UNIT - 1 WEB - D WITH MERN NOTES

UNIT 1: WEB DEVELOPMENT FUNDAMENTALS

1. INTRODUCTION

1.1 Fundamentals of Good Website Design

- **Definition:** Effective website design aims to fulfill its intended function by clearly conveying its message while simultaneously engaging the visitor. It's a blend of aesthetics (how it looks) and functionality (how it works).
- **Core Purpose (Page 1):**

1. **Describing Expertise:** Showcasing knowledge and authority in a specific area.
2. **Building Your Reputation (Page 2):** Establishing trust and credibility.
3. **Generating Leads (Page 2):** Capturing potential customer information.
4. **Sales and After Care (Page 2):** Facilitating transactions and post-sale support.

- **Key Factors (Page 1):**
- **Consistency:** Uniform design elements (fonts, colors, layout) across all pages.
- **Colours (Page 2):**
- Communicate messages and evoke emotions.
- Palette should fit the brand.
- Limit to less than 5 colours. Complementary colours work well.
- Pleasing combinations increase engagement.
- **Typography (Page 2):**
- Visual interpretation of the brand voice.
- Must be legible.
- Use a maximum of 3 different fonts.
- **Imagery (Page 2):**
- Includes photos, illustrations, videos, graphics.
- Should be expressive, capture company spirit, and embody brand personality.
- High-quality images convey professionalism and credibility.
- **Simplicity (Page 2):**
- Best for user experience and usability. Achieved through careful use of color, type, and imagery.
- **Functionality (Page 1):** How easy the website is to use; ensuring features work as expected.

- **User Experience (UX) (Page 1):** Optimizing for usability (form and aesthetics) and ease of use. A well-designed website builds trust and guides visitors to action.
- **Additional Design Principles:**
- **Navigation (Page 2):** [PYQ Q2(b) mentions CSS link states, related to navigation styling]
- The wayfinding system for visitors.
- Key to retaining visitors; confusing navigation leads to abandonment.
- Should be simple, intuitive, and consistent on every page.
- **F-Shaped Pattern Reading (Page 3):**
- Most common way visitors scan text (top and left areas).
- Layout mimics natural reading patterns (left-to-right, top-to-bottom in the West).
- **Visual Hierarchy (Page 3):**
- Arrangement of elements in order of importance (using size, color, contrast, etc.).
- Establishes a focal point, guiding visitors to the most important information.
- **Content (Page 3):**
- Effective websites need great design AND great content.
- Compelling language attracts, influences, and converts visitors.
- **Grid-Based Layout (Page 4):**
- Structures design and organizes content.
- Aligns elements, keeps the page clean, balanced, and imposes order.
- Results in an aesthetically pleasing website.
- **Load Time (Page 4):**
- Slow loading loses visitors (nearly half expect load in 2 seconds or less).
- Optimizing image sizes helps improve load time.
- **Mobile Friendly (Page 4):**
- Essential as more users browse on phones/devices.
- Responsive layout adjusts to different screen sizes.

1.2 Web Page and Website

- **Web Page (Page 5):**
- **Definition:** A document displayable in a web browser (e.g., Firefox, Chrome). Often called just "pages."
- Typically written in HTML (HyperText Markup Language).
- Can embed resources like:
- Style information (CSS for look-and-feel).
- Scripts (JavaScript for interactivity).
- Media (images, sounds, videos).

- Reachable through a unique address (URL).
- Note: While browsers display PDFs/images, "web page" specifically refers to HTML documents.
- **Website (Page 5):**
- **Definition:** A collection of linked web pages (and associated resources) grouped together under a unique domain name. Often called a "site."
- Pages provide explicit links (clickable text/images) for navigation.
- Accessed by typing its domain name into a browser, which usually displays the homepage (main web page).
- Can be a single-page website (dynamically updated content) (Page 6).

1.3 Web Application (Page 7-8)

- **Definition:** An application program stored on a remote server and delivered over the internet through a browser interface.
- Many websites contain web apps; web services are web apps by definition.
- **How they work:**
- Accessed through a browser; no download needed.
- Requires: Web server (manages client requests), Application server (completes tasks), Database (stores information).
- Development: Often JavaScript, HTML5, CSS (front-end). Server-side (scripts) languages like Python, Java, Ruby.
- **Benefits:**
- Multiple users access the same version.
- No installation required by users.
- Accessible via various platforms (desktop, mobile) and browsers.
- **Web App vs. Native App vs. Hybrid App (Page 8):**
- **Native App:** Built for a specific platform/device, installed, can use device-specific hardware (GPS, camera). Can operate offline.
- **Web App:** Accessed via browser, typically needs internet.
- **Hybrid App:** Combines approaches. Installs like native, built with web tech, can use device APIs. Typically doesn't work offline. Shares navigation elements with web apps.

1.4 Client-Server Architecture (Page 9-14) [PYQ Q1(a) asks for MERN's 3-tier architecture, which is a type of client-server architecture]

- **Definition:** A computing model where the server hosts, delivers, and manages resources/services requested by the client. Also known as a networking computing model.
- **Two Key Factors (Page 9):**

1. **Server:** Provides requested services.

2. **Client:** Requests services.

- **Typical Setup (Page 9):** Clients (workstations, PCs) connect to more powerful servers over a network.
- **Examples (Page 10):**
- **Mail Servers:** Send/receive emails.
- **File Servers:** Centralized file storage (e.g., Google Docs).
- **Web Servers (Page 6 & 10):** High-performance computers hosting websites. A web server is a computer hosting one or more websites. "Hosting" means web pages and supporting files are on that computer. The server sends web pages to a user's browser upon request. *Don't confuse websites and web servers.*
- **Components (Page 10):**

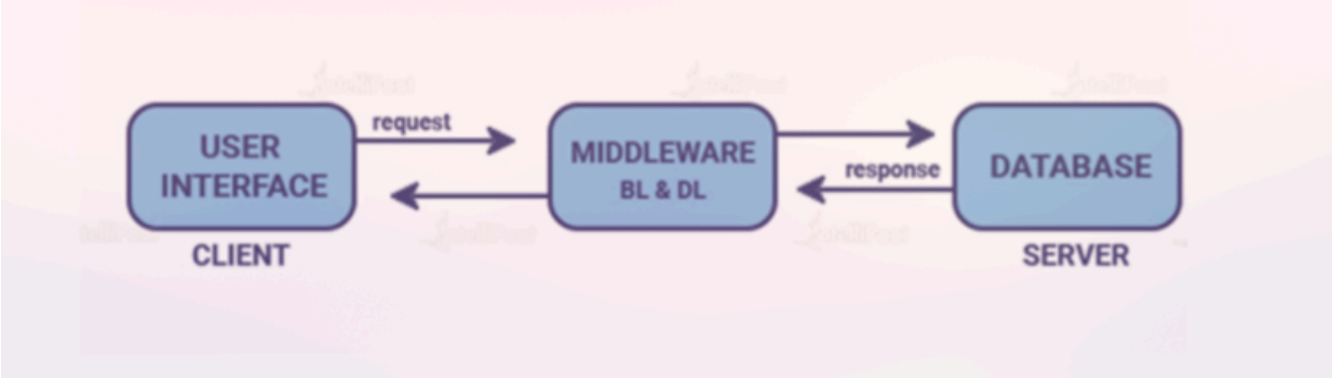
1. **Workstations (Clients):** Request access to shared files/databases from servers.
2. **Servers:** Fast-processing devices; centralized repositories for files, programs, databases, policies. Handle multiple requests.
3. **Networking Devices (Page 11):** Connect workstations and servers (e.g., hubs, repeaters, bridges).

- **How it Works (Page 11):**

1. User enters URL. Browser sends request to DNS server.
2. DNS server finds and returns the IP address of the web server.
3. Browser sends HTTP/HTTPS request to the web server's IP.
4. Server transmits the essential website files.
5. Browser processes files and displays the website.

- **Tiers of Architecture (Page 12-14):**
- **1-Tier (Page 12):** All layers (presentation, business, data) on a single device. Data often local or on a shared drive. Challenging to manage.
- **2-Tier (Page 13):** Client-side stores UI; server houses database. Logic can be on client or server. No intermediaries. Example: Online ticket reservation.
- **3-Tier (Page 13):** Middleware between client and server. Request goes Client -> Middleware -> Server. Response follows reverse. Layers: Presentation (client), Application (middleware/server logic), Database (server). More secure, invisible database structure, data integrity. [This is relevant

for MERN as mentioned in PYQ Q1(a) and Page 19]

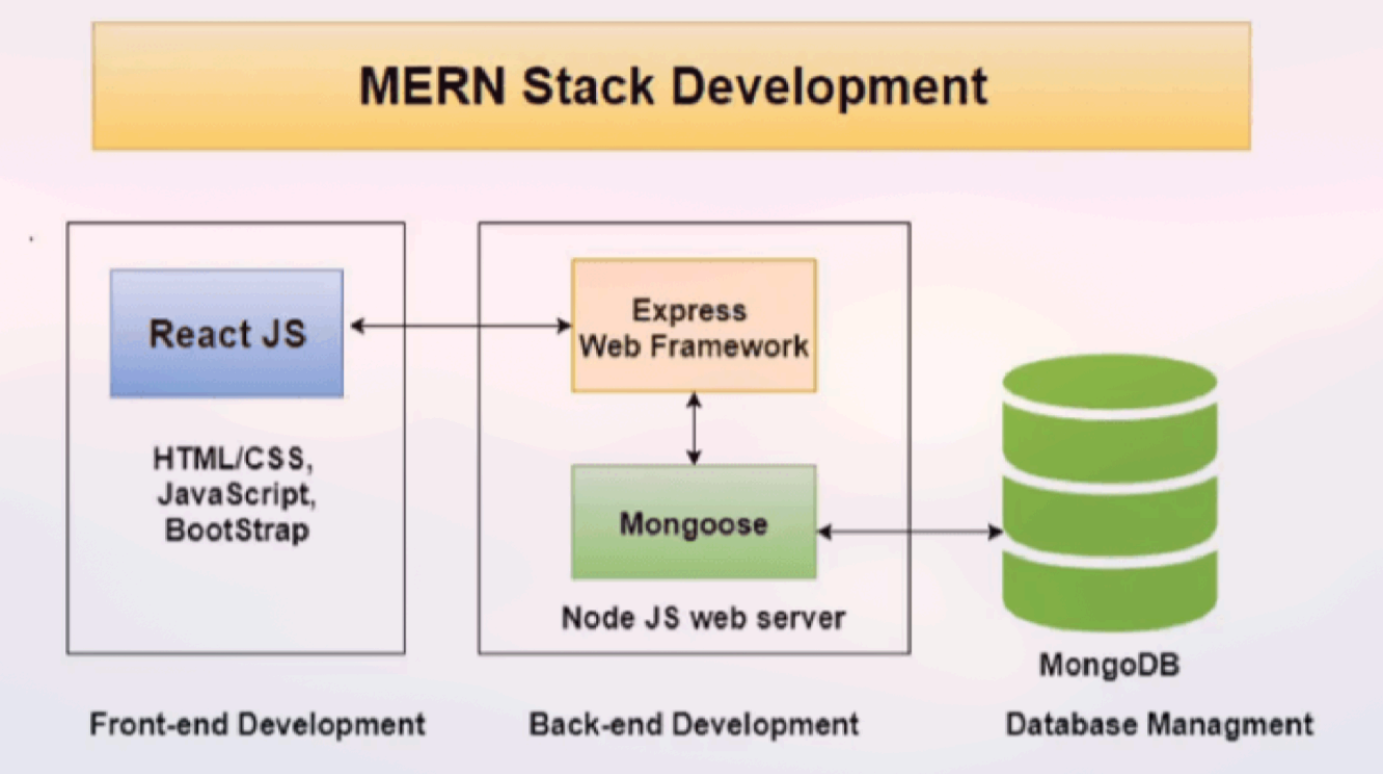


- **N-Tier (Multi-tier) (Page 14):** Scaled form. Each function (presentation, application processing, data management) can be an isolated layer.
- **Difference: Client-Server vs. Peer-to-Peer (Page 14):**

Feature	Client-Server Architecture	Peer-to-Peer Architecture
Roles	Specific clients and servers.	No differentiation; peers are equal.
Data Management	Centralized.	Each peer has its own data/applications.
Objective	Disseminate knowledge/exchange relevant data.	Encourage peer connectivity, relationships.
Data Provision	Only in response to a request.	Peers can request and provide services.
Scalability	Suitable for small and large networks.	Better for limited users (e.g., <10 devices).

