# Week 6 - HTML, CSS, Bootstrap

## HTML

**HTML (HyperText Markup Language)** is the **standard markup language** used to create and structure content on the **web**.

It tells the browser **how to display text, images, links, and other media** on a webpage.

- **HyperText** → links web pages together.
- **Markup Language** → uses *tags* to structure and describe content.

### Usage

HTML is used for:

1. **Structuring web pages** (headings, paragraphs, lists, etc.)
2. **Embedding media** (images, videos, audio).
3. **Creating hyperlinks** to connect web pages.
4. **Building forms** for user input.
5. **Working with CSS & JavaScript** to style and add interactivity.

### Versions

| Version | Year | Key Features |
|---|---|---|
| **HTML 1.0** | 1993 | Basic text, links, and images. |
| **HTML 2.0** | 1995 | Forms, basic tables, and internationalization. |

| Version | Year | Key Features |
| --- | --- | --- |
| **HTML 3.2** | 1997 | Added tables, applets, and scripting support. |
| **HTML 4.01** | 1999 | Introduced CSS and scripting standards. |
| **XHTML 1.0** | 2000 | XML-based stricter syntax of HTML. |
| **HTML5** | 2014 (current) | Audio/video tags, semantic elements (`<header>`, `<article>`), Canvas, SVG, APIs. |

## Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.
0">
  <title>Page Title</title>
</head>
<body>
  <h1>This is main heading</h1>
  <p>This is a paragraph of text on the webpage.</p>
</body>
</html>
```

- `<!DOCTYPE html>`: This is the **document type declaration**. It must be the very first line and tells the browser that the page is written in HTML5.
- `<html>`: This is the **root element** that wraps all the content on the page. The `lang="en"` attribute specifies the page's language.
- `<head>`: This element contains **meta-information** about the page. This content is *not* displayed on the page itself but is used by the browser and search engines.
    - `<meta charset="UTF-8">`: Specifies the character encoding, essential for displaying text correctly.
    - `<meta name="viewport" ...>`: Ensures the page renders correctly on mobile devices (it makes the page responsive).

    - `<title>`: Sets the title of the page, which appears in the browser tab and in search engine results.
- `<body>`: This element contains **all the visible content** of the webpage—what you actually see in the browser window, such as headings (`<h1>`), paragraphs (`<p>`), images (`<img>`), links (`<a>`), and more.

## Img

```
<img src="img/antelope-canyon-9391418_1280.jpg" alt="Antelope canyon" width="50%" height="500px">
```

src: path of image

alt:

- **It is read aloud** by screen readers for visually impaired users.
- **It is displayed** on the page if the image fails to load (e.imga, broken link, slow connection).
- **It helps search engines** understand the content of the image, which is good for SEO.

width: width of image in percentage or px

height: height of image in percentage or px

## Table

```
<table border="1">
    <caption>Opinion Poll Results</caption>

    <tr>
      <th>heading</th>
    </tr>
    <tr>
```

```
    <td>Data</td>
  </tr>

</table>
```

- **Use `<th>` for Headers:** This is the most important step. Screen readers announce `<th>` elements as headers, providing context for the data in the `<td>` cells.
- **Add a `<caption>`:** The `<caption>` tag provides a title for your table. It should be the *first element* inside the `<table>` tag. This is the first thing a screen reader will announce, giving the user an overview of the table's content.
- **Use the `scope` Attribute:** This attribute explicitly tells a screen reader what the header is for.
  - `scope="col"`: Use this on a `<th>` to indicate it's a header for the entire **column** below it.
  - `scope="row"`: Use this on a `<th>` (if the first cell in a row is a header) to indicate it's a header for the **row** to its right.
- `colspan="number"`: Makes a cell span horizontally across a `number` of **columns**.
- `rowspan="number"`: Makes a cell span vertically across a `number` of **rows**.

## Form

```
<form action="/submit-page" method="POST">

  <div>
    <label for="username">Name:</label>
    <input type="text" id="username" name="user_name">
  </div>

  <div>
    <label for="pass">Password:</label>
    <input type="password" id="pass" name="password">
  </div>
```

