1) What is HTML, usage, versions, structure

---

1. Definition:
   - HTML stands for HyperText Markup Language.
   - A markup language (not a programming language) used to structure web pages.
   - Uses tags (like <h1>, , <a>, etc.) to annotate content.

2. Usage:
   - Defines the skeleton/framework of a webpage.
   - Structures content: headings, paragraphs, lists, images, links, forms.
   - Embeds media: images, audio, video, canvas graphics.
   - Defines semantics: this part is a navigation menu, this part is an article, etc.
   - Works with CSS (for styling) and JavaScript (for interactivity).

3. Versions:
   - HTML 1.0 (~1991): very basic content only.
   - HTML 2.0 (~1995): added forms, tables.
   - HTML 3 / 3.2 (~1997): more layout features.
   - HTML 4 / 4.01 (~1999): integration with CSS, scripting, multimedia.
   - HTML5 (~2014 onward): major update – new semantic tags, built-in media support, modern APIs. Current standard.

4. Basic Structure of an HTML Document:

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8" /> <title>Page Title</title> </head> <body> <!-- Visible content goes here --> <h1>Main Heading</h1> This is a paragraph. Link example </body> </html> ``` - <!DOCTYPE html> : declares HTML5 document type. - <html> … </html> : root element. - <head> … </head> : metadata (title, charset, links to CSS/js). - <body> … </body> : actual content visible to users.

5. Good Practices:

    - Use semantic tags when possible: <header>, <footer>, <nav>, <article>, <section> etc.
    - Ensure proper nesting of tags.
    - Use the correct language attribute (lang) on <html> for accessibility.
    - Use <meta charset="UTF-8" /> to support international characters.

- Keep HTML structure separated from CSS (styling) and JavaScript (behavior) for clarity and maintainability.

6. Why it matters:
   - Good HTML structure improves browser rendering, accessibility, SEO, and maintainability.
   - As a developer, starting with clean HTML makes it easier to build larger applications (with CSS/JS frameworks) later.

2) Usage of <img>, attributes (src, alt, width, height), accessibility

---

1. Purpose:
   - The <img> element embeds an image into a web page.
   - It links to an image file via a URL/path using the src attribute.
   - It does not require a closing tag (in HTML5).

2. Key Attributes:
   a) src
      - Specifies the image source (path or URL).
      - Example: src="images/photo.jpg"
   b) alt
      - Specifies alternate text describing the image.
      - If the image fails to load or for screen-readers.
      - Example: alt="A mountain peak at sunrise"
   c) width
      - Specifies the display width (in pixels) of the image.
      - Helps reserve layout space before loading.
      - Example: width="800"
   d) height
      - Specifies the display height (in pixels) of the image.
      - Example: height="450"

3. Example Usage:
   <img src="images/sunset.jpg" alt="Sunset over the ocean" width="800" height="450">
   <img src="decorative-pattern.png" alt="" width="200" height="100">

4. Accessibility Best Practices:
- Always include the alt attribute (either meaningful or empty) for <img> elements.
- For meaningful images: alt should describe the image's purpose/content.
- For purely decorative images: use alt="" so assistive tech skips it.
- Use width and height to avoid layout shifts and improve experience.
- Optimize image files for size and format to improve performance.

5. Performance & Responsiveness Tips:
- Don't use oversized image files just scaled down via HTML attributes — resize the actual image file.
- Use width/height attributes for layout hints; use CSS for responsive scaling (e.g., max-width:100%).
- Use relative URLs for images within your project.
- For advanced scenarios, consider using srcset/sizes for different resolutions or formats.

6. Why It Matters:
- Proper <img> usage supports accessibility, performance, and layout stability.
- A good user experience includes images that load gracefully, don't cause layout jumps, and are accessible to everyone.

3) Structure (<table>, <tr>, <td>, <th>), rowspan/colspan, styling, accessibility

---

1️⃣ Table Structure:
- <table>: Defines the table.
- <tr>: Table row.
- <td>: Table data cell.
- <th>: Table header (bold, centered by default).

