FRONT END WEB TECHNOLOGY

HYPERTEXT MARKUP LANGUAGE[HTML]

What is HTML?

HTML stands for Hypertext Markup Language. It's the language used to create websites, webpages, and web applications. HTML is like the skeleton of a webpage; it gives the structure.

• First Created By: Tim Berners-Lee in 1993.

• Latest Version: HTML5, released in 2014, has some cool new features for videos, forms, and more.

File Extension: .html

When you create an HTML file, you save it with the extension .html. This tells the browser that the file is an HTML document. For example:

• index.html

Website vs. Web Application

Here's the difference between a website and a web application:

• **Website**: A website is usually static. This means it doesn't change unless someone updates it. Think of things like blogs or informational sites that show the same content each time you visit. Example: A homepage.

• **Web Application:** A web app is dynamic, meaning it can change and interact with the user. You can enter information, and the website will give you a response. Examples are Gmail or Facebook, where you log in and interact.

Webpage, HTML Tags/Elements Webpage:

A webpage is just one page of a website. It's made with HTML tags and elements that tell the browser how to show the page.

HTML Tags:

HTML is made of tags. These are little instructions that tell the browser what to do with the content. They are usually in the form of opening and closing tags.

• Opening Tag: This starts the element. For example, <h1> for a heading.

• Closing Tag: This ends the element. It looks like </h1>. The content goes between these tags.

Example: This is a paragraph

Where to Write HTML Code

You can write HTML code in different places, such as:

1. Code Editors
    - Visual Studio Code
    - Sublime Text
    - Atom
2. Text Editors (like Notepad or TextEdit) can work too, but they don't help as much with errors.
3. Online Editors: If you don't want to install anything, you can use online editors that let you

write and test HTML in the browser:

OneCompiler (https://onecompiler.com/) and many more.

Where to View HTML Code

Once you've written your HTML code, you can view it in a web browser. Any browser works, like:

- Chrome
- Firefox
- Safari
- Microsoft Edge

To see your HTML file in a browser:

- Save it with the .html extension.
- Double-click the file, and it will open in your default browser.

HTML STRUCTURE

HTML structure is divided into 4 parts.

- Declaration Part
- Root Part
- Head Part
- Body part

FULL STRUCTURE EXAMPLE

<!DOCTYPE html>: Tells the browser to use HTML5.

<html>: The root of the HTML document, wrapping everything inside.

<head>: Contains metadata like the page title, external resources and does not appear in the browser.

<title>: Sets the title of the webpage (appears in the browser tab).

<meta charset="UTF-8">: Specifies the character encoding used in the document.

<body>: Contains the visible content (text, images, headings, etc.).

HTML TAGS

HTML tags are mainly categorized into 4 major types. They are:

- Text Formatted Tags
- Form Tags
- Table Tags
- Media & Specific Tags

TEXT FORMATTED TAGS

1. **Headings** - Used to create headings for sections.HTML supports 6 levels of headings, from <h1> (largest) to <h6> (smallest).

Example:

- <h1>This is a level 1 heading</h1>
- <h2>This is a level 2 heading</h2>
- <h3>This is a level 3 heading</h3>
- <h4>This is a level 3 heading</h4>
- <h5>This is a level 3 heading</h5>

- <h6>This is a level 3 heading</h6>

2. **Paragraph** - Used to create paragraphs of text. Always starts at a new line. Example:

This is a paragraph

3. **Preformatted Text Tag** - Displays text exactly as typed (preserves spaces, newlines). Example:

<pre> This text will be
displayed as preformatted text.
</pre>

4. **Bold** - Makes text bold. Example:

Bold Text

5. **Italic** - Makes text italicized. Example:

Italic Text

6. **Mark** - Highlights content, typically with a yellow background.

Example:

<mark>Yellow Highlighted Text</mark>

7. **Underline** - Underlines the text

Example:

<u>Underlined Text</u>

8. **Delete** - Strikes through the text.

Example:

~~Line-through Text~~

9.**Superscript Tag** - Displays text in superscript (above the baseline), typically used for powers.

Example:

- H²O is water.
- Output – H2O is water.

10.**Subscript** - Displays text in subscript (below the baseline), typically used for chemical formulas or mathematical notations.

Example:

- CO₂ is carbon dioxide.
- Output – CO2 is carbon dioxide.

11. **List (Group of Items):** Used to group items in a list. There are two types:

- Ordered List (<ol>):

- Type of list which has specific order (A, a, 1, I, i).
- Default order is numbered list.

- Unordered List (<ul>):

- Type of list which has no specific order (square, circle, disc, none).
- Default order is bulleted list.

List Item (<li>): Used to define each item in the list.

Examples: ORDERED LIST

First item

Second item

Third item

 UNORDERED LIST

First item

Second item

Third item

Fourth item

12.Void Tags / Self-Closing Tags - Tags that do not have closing counterparts and work without surrounding content.

Examples:

-
: Line break (forces content to the next line).
- <hr>: Horizontal rule (creates a horizontal line).

**13.Break Tag (
)** - Forces the content to break and continue on the next line. Example:

• This is line one

• This is line two.

14.Horizontal Rule Tag (<hr>) - Creates a horizontal line, typically used to separate sections.

Example:

• <hr>

• This is content after the horizontal line.

FORM TAGS

What is a Form?

• A form in HTML is used to collect information from users.

• Forms are commonly used to gather data like contact information, user feedback, login details, etc.

Form Structure

• Root Tag for Forms: The <form> tag is the root of any form in HTML.

• Syntax:

<form>

<!-- form fields go here -->

</form>

Attributes for <form> Tag:

• method: Defines how the data will be sent to the server.

- GET: Data is sent in the URL (not secure).

- POST: Data is sent in the body (more secure, not visible in URL).

