Week 0

Description	HTML & CSS Basics

HTML

Browser :

A browser is a software application that allows you to access and interact with websites on the internet. Basically it is a tool to explore internet.

The browser's goal is to take these HTML, CSS and JavaScript files, interpret them and display them in a way that users can see and interact with. This process is called **rendering.**

HTML (Hyper Text Markup Language)

- standard language used to create the structure to our web page.
- It's like the skeleton of the web page, laying out all the different parts.

Important tags to keep in mind :

1. `<html>`**:**
    - The root element of an HTML document. Everything on your webpage will be nested inside this tag.
2. `<head>`**:**
    - Contains meta-information about the document, such as the title, character set, linked stylesheets, and scripts. It doesn't display anything on the page itself but is crucial for functionality and SEO.
3. `<title>`**:**
    - Sets the title of the webpage, which appears in the browser tab. It's also used by search engines when indexing your page.

4. `<body>`:
    - Contains all the content that will be displayed on the webpage. This is where you put elements like text, images, links, and other content.
5. `<div>` / `<span>`:
    - `<div>`: A block-level element used to group content. It's useful for organizing sections of your webpage.
    - `<span>`: An inline element used to style or group small portions of text or other inline elements.
6. `<h1>` to `<h6>`:
    - Header tags, used for defining headings on your page. `<h1>` is the highest level, typically used for the main title, while `<h6>` is the lowest, used for sub-subheadings.
7. `<p>`:
    - Defines a paragraph of text.
8. `<img>`:
    - Used to embed images in your webpage. It's a self-closing tag, meaning it doesn't require a closing tag.
9. `<a>`:
    - The anchor tag is used to create hyperlinks, allowing users to navigate to other pages or resources.
10. `<input>`:
    - Defines an input field where users can enter data. It's used in forms and can have various types like text, password, email, etc.
11. `<button>`:
    - Used to create clickable buttons, often in forms or to trigger JavaScript functions.
12. `<b>` / `<i>`:
    - `<b>`: Bold text.

- `<i>`: Italicized text.
- These tags are mainly for styling text and are often replaced by CSS for more control.

Attributes :

- Attributes provide additional information about HTML elements.
- Attributes are the extra things that are provided with the tags. It helps to define what specific tags will have to do.

Common Attributes:

1. `src`:
   - Used with the `<img>` tag to specify the path to the image file.
   - Example: `<img src="photo.png">`
2. `href`:
   - Used with the `<a>` tag to define the URL of the link.
   - Example: `<a href="https://www.google.com">Google</a>`
3. `onclick`:
   - Often used with `<button>` or other interactive elements to specify a JavaScript function that should run when the element is clicked.
   - Example: `<button onclick="myFunction()">Click me</button>`
4. `id`:
   - Used to uniquely identify an element on the page. This is particularly useful for CSS styling or JavaScript targeting.
   - Example: `<input id="username" type="text">`

CSS

1. Why Do We Need CSS

- CSS (**Cascading Style Sheets**) is used to style and format HTML documents.
- It separates **content** (HTML) from **design** (CSS).
- Helps make websites **responsive, beautiful, and consistent**.

Example:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: blue;
      font-size: 20px;
    }
  </style>
</head>
<body>
  <p>Hello, World!</p>
</body>
</html>
```

Output: A blue-colored paragraph text “Hello, World!” in **20px** size.

---

2. Types of CSS

a) Inline CSS

- Applied directly to elements using the `style` attribute.

```
<p style="color:red; font-size:18px;">Inline CSS Example</p>
```

✅ **Best for quick styling**

❌ Avoid using inline CSS in large projects.

---

b) Internal CSS

- Defined inside `<style>` tags within the `<head>` section.

```
<head>
  <style>
    h1 {
      color: green;
      text-align: center;
    }
  </style>
</head>
```

✅ Useful for **single-page websites**.

---

c) External CSS *(Recommended)*

- CSS stored in a separate `.css` file.