2️⃣ Example:
```
<table border="1">
  <tr><th>Subject</th><th>Marks</th></tr>
  <tr><td>Math</td><td>95</td></tr>
</table>
```

3️⃣ Rowspan & Colspan:
- rowspan → merges cells vertically.
- colspan → merges cells horizontally.

Example:
```
<td rowspan="2">Merged</td>
<td colspan="2">Merged</td>
```

4️⃣ Styling Tables:
- Use CSS for borders, background, and alignment.
  table { border-collapse: collapse; width: 50%; }
  th, td { border: 1px solid #333; padding: 8px; }

5️⃣ Accessibility:
- <caption> for table title.
- <th scope="col"> or <th scope="row"> for screen readers.
- Helps users with disabilities navigate tables.

4) Input types, labels, validations, form actions, GET vs POST

📘 HTML Forms — Input Types, Labels, Validations, Actions, and Methods

1️⃣ Input Types:
<input type="text"> → Text input
<input type="password"> → Hidden characters
<input type="email"> → Validates email
<input type="number"> → Numeric input
<input type="date"> → Date picker
<input type="radio"> → One choice from many
<input type="checkbox"> → Multiple selections
<input type="file"> → Upload files
<input type="submit"> → Submit button
<input type="reset"> → Clear all fields

2️⃣ Labels:
<label for="id">Text</label>
→ Connects label to input for accessibility.

Example:
<label for="username">Username:</label>
<input id="username" type="text" name="username">

3️⃣ Validations:
- required → Field must be filled.
- pattern → Restrict input format using regex.
- minlength / maxlength → Control text length.
- type="email" → Checks valid email.
- min / max → Restrict numeric or date range.

Example:

<input type="text" pattern="[A-Za-z]{3,10}" required>

4️⃣ Form Actions:
<form action="/submit" method="POST">
→ action defines where to send data.
→ method defines how data is sent.

5️⃣ GET vs POST:
GET:
- Data visible in URL
- Used for searches
- Less secure

POST:
- Data hidden from URL
- Used for login, registration
- More secure and allows larger data

Example:
<form action="/search" method="GET">
<form action="/register" method="POST">

5) <header>, <nav>, <article>, <footer>, SEO importance

📘 Topic: Semantic HTML — <header>, <nav>, <article>, <footer> and SEO Importance

🔹 What is Semantic HTML?
Semantic HTML uses meaningful tags that describe the purpose of content.
It improves accessibility, code clarity, and SEO ranking.

🔹 Elements Overview

1️⃣ <header>
• Represents the top or introductory part of a webpage.
• Commonly contains site title, logo, and navigation.
🧩 Example:
```
<header>
  <h1>My Website</h1>
</header>
```

2️⃣ <nav>
• Defines a section of navigation links.
• Helps users move between different parts of the site.

🧩 Example:

```
<nav>
  <a href="#home">Home</a> | <a href="#about">About</a>
</nav>
```

3️⃣ <article>

- Represents self-contained, reusable content.
- Used for blog posts, news articles, etc.

🧩 Example:

```
<article>
  <h2>My Blog Post</h2>
  <p>Content goes here...</p>
</article>
```

4️⃣ <footer>

- Represents the bottom section of a webpage.
- Contains copyright, contact, and social links.

🧩 Example:

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

🔹 SEO Importance

- Semantic tags help search engines understand content structure.
- Improves accessibility for screen readers.
- Increases ranking potential by emphasizing key sections.

🎯 Using semantic HTML = Better SEO + Better Accessibility + Cleaner Code.

📘 1) Topic: CSS Selectors and Specificity

🔹 What are Selectors?
Selectors are patterns used to target and style specific HTML elements.