Example:

<form method="post" action="submit.php">

</form>

- action: Specifies the URL or location where the form data will be sent after submission.

Label Tag (<label>)

• Purpose: The <label> tag is used to define labels for form fields. When a user clicks on the label, it focuses on the corresponding input field.

• Example:

<label>Username:</label>

- Input Tag (<input>)
- Purpose: The <input> tag is used to create interactive fields where users can enter data.
- By default, the input type is text, but it can be changed using the type attribute.

Input Tag Attributes

- The <input> tag can have several attributes to control its behavior and appearance. Here are the key 5 attributes you should know:
- type: Specifies the type of input field (e.g., text, email, number).
  Example: <input type="text">
- name: Identifies the input field, and its value is sent to the server when the form is submitted.
  Example: <input type="text" name="username">
- id: A unique identifier for the input field, often used for styling or linking with the <label> tag.
  Example: <input type="text" id="email">
  placeholder: Displays a hint or text inside the input field that disappears when the user starts typing.
  Example: <input type="text" placeholder="Enter your name">
- value: Sets the default value for the input field, or the value sent to the server when the form is submitted.
  Example: <input type="text" value="John">

Input Types

The type attribute of the <input> tag defines the kind of input field to create. Here are some of the most common input types:

• text: A single-line text input field for regular text input.

Ex: <input type="text" name="username" id="username" placeholder="Enter your username" value="John">

• email: Ensures the entered text is a valid email address (useful for email forms).

Ex: <input type="email" name="email" id="email" placeholder="Enter your email" value="example@email.com">

• number: Allows the user to input only numeric values.

Ex: <input type="number" name="age" id="age" placeholder="Enter your age" value="25">

• time: Lets the user select a time.

Ex: <input type="time" name="appointment_time" id="appointment_time" placeholder="Select a time" value="12:00">

• date: Allows the user to pick a date from a date picker.

Ex: <input type="date" name="birthdate" id="birthdate" placeholder="Select your birthdate" value="1995-06-15">

• radio: Lets the user choose one option from a group. Useful for multiple-choice questions.

Ex: <input type="radio" name="gender" id="male" value="male"> Male

<input type="radio" name="gender" id="female" value="female"> Female

• checkbox: Lets the user select multiple options.

Ex: <input type="checkbox" name="interest1" id="sports" value="sports"> Sports

<input type="checkbox" name="interest2" id="music" value="music"> Music

• textarea: Used for multi-line input fields (e.g., for comments, feedback).

Ex: <textarea name="feedback" id="feedback" rows="4" cols="50" placeholder="Enter your feedback">Great website!</textarea>

• select (Dropdown): Used to create a dropdown menu with several options.

Ex: <select name="country" id="country">

<option value="usa">USA</option>

<option value="india">India</option>

<option value="canada">Canada</option>

</select>

• submit: A button that submits the form.

Ex: <input type="submit" value="Submit">

• reset: Resets all form fields to their default values.

Ex: <input type="reset" value="Reset">

• file: Allows the user to upload files (images, documents, etc.).

Ex: <input type="file" name="file_upload" id="file_upload">

Fieldset Tag (<fieldset>)

• Purpose: Used to group related elements in a form and create a border around them.

Example:

<fieldset>

<legend>Personal Information</legend>

<label>Name:</label>

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

</fieldset>

Legend Tag (<legend>)

• Purpose: Defines a caption or heading for the content inside a <fieldset> tag.

Example: <fieldset>

<legend>Contact Details</legend>

<label>Email:</label>

<input type="email" id="email" name="email">

</fieldset>

TABLE TAGS

What is a Table?

- A table is a way to display data in a structured format using rows and columns.
- Tables are used to organize data in a grid-like structure, making it easier to read and compare values.
- Table Structure
- The basic structure of a table is wrapped inside the <table> tag.
- A table consists of rows (<tr>) and each row contains cells (either headings <th> or data <td>).

<table>

<!-- Table rows and data go here -->

</table>

<tr> (Table Row)

• The <tr> tag is used to define a row in a table.

• Each <tr> will contain one or more <th> (table header) or <td> (table data) elements.

<tr>

<td>John</td>

<td>25</td>

</tr>

<th> (Table Heading)

• The <th> tag defines a table header cell. It is used to represent headings for columns or rows.

• Text inside <th> elements is usually bold and centered by default.

<tr>

<th>Name</th>

<th>Age</th>

</tr>

<td> (Table Data)

• The <td> tag is used to define a table data cell. It holds the actual data inside the table.

• A table row (<tr>) can have multiple <td> elements.

```
<tr>
    <td>John</td>
    <td>25</td>
</tr>
```

Border Attribute

• The border attribute adds a border around the table and its cells. You can specify the size of the border (in pixels).

<table border="1"> ….. </table>

Result- The table will have a 1-pixel border around all its cells.

Table Attributes

Rowspan Attribute

• The rowspan attribute allows you to merge two or more rows into one. It is used with <td> or <th> to extend a cell across multiple rows.

• Example: Merging a cell across two rows.

```
<table border="1">
<tr>
</tr>
<tr>
<th rowspan="2">Name</th>
<th>Age</th>
<td>John</td>
</tr>
</table>
```

In this example, the "Name" cell will span two rows.

Colspan Attribute

• The colspan attribute allows you to merge two or more columns into one. It is used with <td> or <th> to extend a cell across multiple columns.

• Example: Merging a cell across two columns

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

In this example, the "Personal Information" cell will span two columns.

MEDIA & SPECIFIC TAGS

<img> Tag (Image Tag)

- The <img> tag is a void tag (self-closing) used to insert images into a webpage.
- Attributes:

  src: Specifies the source of the image. The source can be:

- By Name: Using just the image file name if the image is in the same directory as the HTML file.
- By Path: Using the file path if the image is in a different folder or location.

  alt: Provides alternate text when the image cannot be displayed (e.g., if the image fails to load or for screen readers).

title: Provides extra information when you hover over the image.

height: Defines the height of the image.

width: Defines the width of the image.