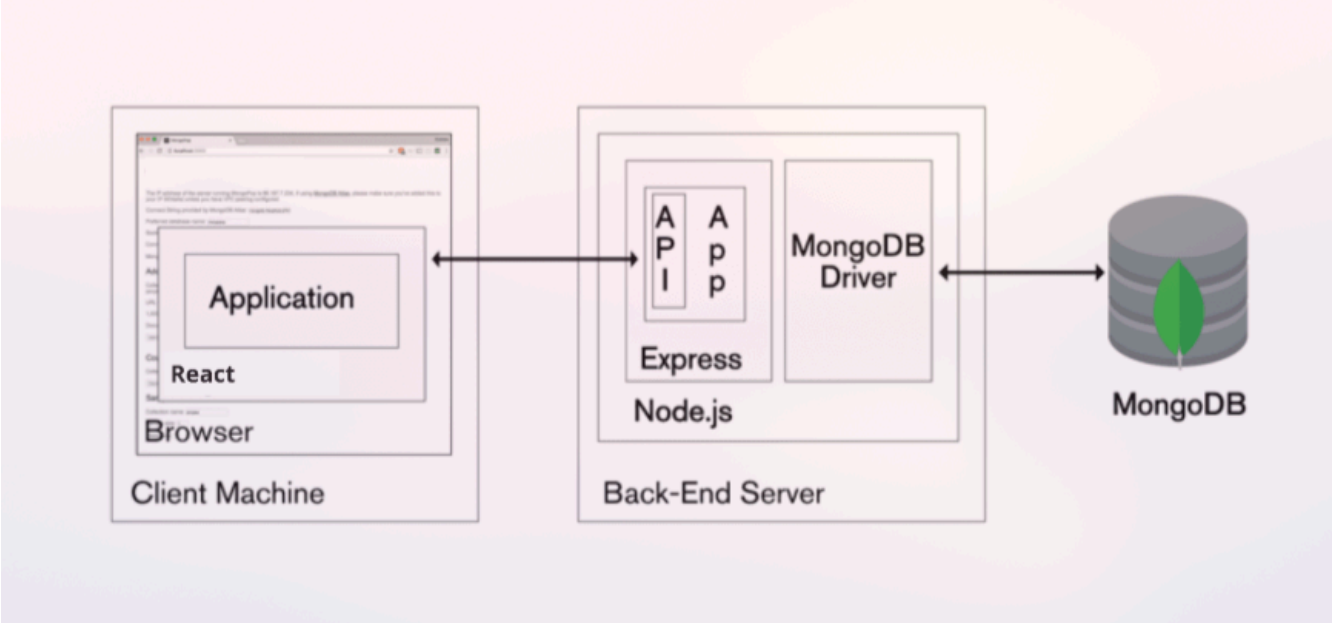
- **Advantages & Disadvantages (Page 15):**
- **Advantages:** Centralized control, central access to files, good UI, easy resource sharing.
- **Disadvantages:** If primary server fails, architecture disrupted; expensive hardware/software; needs specific networking OS; traffic congestion with many users; requires technical expertise for maintenance.

1.5 Introduction to MERN Stack (Page 17-22) [PYQ Q1(a) directly asks for MERN's 3-tier architecture]



- **Definition:** MERN is a popular JavaScript stack used for building full-stack web applications. The acronym MERN stands for its core components:
  1. **M - MongoDB (Page 17, 20):** A NoSQL (Non-Structured Query Language), document-oriented database system.
  2. **E - Express.js (Page 17, 21):** A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It acts as middleware on top of Node.js web APIs.
  3. **R - React.js (Page 17, 21):** A client-side JavaScript library for building user interfaces (UIs) and single-page applications (SPAs).
  4. **N - Node.js (Page 17, 22):** A cross-platform, open-source JavaScript runtime environment that executes JavaScript code outside a web browser, primarily used for server-side development.
- The MERN stack facilitates smoother and simpler development as both frontend and backend primarily use JavaScript.
- **Other Stacks Mentioned (Page 17):**
  - **LAMP:** Linux, Apache, MySQL, PHP
  - **MEAN:** MongoDB, ExpressJS, AngularJS, NodeJS

- **MERN Stack Architecture (3-Tier) (Page 19-20, Diagrams on Page 18 & 19):** [PYQ Q1(a)]



  - MERN adheres to a 3-tier architecture system:
    1. **Tier 1: Web or Front-end Tier (Handled by React.js) (Page 20):**
       - **Definition:** The top tier responsible for the user interface and user experience.
       - **React.js:** An open-source, front-end JavaScript library for building dynamic client-side web applications and complex UIs.
       - **Key Feature:** Allows reusability of code (components), which is time-saving and beneficial.
       - (Diagram on Page 18 shows: React JS with HTML/CSS, JavaScript, Bootstrap for Front-end Development).
       - (Diagram on Page 19 shows: Client Machine with Browser running React Application).
    2. **Tier 2: Server or Middle-Tier (Handled by Express.js and Node.js) (Page 20):**
       - **Definition:** The layer that handles application logic, processes client requests, and interacts with the database.
       - **Express.js:** Maintains the server-side framework and middleware.
       - **Node.js:** The server environment where Express.js runs. Node.js is a cross-platform runtime environment for executing JavaScript code outside a browser.
       - This combination empowers developers to create dynamic APIs and web servers.
       - (Diagram on Page 18 shows: Express Web Framework and Mongoose (an ODM for MongoDB) running on Node JS web server for Back-end Development).
       - (Diagram on Page 19 shows: Back-End Server with Node.js hosting Express (with API logic) and MongoDB Driver).
    3. **Tier 3: Database as Backend Tier (Handled by MongoDB) (Page 20):**
       - **Definition:** The data storage layer of the application.
       - **MongoDB:** Stores all application-related data.

        - Maintains proper records, can generate replica files for data redundancy, and retrieves specific information as requested. It's a popular NoSQL database.
        - (Diagram on Page 18 shows: MongoDB for Database Management).
        - (Diagram on Page 19 shows: MongoDB as the database).

- **Impacts/Benefits of MERN Stack (Page 18):**
    1. **Cost-effective:** All four technologies are open source. Using JavaScript for both frontend and backend can reduce development costs.
    2. **SEO friendly:** Websites created using MERN technologies can be designed to be easily crawlable by search engines like Google and Yahoo.
    3. **Better performance:** Fast exchange of responses between backend, frontend, and database improves website speed and user experience.
    4. **Improves Security:** Refers to web application security measures protecting web servers, applications, and APIs.
    5. **Provide the fastest delivery:** Web and mobile applications can be built and delivered faster.
    6. **Provides faster Modifications (Page 19):** Technologies support quick modifications and agile development.
    7. **Open Source (Page 19):** All four technologies are open-source, leading to better community and development support, making development faster and more efficient.
    8. **Easy to switch between client and server (Page 19):** Since it's written in one primary language (JavaScript), switching context between client and server development is simpler.
- **Detailed Components of MERN Stack:**
    - **MongoDB (Page 20):**
        1. Popular NoSQL, open-source, document-oriented database.
        2. 'NoSQL' means non-relational; doesn't require a fixed schema, stores data in a different format than relational tables (uses BSON).
        3. Data stored in **BSON (Binary JavaScript Object Notation)** format.
        4. BSON structure encodes length and type of information, allowing quick parsing and a highly scalable, flexible document structure.
        5. Faster than RDBMS due to efficient storage and better indexing.
        6. Uses JavaScript as a coding language (big advantage for the stack).
        7. Schemaless: data stored in separate documents.
        8. Supports a flexible document model, fast for developers.
        9. Scales easily by adding more servers; increases productivity with its flexible model.
    - **Express.js (Page 21):**
        1. JavaScript-based server-side framework running within Node.js.
        2. A leading backend development framework for creating and maintaining robust servers.
        3. Used for building web and mobile applications efficiently.

4. Allows developers to spin up robust APIs and web servers with ease.
5. Adds robust functionalities like routing and middleware, making it easy to connect with databases (like MongoDB) and handle errors.
6. Large suite of third-party add-ons provides better functionality, security, and speed.
7. Built-in router promotes code reusability.

- **React.js (Page 21):**
  1. Popular open-source JavaScript library for building UIs, especially for single-page web applications.
  2. It's a library developed by Facebook, not a full framework.
  3. Allows creation of reusable UI components that are fast, simple, and scalable.
  4. Can be used with other JavaScript libraries or frameworks.
  5. Offers fast performance due to its use of a virtual DOM and efficient data update mechanisms (immutability of data concept).
- **Node.js (Page 22):**
  1. Open-source, cross-platform runtime environment that executes JavaScript code outside the browser (server-side).
  2. It is NOT a programming language or a framework itself.
  3. Used for building numerous backend services for web and mobile.
  4. Incredibly consistent, provides robust and quick service.
  5. Key role in building real-time web apps due to faster synchronization.
  6. Non-blocking/Asynchronous nature (due to the event loop) allows fast processing of operations.
  7. Provides caching properties (single modules can be cached), reducing re-execution of code and quickening response time.

2. MARKUP LANGUAGES

2.1 Introduction to HTML (HyperText Markup Language)

- **Definition:** The standard markup language used to create web pages and web applications. It describes the structure of a web page semantically.
- HTML documents are plain text files that can be created using any text editor.
- HTML files typically have `.html` or `.htm` extensions.
- (Handwritten notes, "HTML Jan 16"): HTML is the code used to structure a web page and its contents. The components used to design a website are called "elements".
- **Basic Structure (Handwritten notes, "Basic HTML Page on VS Code Jan 19"):**

```
<!DOCTYPE html> <!-- Tells the browser you are using HTML5 -->
<html> <!-- Root element of an HTML page -->
<head> <!-- Contains meta-information about the HTML document (not
```

```
displayed) -->
<title>My 1st Page</title> <!-- Title shown in browser tab or window title
bar -->
</head>
<body> <!-- Contains the visible page content -->
<p>Hello world!</p> <!-- A paragraph element -->
</body>
</html>
```

- (Handwritten notes, Jan 21 "Quick points about HTML"):
- It is not case sensitive (though lowercase is recommended for consistency and XHTML compatibility).
- `<html>` tag is the parent of `<head>` and `<body>` tags.
- Most HTML elements have opening and closing tags with content in between.
- Some tags have no content (empty elements), e.g., `<br>` (break line tag).

