**Day 1**

08/10/2024

* **HTML –** Hyper Text Markup Language

Basic building block of the web.

Hypertext : Link that connect web pages to one another.

Markup : To annotate text ,images ,and other content for display in a web browser. It include special elements like <head>,<title>,<body>,etc.

* **CSS – Cascading Style Sheets**

Use to describe the presentation of a document written in HTML or XML.

* **Bootstrap –** Freeand open sourceframework ,helps to build website and web apps faster and more efficient.

Bootstrap’s CSS : <link> tag in <head>

JavaScript : <script> before closing </body>

**1. HTML**

**Key Topics:**

* **Basic Structure:**
  + <html>: Root element that contain all other html content.
  + <head>: Contains metadata – Information about document (such as its title, character set ,links).
  + <body>: Contain all the visible content of the web page like text ,image ,links.
  + Elements like <div>, <span>, <p>, <a>, etc.
* **Forms and Input Types:**

**Elements :**

* <form> - Forms used to collect user input.

**Attributes** : Action – Specify where to send the form data.

: Method -Defines how to send the data (GET or POST).

Example: <form action="/submit" method="POST">

* + <input> - Capture types of user inputs ,like text ,email ,password.

- Type define input ‘s behaviour.

Example: <input type="text" placeholder="username">

* + <button> - Clickable button.
  + <select> - Create a dropdown menu.
  + <textarea> - Multiline text area for descriptions , comments messages ,etc.
  + Input types: text, email, password, radio, checkbox, etc.
* **Semantic Elements:**

These elements help improve accessibility, SEO, and maintainability of web pages by giving structure and meaning to the content.

* + <header> - Represents the introductory content or a group of navigational links for a webpage or a section of the webpage.
  + <footer> - Represents the footer section of a webpage or an individual section.
  + <article> - Represents an independent piece of content.
  + <section> - Divide a webpage into different sections.
  + <nav> - Represents a section of the page dedicated to navigation links.
  + <main> - Represents the main content of a webpage, excluding headers, footers, and navigation links.
  + <aside> - Placed as a sidebar.
* **Tables:**

Tables in HTML are used to display data in a structured grid format consisting of rows and columns.

* + <table> - Root element.
  + <tr> - Defines a row in the table.
  + <th> - Table Header Cell.
  + <td> - Table Data Cell.<td> elements hold the actual data.
  + colspan, rowspan - Merges cells across columns and rows.
* **Lists:**

**Ordered List and Unordered List**

* + **Ordered (<ol>)** : Items in an ordered list are automatically numbered by the browser.

: Attributes for Ordered Lists: **type**: Defines the type of numbering (e.g., numbers, letters, or Roman numerals)

:Example: <ol>

<li>item1</li>

<li>item2</li>

<li>item3</li>

</ol>

* + **Unordered Lists (<ul>) :** Items in an unordered list are marked with bullet points.

**:** Attributes for Ordered Lists: **type** : Defines the type of bullet (circle, square, or disc).

**:** Example: <ul>

<li>item1</li>

<li>item2</li>

<li>item3</li>

</ul>

* **Links and Navigation:**
  + **Internal and external linking**

linking to pages within the same website and linking to other websites.

* + **anchor tags (<a>)**

To create hyperlinks.

Example : <a href="URL">Link Text</a>

* **Images and Media:**
  + **<img>** - Used to embed images in a webpage.

Example : <img src="image.jpg" alt="Description of image" width="500" height="300">

* + **<video>** - Used to embed vedios in a webpage.

Example: <video src="video.mp4" controls width="640" height="360"></video>

**controls**: Displays video controls (play, pause, volume, etc.)

* + **<audio>** - Used to embed audio in a webpage.

Example : <audio src="audio.mp3" controls></audio>

**controls**: Displays audio controls (play, pause, volume, etc.).

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

**Key Topics:**

* **Selectors:**
  + **Element Selector :** (also known as the **type selector**) targets HTML elements directly by their tag name.