Unit: The values for height and width can be in pixels (e.g., 1px = 0.01cm) or other units like %.

Ex: <img src="image.jpg" alt="A beautiful sunset" title="Sunset" height="200" width="300">

<a> Tag (Anchor Tag)

- The <a> tag is used to create hyperlinks that navigate from one page to another or to an external resource.
- Attributes:
  - href: Hypertext reference, the destination URL to which the link points.
  - title: Provides additional information about the link when hovered over.
  - target: Defines where to open the linked document.
- _blank: Opens the link in a new tab.

Ex: Click here In above example, “Click here” can be called as Anchor Point.

<iframe> Tag (Inline Frame)

• The <iframe> tag is used to embed or display one webpage inside another webpage.

• Attributes:

src: The source URL of the page to embed.

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

Steps to Embed Google Maps Using <iframe>:

1. Go to Google Maps:
   a. Open Google Maps.
   b. Search for the location or address you want to embed.
2. Get the Embed Code:
   a. Once you've found the location, click on the Share button (usually represented by a share icon or button).
   b. From the share options, choose Embed a map.

c. You will be given an HTML iframe code.

3. Copy the Embed Code: The embed code will look like this:

<iframe src="https://www.google.com/maps/embed?pb=...your-location-code..." width="600" height="450" style="border:0;" allowfullscreen="" loading="lazy"></iframe>

4. Embed the Map in Your Website:

Copy the entire iframe code provided by Google and paste it into your webpage's HTML where you want the map to appear.

<audio> Tag (Audio Tag)

- The <audio> tag is used to embed audio files into a webpage.
- Attributes:

controls: Provides controls like play/pause, volume, and time display.

loop: Makes the audio loop continuously.

Ex: <audio src="audio.mp3" controls loop>

Your browser does not support the audio element.

</audio>

<video> Tag (Video Tag)

The <video> tag is used to embed video files into a webpage.

Attributes:

- controls: Provides play/pause, volume, and time display controls.
- loop: Makes the video loop continuously.
- height: Defines the height of the video.
- width: Defines the width of the video.
- muted: Mutes the video sound by default.
- autoplay: Automatically starts the video as soon as it's ready.

Ex: <video src="video.mp4" controls loop height="300" width="500" muted autoplay> Your browser does not support the video tag.

</video>

<meta> Tag

- The <meta> tag is used to provide metadata about the HTML document.
- It is placed within the <head> section of the webpage.
- The <meta> tag is a void tag (self-closing) and does not have visible content on the page.

Common uses:

charset: Defines the character encoding.

name and content: Provides additional metadata like author, description, etc.

Ex: <meta charset="UTF-8">

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

<meta name="description" content="This is a webpage about HTML tags.">

** Tag (Division Tag)**

• The tag is a block-level element used to group other elements together. It serves as a container for content, allowing for styling or structuring.

Ex:

<h2>Welcome to My Website</h2>

This is some introductory text.

** Tag (Span Tag)**

• The tag is an inline element used to group smaller portions of text or content within a line. It serves as a mini container for styling or structuring smaller sections of the page.

Ex: This is a red word in the sentence.

CASCADING STYLE SHEET [CSS]

- CSS stands for Cascading Style Sheets.
- It is used to style webpages, websites, and web applications.
- With CSS, we can change the appearance of elements like text, background, images, borders, layout, and animations.
- CSS was first introduced by Håkon Wium Lie in 1996 as CSS1.
- Over the years, CSS has evolved, with the latest version being CSS3 (released in 2005).

Why Use CSS:

CSS helps in:

- Enhancing the look and feel of a webpage.
- Separating content from design for better organization.
- Reducing code repetition, making development easier.
- Improving website performance by loading styles separately.

CSS Syntax:

CSS works on properties and values.

property-name: value;

Ex: color: red; /* changes text to red */

Ways to add CSS to HTML:

There are three ways to connect CSS to HTML.

- Inline CSS
- Internal CSS
- External CSS

Inline CSS:

- Written directly inside an HTML tag using the style attribute.
- Has the highest priority, overriding other styles.
- Used for quick styling but not recommended for large projects.
- Ex: This is a blue paragraph.

Internal CSS:

- Written inside the <head> section of an HTML document using the <style> tag.
- Applies to the whole page but only to that specific HTML file.

Ex:

```
<head>
    <style>
        h1 { color: green; font-size: 24px; }
    </style>
</head>
<body>
    <h1>This is a green heading</h1>
</body>
```

External CSS:

- Written in a separate file with a .css extension.
- Linked to an HTML file using the <link> tag.
- Best practice for large websites because it keeps styles organized.
- Easy to manage and apply styles across multiple pages.

```
/* styles.css */
h2 { color: orange; font-size: 20px; }
/* index.html */
<head>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h2>This is an orange heading</h2>
</body>
```

CSS Selectors:

CSS selectors are used to target and style specific elements in an HTML document. We can select elements in five different ways:

- Universal Selector (*)
- Element Selector (Tag Name)
- Class Selector (.classname)

- ID Selector (#idname)
- Group Selector (,)

1. Universal Selector (*):

- The universal selector (*) applies styles to all elements in the HTML document.
- It is useful for setting default styles for the entire page.