🔹 Types of Selectors:

1️⃣ Element Selector
- Targets elements by tag name.
- Example:
  p { color: blue; }

2️⃣ Class Selector (.)
- Targets elements with a specific class.
- Reusable across multiple elements.
- Example:
  .highlight { background-color: yellow; }

3️⃣ ID Selector (#)
- Targets one unique element with a specific ID.
- Example:
  #header { text-align: center; }

4️⃣ Attribute Selector
- Targets elements based on their attributes.
- Example:
  input[type="text"] { border: 1px solid black; }

🔹 Specificity (Priority Order)
- Element selector → Lowest (1)
- Class/Attribute selector → Medium (10)
- ID selector → High (100)
- Inline style → Highest (1000)

🧠 Rule:
If two rules conflict → the one with higher specificity wins.

✅ Summary:
- Element: Targets by tag name.
- Class: Targets multiple elements with same class.
- ID: Targets a unique element.
- Attribute: Targets based on HTML attribute.
- Specificity controls which style is applied when conflicts occur.

📘 2) Topic: CSS Box Model — Content, Padding, Border, Margin, Box-Sizing

📘
🔹 Box Model Overview:
Every element in CSS is a rectangular box composed of:
1. Content
2. Padding
3. Border
4. Margin

🔹 Content:
• The actual text, image, or data inside the element.
• Defined using width and height.
🧩 Example:
div { width: 200px; height: 100px; }

🔹 Padding:
• Space between content and border (inner spacing).
• Increases element's total size.
🧩 Example:
div { padding: 20px; }

🔹 Border:
• A visible outline around the element.
• Sits between padding and margin.
🧩 Example:
div { border: 2px solid black; }

🔹 Margin:
• Space outside the border (outer spacing).
• Creates distance between elements.
🧩 Example:
div { margin: 30px; }

🔹 Box-Sizing:
• Defines how width and height are calculated.
  ▪ content-box → width = content only (default)
  ▪ border-box → width = content + padding + border

🧩 Example:
```
div {
  width: 200px;
  padding: 20px;
  border: 10px solid black;
  box-sizing: border-box;
```

}

✅ Summary:
• Padding, Border, Margin add space.
• box-sizing: border-box is preferred — it simplifies layout control.
• The box model determines how elements occupy space on a page.

📘 3) CSS Flexbox & Grid Notes

🔹 Flexbox → 1D layout system (row/column)
- Container: display: flex
- justify-content → Horizontal alignment
  • flex-start | flex-end | center | space-between | space-around | space-evenly
- align-items → Vertical alignment
  • stretch | flex-start | flex-end | center | baseline
- Common combo:
  display: flex;
  justify-content: center;
  align-items: center;

🔹 CSS Grid → 2D layout system (rows + columns)
- Container: display: grid
- grid-template-columns → Define columns
  • Example: grid-template-columns: 1fr 1fr 1fr;
- grid-template-rows → Define rows
- gap → Space between rows/columns
- place-items: center → Center items in both axes

🔹 Flexbox vs Grid
- Flexbox → Align items in one direction
- Grid → Align items in both directions
- Use Flexbox for smaller UI components, Grid for full-page structure

📘 4) CSS Responsive Design — Media Queries, Relative Units, Mobile-First Design

1️⃣ Media Queries:
• Used to apply styles based on screen size, resolution, or device type.
• Syntax:
  @media (max-width: 600px) { ... }
• Common breakpoints:
  - Phones: max-width: 480px
  - Tablets: max-width: 768px
  - Laptops: max-width: 1024px

- Desktops: max-width: 1200px+

2️⃣ Relative Units:
• Help create scalable layouts.
• % → Relative to parent size
• em → Relative to element font size
• rem → Relative to root (html) font size
• vw → % of viewport width
• vh → % of viewport height

3️⃣ Mobile-First Design:
• Design for small screens first.
• Use min-width media queries to scale up.
• Example:
  Base (mobile) → Tablet (768px+) → Desktop (1024px+)

✅ Best Practices:
• Use rem for consistent font scaling.
• Use vw/vh for full-width/height layouts.
• Always start mobile-first, then expand.
• Test on different devices to ensure usability.