Example : p {

color: blue;

font-size: 16px;

}

* + **Class selector:** The class selector targets elements that have a specific class attribute.

Example : .highlight {

background-color: yellow;

}

* + **ID selector :** The ID selector targets a specific element with a unique id attribute.

Example: #header {

background-color: lightgray;

padding: 10px;

}

* + **Pseudo-classes (:hover, :focus, etc.) and Pseudo-elements (::before, ::after)**

used to style specific states of an element or parts of an element.

**Pseudo-classes :** apply styles based on the element's state or user interaction (e.g., hovering, focusing, or visited links).

Example: a:hover {

color: red;

}

**hover:** Styles an element when the user hovers over it with a mouse.

**Pseudo-Elements :** Style specific parts of an element, such as adding content before or after the element, or styling the first letter/line of a text block.

Example: h1::before {

content: "Chapter: ";

color: gray;

}

**::before:** Inserts content before the element’s actual content.

* **Box Model:**

Describes the rectangular boxes generated for elements in a webpage.

**Components of the Box Model**

* + **Margins** : The outermost space that separates the box from other elements. Margins create space around the element and can also collapse with adjacent margins. The margin property is used to set this space.
  + **Padding** : The space between the content and the border. Padding creates an inner space that adds extra room inside the box without affecting the overall size. It is transparent and can be adjusted using the padding property.
  + **Borders** : The edge surrounding the padding (if any) and the content. It can be styled with width, color, and type (solid, dashed, etc.). The border takes space and can be adjusted using the border property.
  + **Content** : The area where text and images appear. The size of the content area is defined by the width and height properties.
  + **box-sizing: border-box**;

The box-sizing property determines how the total width and height of an element are calculated, affecting how padding and borders are handled.

**Default Value**: The default value is content-box.

**Value: border-box**: When set to border-box, the width and height properties include the content, padding, and border. This makes layout management easier because the specified dimensions represent the total size of the element.

Example: .box {

box-sizing: border-box; /\* Include padding and border in the total width and height \*/

width: 300px;

height: 200px;

padding: 20px; /\* Total size will still be 300px by 200px \*/

border: 5px solid black;

}

* **Positioning:**

CSS positioning allows you to control the placement of elements on a webpage.

**Positioning Types :**

* + **Static positioning :** This is the default positioning method for all elements. Elements are positioned according to the normal flow of the document.

No top, right, bottom, or left properties apply.

* + **Relative positioning :** The element is positioned relative to its normal position in the flow of the document.

Can use top, right, bottom, and left properties to adjust its position.

* + **Absolute positioning :** The element is positioned relative to its nearest positioned ancestor .
  + **z-index :** The z-index property controls the stacking order of positioned elements.

Example: .box1 {

position: absolute;

z-index: 1; /\* Lower stacking order \*/

}

.box2 {

position: absolute;

z-index: 2; /\* Higher stacking order, appears on top of box1 \*/

}

* **Flexbox:**

**Flexbox** is a powerful CSS layout module designed to create responsive and flexible layouts. It allows to arrange items in a container efficiently, providing control over alignment, direction, size, and order, even when the sizes of the items are unknown or dynamic.

* + **Main Axis**: The primary axis along which flex items are laid out. By default, it runs horizontally from left to right. The direction of the main axis can be changed using the flex-direction property.
  + **Cross Axis**: The axis perpendicular to the main axis. If the main axis is horizontal, the cross axis is vertical, and vice versa. This axis determines how flex items align along the vertical (or horizontal) direction.
  + **justify-content :** This property aligns flex items along the main axis**.**

Example: .flex-container {

justify-content: space-between; /\* Space between items \*/

}

* + **align-items:** This property aligns flex items along the cross axis.

Example: .flex-container {

align-items: center; /\* Center items along the cross axis \*/

}

* + **flex-direction:** Defines the direction of the main axis (row, row-reverse, column, column-reverse).
  + **flex-wrap:** This property determines whether flex items should wrap onto multiple lines or stay in a single line.
* **Grid Layout:**

CSS layout system that enables to create complex, responsive web designs using a two-dimensional grid-based approach.