Ex: * {

margin: 0;

padding: 0;

}

2. Element Selector (Tag Name):

- Selects all elements of a specific type.
- Applies the same style to all occurrences of that tag.
- Ex: p { color: blue; font-size: 18px; }

3. **Class Selector (.classname):**

- Used to style multiple elements that share the same class.
- Class names are defined in the class attribute of HTML elements.
- A class can be used multiple times on a page.

```
/* CSS */
.red-text { color: red; Font-size: large; }
/* HTML */
<p class="red-text">This is a red paragraph</p>
```

4. ID Selector (#idname):

- Targets a single, unique element using the id attribute.
- IDs must be unique (one per page).

```
/* CSS */
#main-heading { color: green; text-align: center; }
/* HTML */
<h1 id="main-heading">Welcome to My Website</h1>
```

5. Group Selector (,):

- Used when multiple selectors share the same styles.
- Avoids repeating the same styles for each selector separately.

```
/* CSS */
h1, h2, h3 { color: purple; font-family: Arial, sans-serif; }
/* HTML */
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
```

CSS PROPERTIES:

1. Color:

- The color property applies color to text.
- Colors can be set by name or by hex code.
- For Hex codes, use Google Color Picker to select colors.

```
p {
color: red;
} /* By Name */
h1 {
color: #ff5733;
 } /* By Hex Code */
```

2. Background-color:

The background-color property changes the background of an element.

```
body {
        background-color: lightblue;
 }
h1 {
 background-color: yellow;
}
```

3. ! important (Highest Priority):

The !important rule overrides all other CSS rules, even inline styles.

```
p {
        color: green !important;
}
p {
        color: red;
 }/* This will be ignored because !important is used above */
```

The paragraph will always be green because of !important rule.

4. Text-align:

- The text-align property controls text alignment.
- Values: left, right, center, justify.

```
h1 {
        text-align: center;
 } /* Centers the heading */
p {
        text-align: justify;
}/* Makes text evenly spaced */
```

5. Text-decoration:

- The text-decoration property adds or removes text effects.
- Values: underline, overline, line-through, none.

```
h1 {
        text-decoration: underline;
 }
p {
        text-decoration: line-through;
 }
```

6. Text-transform:

- The text-transform property changes text capitalization.
- Values: uppercase, lowercase, capitalize.

```
h1 {
        text-transform: uppercase;
 } /* All letters become uppercase */
p {
        text-transform: capitalize;
} /* First letter of each word is capitalized */
```

7. Line-height:

The line-height property controls the space between lines of text.

```
p {
        line-height: 1.5;
} /* Increases space between lines */
```

8. Opacity:

- The opacity property controls the transparency of an element.
- Values range from 0 (fully transparent) to 1 (fully visible).

```
img {
opacity: 0.5;
} /* 50% transparent */
```

9. Font-size:

- The font-size property controls the size of the text.
- Measured in pixels (px).

```
p {
        font-size: 20px;
 } /* Text is 20 pixels in size */
```

10. Font-style:

- The font-style property changes the appearance of the text.

- Common values: italic, normal.

```
p {
        font-style: italic;
} /* Text appears in italics */
```

11. Font-family:

- The font-family property sets the typeface of the text.
- Multiple fonts can be listed, and the browser will use the first available one. p { font-family: "Times New Roman", serif; }

```
h1 {
         font-family: Arial, sans-serif;
}
```

12. Link Styling:

- Links (<a> tags) are an important part of web pages. CSS allows us to style links in different states:
- a: link (unvisited Link)
- Styles links that haven't been clicked yet.
- Commonly used to set the default color of links.

Ex:

```
a:link {
 color: red;
 } /* Unvisited links will be red */
```

a:hover (Mouse Over)

- Styles links when the user moves the mouse over them.
- Commonly used to change color or add underlines.

Ex:

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

a:visited (Clicked Links)

- Styles links that have already been visited by the user.
- Useful for differentiating visited links from unvisited ones.

Ex:

```
a:visited {
        color: red;
}
```

a:active (Clicking the Link)

- Styles links when they are being clicked.

- The effect lasts only while the mouse button is pressed down.

Ex:

```
a:active {
        color: yellow;
 }
```

13. Lists:

- Lists are used to display items in a structured way. CSS allows us to customize lists using properties like:
- list-style-type (Bullet or Number Style)
    a. The list-style-type property changes the bullet or number style of a list.
    b. Works for both unordered (<ul>) and ordered (<ol>) lists.

Ex:

```
ul {
        list-style-type: square;
} /* Squares instead of default bullets */
ol {
 list-style-type: upper-roman;
} /* Numbers become Roman numerals */
```

list-style-position (Number Position)

a. The list-style-position property controls where the bullet or number appears.
b. inside → Bullets/numbers inside the text block.
c. outside → Bullets/numbers outside the text block (default).

Ex:

```
ul {
 list-style-position: inside;
} ol {
 list-style-position: outside;
 }
```

14. Display Properties:

The display property in CSS controls how an element appears on a webpage. It determines if the element takes up the full width, appears inline, or combines both behaviors.

display: block; (Block-Level Elements)

The element starts on a new line. Takes up the entire width available. Accepts height and width modifications.

Examples:

, <p>, <h1> to <h6>,

Ex:

```
<div class="block">Block Element 1 </div>
```

Block Element 2

```
.block {
        display: block;
        background-color; lightblue;
        width: 200px;
        height: 50px;
}
```

Now, Both elements starts on new line occupying full width.

2. display: inline; (Inline Elements)

Elements appear side by side (inline). Cannot accept height and width modifications.

Examples:

, <a>, , .

Ex:

Inline Element 1

Inline Element 2

```
.inline {
        display: inline;
        background-color: lightgreen;
        width: 200px;
        height: 50px;
}
```

Now, both elements appear side by side and doesn't apply height and width.

3. display: inline-block; (Mix of Both)

Behaves like an inline element (side by side). Accepts width and height like a block element.

Examples:

<button>, <img>, <input> (by default).

Ex:

Inline-Block 1

Inline-Block 2

```
.inline-block {
        display: inline-block;
        background-color: lightcoral;
        width: 200px;
        height: 50px;
 }
```

Now both elements appear on side by side with given height and width.