2.2 HTML Elements and HTML Tags

- **HTML Element (Handwritten notes, "HTML tag Jan 17"):**
- **Definition:** An individual component of an HTML document. It usually consists of a start tag, content, and an end tag.
- Example: `<p>This is a paragraph.</p>` (Here, `<p>` is the start tag, `</p>` is the end tag, and "This is a paragraph." is the content).
- Empty elements (e.g., `<br>`, `<img>`, `<input>`) do not have end tags in HTML5 (they are self-closing in XHTML: `<br />`).
- **HTML Tag:**
- **Definition:** Tags are the keywords surrounded by angle brackets `< >` used to define HTML elements. They tell the browser how to format and display the content.
- `<head>` tag (Handwritten notes, after Jan 17): Tells information which is not to be displayed on the Web Page but must be known by the code.
- `<body>` tag (Handwritten notes, after Jan 17): Tells the information which actually displays on the web page.
- **Attributes (Handwritten notes, Jan 19 "lang attribute"):**
- Provide additional information about HTML elements.
- Always specified in the start tag.
- Usually come in name/value pairs like: `name="value"`.
- Example: `<html lang="en">` (specifies the language of the document is English).
- `<meta charset="UTF-8">` (Handwritten notes, Jan 19): Specifies character encoding. UTF-8 is universal and covers almost all characters and symbols. [PYQ indirectly related if discussing web page rendering]

- `<meta name="viewport" content="width=device-width, initial-scale=1.0">` (Handwritten notes, Jan 19): Configures the viewport for responsiveness on different devices. "Viewport means website must be responsive."

2.3 Basics of XHTML (Extensible HyperText Markup Language) (Handwritten notes, Jan 26)

- **Definition:** A stricter, more XML-based version of HTML. It's HTML defined as an XML application.
- Developed in 2000 (HTML5 in 2014).
- It is case sensitive (unlike HTML).
- **Key Differences/Rules (Major Differences from HTML):**

1. **`<!DOCTYPE>` is mandatory.**
2. **XML namespace (`xmlns`) attribute in `<html>` is mandatory.**
3. **Elements must be properly nested.** (e.g., `<b><i>text</i></b>` is correct, `<b><i>text</b></i>` is not).
4. **Elements must always be closed.** (e.g., `<p>text</p>`, `<br />`).
5. **Attribute names must be in lowercase.** (e.g., `<img src="...">` not `<img SRC="...">`).
6. **Attribute values must be quoted.** (e.g., `<input type="text">`).
7. **Attribute minimization is forbidden.** (e.g., use `checked="checked"`, not just `checked`).
8. **Documents must have one root element** (typically `<html>`).

- **Steps to convert from HTML to XHTML (Handwritten notes, Jan 26):**

1. Add XHTML `<!DOCTYPE>` to the first line.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- Or Transitional, Frameset depending on needs -->
```

(Handwritten notes Jan 28 shows `XHTML 1.1 //EN` with DTD URL, and for `XHTML 1.0`, three types: Strict, Transitional, Frameset).
2. Add `xmlns` attribute to html elements.

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

3. Change all element names to lowercase.
4. Close all elements.
5. Change all attribute names to lowercase and quote attribute values.

- **XHTML Document Structure Example (Handwritten notes, Jan 27):**

```
<?xml version="1.0" encoding="UTF-8"?> <!-- XML declaration, UTF-8 encoding
must be specified -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
```

```
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Title of a Web Page</title>
</head>
<body>
Contents to be displayed on the web page...
</body>
</html>
```

2.4 HTML Lists (Page 23-34)

- **Definition:** Used to group related pieces of information, making them clearly associated and easy to read. Important for navigation and general content.
- **Advantages (Page 23):**
- **Flexibility:** Easy to reorder items in ordered lists.
- **Styling:** Specific tags allow targeted CSS styling.
- **Semantics:** Provides proper structure, aiding accessibility (e.g., screen readers).
- **Types of Lists (Page 23):**

1. **Unordered List (`<ul>`) (Page 24):** [PYQ Q2(c)(ii) `<em>` `<i>` are often used within list items]

- Used when item order doesn't matter (e.g., shopping list).
- Displays with bullets by default (can be changed with CSS).
- Each item is an `<li>` (list item) element.
- Example (Page 25):

```
<h2>Shopping List</h2>
<ul>
<li>bread</li>
<li>coffee beans</li>
<li>milk</li>
<li>butter</li>
</ul>
```

2. **Ordered List (`<ol>`) (Page 26):**

- Used when item order is specific (e.g., cooking instructions).
- Displays with numbers by default. Browser auto-numbers.
- Each item is an `<li>` element.
- Attributes:
- `type`: Can change numbering style (e.g., `a`, `A`, `i`, `I`). (Page 27)

- `start`: Specifies starting number for the list (e.g., `start="4"`). (Page 28)
- Example (Page 27):

```
<h2>Cooking Instructions</h2>
<ol>
<li>Gather ingredients</li>
<li>Mix ingredients together</li>
<li>Place ingredients in a baking dish</li>
</ol>
```

3. **Description List (`<dl>`) (Page 29-30):**

- Used for name/value pairs (terms and definitions, questions/answers).
- Consists of:
- `<dl>`: The description list itself.
- `<dt>`: Defines a term/name.
- `<dd>`: Describes/defines the term/name.
- Example (Page 30):

```
<h2>List of Single Names with Single Values</h2>
<dl>
<dt>Bread</dt>
<dd>A baked food made of flour.</dd>
<dt>Coffee</dt>
<dd>A drink made from roasted coffee beans.</dd>
</dl>
```

- **Nesting Lists (Page 33-34):**
- An `<li>` item can contain another entire list (`<ul>` or `<ol>`).
- Useful for hierarchical structures like tables of contents.
- Good rule: Don't nest lists deeper than three levels to avoid confusion.
- Example (Page 33):

```
<h4>Nesting List Example</h4>
<ol>
<li>Chapter One
<ol>
<li>Section One</li>
<li>Section Two</li>
</ol>
</li>
<li>Chapter Two</li>
</ol>
```

- **Summary of List Tags (Page 32):**

Tag	Description
`<ul>`	Defines an unordered list
`<ol>`	Defines an ordered list
`<li>`	Defines a list item
`<dl>`	Defines a description list
`<dt>`	Defines a name in a description list
`<dd>`	Describes the value in a description list

2.5 HTML Tables (Page 43-49)

- **Definition:** Allow web authors to arrange data (text, images, links, other tables) into rows and columns of cells.
- **Basic Structure:**
- `<table>`: Defines the table.
- `<tr>`: Defines a table row.
- `<td>`: Defines a table data cell.
- `<th>`: Defines a table header cell (typically bold and centered).
- **Attributes:**
- `border`: Specifies a border for the table (e.g., `border="1"`).
- `cellpadding` (Page 44): Distance between cell borders and content.
- `cellspacing` (Page 44): Width of the border between cells.
- `colspan` (Page 44): Merges two or more columns into a single column.
- `rowspan` (Page 45): Merges two or more rows into a single row.
- `bgcolor` (Page 45): Sets background color for table or cell.
- `background` (Page 45): Sets background image for table or cell.
- `bordercolor` (Page 45): Sets border color.
- `width`, `height` (Page 46): Sets table dimensions (pixels or percentage).
- **Table Elements:**
- `<caption>` (Page 47): Title or explanation for the table (appears at the top).
- `<thead>` (Page 48): Groups header content in a table.
- `<tbody>` (Page 48): Groups body content in a table.
- `<tfoot>` (Page 48): Groups footer content in a table. (Note: `<thead>` and `<tfoot>` should appear before `<tbody>` in the code).

- **Example with `<th>`, `colspan`, `rowspan` (adapted from Page 43 & 45):**

```
<table border="1">
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td rowspan="2">Row 1 & 2, Cell 1</td>
<td>Row 1, Cell 2</td>
<td>Row 1, Cell 3</td>
</tr>
<tr>
<td>Row 2, Cell 2</td>
<td>Row 2, Cell 3</td>
</tr>
</tbody>
<tfoot>
<tr>
<td colspan="3">Row 3, Spanning all 3 Columns</td>
</tr>
</tfoot>
</table>
```

- **Nested Tables (Page 48):** A `<td>` can contain another `<table>`.

2.6 HTML Forms (Page 15-16, 35-42)

- **Definition:** Used to collect data from site visitors (e.g., user registration, contact forms).
- Input is taken from the visitor and posted to a back-end application (CGI, ASP, PHP script) for processing.
- The `<form>` Tag (Page 35):
- Used to create an HTML form.
- Syntax: `<form action="Script URL" method="GET|POST"> ...form elements... </form>`
- **Form Attributes (Page 35):**
- `action`: URL of the script that will process the form data.
- `method`: HTTP method to use when submitting the form (GET or POST).
- **GET:** Appends form data to the URL; good for short, non-sensitive data; limited size.

- **POST:** Sends form data in the HTTP request body; more secure for sensitive data; no size limitations.
- `target`: Where to display the response (`_blank`, `_self`, `_parent`).
- `enctype`: How form-data should be encoded when submitting (especially for file uploads).
- `application/x-www-form-urlencoded`: Standard for simple scenarios.
- `multipart/form-data`: Used for uploading files.
- **HTML Form Controls (Page 35):**
- **Text Input Controls (`<input type="text">`, `<input type="password">`, `<textarea>`) (Page 36-38):**
- `<input type="text">`: Single-line text input (e.g., names, search boxes).
- Attributes: `name`, `value`, `size`, `maxlength`, `placeholder` (Page 15).
- `<input type="password">`: Single-line text input that masks characters.
- Attributes: Same as `text`.
- `<textarea>`: Multi-line text input (e.g., descriptions, comments).
- Attributes: `name`, `rows`, `cols`.
- (Page 15 shows an example `Basic Form in HTML` with these attributes for input type text).
- **Checkbox Control (`<input type="checkbox">`) (Page 38):**
- Used when multiple options can be selected.
- Attributes: `name`, `value`, `checked`.
- **Radio Button Control (`<input type="radio">`) (Page 39):**
- Used when only one option from many can be selected.
- Radio buttons in a group must share the same `name` attribute.
- Attributes: `name`, `value`, `checked`.
- **Select Box Control (`<select>`, `<option>`) (Page 39-40):**
- Dropdown list.
- `<select>` attributes: `name`, `size`, `multiple`.
- `<option>` attributes: `value`, `selected`, `label`.
- **File Select Box (`<input type="file">`) (Page 40):**
- Allows users to upload files.
- Attributes: `name`, `accept` (specifies file types).
- **Button Controls (`<input type="submit">`, `<input type="reset">`, `<input type="button">`, `<input type="image">`, `<button>`) (Page 41):**
- `<input type="submit">`: Submits the form.
- `<input type="reset">`: Resets form controls to initial values.

- `<input type="button">`: Generic button, often used with JavaScript.
- `<input type="image">`: Clickable image button (submits the form).
- `<button>` element: More flexible button, can contain HTML content.
- **Hidden Controls (`<input type="hidden">`) (Page 42):**
- Hides data within the page, submitted with the form but not visible to the user.
- Used to pass information like current page number, user ID, etc.
- **Basic Form Example (Page 15-16, adapted):**