* + **grid-template-rows**: This property defines the number and size of the rows in the grid.

Example: .grid-container {

display: grid;

grid-template-rows: 100px 200px; /\* Two rows: first 100px, second 200px \*/

}

* + **grid-template-columns:** This property defines the number and size of the columns in the grid.

Example: .grid-container {

display: grid;

grid-template-columns: 150px auto; /\* First column 150px, second column takes remaining space \*/

}

* + **grid-area:**  This property is used to specify where a grid item should be placed within the grid layout.

Example: .grid-item {

grid-area: 1 / 1 / 2 / 3; /\* Start at row 1, column 1 and span to row 2, column 3 \*/

}

* + **Gap:** The gap property (previously known as grid-gap) defines the space between rows and columns in a grid layout.

Example: .grid-container {

display: grid;

grid-template-rows: 100px 200px;

grid-template-columns: 150px auto;

gap: 10px 20px; /\* 10px gap between rows and 20px between columns \*/

}

* + **alignment properties for grid**: Control how items align within their grid areas (e.g., justify-items, align-items, justify-content, align-content)
* **Typography:**

Typography refers to the style and appearance of text in web design.

* + **Fonts (font-family, font-size)**

font-family: Specifies the typeface (font) to be used for an element.

font-size :Specifies the size of the font.

Example: .text {

font-family: "Arial", sans-serif; /\* Arial font with fallback to sans-serif \*/

font-size: 16px; /\* Fixed font size \*/

}

* + **Text properties (color, line-height, text-align, text-decoration)**

color: Specifies the color of the text.

line-height: Specifies the height of a line of text, affecting spacing between lines.

text-align: Specifies the horizontal alignment of text within its containing element.

text-decoration : Specifies decorations added to text, such as underline, overline, or line-through.

Example: .text {

font-family: "Arial", sans-serif; /\* Font family \*/

font-size: 16px; /\* Font size \*/

color: #333; /\* Text color \*/

line-height: 1.5; /\* Line height \*/

text-align: left; /\* Left-align text \*/

}

* **Responsive Design:**
  + **Media Queries (@media)**

Media queries are a CSS technique used to apply styles based on the conditions such as viewport size, resolution, orientation, and device characteristics.

Common Media Types : screen ,print

Example:

@media screen and (max-width: 600px) {

body {

background-color: lightblue; /\* Change background color on small screens \*/

}

.text {

font-size: 14px; /\* Adjust font size for small screens \*/

}

}

* + **Breakpoints for responsiveness**

Breakpoints are specific points in the CSS where styles change based on the viewport width. They allow to create different layouts for various screen sizes.

* + **Relative units (rem, em, %, vw, vh)**

Relative units provide a flexible way to size elements and allow them to scale better with different screen sizes, ensuring a more responsive design.

A relative unit based on the font size of the root element (<html>). It allows for **rem (root em):**consistent scaling across the entire document.

Example: html {

font-size: 16px; /\* Base font size \*/

}

body {

font-size: 1rem; /\* 16px \*/

margin: 2rem; /\* 32px \*/

}

**em** :A relative unit based on the font size of the parent element. This means that nested elements can have compounded sizing.

Example: .container {

font-size: 16px; /\* Base font size for container \*/

}

.text {

font-size: 1.5em; /\* 1.5 times the font size of the container (24px) \*/

}

**%(percentage):**A percentage value used for sizes and spacing, relative to the parent element.

Example: .container {

width: 80%; /\* 80% of the parent element’s width \*/

padding: 5%; /\* 5% padding around the container \*/

}

**vw (viewport width):**A unit that represents a percentage of the viewport's width. 1vw is equal to 1% of the viewport width.

**vh (viewport height):**A unit that represents a percentage of the viewport's height. 1vh is equal to 1% of the viewport height.