15. Box Model

The box model is a foundation of layout in CSS. It consists:

1. Border

The border surrounds the content of an element.

- border-width: Sets the thickness (e.g., 5px, 10px, 2px).
- border-style: Defines the style (e.g., solid, double, dotted, dashed).
- border-color: Determines the color (e.g., red, green, blue, black).
- Short-hand property: border: 4px solid black;

Individual sides:

border-top-width: 5px;

border-top-style: dashed;

border-top-color: red;

Like this, we change it to any side.

Mixed Property / Short-hand Property:

border: 2px solid #efefef;

2. Margin

Margin creates space around the border. It pushes other elements away from the box.

Margin applies a uniform space around all four sides based on the TRBL rule.

TRBL = TOP RIGHT BOTTOM LEFT

Ex: margin: 5px;

Individual Sides:

margin-top: 10px;

margin-right: 15px;

margin-bottom: 10px;

margin-left: 15px;

Mixed Property / Shortend Property:

margin: 10px 5px 4px 3px;

margin: 10 px 5px;

3. Padding

Padding creates space between the content and the border.

Padding applies uniform space inside the box on all sides.

Ex: padding: 5px;

Individual Sides:

padding-top: 10px;

padding-right: 15px;

padding-bottom: 10px;

padding-left: 15px;

Mixed or Shortend Property:

Padding: 10px 8px 5px 9px;

Padding: 5px 4px;

4. Border-Radius

Border-radius is used to round the corners of the border.

Uniform Rounding: border-radius: 20px;

Individual corners:

border-top-right-radius: 10px;

border-bottom-left-radius: 10px;

5. Float

The float property allows an element to be taken out of the normal document flow and aligned to the left or right.

Floating an Element

Ex:

.float-left { float: left; }

.float-right { float: right; }

Clearing Floats:

To break the float's flow and prevent subsequent elements from wrapping around the floated element, use the clear property:

Ex: .clear { clear: both; }

16. Border-Collapse

The border-collapse property is used to control how table borders are displayed. It determines whether the borders of adjacent cells are merged into a single line (collapsed) or kept separate(default).

Ex:

```
 table {
border-collapse: separate;
} /* Keeps the borders separate (default) */
table {
border-collapse: collapse;
} /* Merges the borders */
```

17. nth-child(n)

The nth-child(n) selector is used to target elements based on their position within a parent.

Target even and odd rows:

```
        tr:nth-child(odd) {
        background-color: lightblue;
 }
        tr:nth-child(even) {
 background-color: lightgray;
 }
```

18. Icons in CSS using CDN

Icons are small graphical symbols used to enhance the user interface of websites. Instead of using images, we can use icon libraries such as Font Awesome, Boxicons, Bootstrap Icons, etc., by linking them via a CDN (Content Delivery Network).

What is a CDN?

A CDN (Content Delivery Network) is a network of servers that delivers content faster by distributing it worldwide. Instead of downloading icon files, we link to a CDN, which loads icons directly from the internet. Font Awesome 4 (Using CDN)

Font Awesome is a popular icon library that provides scalable vector icons that can be customized with CSS.

1. Include the CDN link in the <head> section of your HTML file:

<link rel="stylesheet"href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" >

2.Use icons with the tag and class names:

Ex: <!-- Home icon -->

 <!-- User icon -->

19. background-image (Adding a Background Image)

The background-image property is used to set an image as the background of an element.

```
Ex: body {
        background-image: url("background.jpg");
 }
```

20. background-repeat (Controlling Image Repetition)

This property controls whether the background image repeats or not.

Ex: background-repeat: repeat | no-repeat ;

21. background-size (Resizing the Background Image)

The background-size property defines the height and width of the background image.

Ex: background-size: width height;

22. Overflow – Handling Overflowing Content

The overflow property controls what happens when content overflows the defined width/height of an element.

Syntax: overflow: visible | hidden | scroll | auto;

visible → (Default) Content spills outside the box.

hidden → Cuts off the overflowing content (no scrollbar). scroll → Adds both horizontal & vertical scrollbars.

auto → Adds scrollbars only when needed.

23. Position – Controlling Element Positioning

The position property determines how an element is placed in the document.

Syntax:

position: static | absolute | sticky;

static → (Default) Normal document flow.

absolute → Moves relative to the nearest positioned ancestor. sticky → Sticks in place until a scroll threshold is met.

24. Z-index – Controlling Layer Order

The z-index property controls the stacking order of elements (which one appears on top).

Syntax: Z-index: value;

Higher the value, appears on the top. Works on absolute position.

BOOSTRAP

What is Bootstrap?

- Bootstrap is a CSS & JavaScript framework for building responsive and mobile-first websites quickly.
- It comes with predefined styles and components like buttons, cards, forms, and grids.
- No need to write CSS from scratch—Bootstrap provides ready-made classes.
- Ex: <button class="btn btn-primary">Click Me</button> // Makes the button blue

What is a Framework?

A framework is a collection of prewritten code that helps developers build applications faster. Instead of writing everything from scratch, we use frameworks to save time and effort.

Benefits of Bootstrap Framework:

- Pre-built styles and components
- Faster development
- Responsive design (fits all screens)
- Open-source and free

Why Use Bootstrap?

- Time-Saving – No need to write complex CSS.
- Fast Development – Ready-made classes for layout, colors, and responsiveness.
- Built-in Components – Buttons, navbars, modals, carousels, etc.
- Responsive Design – Works well on mobile, tablet, and desktop.
- Customizable – You can modify Bootstrap style**s as per your needs.**

Who Created Bootstrap?