```
<body>
<form action="submit_form.php" method="post">
Name:
<input type="text" name="Name" size="15" maxlength="30" placeholder="Name">
<br><br>
Email:
<input type="email" name="email" size="15" maxlength="30"
placeholder="Email"><br><br>
Gender:
<input type="radio" name="gender" value="male"> Male
<input type="radio" name="gender" value="female"> Female<br><br>
Date of Birth:
<input type="date" name="date_of_birth"><br><br>
<input type="submit" value="Submit">
</form>
</body>
```

2.7 Defining XHTML's Abstract Syntax

- **Definition:** XHTML's abstract syntax refers to the underlying rules and structure that define what constitutes a valid XHTML document, independent of how it's serialized (written as text). It's primarily concerned with the hierarchical tree structure of elements and their relationships.
- **Key Aspects (derived from XHTML rules):**

1. **Well-formedness:** All XHTML documents must be well-formed XML. This includes:

- A single root element (e.g., `<html>`).
- All elements must have closing tags (or be self-closed if empty, e.g., `<br />`).
- Elements must be properly nested (no overlapping tags).
- Attribute values must be quoted.
- Element and attribute names are case-sensitive (typically lowercase).

2. **Tree Structure:** The document is a hierarchical tree of elements, starting from the root element. Each element can have parent, child, and sibling relationships.

3. **DTD (Document Type Definition):** XHTML documents must declare a DTD (Strict, Transitional, or Frameset for XHTML 1.0; XHTML 1.1 has its own) which specifies the allowed elements, attributes, and their nesting rules, further constraining the syntax.

- Essentially, the strict rules of XHTML (lowercase tags, closing all tags, proper nesting, quoted attributes) are practical manifestations of its rigorously defined abstract syntax based on XML.

2.8 XML (Extensible Markup Language)

- **Definition:** A markup language designed to carry data, not to display data. It allows users to define their own tags to describe the structure and meaning of the data.
- **Key Characteristics:**
- **Extensible:** You can create your own tags relevant to your data.
- **Text-based:** XML files are plain text.
- **Human-readable and Machine-readable.**
- **Self-descriptive:** Tags describe the content they enclose.
- **Hierarchical:** Data is structured in a tree-like format.
- **Strict Syntax:** Must be well-formed (similar rules to XHTML: proper nesting, closing tags, single root element, case-sensitive).
- **Purpose:** Often used for data exchange between different systems, configuration files, and as a base for other markup languages (like XHTML, SVG).
- **Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

- **Relation to XHTML:** XHTML is an application of XML; it's HTML rewritten to conform to XML's stricter rules.

3. CSS STYLE SHEETS

3.1 Introduction to CSS (Cascading Style Sheets) (Handwritten notes, "Abhay CSS")

- **Definition:** CSS is a stylesheet language used to describe the presentation (look and formatting) of a document written in a markup language like HTML or XML. It controls how HTML elements are displayed on screen, paper, or in other media.
- CSS beautifies the web page or website.
- It can control the layout of multiple web pages all at once.

- Developed by Håkon Wium Lie in 1996.
- Nowadays, every browser supports CSS.

3.2 CSS Core Syntax (Handwritten notes, "Internal CSS")

- A CSS rule consists of a selector and a declaration block.
- **Selector:** Points to the HTML element(s) you want to style.
- **Declaration Block:** Contains one or more declarations separated by semicolons, enclosed in curly braces `{}`.
- **Declaration:** Consists of a CSS property name and a value, separated by a colon.
- Syntax:

```
selector {
property1: value1;
property2: value2;
}
```

- Example:

```
h1 {
color: red; /* Declaration 1: property 'color', value 'red' */
background-color: yellow; /* Declaration 2 */
}
```

3.3 Types of CSS (How to apply CSS) (Handwritten notes, "Abhay CSS")

1. **Inline CSS:**

- Applies styles directly to a single HTML element using the `style` attribute.
- Used for quick, specific styling but generally not recommended for large-scale styling due to poor maintainability.
- Example (Handwritten notes, "Inline CSS"):

```
<h1 style="color: red; font-size: 24px;">Welcome to our class</h1>
```

2. **Internal CSS (Embedded CSS):**

- CSS rules are placed within a `<style>` tag in the `<head>` section of an HTML document.
- Affects only the HTML document in which it is embedded.
- Example (Handwritten notes, "Internal CSS"):

```
<head>
<title>CSS</title>
<style>
h1 {
```

```
color: red;
background-color: yellow;
}
</style>
</head>
<body>
<h1>Welcome to our class</h1>
<h1>Computer class</h1>
</body>
```

3. **External CSS:**

- CSS rules are defined in a separate `.css` file (e.g., `style.css`).
- The HTML document links to this external stylesheet using the `<link>` tag in the `<head>` section.
- Best for styling multiple pages, promoting consistency and maintainability.
- Example (Handwritten notes, "External CSS May 21"):
- `abc.css` file:

```
h1 {
color: white;
background-color: blue;
text-align: center;
height: 2cm;
line-height: 2cm; /* for vertical centering */
}
```

- `abc.html` file:

```
<head>
<title>CSS</title>
<link rel="stylesheet" type="text/css" href="abc.css">
</head>
<body>
<h1>Welcome to our class</h1>
</body>
```

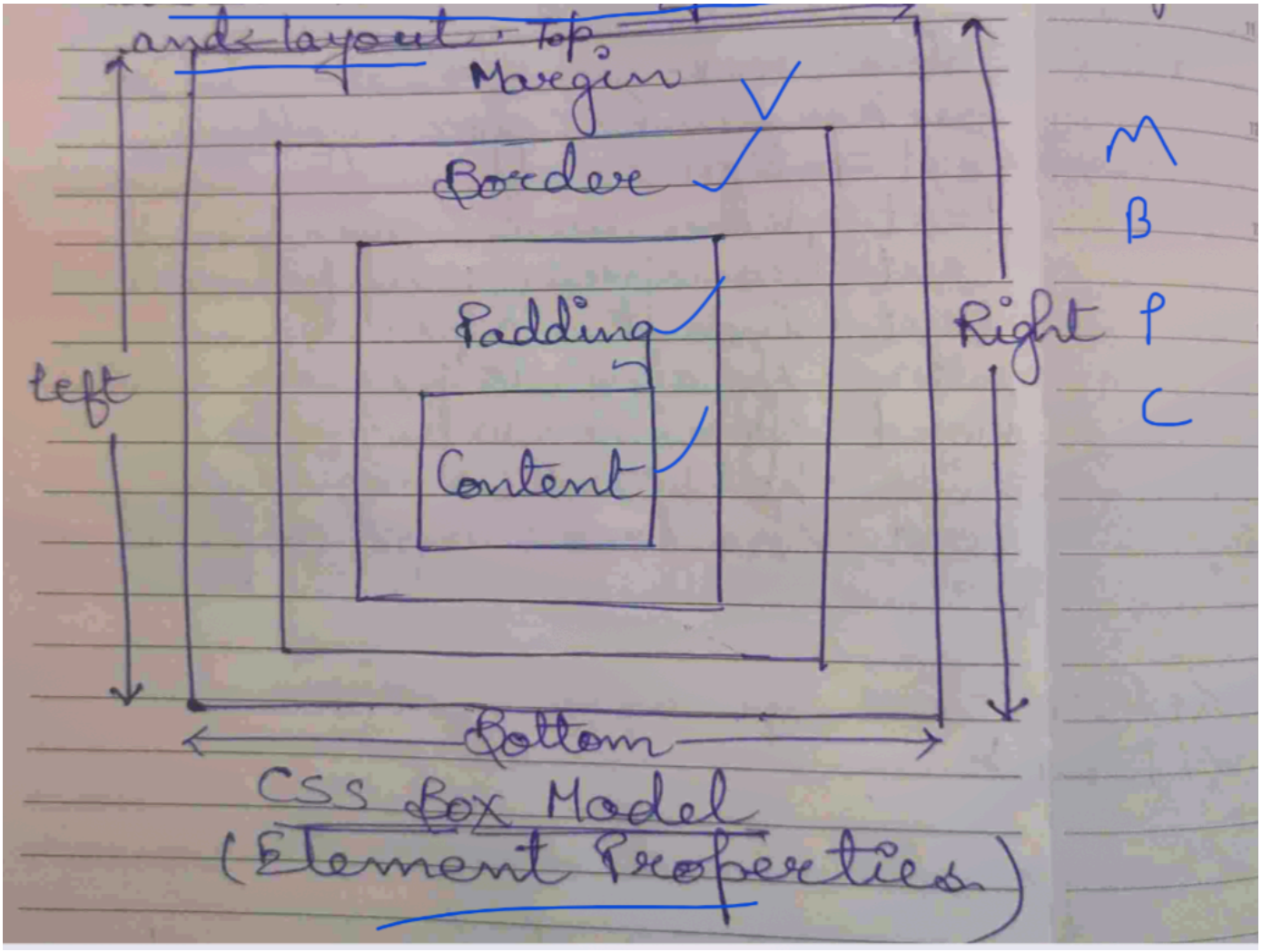
3.4 CSS Text Properties (Handwritten notes, "CSS Text Properties May 23")

- `color`: Sets the color of the text.
- `background-color`: Sets the background color of the text element.
- `text-align`: Aligns the text within its element (e.g., `left`, `right`, `center`, `justify`).
- `text-decoration`: Adds decoration to text (e.g., `none`, `underline`, `overline`, `line-through`).

- `text-transform`: Controls capitalization of text (e.g., `none`, `capitalize`, `uppercase`, `lowercase`).
- `text-indent`: Indents the first line of text in a block. (e.g., `5px`)
- `letter-spacing`: Adjusts space between characters. (e.g., `2px`)
- `word-spacing`: Adjusts space between words. (e.g., `10px`)
- `line-height`: Sets the distance between lines of text. (e.g., `60px` or `1.5`)
- `text-shadow`: Adds shadow to text (e.g., `text-shadow: 5px 4px red;` or `10px 10px 30px blue;` for horizontal offset, vertical offset, blur radius, color).
- Example (Handwritten notes, after "CSS Text Properties"):

```
h1 {
color: red;
background-color: yellow;
text-align: left;
text-decoration: overline;
text-transform: uppercase;
text-indent: 10px;
letter-spacing: 15px;
word-spacing: 10px;
line-height: 60px;
text-shadow: 10px 10px 30px blue;
}
```

3.5 CSS Box Model (Page 2, Q3(b)(ii) in PYQ) (Handwritten notes, "CSS Box Model May 26")



- **Definition:** In CSS, all HTML elements can be considered as boxes. The CSS box model is a box that wraps around every HTML element. It describes how elements are sized and spaced.
- It consists of (from innermost to outermost):

1. **Content:** The actual content of the box, where text and images appear. (Defined by the element itself, includes text, media, etc. It is the visible part.)
2. **Padding:** The blank space around the content, inside the border. Clears an area around the content. Padding is transparent. (Padding shows outside the content and inside the border.)
3. **Border:** A border that goes around the padding and content. (Border is declared around the Element Content and Padding. Border shows around padding and inside the Margin.)
4. **Margin:** The blank space outside the border. Clears an area outside the border. The margin is transparent. (The free space inside the browser window and outside the Border.)

- (Diagram is present in handwritten notes "CSS Box Model May 26" and "Box Model Manages layout May 29")
- Example (Handwritten notes, "div span May 28, 29"):