```
  <div>
    <button type="submit">Log In</button>
  </div>

</form>
```

- `<form>`: This is the main container for all your form inputs.
- `<label>`: This element is **critical for accessibility**. It provides a text description for an input. By using the `for` attribute, we link the label to a specific input's `id`. This has two benefits:
  1. Screen readers can announce the label when the user focuses on the input.
  2. Users can click the label text to focus on the corresponding input, which is especially helpful for small elements like radio buttons or checkboxes.

## Input types

| Type | Description |
| --- | --- |
| `type="text"` | A single-line text field. Used for names, usernames, etc. |
| `type="password"` | A single-line text field that masks the input (e.g., `••••••••`). |
| `type="email"` | A text field that requires a valid email format for submission. |
| `type="number"` | A field for numbers. Often shows spinner controls. |
| `type="checkbox"` | A toggleable box. Use for "select one or more" options (e.g., "Agree to terms"). |
| `type="radio"` | A radio button. Used in a group (by sharing the same `name` attribute) for "select only one" options. |
| `type="date"` | Provides a date picker calendar for selecting a date. |
| `type="file"` | Allows the user to select one or more files from their device. |
| `type="submit"` | A button that, when clicked, submits the form. |

## Form validation

HTML5 provides built-in browser validation, which checks the data *before* the form is submitted. This improves user experience by providing immediate feedback.

- `required`: The field must be filled out.
- `minlength` / `maxlength`: Specifies the minimum and maximum number of characters for a text field.
- `min` / `max`: Specifies the minimum and maximum value for a number or date input.
- `type="email"` / `type="url"`: The browser automatically checks for a valid email or URL format.
- `pattern="[A-Za-z]{3}"`: Allows you to specify a **regular expression (regex)** for the input to match. The example `[A-Za-z]{3}` would require exactly three letters.

## Form action

The `<form>` tag has two key attributes that control what happens on submission.

1. `action`: This attribute specifies the **URL** (or "endpoint") where the form data will be sent for processing. This is typically a server-side script (e.g., `/login`, `submit-contact.php`).
2. `method`: This attribute specifies the **HTTP method** to use when sending the data. The two most common methods are `GET` and `POST`.

## GET vs POST

| Feature | method="GET" | method="POST" |
| --- | --- | --- |
| **Visibility** | Form data is **appended to the URL** (e.g., `.../search?query=html`). | Form data is sent in the **body of the HTTP request**. It is **not visible** in the URL. |

| | | |
|---|---|---|
| **Use Case** | Good for **non-sensitive** data, like search queries or filtering, where we want the URL to be shareable or bookmarkable. | **Must be used** for **sensitive** data (passwords, personal info, payment details) and for any action that changes data on the server (creating a new user, posting a comment). |
| **Data Limit** | Limited by the maximum length of a URL (varies by browser, but ~2000 characters). | No practical limit on data size (can send large amounts of text or files). |
| **Caching** | Can be cached by the browser and will show up in browser history. | Not cached and does not show up in browser history (which is safer for sensitive data). |
| **Security** | **Insecure** for sensitive data. Never use it for passwords. | **More secure** because the data isn't exposed in the URL. (Note: Use `https` to encrypt the data in transit). |

# Semantic tags

| Tag | Meaning | Typical Use |
|---|---|---|
| `<header>` | Page or section header | Contains logo, navigation, title |
| `<nav>` | Navigation section | Contains main menus, links |
| `<article>` | Self-contained content | Blog post, news, forum entry |
| `<section>` | Thematic grouping of content | Chapter, group of articles |
| `<aside>` | Side content | Sidebar, ads, related links |
| `<footer>` | Page or section footer | Copyright, contact info, links |
| `<main>` | Main content area | Central unique page content |

## SEO importance

- **Improves search ranking** — Search engines like Google understand page structure and relevance better.

- **Enhances accessibility** — Screen readers use semantic tags to navigate pages easily.
- **Improves readability** — Developers and browsers understand content sections clearly.
- **Rich snippets & structured data** — Helps search engines identify key content such as articles, headers, and footers.

# CSS

## Selector

A **CSS selector targets HTML elements** to apply **styles** to them. It tells the browser *which elements should receive these styles.*

There is type of selector

1. Element selector

```
h1{}
```

2. Class selector

```
.className{}
```

3. Id selector

```
#idName{}
```

4. Attribute selector

```
a[target="_blank"]{}
```

5. Descendant selector

```
/* select all with class link present inside nav */
nav .link{}
```

6. Child selector