- Created by Mark Otto & Jacob Thornton (Twitter developers) in August 2011.
- Initially developed for Twitter's internal tools, later released for public use.
- The latest version is Bootstrap 5.3 (removes jQuery dependency and improves performance).

How to Link Bootstrap to HTML?

There are two ways to include Bootstrap in a project:

1. Using CDN (Best Practice)
2. CDN (Content Delivery Network) loads Bootstrap directly from the internet. Just add inside the <head> of your HTML file.

- Download & Link Bootstrap Locally
- Download Bootstrap from getbootstrap.com

- Extract and save the Bootstrap folder in your project directory.
- Link Bootstrap CSS in your HTML file. Using CDN is best practice.
- Faster loading
- Always up-to-date
- No need to download Bootstrap files

Bootstrap Color Classes

Bootstrap provides predefined color classes for text, backgrounds, and buttons.

Ex: Dark Background with White Text // black background, white text

Light Blue Background with Dark Text // skyblue background, black text

Bootstrap Containers

A container is used to structure and align elements properly.

1. Fixed Container -- Centered content with a fixed width.

```
<div class="container">
        <h2>Fixed Container</h2>
</div>
```

2. Fluid Contianer – Takes full width of the screen.

```
<div class="container-fluid">
        <h2>Full Width Container</h2>
</div>
```

Bootstrap Layout & Grid System

The Bootstrap Grid System is used to create responsive layouts by dividing the webpage into rows and columns.

Why Use the Grid System?

- Helps in structuring web pages properly.
- Automatically adjusts the layout based on screen size.
- No need to manually set widths using CSS.
- Provides 12 equal parts (columns) for flexible design.

How Does the Grid System Work?

- Bootstrap divides the webpage into 12 columns.
- We can use different column classes based on screen size.
- A row ( ) is required to use the grid system.
- Columns ( ) go inside a row.

Screen Sizes in Bootstrap Grid

Bootstrap categorizes screens into 6 sizes:

Screen Type	Class Prefix	Width
Extra Small (XS)	col-	Less than 576px
Small (SM)	col-sm-	576px and above

Medium (MD)	col-md-	768px and above
Large (LG)	col-lg-	992px and above
Extra Large (XL)	col-xl-	1200px and above

- Each screen is divided into 12 equal columns.
- You can combine columns (e.g., col-md-6 takes 6 out of 12 columns).
- Ex: 3 equal columns

```
<div class="row">
    <div class="col-md-4 bg-primary text-white">Column 1</div>
    <div class="col-md-4 bg-secondary text-white">Column 2</div>
    <div class="col-md-4 bg-success text-white">Column 3</div>
</div>
```

JAVASCRIPT

What is JavaScript?

- JavaScript (JS) is a programming language used to make web pages interactive.
- It allows users to interact with web pages using buttons, pop-ups, forms, animations, and more.
- Without JavaScript: Web pages are static (only text, images, and layouts).
- With JavaScript: Web pages become interactive (can respond to user actions).

History of JavaScript

- JavaScript was originally called LiveScript but was later renamed to JavaScript.
- Created by Brendan Eich in 1995 while working at Netscape.
- Initially developed for web browsers but is now widely used in backend (Node.js), mobile apps, and games.

Why Use JavaScript?

JavaScript is used for multiple purposes:

- Interactiveness – Adding actions to buttons, popups, alerts, and dropdowns.
- Real-time Updates – Chat applications, notifications, stock price updates, etc.
- Validations – Checking user inputs in forms before submitting.
- Multimedia Content – Controlling audio, video, and animations.

Where to Write JavaScript?

To write JavaScript code, you can use:

- VS Code (Visual Studio Code) – A popular code editor for JS.
- Any text editor like Notepad++, Sublime Text, Atom, etc.
- Browser Developer Console (Right-click on a page → Inspect → Console).

Ways to Connect JavaScript to HTML & CSS

JavaScript can be added to an HTML page in three ways:

1. Inline JavaScript

- Written directly inside an HTML tag using the onclick, onmouseover, etc. attributes.

2. Internal JavaScript

- Written inside <script> </script> tags in the same HTML file.
- Best placed before the closing </body> tag for better performance.
- Not recommended in the <head> part of HTML file.

3. External JavaScript

- JavaScript is written in a separate file (.js extension).
- The HTML file links to the JavaScript file using <script src="filename.js"></script> before the closing </body> tag.

Console.log() Method

The console.log() method is used to display messages, debug code, and test outputs in the browser's developer console. Useful for checking program flow during development.

Ex: console.log("Hello, JavaScript!");

To see output: Open the Browser Developer Console (Right-click → Inspect → Console) to see the output.

What is the DOM?

- DOM = Document Object Model.
- DOM = A structured model of a webpage.
- It allows JavaScript (JS) to change the structure, content, and style of a webpage.
- Acts as a bridge between HTML (structure), CSS (styling), and JavaScript (behavior).
- Without the DOM, JavaScript cannot modify HTML or CSS dynamically.
- The DOM represents an HTML document as a hierarchy of nodes in a tree-like structure.
- Think of it like a construction site:

Role	Description
HTML	The structure (building) of the webpage.
CSS	The styling (paint, decorations) applied to the building**.**
JavaScript	The boss who decides what should happen.

DOM The worker who actually makes things happen.

Why is the DOM Important?

- Allows JavaScript to interact with web pages dynamically.
- Enables real-time updates to web pages (like adding/removing elements).
- Helps in form validation, animations, and event handling. How JavaScript Interacts with the DOM?

DOM Methods:

1. document.write();

- Directly writes content to the webpage.

- Mostly used for testing but not recommended in real projects.
- Ex: <script> document.write("Hello, this is JavaScript!"); </script>
- Output: Displays "Hello, this is JavaScript!" on the page.