```
<style>
div {
height: 100px;
```

```
width: 100px;
background: red;
margin-bottom: 20px; /* Space between div and span */
padding: 10px;
border: 10px solid black;
outline: 10px solid blue; /* Outline is drawn outside the border */
}
span {
height: 100px; /* May not apply if display is inline */
width: 100px;  /* May not apply if display is inline */
background: green;
display: block; /* To make height/width apply for span */
}
</style>
<body>
<div>Hey, I am box</div>
<span></span>
</body>
```

- Output depiction shows:
- `div`: `outline` (blue), `border` (black), `padding` (red color shows here), `content` (text).
- `span`: `background` (green).
- `margin-bottom` shows distance between the two elements.

3.6 Normal Flow Box Layout (Handwritten notes, "Basic CSS Layouts May 30", "Block Vs Inline Elements June")

- **Definition:** Normal flow is the default way browser layout HTML elements on a page. Elements are laid out one after another as they appear in the HTML source.
- **Types of Elements in Normal Flow:**

1. **Block-level Elements:**

- Always start on a new line and take up the full width available.
- Examples: `<div>`, `<h1>` - `<h6>`, `<p>`, `<form>`, `<ul>`, `<ol>`, `<li>`.
- Height and width properties apply.

2. **Inline Elements:**

- Do not start on a new line and only take up as much width as necessary.
- Flow along with surrounding text/content.
- Examples: `<span>`, `<a>`, `<img>`, `<strong>`, `<em>`, `<i>`.
- Height and width properties generally do not apply directly (padding/margin work horizontally but might behave unexpectedly vertically).

- **CSS `display` Property:** Can change the default layout behavior.
- `display: block;` (Makes an inline element behave like block)
- `display: inline;` (Makes a block element behave like inline)
- `display: inline-block;` (Allows setting width/height but flows inline)
- `display: none;` (Hides the element and removes it from the flow)
- Example (Handwritten notes, "Block Vs Inline Elements June", "style.css June Mon"):
- `index.html`:

```
<head>
<title>Basic layout in CSS</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>This is a block-level element.</h1>
<p>This is also a block-level element.</p>
<span>This an inline-level element.</span>
<span>This is another inline-level element.</span>
</body>
```

- `style.css`:

```
h1, p { /* Block elements */
display: block;
background-color: red; /* For visibility */
}
span { /* Inline elements */
display: inline;
background-color: lightblue; /* For visibility */
padding: 5px;
margin: 5px;
}
```

The output would show `h1` and `p` each on new lines, taking full width. `span` elements would be on the same line, taking only necessary width.

3.7 Other Properties (like list, tables)

- **Styling Lists with CSS:**
- `list-style-type`: Changes the marker of list items (e.g., `none`, `disc`, `circle`, `square` for `<ul>`; `decimal`, `lower-roman`, `upper-alpha` for `<ol>`).
- `list-style-image`: Uses an image as the list item marker.

- `list-style-position`: Specifies if markers appear inside or outside the content flow (`inside`, `outside`).
- Padding and margin can be used to adjust spacing.

```
ul.custom {
list-style-type: square;
padding-left: 20px;
}
ol.custom {
list-style-type: lower-roman;
margin-left: 30px;
}
```

- **Styling Tables with CSS:**
- `border`: Can be applied to `table`, `th`, `td`.
- `border-collapse`: Specifies if table borders should be collapsed into a single border or separated (`collapse`, `separate`).
- `padding`: Adds space inside `th` and `td` cells.
- `text-align`: Aligns text within cells.
- `background-color`: Can be set for `table`, `tr`, `th`, `td`.
- `:hover` pseudo-class can be used for row/cell highlighting.

```
table.styled {
width: 100%;
border-collapse: collapse;
}
table.styled th, table.styled td {
border: 1px solid black;
padding: 8px;
text-align: left;
}
table.styled th {
background-color: #f2f2f2;
}
table.styled tr:hover {
background-color: #ddd;
}
```

3.8 XSLT (Extensible Stylesheet Language Transformations)

- **Definition:** XSLT is a language used for transforming XML documents into other XML documents, or other formats such as HTML, plain text, or XSL Formatting Objects (XSL-FO) which can then be converted to PDF.

- **Core Components:**
- **XSLT:** The transformation language itself.
- **XPath:** A language for navigating and selecting parts of an XML document. XSLT uses XPath to match and select elements for processing.
- **XSL-FO (Optional):** An XML vocabulary for specifying formatting semantics (for print/PDF output).
- **How it Works:**

1. An XSLT processor takes an XML input document and an XSLT stylesheet.
2. The stylesheet contains "templates" that define how to transform specific XML elements or patterns found via XPath.
3. The processor applies these templates to the input XML and produces an output document.

- **Purpose:**
- Converting XML data into an HTML page for display in a browser.
- Transforming XML from one schema (structure) to another.
- Generating reports or summaries from XML data.
- **Basic Example (Conceptual):**
- If you have XML data like:

```
<book>
<title>Learning XSLT</title>
<author>John Doe</author>
</book>
```

- An XSLT stylesheet could transform this into HTML:

```
<div>
<h1>Learning XSLT</h1>
<p>By: John Doe</p>
</div>
```

- The XSLT stylesheet would contain rules like:

```
<xsl:template match="book">
<div>
<h1><xsl:value-of select="title"/></h1>
<p>By: <xsl:value-of select="author"/></p>
</div>
</xsl:template>
```

4. CLIENT-SIDE PROGRAMMING: JAVASCRIPT [PYQ Q1(e), Q2(a), Q3(a), Q3(b)(iii) are from this section]

4.1 Introduction to JavaScript (JS) (Page 118-120, Handwritten "Live Script Feb 2")

- **Definition:** JavaScript is a high-level, interpreted scripting language primarily known for its use in web browsers to create dynamic and interactive web pages. It can also be used in other environments (e.g., Node.js for server-side programming).
- (Page 119): Used in millions of web pages to improve design, validate forms, detect browsers, create cookies, etc. It is the most popular scripting language on the internet and works in all major browsers.
- (Page 120):
- Designed to add interactivity to HTML pages.
- A scripting language (lightweight programming language).
- Consists of lines of executable computer code.
- Usually embedded directly into HTML pages.
- Interpreted language (scripts execute without preliminary compilation).
- Free to use (no license purchase needed).
- (Handwritten "Live Script Feb 2"):
- Invented by Brendan Eich (Netscape Communications).
- It is an interpreted programming language with object-oriented capabilities.
- It is a dynamic programming language.
- Supports multiple inheritance concept (conceptually, through prototypes rather than classical inheritance).
- Primarily client-side scripting.
- Browser -> JavaScript Engine (executes JS code).
- JS programs are run by an interpreter built into the user's web browser.
- It is case sensitive. JS ignores whitespace in your program (mostly, except within strings).
- **JavaScript is NOT Java.** (Page 121) They are completely different languages in concept and design. Java is a more complex, compiled programming language (like C++).

4.2 Basic Syntax & Embedding JS in HTML

- **Embedding in HTML (Page 122, Handwritten "Java Script Jan 29"):**
- JS code is placed within `<script>` tags.
- The `type="text/javascript"` attribute was common, but is optional in HTML5 as JavaScript is the default scripting language.
- (Handwritten "Basic Syntax of Java Script Feb 4") Type I (common):

```
<script type="text/javascript"> // or just <script>
document.write("Hello World!");
alert('Welcome to JS');
</script>
```

- Type II (older, less common):

```
<script language="JavaScript">
document.write("Education for you");
</script>
```

- `<script>` tag attributes (`language`, `type`):
- `language`: Specifies which language you are using (value is `JavaScript`). (Older attribute)
- `type`: Indicates which script language you are using with type attribute (value `text/javascript`). (More common, but optional in HTML5)
- Scripts can be placed in `<head>` or `<body>`.
- **In `<head>`:** Often for functions or code that needs to be loaded before the body.
- **In `<body>`:** Usually at the end, for code that manipulates DOM elements after they've loaded, or for better perceived page load speed.
- **External JS File:**

```
<script src="myscript.js"></script>
```

- **Statements and Semicolons (Page 123, Handwritten "Semicolons Feb 5"):**
- JS statements are commands to be executed.
- Semicolons (`;`) are used to separate statements.
- They are technically optional at the end of a line if each statement is on its own line. However, they are *required* if you put more than one statement on a single line.
- It's a good practice to always use semicolons to avoid ambiguity.
- **Comments (Handwritten "Semicolons Feb 5"):**
- Same as C++ and other languages.
- Single-line comment: `// This is a comment`
- Multi-line comment: `/* This is a multi-line comment */`
- **Case Sensitivity:** JavaScript is case-sensitive (e.g., `myVariable` is different from `myvariable`).

4.3 Variables & Data Types (Page 124, Handwritten "Variables, Data Types, Literals Feb 12")

- **Variables (Page 124):**
- **Definition:** Used to store data values. A variable is like a "container" for information. Its value can change during script execution.
- **Declaration:**
- `var variableName;` (Older way, function-scoped or globally-scoped)
- `let variableName;` (ES6+, block-scoped)
- `const variableName = value;` (ES6+, block-scoped, constant value, must be initialized)
- **Naming Rules:**

- Names can contain letters, digits, underscores, and dollar signs.
- Must begin with a letter, `$`, or `_`.
- Case-sensitive (e.g., `strname` is not `STRNAME`).
- Reserved keywords (like `var`, `let`, `function`) cannot be used as names.
- (Handwritten "Var, let, const Mar 6"):
- `var`: Functional scope. If declared outside a function, it's global. Can be redeclared and updated. Hoisted with `undefined`.
- `let`: Block scope (`{}`). Cannot be redeclared in the same scope. Can be updated. Hoisted but not initialized (Temporal Dead Zone).
- `const`: Block scope. Cannot be redeclared or updated. Must be initialized. Hoisted but not initialized.
- **Data Types (Handwritten "Variables, Data Types, Literals Feb 12"):** JavaScript is dynamically typed (variable types are determined at runtime).

1. **Primitive Types:**

- **String:** Sequence of characters (e.g., `"Hello"`, `'world'`).

```
let name = "Alice";
console.log(typeof name); // "string"
```

- **Number:** Numeric values (integers or floating-point) (e.g., `100`, `3.14`). Special values: `Infinity`, `-Infinity`, `NaN` (Not a Number).

```
let age = 30;
let price = 19.99;
console.log(typeof age); // "number"
```

- **BigInt:** For integers larger than the `Number` type can hold (add `n` to end). (e.g., `123n`).

```
let bigNumber = 1234567890123456789012345678901234567890n;
console.log(typeof bigNumber); // "bigint"
```

- **Boolean:** Logical values: `true` or `false`.

```
let isActive = true;
console.log(typeof isActive); // "boolean"
```

- **Undefined:** A variable that has been declared but not assigned a value. Also, the value of a function that doesn't explicitly return anything.

```
let x;
console.log(x); // undefined
console.log(typeof x); // "undefined"
```

- **Null:** Represents the intentional absence of any object value. It's an assignment value. (Typeof null is "object" - a known quirk).

```
let car = null;
console.log(car); // null
console.log(typeof car); // "object"
```

- **Symbol (ES6+):** Unique and immutable primitive value, often used as object property keys.

```
let sym = Symbol("id");
console.log(typeof sym); // "symbol"
```

2. **Object Type (Reference Type):**

- **Object:** A collection of key-value pairs (properties and methods).