```
<link rel="stylesheet" href="style.css">
```

style.css

```
h1 {
  color: purple;
}
```

✅ Best practice for **maintainable and scalable** projects.

---

3. CSS Selectors

Selectors decide **which elements** to style.

a) Element Selector

```
p { color: red; }
```

Applies to **all** `<p>` **tags**.

---

b) Class Selector (`.class`)

```
.highlight {
  background-color: yellow;
}
```

```
<p class="highlight">This text is highlighted</p>
```

---

c) ID Selector (`#id`)

```
#title {
  color: blue;
}
```

```
<h1 id="title">Welcome</h1>
```

⚡ IDs are **unique per page**.

---

d) Attribute Selector

```
input[type] {
  border: 1px solid black;
}
```

Applies styles to **all inputs** with a `type` attribute.

---

e) Attribute Value Selector

```
input[type="text"] {
  background-color: lightyellow;
```

```
}
```

Targets inputs **only of type** `text`.

f) Universal Selector (*)

```
* {
  margin: 0;
  padding: 0;
}
```

Resets **default browser styling**.

4. CSS Colors & Backgrounds

- **Named colors:** `red`, `blue`, `green`
- **Hex codes:** `#ff0000`
- **RGB:** `rgb(255, 0, 0)`
- **RGBA:** `rgba(255, 0, 0, 0.5)` *(adds transparency)*

```
body {
  background-color: #f1f1f1;
  color: #333;
}
```

5. Fonts & Text Styling

Font Size Units

Unit	Description	Example
`px`	Fixed size	`font-size:16px;`
`pt`	Print-based	`font-size:12pt;`
`em`	Relative to **parent**	`font-size:2em;`

Unit	Description	Example
`rem`	Relative to **root**	`font-size:1.5rem;`

Font Weight

```
h1 { font-weight: bold; }
p { font-weight: lighter; }
span { font-weight: 700; } /* 100 - 900 */
```

Font Family

```
p {
  font-family: 'Poppins', sans-serif;
}
```

6. CSS Box Model

Every HTML element is a **box** consisting of:
Content → Padding → Border → Margin

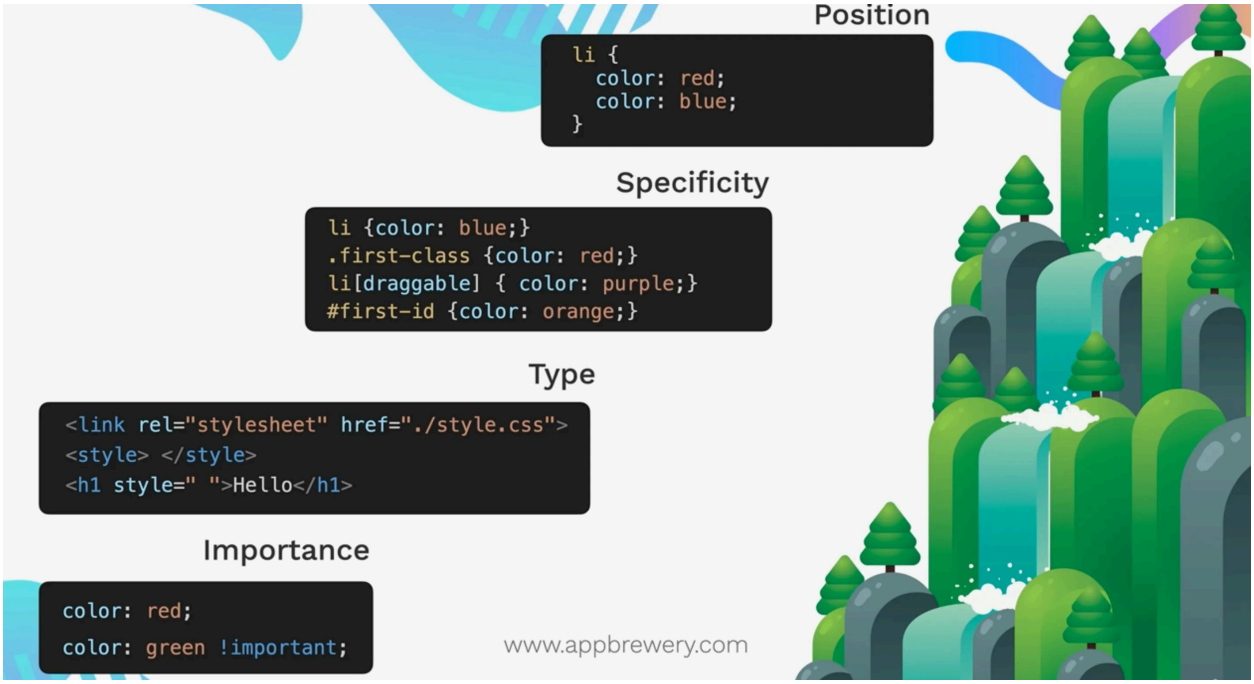
Example:

```
div {
  margin: 20px;
  padding: 15px;
  border: 2px solid black;
}
```

- **Content** → Actual data inside.
- **Padding** → Space **inside** border.
- **Border** → Visible boundary.
- **Margin** → Space **outside** border.

7. CSS Specificity

When multiple rules conflict, **specificity decides which wins**.



Priority Order

1. **Inline CSS** (highest)
2. **ID selectors**
3. **Classes, attributes, pseudo-classes**
4. **Element selectors** (lowest)

Example:

```
p { color: blue; }          /* 4th priority */
.text { color: green; }     /* 3rd priority */
#main { color: red; }       /* 2nd priority */
<p style="color: purple;">  /* 1st priority */
```

8. Combining & Chaining Selectors

Grouping

```
h1, h2, h3 { color: blue; }
```

Child Selector (`>`)

```
div > p { color: red; }
```

Targets only **direct children**.

Descendant Selector

```
div p { color: green; }
```

Targets **all nested** `<p>` **tags**.

Chaining

```
h1.title#main { color: orange; }
```

Applies only when **all conditions match**.

---

9. CSS Positioning

Controls **where elements appear**.

Static (default)

```
div { position: static; }
```

Relative

- Moves element **relative to original position**.

```
div { position: relative; top: 20px; left: 10px; }
```

Absolute

- Positioned **relative to nearest positioned ancestor**.

```
div { position: absolute; top: 50px; left: 30px; }
```

Fixed

- Stays **fixed to viewport**.

```
nav { position: fixed; top: 0; }
```

z-index

Controls **stacking order**:

```
.box1 { position: absolute; z-index: 2; }
.box2 { position: absolute; z-index: 1; }
```

---

10. Display, Float & Clear

Display

Value	Behavior
`inline`	No height/width, flows inline
`block`	Starts new line, supports width/height
`inline-block`	Acts inline but allows width/height
`none`	Hides the element

Float

```
img { float: left; }
```

Moves image to **left** and wraps text around it.

Clear

```
div { clear: left; }
```

Prevents overlapping with floated elements.

---

11. Responsive Design – Media Queries

Used to make layouts **mobile-friendly**.

```
@media screen and (max-width: 768px) {
  body {
    background-color: lightblue;
  }
}
```

Tip: Always design **mobile-first**.

Best Practices

✅ Always use **external CSS** for scalability.

✅ Prefer **classes** over IDs for reusability.

✅ Use **rem** instead of `px` for responsiveness.

✅ Reset default browser styles using:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

CSS Flexbox

Flexbox (**Flexible Box Layout**) is used to create **one-dimensional layouts** (either **row** or **column**).

a) Enable Flexbox

```
.container {
  display: flex;
}
```

b) Flex Direction

Defines the **main axis**:

```
.container {
  display: flex;
  flex-direction: row;        /* Default: left → right */
  flex-direction: row-reverse;/* right → left */
  flex-direction: column;     /* top → bottom */
  flex-direction: column-reverse;
}
```

Diagram:

```
row: → → →
column: ↓
        ↓
        ↓
```

c) Flex Wrap

Controls whether items **wrap** onto multiple lines:

```
.container {
  flex-wrap: nowrap;  /* Default: single line */
  flex-wrap: wrap;    /* Items wrap onto new lines */
```

```
  flex-wrap: wrap-reverse; /* Wraps in reverse order */
}
```

d) Justify-Content *(Horizontal Alignment)*

Aligns items **along the main axis**:

```
.container {
  justify-content: flex-start; /* Default */
  justify-content: flex-end;
  justify-content: center;
  justify-content: space-between; /* Equal gaps between items */
  justify-content: space-around;  /* Equal space around items */
  justify-content: space-evenly;  /* Equal space everywhere */
}
```

e) Align-Items *(Vertical Alignment in a Row)*

Aligns items **along the cross-axis**:

```
.container {
  align-items: stretch;    /* Default */
  align-items: flex-start; /* Top */
  align-items: flex-end;   /* Bottom */
  align-items: center;     /* Middle */
  align-items: baseline;   /* Align text baselines */
}
```

f) Align-Content

Controls spacing **between multiple rows**:

```
.container {
  align-content: stretch;    /* Default */
  align-content: flex-start;
```

```
  align-content: flex-end;
  align-content: center;
  align-content: space-between;
  align-content: space-around;
}
```

g) Flex Sizing

Order of **flex sizing** priority:

```
content width < width < flex-basis < min/max-width
```

h) Flex-Grow, Flex-Shrink, Flex-Basis

Applied to flex items, not the container.

```
.item {
  flex-grow: 1;   /* Item expands if space is available */
  flex-shrink: 1; /* Item shrinks if space is tight */
  flex-basis: 200px; /* Default size before growth/shrink */
}
```

Or shorthand:

```
.item {
  flex: 1 0 150px; /* grow | shrink | basis */
}
```

Example:

```
<div class="container">
  <div class="item">1</div>
  <div class="item">2</div>
```

```
  <div class="item">3</div>
</div>
```

```
.container {
  display: flex;
  justify-content: space-around;
}
.item {
  background: lightblue;
  flex: 1;
  margin: 10px;
}
```

---

13. CSS Grid

CSS Grid is used for **two-dimensional layouts** (rows **and** columns).

---

a) Enable Grid

```
.container {
  display: grid;
}
```

---

b) Define Grid Rows & Columns

```
.container {
  display: grid;
  grid-template-rows: 100px 200px auto;
  grid-template-columns: 1fr 2fr 1fr;
}
```

Units:

- `px` → fixed size

- `%` → relative
- `fr` → **fraction of remaining space**

---

c) Repeat Function

```
.container {
  grid-template-columns: repeat(4, 1fr);
}
```

➡️ Creates **4 equal-width columns**.

---

d) Auto Rows

```
.container {
  grid-auto-rows: 300px;
}
```

Defines the **height** of rows created **automatically**.

---

e) Spanning Rows & Columns

```
.item {
  grid-column: span 2; /* Takes 2 columns */
  grid-row: span 3;    /* Takes 3 rows */
}
```

---

f) Start & End Positions

```
.item {
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 2;
```

```
  grid-row-end: 4;
}
```

Or shorthand:

```
.item {
  grid-column: 1 / 3; /* From column 1 to 3 */
  grid-row: 2 / 4;    /* From row 2 to 4 */
}
```

---

g) Gap Between Items

```
.container {
  grid-gap: 20px;
}
```

Shorthand for:

```
grid-row-gap: 20px;
grid-column-gap: 20px;
```

---

h) Combining Flexbox + Grid

- Use **Grid** for **overall page structure**.
- Use **Flexbox** inside **Grid items** for finer control.

Flexbox vs Grid — When to Use What

Feature	Flexbox	Grid
Layout Type	One-dimensional (row/col)	Two-dimensional (row + col)
Alignment	Best for content alignment	Best for full-page layouts
Responsiveness	Easier with `flex-wrap`	Easier with `auto-fit` & `fr`
Use Case	Navbars, toolbars, buttons	Dashboards, page layouts