2. getElementById()

- Selects an element using its ID.
- Used to modify text, styles, or apply events.
- Ex: Original Text
- document.getElementById("text").innerText = "Text Changed!"; It selects the paragraph tag based on the id.

3. InnerText

- Changes the text content of an HTML element.
- Does not interpret HTML tags inside it.
- Ex: Original Text

document.getElementById("text").innerText = "Text Changed!";

4. InnerHTML

- Similar to innerText but can insert HTML elements as well.
- Ex: Original Content

document.getElementById("content").innerHTML = "New Bold Content!";

5. getElementsByClassName()

- Selects multiple elements using their class name. Access elements using index ([0], [1], etc.).
- Ex: First

document.getElementsByClassName("myClass")[0].innerText = "Updated First!";

6. querySelector()

- Selects the first matching element based on CSS selectors.
- Ex: First
- Second

document.querySelector("#myId").innerText = "First Updated!";

document.querySelector(".myClass").innerText = "First Updated!";

7. style Property

- Used to apply CSS styles directly to elements.
- Ex: Change my color

document.getElementById("styledText").style.color = "red";

8. style.cssText

- Allows multiple CSS properties at once.
- Ex: Styled Text

document.getElementById("styledText").style.cssText = "color: blue; font-size: 20px; background: yellow;";

What are Variables?

- Variables are containers that store data or values.
- They help in saving information and reusing it in the program.
- Syntax: variableType variableName = value;

Ways to Declare Variables in JavaScript

1. var (Old Version)

- Introduced in older JavaScript versions. We can reassign the variable name. We can update the variable value.
- Ex: var x = 10; x = 20; // Value updated document.write(x); // Output: 20

2. let (Modern Version)

- Introduced in 2015, recommended for most cases. We cannot reassign the variable name. We can update the variable value.
- Ex: let y = 30; y = 40; // Allowed (value updated) document.write(y); // Output: 40

let y = 50; // ❌ Error! Cannot redeclare a `let` variable.

3. const (Constant)

- Used for fixed values that do not change. We cannot redeclare the variable.
- We cannot update the value.
- Ex: const z = 100;

z = 200; // ❌ Error! Cannot update a `const` variable. Document.write(z);

Rules for Variable Names

✅ Allowed:

- Must start with a letter (a-z, A-Z), underscore (_), or dollar sign ($).
- Can contain letters, numbers (0-9), _, and $.
- No spaces allowed.

🚫 **Not Allowed:**

- Cannot start with a number.
- No special characters except _ and $.
- Case-sensitive (myVar and myvar are different).

What is a Data Type?

A data type defines what kind of value a variable can store. JavaScript has 7 main data types.

1. String ("text")

- Stores textual data. Enclosed in single (') or double (") quotes.
- Ex: let name = "John"; document.write(name); // Output: John

2. Number (123, 4.56)

- Stores integer and decimal values. No quotes required.
- Ex: let price = 99.99; document.write(price); // Output: 99.99

3. Boolean (true / false)

- Stores two values: true or false. Used in conditions and decision-making.
- Ex: let isStudent = true; document.write(isStudent); // Output: true

4. Null (null)

- Represents an empty value or no value.
- Ex: let car = null; document.write(car); // Output: null

5. Undefined (undefined)

- A variable declared but not assigned a value.
- Ex: let city; document.write(city); // Output: undefined

6. Object ({ key: value })

- Stores multiple related data in key-value pairs. Enclosed in curly braces {}.
- Access using dot (.) or bracket ([]) notation. We can add or update values.
- Ex: let student = { name: "Alice", age: 22, isGraduated: false };

// Accessing values

Document.write(student.name); // Output: Alice Document.write (student["age"]); // Output: 22

// Updating values student.age = 23;

Document.write (student.age); // Output: 23

7. Array ([value1, value2])

- Stores multiple values in square brackets []. Values can be of different types.
- Access using index numbers (starting from 0). We can add (push) or update values.
- Ex: let colors = ["red", "blue", "green"];

Document.write (colors[1]); // Output: blue

// Adding a new value colors.push("yellow");

Document.write (colors); // Output: ["red", "blue", "green", "yellow"]

// Updating a value

colors[0] = "purple";

Document.write (colors); // Output: ["purple", "blue", "green", "yellow"]

What are Operators?

Operators are symbols that tell JavaScript to perform a specific task.

1. Arithmetic Operators (For Calculations)

- Used to perform mathematical operations.
- Operators Like + , - , /, % , * , **

2. Comparison Operators (For Checking Conditions)

- Used to compare values. Returns Boolean values (true or false).
- Operators like ==, ===, !=, > , <, >=, <=

3. Logical Operators (For Multiple Conditions)

- Used to combine multiple comparisons. Returns true or false.

- Operators like &&, ||, !

4. Assignment Operators (For Assigning Values)

- Used to assign values to variables.
- Operators like =, +=, -=, *=, /=, %=, **=

Conditional Statements (Decision-Making)

Conditional statements allow JavaScript to execute different actions based on conditions.

if Statement - Executes a block of code only if the condition is true.

Syntax:

```
if (condition) {
    // Code to execute if condition is true
 }
```

if...else Statement - Executes one block if the condition is true, otherwise executes the else block.

Syntax:

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

if...else if...else Statement - Used for checking multiple conditions.

Syntax:

```
if (condition1) {
    // Code if condition1 is true
} else if (condition2) {
    // Code if condition2 is true
} else {
    // Code if none of the conditions are true
}
```

switch Statement - A better alternative to multiple if...else conditions.

Syntax:

```
switch (expression) {
case value1:
    // Code to execute if expression == value1 break;
case value2:
    // Code to execute if expression == value2 break;
default:
    // Code if no case matches
}
```

Control Statements (Loops - Repeating Code)

Loops allow JavaScript to execute a block of code multiple times.

for Loop - Used when we know how many times to repeat.

