows developers to create interactive and dynamic web applications. Among the most commonly used events are those related to keyboard and mouse interactions. Understanding these events is crucial for enhancing user experience and building responsive interfaces.

🧩 Keyboard Events:

Keyboard events are triggered by user interactions with the keyboard. The primary keyboard events include:

1. `keydown`:
   - **Description:** Fires when a key is pressed down.
   - **Usage:** Detecting when a key is initially pressed, regardless of whether it produces a character.
   - **Example:**

     ```
     javascript
     CopyEdit
     document.addEventListener('keydown', function(event) {
       console.log(`Key down: ${event.key}`);
     });
     ```

2. `keypress`:
   - **Description:** Fires when a key that produces a character value is pressed down.
   - **Usage:** Capturing character input from the user.
   - **Note:** This event is deprecated and may not be supported in all browsers. Use `keydown` or `beforeinput` instead.

     MDN Web Docs
   - **Example:**

     ```
     javascript
     CopyEdit
     document.addEventListener('keypress', function(event) {
       console.log(`Key press: ${event.key}`);
     });
     ```

3. `keyup`:
   - **Description:** Fires when a key is released.

- **Usage:** Performing actions after a key is released, such as validating input.
- **Example:**

```
javascript
CopyEdit
document.addEventListener('keyup', function(event) {
  console.log(`Key up: ${event.key}`);
});
```

For a comprehensive list of key values associated with keyboard events, refer to the MDN documentation.

MDN Web Docs

🧩 Mouse Events:

Mouse events are triggered by user interactions with a pointing device, such as a mouse. Common mouse events include:

1. `click`:
   - **Description:** Fires when a pointing device button (usually the primary button) is pressed and released on an element.
   - **Usage:** Handling user clicks on buttons, links, or other interactive elements.
   - **Example:**

```
javascript
CopyEdit
document.addEventListener('click', function(event) {
  console.log('Element clicked:', event.target);
});
```

2. `dblclick`:
   - **Description:** Fires when a pointing device button is clicked twice on an element.
   - **Usage:** Implementing double-click functionality, such as opening files or folders.
   - **Example:**

```
javascript
CopyEdit
document.addEventListener('dblclick', function(event) {
  console.log('Element double-clicked:', event.target);
```

```
});
```

3. `mousedown` and `mouseup`:
   - **Description:** `mousedown` fires when a pointing device button is pressed on an element; `mouseup` fires when the button is released.
   - **Usage:** Detecting the start and end of a mouse press, useful for implementing drag-and-drop functionality.
   - **Example:**

```
javascript
CopyEdit
document.addEventListener('mousedown', function(event) {
  console.log('Mouse button pressed:', event.button);
});

document.addEventListener('mouseup', function(event) {
  console.log('Mouse button released:', event.button);
});
```

4. `mousemove`:
   - **Description:** Fires when the mouse pointer is moved over an element.
   - **Usage:** Tracking mouse movement, such as for drawing applications or custom cursor effects.
   - **Example:**

```
javascript
CopyEdit
document.addEventListener('mousemove', function(event) {
  console.log(`Mouse moved to: (${event.clientX}, ${event.clientY})`);
});
```

For a detailed overview of mouse events, refer to the MDN documentation.

MDN Web Docs

🚀 Best Practices:

- **Debounce Frequent Events:** For events that fire frequently, such as `mousemove` or `keydown`, implement debouncing to improve performance.
- **Use Event Delegation:** Attach a single event listener to a parent element to manage events for multiple child elements efficiently.

- **Consider Accessibility:** Ensure that keyboard and mouse interactions are accessible to all users, including those relying on assistive technologies.

❌ Common Mistakes:

Mistake	Why It's Incorrect
Overusing the `keypress` Event	The `keypress` event is deprecated and may not be supported in all browsers. Use `keydown` or `beforeinput` instead.
Not Removing Event Listeners	Failing to remove event listeners can lead to memory leaks and unintended behavior.
Ignoring Event Object Properties	Not utilizing properties like `event.key` or `event.button` can result in less precise event handling.