```
let person = {firstName: "John", lastName: "Doe"};
console.log(typeof person); // "object"
```

- **Array:** A special type of object used to store ordered collections of values. (See section 4.7)
- **Function:** A block of code designed to perform a particular task. (See section 4.5)

4.4 Literals (Handwritten "Variables, Data Types, Literals Feb 12")

- **Definition:** Literals are fixed values in source code, not variables. They represent values directly.
- Examples:
- `100` (Number literal)
- `3.14` (Number literal)
- `"Hello"` (String literal)
- `'World'` (String literal)
- `true` (Boolean literal)
- `false` (Boolean literal)
- `null` (Null literal)
- `{name: "John"}` (Object literal)
- `[1, 2, 3]` (Array literal)
- (Handwritten "Console.log(234+456) Feb 13" - `234` and `456` are number literals, the expression evaluates to `690`)
- (Handwritten "Console.log("A" + "A") Feb 13" - `"A"` is a string literal, evaluates to `"AA"`)

4.5 Operators (Page 125-128)

- **Arithmetic Operators (Page 125):**

Operator	Description	Example	Result
`+`	Addition	`x=2, y=2; x+y`	`4`
`-`	Subtraction	`x=5, y=2; x-y`	`3`
`*`	Multiplication	`x=5, y=4; x*y`	`20`
`/`	Division	`15/5` -> `3`, `5/2` -> `2.5`	
`%`	Modulus (remainder)	`5%2` -> `1`, `10%8` -> `2`	
`++`	Increment	`x=5; x++` (now x is 6)	
`--`	Decrement	`x=5; x--` (now x is 4)	

- **Assignment Operators (Page 126):**

Operator	Example	Same As
`=`	`x = y`	`x = y`
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

- **Comparison Operators (Page 127):**

Operator	Description	Example
`==`	Equal to (value, type conversion)	`5 == "5"` -> `true`
`===`	Strictly equal to (value and type)	`5 === "5"` -> `false`
`!=`	Not equal to (value, type conversion)	`5 != "5"` -> `false`
`!==`	Strictly not equal to (value and type)	`5 !== "5"` -> `true`
`>`	Greater than	`5 > 8` -> `false`
`<`	Less than	`5 < 8` -> `true`
`>=`	Greater than or equal to	`5 >= 8` -> `false`
`<=`	Less than or equal to	`5 <= 8` -> `true`

- **Logical Operators (Page 128):**

Operator	Description	Example
`&&`	Logical AND	`(x < 10 && y > 1)`
`!`	Logical NOT	`!(x == y)`
'or'	Logical OR	`(x == 5 or y == 5)`

- **String Concatenation:** The `+` operator is also used to concatenate (join) strings.

```
let greeting = "Hello" + " " + "World!"; // "Hello World!"
```

- **Conditional (Ternary) Operator:**

```
let age = 20;
let voteable = (age < 18) ? "Too young" : "Old enough"; // "Old enough"
```

4.6 Functions (Handwritten "Function Syntax in Java Script Feb 18", "Function with Parameter Feb 19") [PYQ Q1(e) "closures in JavaScript"]

- **Definition:** A block of reusable code designed to perform a particular task. Functions are executed when they are "called" (invoked).
- **Declaration (Function Definition):**

```
function functionName(parameter1, parameter2) {
// code to be executed
return result; // optional
}
```

- **Function Invocation (Calling):**

```
functionName(argument1, argument2);
```

- **Parameters vs. Arguments:**
- **Parameters:** Variables listed inside the parentheses `()` in the function definition.
- **Arguments:** The actual values passed to the function when it is invoked.
- **Return Value:** Functions can return a value using the `return` statement. If no `return` statement is used, or `return` is used without a value, the function returns `undefined`.
- **Function Scope:** Variables declared inside a function (using `var`, `let`, or `const`) are typically local to that function and not accessible from outside.
- **Example (Handwritten "Function Syntax in Java Script Feb 18"):**

```
<script>
function myIntro() {
document.write('I am Shyam.<br>');
document.write('I am an Animation Specialist.<br>');
```

```
}
myIntro(); // Calling the function
myIntro(); // Calling again
</script>
<!-- O/P: I am Shyam.
I am an Animation Specialist.
I am Shyam.
I am an Animation Specialist. -->
```

- **Example with Parameters (Handwritten "Function with Parameter Feb 19"):**

```
function sum(a, b) {
document.write(a + b + "<br>");
}
sum(10, 20); // O/P: 30
sum(20, 40); // O/P: 60
```

- **Function Expressions (Anonymous Functions):**

```
const greet = function(name) {
return "Hello " + name;
};
console.log(greet("Alice")); // "Hello Alice"
```

- **Arrow Functions (ES6+):** A more concise syntax for writing functions.

```
const add = (a, b) => a + b;
console.log(add(5, 3)); // 8

const greetByName = name => "Hello " + name; // Single parameter, no
parentheses
console.log(greetByName("Bob")); // "Hello Bob"
```

- **Closures [PYQ Q1(e)]**
- **Definition:** A closure is an inner function that has access to the outer (enclosing) function's variables and parameters (its scope chain). This access persists even after the outer function has finished executing.
- **How it works:** When a function is created, a closure is created, packaging the function together with references to its surrounding state (the lexical environment).
- **Use Cases:**
- Creating private variables and methods (data encapsulation).
- Maintaining state in asynchronous operations (e.g., callbacks, event handlers).
- Currying and partial application.
- **Example:**

```
function outerFunction(outerVariable) {
return function innerFunction(innerVariable) {
console.log("Outer Variable: " + outerVariable);
console.log("Inner Variable: " + innerVariable);
console.log("Sum: " + (outerVariable + innerVariable));
}
}

const newFunction = outerFunction(10); // outerFunction has executed
// newFunction is now the innerFunction, but it still "remembers"
outerVariable = 10
newFunction(5);
// Output:
// Outer Variable: 10
// Inner Variable: 5
// Sum: 15
```

4.7 Objects (Handwritten "Object Advance Level of Array Feb 21", "Object Feb 23", "Object Mar 5")

- **Definition:** In JavaScript, an object is a standalone entity with properties and type. It's a collection of key-value pairs where keys are strings (or Symbols) and values can be any data type, including other objects or functions (methods).
- **Creating Objects:**

1. **Object Literal (most common):**

```
// Handwritten "Object Mar 5"
let person = {
firstName: "Rohit", // property
lastName: "Sharma",
age: 60,          // property
greet: function() { // method
console.log("Hello, my name is " + this.firstName);
}
};
console.log(person.firstName); // "Rohit"
console.log(person.age);       // 60
person.greet();                // "Hello, my name is Rohit"
document.write(person.age);    // 60 on page
```

2. **Using `new Object()`:**

```
let car = new Object();
car.make = "Toyota";
```

```
car.model = "Camry";
```

3. **Constructor Functions (ES5 style "classes"):**

```
function Person(first, last) {
this.firstName = first;
this.lastName = last;
}
let myFather = new Person("John", "Doe");
```

4. **ES6 Classes:**

```
class Rectangle {
constructor(height, width) {
this.height = height;
this.width = width;
}
get area() {
return this.calcArea();
}
calcArea() {
return this.height * this.width;
}
}
const square = new Rectangle(10, 10);
console.log(square.area); // 100
```

- **Accessing Properties:**
- Dot notation: `objectName.propertyName`
- Bracket notation: `objectName["propertyName"]` (useful if property name is dynamic or has spaces/special chars).
- **`this` Keyword:** Inside an object's method, `this` refers to the object itself.
- **Types of Objects (Handwritten "Object Feb 23"):**

1. **Built-in Objects (Native Objects):** Standard objects provided by JS (e.g., `String`, `Number`, `Boolean`, `Date`, `Math`, `Array`, `RegExp`).
2. **Host Objects:** Objects provided by the browser environment (e.g., `window`, `document`, `location`, `history`, `NodeList`, `HTMLElement`).
3. **User-defined Objects:** Objects created by the programmer.

- (Handwritten "Object Feb 21"): An object is a collection of properties. A property is an association between a name (key) and a value. A property's value can be a function, which is then called a method.

4.8 Arrays (Handwritten "Array in Java Script Feb 27", "let subject new Array Feb 28", "var a = [10,20,30] Mar")

- **Definition:** An array is a special type of object used to store an ordered collection of multiple values under a single variable name. Array items are indexed, starting from 0.
- **Creating Arrays:**

1. **Array Literal (most common):**

```
// Handwritten "let subject = ['Math', 'Physics'];"
let fruits = ["Apple", "Banana", "Orange"];
let numbers = [10, 20, 30, 40, 50];
let mixed = [10, "Harry", "Sarah", true, null]; // Handwritten "var. ary = [10, "Harray", ..."] Mar Mo"
```

2. **Using `new Array()`:**

```
// Handwritten "let subject = new Array('Math', 'Physics');"
let colors = new Array("Red", "Green", "Blue");
```

- **Accessing Array Elements:** Using zero-based index.

```
console.log(fruits[0]); // "Apple"
console.log(numbers[2]); // 30
```

- **Modifying Array Elements:**

```
// Handwritten "Update: subject[3] = 'English';"
fruits[1] = "Mango"; // fruits is now ["Apple", "Mango", "Orange"]
```

- **Array Properties and Methods:**
- `length`: Returns the number of elements in the array.

```
console.log(fruits.length); // 3
```

- Common Methods: `push()`, `pop()`, `shift()`, `unshift()`, `splice()`, `slice()`, `concat()`, `join()`, `indexOf()`, `forEach()`, `map()`, `filter()`, `reduce()`.
- **Iterating through an Array (Handwritten "for (var a=0; a<4; a++) Mar"):**

```
// Handwritten "var ary = [10,20,30,40,50]; ... document.write(ary[a] + "<br>");"
let nums = [10, 20, 30, 40, 50];
for (let i = 0; i < nums.length; i++) {
console.log(nums[i]);
}
// Using forEach (ES5+)
nums.forEach(function(num) {
```

```
console.log(num);
});
```

- **Nested Arrays (Handwritten "Nested Array in Java Script Mar 3"):** Arrays can contain other arrays.

```
// Handwritten "animals = ["cat", "dog", ["hawk", "eagle"]];"
let matrix = [
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]
];
console.log(matrix[1][1]); // 5 (accessing element at row 1, column 1)
// Handwritten "console.log(cities[3][1]) -> 'b'"
// var cities = ["Delhi", "Mumbai", "Chennai", ["a", "b", "c"]];
```

4.9 Built-in Objects

- JavaScript provides several standard built-in objects.
- `Math` **Object:** Provides properties and methods for mathematical constants and functions.

```
console.log(Math.PI);         // 3.141592653589793
console.log(Math.sqrt(16));  // 4
console.log(Math.random());  // Random number between 0 (inclusive) and 1
(exclusive)
console.log(Math.floor(4.7)); // 4
console.log(Math.ceil(4.2));  // 5
console.log(Math.round(4.5)); // 5
```

- `Date` **Object:** Used for working with dates and times.

```
let now = new Date();
console.log(now.toString());
console.log(now.getFullYear());
console.log(now.getMonth()); // 0-11 (January is 0)
console.log(now.getDate());   // Day of the month
console.log(now.getHours());
```

- `String` **Object:** Strings are primitive, but JS provides String object wrappers for methods.

```
let text = "Hello World";
console.log(text.length);         // 11
console.log(text.toUpperCase());  // "HELLO WORLD"
console.log(text.indexOf("World")); // 6
console.log(text.slice(0, 5));    // "Hello"
```

- `Number` **and** `Boolean` **Objects:** Similar to String, these primitives have object wrappers.

- `JSON` **Object (ES5+):** For parsing JSON strings and converting objects to JSON strings.