Syntax:

```
for (initialization; condition; increment/decrement) {
        // Code to execute in each iteration
}
```

while Loop - Used when we don't know how many times to repeat.

Syntax:

```
while (condition) {
        // Code to execute while the condition is true
}
```

do...while Loop - Executes at least once, even if the condition is false.

Syntax:

```
do {
        // Code to execute at least once
} while (condition);
```

Jump Statements (Controlling Loop Execution)

-- Jump statements help break or skip loop iterations.

break Statement - Stops the loop completely when a certain condition is met.

Syntax:

```
for (initialization; condition; increment/decrement)
 {
        if (exitCondition) {
        break; // Stops the loop
}
}
```

continue Statement - Skips the current iteration and moves to the next one.

Syntax:

```
for (initialization; condition; increment/decrement) {
         if (skipCondition) {
                continue; // Skips current iteration
        }
}
```

Functions

Functions are reusable blocks of code that perform a specific task.

Instead of writing the same code multiple times, we can define a function once and call it whenever needed.

Why Use Functions?

- Reusability – Write once, use multiple times.
- Reduces Repetition – Less duplicate code, easier maintenance.
- Saves Time – No need to rewrite logic every time.
- Improves Readability – Breaks complex code into smaller, manageable pieces.

Function Definition & Calling

A function must be defined before it can be used. To execute the function, we must call it.

Syntax:

```
function functionName() {
        // Code inside function
}
functionName(); // Calling the function
```

Function with Parameters

Functions can take parameters (variables passed into the function) to work with dynamic values.

Syntax:

```
function functionName(parameter1, parameter2) {
        // Use parameters inside function
}
functionName(value1, value2); // Call function with arguments
```

Function with Return Value

A function can return a value instead of just displaying it.

Syntax:

```
function functionName() {
        return value; // Function returns a value
}
let result = functionName(); // Store the returned value
document.write(result); // print the result
```

Events

Events are user actions that trigger JavaScript functions.

They allow us to make web pages interactive by responding to user actions like clicking, hovering, typing, scrolling, etc.

Why Use Events?

- Make web pages dynamic and interactive
- Perform real-time actions based on user behavior
- Control form validations, animations, popups, etc.

1.Click Event (onclick)

Runs when an element is clicked (e.g., a button).

Ex: <button id="myBtn">Click Me</button>

<script> document.getElementById("myBtn").onclick = function() { alert("Button clicked!"); }; </script>

2.Mouse Hover Event (onmouseover, onmouseout)

Runs when the mouse enters or leaves an element.

Ex: Hover over me!

```
<script>
    let text = document.getElementById("hoverText");
    // When mouse enters
    text.onmouseover = function() { text.style.color = "red"; };
    // When mouse leaves
    text.onmouseout = function() { text.style.color = "black"; };
</script>
```

OOP – Object Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that helps organize code using objects, classes, methods, and properties to make it structured, reusable, and scalable.

Why Use OOP?

- Cleaner & structured code
- Reduces repetition (DRY - Don't Repeat Yourself)
- Makes debugging & maintenance easier
- Encapsulates related data & methods inside objects

Better for real-world applications Ex:

```
class Student {
    constructor(name, age) {
    this.name = name; this.age = age;
    }
    greet() {
        document.write('Hello, my name is' + this.name + ' and I am ' + this.age + ' years old.`);
    }
}
let student1 = new Student("John", 20);
student1.greet(); // Output: Hello, my name is John and I am 20 years old.
```

Student – blueprint

Constructor – initializes the object's properties

this – refers to the current object

greet() – method

new Student("John", 20); -- creates a new object.

JavaScript Date & Time

JavaScript provides the Date() object to work with dates and times.

Creating a Date Object

let currentDate = new Date(); // Creates a new Date object with the current date & time document.write(currentDate);

// Output: Fri Mar 02 2025 15:30:45 GMT+0530 (IST)

Date & Time Components

getFullYear()	Gets the year	2025
getMonth()	Gets the month (0-based)	0-11
getDate()	Gets the day of the month	1-31
getDay()	Gets the weekday (0-based)	0-6
getHours()	Gets the hour (24-hour format)	0-23
getMinutes()	Gets the minutes	0-59
getSeconds()	Gets the seconds	

Form Validation

Form validation is used to check if user inputs are correct and complete before submitting data to the server.

Why Form Validation?

- Prevents empty fields (Required fields)
- Ensures correct format (Emails, phone numbers)
- Restricts unwanted inputs (Only numbers, text)
- Improves security & user experience
- Required Fields Validation - Ensures that the user fills in all required fields
- onsubmit Event (Trigger Validation Before Submission) - The onsubmit event runs validation when the form is submitted.
- Value Property (Getting Input Values) - We use .value to fetch user input.
- Length Validation (Min/Max Characters) - Ensures correct input length (e.g., username must be at least 5 characters).
- isNaN (Check if Input is a Number) - Ensures only numbers are entered (e.g., phone numbers, age).
- Trim() – remove spaces.

Example:

```
<form onsubmit="return validateForm()">
        <input type="text" id="name" placeholder="Enter name">
        <input type="text" id="age" placeholder="Enter age">
        <input type="password" id="password" placeholder="Enter password">
        <button type="submit">Submit</button>
```

```
</form>
<script>
function  validateForm() {
        let name = document.getElementById("name").value.trim();
        let age = document.getElementById("age").value;
        let password = document.getElementById("password").value;
        if (name === "") {
                alert("Name cannot be empty."); return false;
        }
        if (isNaN(age) || age < 1 || age > 120) {
        alert("Please enter a valid age (1-120)."); return false;
        }
        if (password.length < 6) {
        alert("Password must be at least 6 characters long."); return false;
        }
return true; // If all checks pass, form submits successfully
}
</script>
```