```
/* select all direct element with class link inside nav */
nav > .link {}
```

## Specificity

**Specificity** is the algorithm the browser uses to decide which CSS rule to apply when multiple rules target the same element. It is a scoring system. The selector with the **highest score wins**.

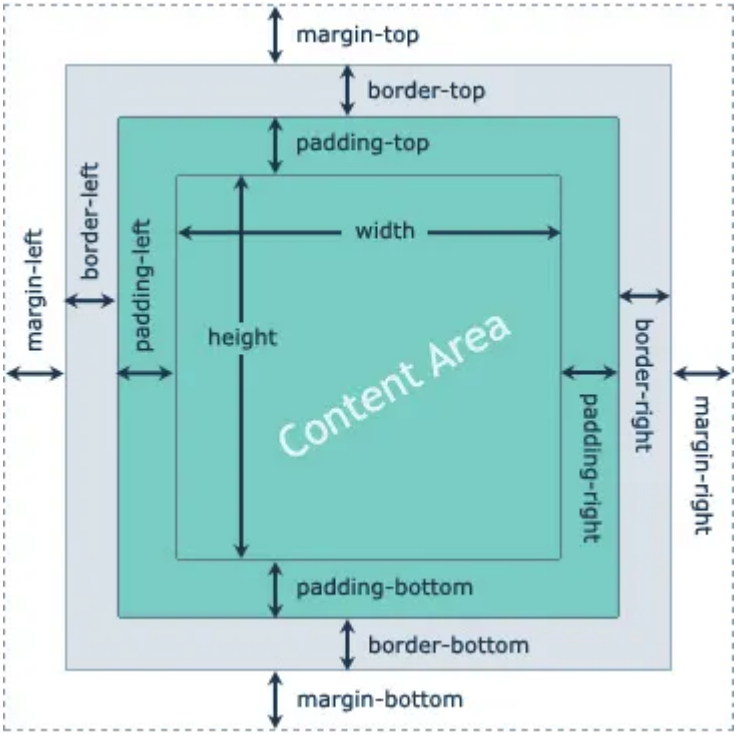
If we have two rules, the browser doesn't just pick the last one it sees; it calculates the specificity of each selector first.

## Specificity Hierarchy

Here is the hierarchy from highest to lowest specificity:

1. **Inline Styles** (Highest)
    - Written directly on the HTML element: `<p style="color: red;">`
    - This "wins" against almost everything.
2. **ID Selectors (`#main-header`)**
    - Very high specificity. An ID will override any class or element selector.
3. **Class (`.highlight`), Attribute (`[type="text"]`), and Pseudo-class (`:hover`) Selectors**
    - These all have the same level of specificity. They will override element selectors.
4. **Element (`p`) and Pseudo-element (`::before`) Selectors**
    - The lowest specificity. These are the easiest to override.

# Box model




```css
.box {
  width: 200px;              /* Content width */
  padding: 10px;             /* Inside space */
  border: 5px solid blue;    /* Border around */
  margin: 20px;              /* Outer space */
  background-color: lightyellow;
  box-sizing: border-box;
}
```

| Value | Meaning |
| --- | --- |
| `content-box` (default) | Width includes only content. Padding and border are added outside. |
| `border-box` | Width includes content + padding + border (easier for layouts). |

# Layouts

## Flex

Flexbox is designed to distribute space among items in a single dimension.

**Main Axis:** The primary direction your flex items are laid out in. By default, this is **horizontal** ( `flex-direction: row` ).

## Key Properties

- `display: flex;` (on the parent): Turns on Flexbox.
- `flex-direction: row | column;` (on the parent): Sets the main axis.
- `justify-content:` (on the parent): Aligns items along the **main axis** (e.g., `center` , `space-between` ).
- `align-items:` (on the parent): Aligns items along the **cross axis** (e.g., `center` , `stretch` )

```
<style>
    #flexSection {
      border: 2px solid black;
      display: flex;
      height: 500px;
      width: 500px;
      align-items: center;
      justify-content: space-evenly;
      flex-direction: row;
    }
    .box {
      background-color: red;
      width: 100px;
      height: 100px;
      margin: 10px;
      box-sizing: border-box;
      border: 2px solid blue;
    }
</style>
<body>
   <section id="flexSection">
    <div class="box"></div>
```

```
    <div class="box"></div>
    <div class="box"></div>
    <div class="box"></div>
  </section>
</body>
```

## Grid

Grid is a powerful system that lets us create complex, two-dimensional layouts. We define a grid with columns and rows on the parent, and then we can place items *precisely* within that grid.

Key Properties

- `display: grid;` (on the parent): Turns on Grid.
- `grid-template-columns:` (on the parent): Defines columns. (e.g., `100px 100px 100px` for three 100px columns, or `1fr 2fr` for two columns where one is twice as large as the other).
- `grid-template-rows:` (on the parent): Defines rows.
- `gap:` (on the parent): Sets the space *between* your rows and columns.
- `grid-column:` / `grid-row:` (on the child): Specifies which grid lines an item should start and end on, allowing it to span multiple cells.