```
let myObj = { name: "John", age: 30 };
let myJSON = JSON.stringify(myObj); // '{"name":"John","age":30}'
let parsedObj = JSON.parse(myJSON); // { name: "John", age: 30 }
```

- Others include `RegExp` (Regular Expressions).

4.10 JavaScript Form Programming (Interaction with HTML Forms)

- **Definition:** Using JavaScript to interact with HTML form elements, primarily for validation before submission, dynamic manipulation, or enhancing user experience.
- **Accessing Form Elements:**
- Using `document.getElementById('elementId')`
- Using `document.forms['formName'].elements['elementName']`
- Using `document.querySelector()` or `document.querySelectorAll()`
- **Getting/Setting Values:**
- For most input types: `element.value`
- For checkboxes: `element.checked` (boolean)
- **Form Validation Example:**

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

<script>
function validateForm() {
let x = document.forms["myForm"]["fname"].value;
if (x == "") {
alert("Name must be filled out");
return false; // Prevents form submission
}
return true; // Allows form submission
}
</script>
```

- (Page 119 mentions "validate forms" as a key use of JavaScript).

4.11 Intrinsic Event Handling (Handwritten "Java Script Event Handling Mar 10", "Java Script Basic Events Mar 12") [PYQ Q3(a)]

- **Definition:** Events are actions that occur in the browser, such as a user clicking a button, a page finishing loading, or a mouse moving over an element. Event handling is the process of responding

to these events.

- **Intrinsic event handlers** are HTML attributes that execute JavaScript code when a specific event occurs on that element.
- **Event Handling Identifies (Handwritten "Java Script Event Handling Mar 10"):**
- Where an event should be founded.
- It makes the forward event.
- It takes some kind of appropriate action in circumstances such as writing to a log or sending a message.
- The event handler may ultimately forward the event to an event consumer.
- **Common Event Attributes:**
- `onclick`: User clicks an element.

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

- `onload`: Page or image finishes loading. (Used on `<body>` or `<img>`)

```
<body onload="displayTime()"> <!-- Handwritten "Body onload="time()" Mar 12"
-->
```

- `onunload`: User closes the browser window or navigates away.
- `onchange`: Value of an input field, select, or textarea changes.
- `onmouseover`: Mouse pointer moves over an element.
- `onmouseout`: Mouse pointer moves out of an element.
- `onmousedown`: Mouse button is pressed down on an element.
- `onmouseup`: Mouse button is released over an element.
- `onfocus`: Element gets focus.
- `onblur`: Element loses focus.
- `onsubmit`: Form is submitted (used on `<form>`).
- `onkeydown`, `onkeypress`, `onkeyup`: Keyboard events.
- **Example (Handwritten "button onclick="hello()" Mar 13"):**

```
<head>
<title>Event Example</title>
<script>
function hello() {
document.write("Hello Everyone");
}
</script>
</head>
<body>
```

```
<button onclick="hello()">Click Me</button> <!-- O/P on click: Page shows "Hello Everyone" -->
</body>
```

- **Modern Approach (addEventListener):** While intrinsic event handlers are simple, `addEventListener` is more flexible and preferred for complex applications.

```
document.getElementById("myBtn").addEventListener("click", function() {
alert("Button clicked via addEventListener!");
});
```

4.12 Modifying Element Style (Handwritten "Modifying Element Style using JS Mar 14")

- **Definition:** JavaScript can dynamically change the CSS style properties of HTML elements.
- **Accessing Style Property:**
- Each HTML element has a `style` object property.
- CSS properties are accessed using camelCase (e.g., `backgroundColor` for `background-color`).
- **Syntax:** `document.getElementById("elementId").style.propertyName = "newValue";`
- **Example (Handwritten "Modifying Element Style using JS Mar 14"):**

```
<h1 id="hid">JS</h1>
<button onclick="changeStyle()">Change Color</button>

<script>
function changeStyle() {
// Syntax: document.getElementById(id).style.property = newStyle
document.getElementById("hid").style.color = "red";
document.getElementById("hid").style.fontSize = "50px";
document.getElementById("hid").style.backgroundColor = "yellow";
}
</script>
```

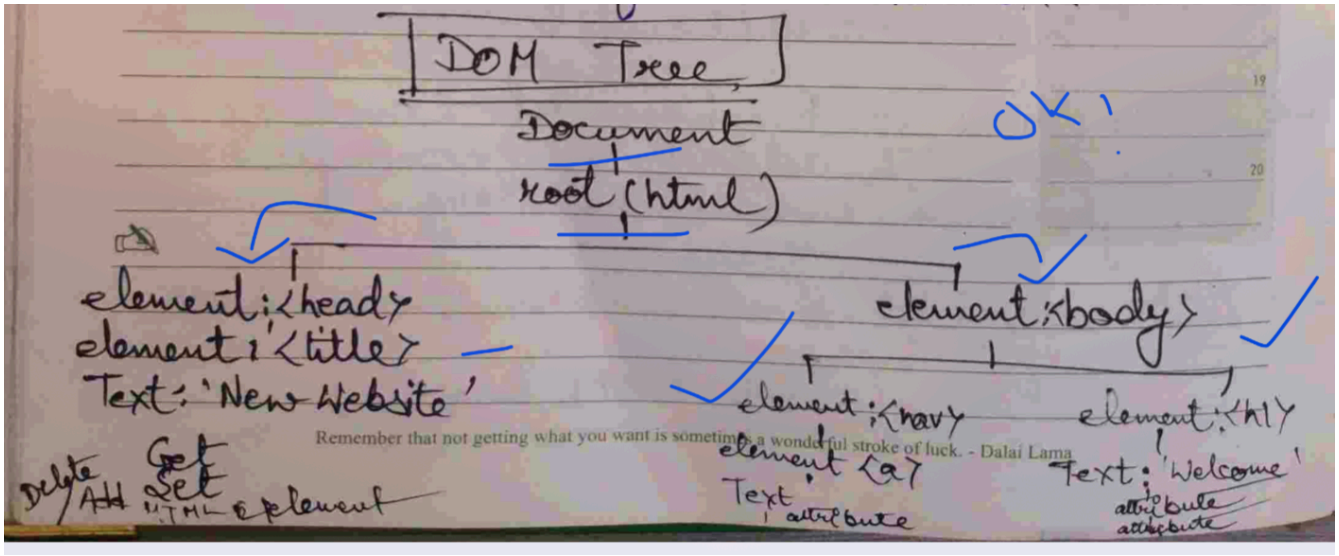
(Handwritten notes also show `<button style="..." onclick="document.getElementById('hid').style.color='red'">Click Me</button>` for direct modification).

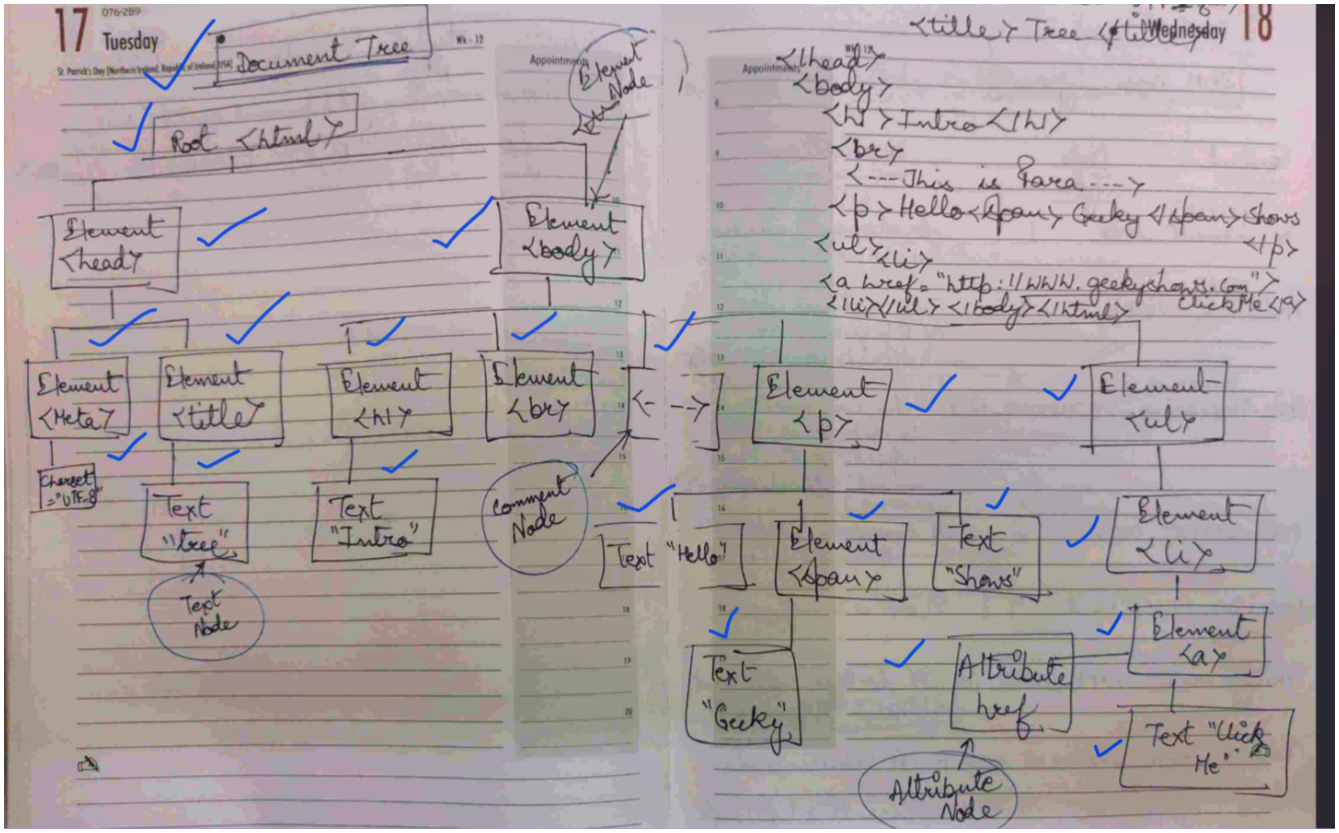
4.13 Document Trees (DOM - Document Object Model) (Handwritten "DOM Feb 24", "Categories of DOM Feb 26", "Document Tree Mar 17") [PYQ Q3(b)(iii)]

- **Definition (DOM):** The Document Object Model is a programming interface (API) for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as a tree of objects (nodes).
- (Handwritten "DOM Feb 24"): Affiliate programming I/F (API) for HTML & XML documents. With DOM, programmers can create, build documents, navigate their structure, & add, modify, or delete

elements & content. DOM is an OO (Object Oriented) representation of the web page, which can be modified with a scripting language such as JS. Acc. to DOM, every HTML tag is an object.