📘 1) Bootstrap: History & Purpose Notes

1. Definition:
   - Bootstrap is a free, open-source front-end framework (HTML/CSS/JS) for building responsive, mobile-first websites.

2. History:
   - Mid 2010: Created internally at Twitter by Mark Otto & Jacob Thornton.
   - Originally called Twitter Blueprint, later renamed Bootstrap.
   - August 2011: Released open-source on GitHub.
   - Version 2: Added responsive grid & more components.
   - Version 3: Emphasized mobile-first design.
   - Version 4+: Major rewrite (Sass, flexbox, dropped older browsers).
   - Used widely online; part of many websites & UI toolkits.

3. Purpose / Why use it:
   - Speeds up UI development (ready components, grid system).
   - Brings consistency to UI across pages or teams.
   - Built for responsive layouts across devices.
   - Provides many reusable components (buttons, navbars, forms, etc).
   - Strong community, documentation, templates.

4. When to use:
   - Good for rapid development, standard websites, dashboards.
   - Great when multiple developers/designers need consistency.
   - Ideal when responsive behaviour is required out-of-the-box.

5. Things to weigh:
   - Customisation may require overriding many defaults.
   - May include more than needed for small or highly customised designs.
   - Alternatives exist if you want ultra-lightweight or fully custom UI.

📘 2) Bootstrap: Setup Notes

1. What is Bootstrap setup?
   - Including Bootstrap means adding its CSS and JS into your HTML so you can use its components and grid.

2. Methods:
   a) CDN (Content Delivery Network)
      • Use <link> in <head> for CSS.
      • Use <script> before </body> for JS (if needed).
      • Fast, no download required, good for quick setup.

• Example:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
      rel="stylesheet" integrity="..." crossorigin="anonymous">
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
        integrity="..." crossorigin="anonymous"></script>
```

b) Local / Download

- Download Bootstrap CSS/JS files into your project (css/, js/ folders).
- Include them via relative paths.
- Allows customization and offline use.

3. Quick Start (CDN):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap Demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.x/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="..." crossorigin="anonymous">
  </head>
  <body>
    <h1 class="text-center">Hello, Bootstrap!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.x/dist/js/bootstrap.bundle.min.js"
integrity="..." crossorigin="anonymous"></script>
  </body>
</html>
```

4. Why choose which method?

- Use CDN: Quick, standard UI, minimal work.
- Use Local: Custom theme, offline, full control.

5. Important notes:

- Always include the meta viewport tag for mobile responsiveness.
- Place CSS <link> in <head>, JS <script> just before closing </body>.
- For Bootstrap 5, you do *not* need jQuery; bundle includes required dependencies.

📘 3) Bootstrap Grid System, Responsive Design, and Components

1️⃣ Grid System:
• Layout based on 12 columns.
• Containers hold rows, rows hold columns.
• .container → Fixed width
• .container-fluid → Full width
• Use .row and .col classes to create layout.
• Responsive column prefixes:
  .col- (mobile), .col-sm-, .col-md-, .col-lg-, .col-xl-

Example:
```
<div class="container">
  <div class="row">
    <div class="col-md-6">Half width on tablets+</div>
    <div class="col-md-6">Half width on tablets+</div>
  </div>
</div>
```

2️⃣ Responsive Design:
• Makes pages adjust to different screen sizes.
• Bootstrap is mobile-first.
• Key classes:
  .d-none / .d-md-block → Hide/show by screen size
  .img-fluid → Makes images responsive
• Uses breakpoints and media queries internally.

3️⃣ Components:
• Prebuilt UI elements like buttons, cards, navbars, modals.
• Reduces coding effort and ensures consistent design.

Common Components:
  Buttons: <button class="btn btn-primary">
  Alerts:
  Cards: ...
  Navbar: <nav class="navbar navbar-expand-lg navbar-light bg-light">

✅ Summary:
• Grid → Structure (rows & columns)
• Responsive design → Adjusts to screens
• Components → Ready-made elements to build UI quickly