```
<style>
        #gridSection {
    border: 2px solid black;
    display: grid;
    grid-template-columns: 1fr 2fr 1fr 1fr;
    grid-template-rows: 100px 100px 100px 100px 100px;
    gap: 10px;
  }
</style>

<body>
  <section id="gridSection">
```

```
    <div class="box" style="grid-column: 2; grid-row: 2">1</div>
    <div class="box">2</div>
    <div class="box">3</div>
    <div class="box">4</div>
    <div class="box">5</div>
    <div class="box">6</div>
    <div class="box">7</div>
    <div class="box">8</div>
    <div class="box">9</div>
    <div class="box">10</div>
    <div class="box">11</div>
    <div class="box">12</div>
    <div class="box">13</div>
    <div class="box">14</div>
    <div class="box">15</div>
  </section>
</body>
```

## Responsive Design

Responsive design is an approach to web design that makes site look good and work well on **all devices**, from tiny mobile phones to wide desktop monitors.

### Media query

Media query is a CSS feature that allows to apply different styles **only when certain conditions are met**.

Ex:

```
/* Tablet styles */
@media (min-width: 768px) and (max-width: 1024px) {
  #navBar {
    background-color: brown;
  }
  #navBar::before {
    content: "Tablet";
```

```
  }
}
```

The css only applied if screen size is more than 768px and less than 1024 px

## Relative units

- `%` **(Percentage):** The most common relative unit. It sets the size of an element as a percentage of its **parent element's** size.
  - `width: 50%;` means "make me half as wide as my parent container."
- `vw` **(Viewport Width) /** `vh` **(Viewport Height):** This unit is relative to the **browser window's (viewport's) size**.
  - `1vw` = 1% of the viewport's width.
  - `font-size: 5vw;` will make text get larger as the browser window gets wider.
- `rem` **(Root Em):** This unit is relative to the **root** `<html>` **element's font-size**. This is the best unit for creating scalable text and spacing that respects a user's browser settings.
  - If the root font-size is `16px`, then `2rem` is `32px`.
- `em` **(Em):** This unit is relative to the **parent element's font-size**. It can be tricky because it compounds (if a parent is `1.2em` and its child is also `1.2em`, the child will be 1.44x the base size). `rem` is usually preferred.

```
font-size: 1.5rem;  /* font size will increse as use zoom */
```

## Mobile first design

Mobile-first is a **design strategy**. Instead of designing the large desktop site and then trying to "shrink it down" (which is difficult), we do the opposite:

**We design the small-screen, mobile layout first, and *then* add more features and complexity as the screen gets larger.**

```css
/* Mobile (default) */
body {
  font-size: 16px;
}

/* Larger screens */
@media (min-width: 768px) {
  body {
    font-size: 18px;
  }
}
```

# Bootstrap

- **Created by:** Mark Otto and Jacob Thornton at **Twitter** in **2011**.
- **Original name:** *Twitter Blueprint*.
- **Purpose:** To help developers at Twitter maintain **consistent design and code** across multiple internal tools.
- **Open-sourced:** In **August 2011** on GitHub.

## Usage/Setup

1. Use CDN

In head

```
<link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-sRIl4kxILFvY47J16cr9ZwB07vP4J8+LH7qKQnuqkuIA
```

```
vNWLzeN8tE5YBujZqJLB"
      crossorigin="anonymous"
    />
```

In body at last

```
<script
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.min.js"
      integrity="sha384-FKyoEForCGlyvwx9Hj09JcYn3nv7wiPVlz7YYwJrWVcXK/BmnVDxM+D2scQbITxI"
      crossorigin="anonymous"
    ></script>
```

2. Offline setup

Download bootstrap files from https://getbootstrap.com/docs/5.3/getting-started/download/

Then use it by giving relative path

```
<link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
<script src="bootstrap/js/bootstrap.bundle.min.js"></script>
```

3. Using package manager