- **DOM Tree (Document Tree):**

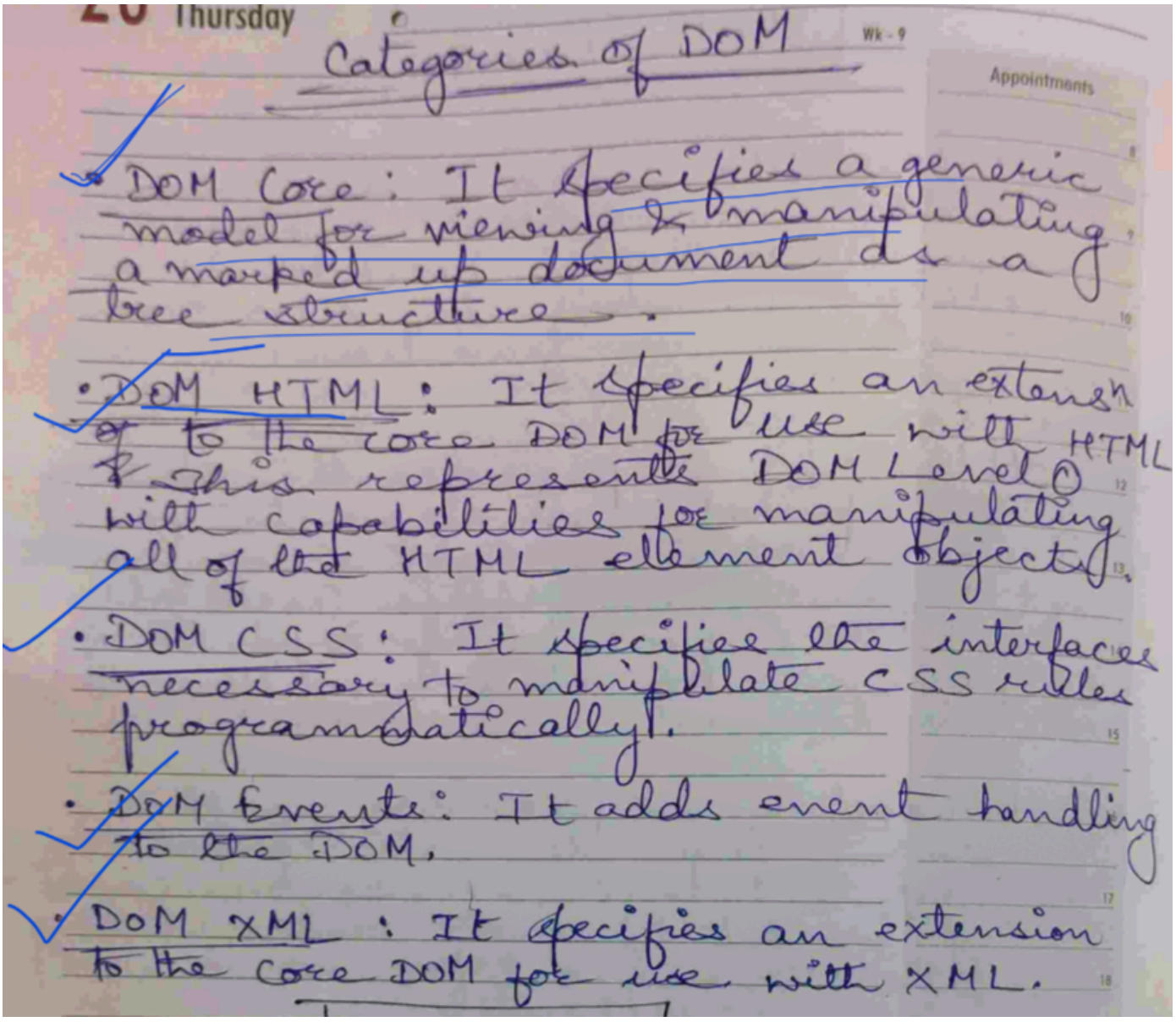




- The browser parses an HTML document and creates a tree-like structure of nodes.
- The root node is typically the `document` object itself, with `<html>` as its main child.
- Each HTML element becomes an element node. Text within elements becomes text nodes. Attributes become attribute nodes. Comments become comment nodes.
- (Handwritten "Document Tree Mar 17" shows a diagram: `Root (html)` -> `Element (head)`, `Element (body)`. `head` -> `Element (title)` -> `Text ("Tree")`. `body` -> `Element (h1)` -> `Text ("Intro")`, etc.)
- **Node Relationships (Handwritten "Node Relationships Mar 20"):**
- **Parent:** A node that directly contains another node.

- **Child:** A node directly contained by another node.
- **Siblings:** Nodes that share the same parent.
- **Ancestor:** A parent, grandparent, etc.
- **Descendant:** A child, grandchild, etc.
- Properties like `parentNode`, `childNodes`, `firstChild`, `lastChild`, `nextSibling`, `previousSibling` are used to navigate the tree.
- **DOM Node Types (Handwritten "DOM Node Types Mar 19"):**

Constant	Node Value	Description
`ELEMENT_NODE`	`1`	An element such as `<h1>` or `<div>`
`TEXT_NODE`	`3`	The actual Text of an Element or Attribute.
`COMMENT_NODE`	`8`	A Comment node.
`DOCUMENT_NODE`	`9`	A Document node.
`DOCUMENT_TYPE_NODE`	`10`	A `<!DOCTYPE html>` node.
`DOCUMENT_FRAGMENT_NODE`	`11`	A lightweight container for DOM nodes.
(Note: `ATTRIBUTE_NODE` (value 2) exists but attributes are not typically part of the main document tree in the same way as element/text nodes; they are properties of element nodes).		



- **Categories/Levels of DOM (Handwritten "DOM Levels Feb 25", "Categories of DOM Feb 26"):**
- **DOM Level 0:** The earliest, informal specification. Basic event handling.
- **DOM Level 1:**
- **Core:** Generic model for any structured document (XML, HTML).
- **HTML:** Specific additions for HTML documents.
- **DOM Level 2:** Added event handling, style sheet manipulation, views, traversal.
- **Events:** Standardized event model.
- **Style:** Interface to CSS.
- **Traversal and Range:** For navigating and selecting parts of a document.
- **DOM Level 3:** Added content models (DTDs, schemas), XPath support, saving/loading.
- **DOM Level 4+ (Living Standard):** Maintained by WHATWG, continuously updated. Includes features like query selectors (`querySelector`, `querySelectorAll`).
- **Accessing HTML Elements in JS [PYQ Q2(a)]:**
- `document.getElementById(id)`: Selects an element by its `id`.

- `document.getElementsByTagName(tagName)`: Selects all elements with the given tag name (returns an HTMLCollection).
- `document.getElementsByClassName(className)`: Selects all elements with the given class name (returns an HTMLCollection).
- `document.querySelector(cssSelector)`: Returns the first element that matches a CSS selector.
- `document.querySelectorAll(cssSelector)`: Returns all elements that match a CSS selector (returns a NodeList).

4.14 ECMAScript (ES)

- **Definition:** ECMAScript is a scripting language specification standardized by Ecma International. JavaScript is the most well-known implementation of the ECMAScript standard.
- **Versions:**
- **ES1-ES3 (1997-1999):** Early versions, laid the foundation.
- **ES4 (Abandoned):** Ambitious but never finalized.
- **ES5 (ECMAScript 2009):** A significant update, added strict mode, JSON support, new Array methods (`forEach`, `map`, `filter`, etc.), Object methods (`Object.keys`, `Object.defineProperty`). Much of the JavaScript used in the provided notes aligns with ES5.
- **ES6 (ECMAScript 2015):** A major overhaul, often called ES2015. Introduced many new features.
- **ES7 (ES2016) onwards:** Annual releases with smaller, incremental updates.

Key ES6 (ECMAScript 2015) Features:

- **`let` and `const`:** Block-scoped variable declarations.
- **Arrow Functions:** Concise function syntax (`=>`).
- **Template Literals (Template Strings):** Easier string interpolation and multi-line strings (using backticks `` `${variable}` ``).
  (Handwritten "Template Literals Mar 23" shows:

```
const fname = "Vinod";
const lname = "Thapa";
// ReactDOM.render( // This is React specific, but shows template literal
concept
//  `<h1>My name is ${fname} ${lname}</h1>
//   <p>my lucky number is ${5+5}</p>`,
//  document.getElementById("root")
// );
let message = `My name is ${fname} ${lname}. My lucky number is ${5+5}.`;
console.log(message);
```

)

- **Default Parameters:** For function parameters.
- **Rest and Spread Operators (`...`):** For handling multiple arguments or expanding iterables.
- **Destructuring Assignment:** For arrays and objects.
- **Classes:** Syntactic sugar over JavaScript's existing prototype-based inheritance.
- **Modules (import/export):** For organizing code into reusable pieces.
- **Promises:** For easier asynchronous programming.
- **New Collections:** `Map`, `Set`, `WeakMap`, `WeakSet`.
- **`for...of` loop:** For iterating over iterable objects (Arrays, Strings, Maps, Sets).
- **ECMAScript6 (and later versions like ES2016, ES2017, etc.)** continue to evolve JavaScript, making it more powerful and easier to write complex applications. Modern web development heavily relies on these newer features.

ES5 vs. ES6 (ECMAScript 2015) - Comparison Table

Feature	ES5 (ECMAScript 2009)	ES6 (ECMAScript 2015)
Variable Declaration	`var` (function-scoped or global-scoped, hoisted with `undefined`, can be re-declared/updated)	`let` (block-scoped, hoisted but not initialized - TDZ, can be updated, not re-declared in same scope) `const` (block-scoped, hoisted but not initialized - TDZ, must be initialized, cannot be re-declared or reassigned)
Function Syntax	Standard `function` declarations and expressions.	**Arrow Functions (`=>`):** Concise syntax, lexical `this` binding, no `arguments` object, cannot be used as constructors.
`this` in Functions	Depends on how the function is called (invocation context).	Arrow functions inherit `this` from the surrounding (enclosing) scope. Traditional functions still behave like ES5.
String Handling	Concatenation with `+`. Multi-line strings via `\n` or `+`.	**Template Literals (`` ` ``):** Easy string interpolation (`${expression}`), direct multi-line string support.
Default Parameters	Manual check for `undefined` to assign default values.	Default values can be set directly in the function signature (e.g., `function greet(name = 'Guest')`).
Handling Multiple Arguments	`arguments` object (array-like, not a true array).	**Rest Parameters (`...args`):** Collects an indefinite number of arguments into a true array.
Expanding Iterables	Manual iteration or methods like `concat` for arrays.	**Spread Operator (`...iterable`):** Expands iterables (arrays, strings) into individual elements for array literals, function calls, etc.
Object/Array Value Extraction	Individual assignments (e.g., `var name = person.name;`).	**Destructuring Assignment:** Concise syntax to extract values from objects or

Feature	ES5 (ECMAScript 2009)	ES6 (ECMAScript 2015)
		arrays into variables (e.g., `const {name, age} = person;`).
Object-Oriented Programming	Constructor functions and prototype-based inheritance.	**`class` syntax:** Syntactic sugar over prototype-based inheritance, providing a more familiar OOP structure (includes `constructor`, `extends`, `super`).
Modules	No built-in module system (used CommonJS, AMD via libraries).	**Native Modules:** `import` and `export` keywords for creating and using reusable code modules.
Asynchronous Operations	Primarily callbacks, prone to "callback hell".	**Promises:** For cleaner handling of asynchronous operations, representing eventual completion or failure.
Looping over Iterables	`for` loop with index, `forEach` (for arrays).	**`for...of` loop:** A simpler syntax to iterate directly over the values of iterable objects (Arrays, Strings, Maps, Sets).
New Data Structures	Limited built-in data structures beyond Array and Object.	**`Map`, `Set`, `WeakMap`, `WeakSet`:** New collection types for more specialized data handling.
Unique Identifiers	No dedicated primitive type for unique IDs.	**`Symbol`:** A new primitive data type for creating unique and immutable identifiers, often used as object property keys.
Object Literal Enhancements	Standard property definition.	Shorthand syntax for defining properties (if variable name matches key) and methods.