```
npm install bootstrap
```

Then import it

```
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';
```

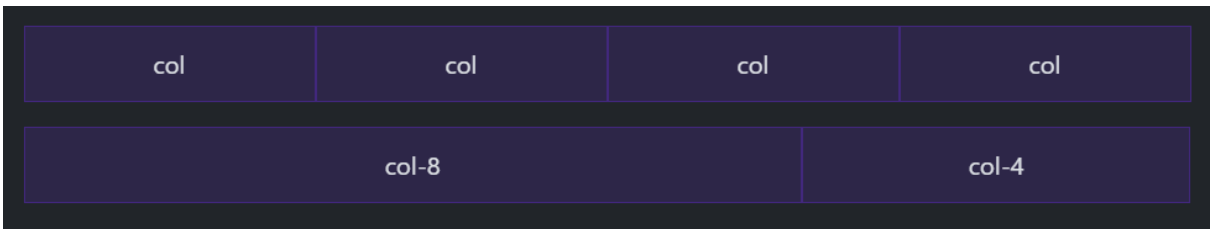
# Grid system

Bootstrap's **grid system** makes it easy to **create responsive page layouts**. It divides the page into **12 equal columns** that adjust automatically based on screen size.

The grid is built using three key classes:

1. `.container` **(or** `.container-fluid` **):**
    - This is the **outermost wrapper**.
    - `.container` provides a responsive, fixed-width box that centers your content.
    - `.container-fluid` provides a full-width box that always spans 100% of the screen.
2. `.row` :
    - Rows are the **horizontal groups** that hold your columns.
    - You must place columns *inside* a `.row` .
    - A `.row` acts as the flex container ( `display: flex` ) for your columns.
3. `.col-*` :
    - These are your **columns**, and they are the immediate children of a `.row` .
    - This is where your actual content (text, images, cards) goes.
    - The * is replaced by a number (1-12) to specify how many of the 12 available columns that element should span.

Ex:



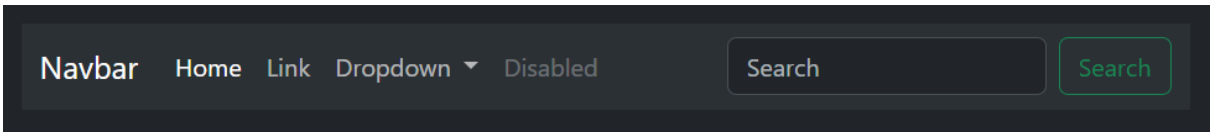
```
<div class="container text-center">
  <div class="row">
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
  </div>
  <div class="row">
    <div class="col-8">col-8</div>
    <div class="col-4">col-4</div>
  </div>
</div>
```

## Components

Bootstrap provide component with html code which can be copied and used as per need

Ex:

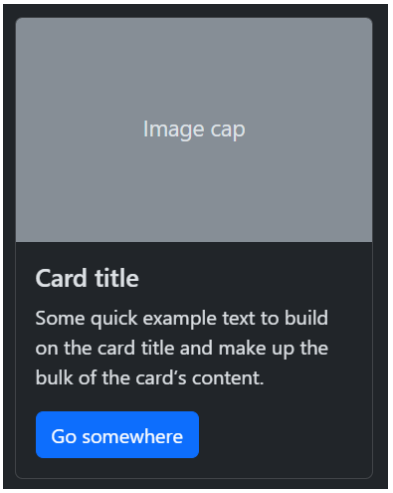
Navbar




```
<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collaps
e" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupp
ortedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
```

```
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Link</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" role="button" data-bs
-toggle="dropdown" aria-expanded="false">
            Dropdown
          </a>
          <ul class="dropdown-menu">
            <li><a class="dropdown-item" href="#">Action</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a>
</li>
          </ul>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" aria-disabled="true">Disabled</a>
        </li>
      </ul>
      <form class="d-flex" role="search">
        <input class="form-control me-2" type="search" placeholder="Searc
h" aria-label="Search"/>
        <button class="btn btn-outline-success" type="submit">Search</butt
on>
      </form>
    </div>
  </div>
</nav>
```

Card

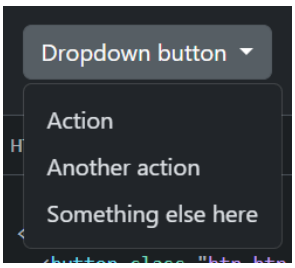



```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Spinner

```
<div class="spinner-border" role="status">
  <span class="visually-hidden">Loading...</span>
</div>
```

Dropdown



```
<div class="dropdown">
  <button class="btn btn-secondary dropdown-toggle" type="button" data-bs-toggle="dropdown" aria-expanded="false">
    Dropdown button
  </button>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" href="#">Action</a></li>
    <li><a class="dropdown-item" href="#">Another action</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
  </ul>
</div>
```