



Week 6 - HTML, CSS, Bootstrap

Created by  Dakshil Gorasiya

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. |
| 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 (<code><header></code> , <code><article></code>), 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 (``), links (`<a>`), and more.

Img

```

```

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

crossorigin: handles CORS settings for the image(anonymous, use-credentials)

fetchpriority: indicates the priority of the image loading(high, low, auto)

loading: specifies whether a browser should load an image immediately or to defer loading(lazy, eager, auto)

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 |
|------------------------------|--|
| <code>type="text"</code> | A single-line text field. Used for names, usernames, etc. |
| <code>type="password"</code> | A single-line text field that masks the input (e.g., <code>.....</code>). |
| <code>type="email"</code> | A text field that requires a valid email format for submission. |
| <code>type="number"</code> | A field for numbers. Often shows spinner controls. |
| <code>type="checkbox"</code> | A toggleable box. Use for "select one or more" options (e.g., "Agree to terms"). |

| | |
|----------------------------|--|
| <code>type="radio"</code> | A radio button. Used in a group (by sharing the same <code>name</code> attribute) for "select only one" options. |
| <code>type="date"</code> | Provides a date picker calendar for selecting a date. |
| <code>type="file"</code> | Allows the user to select one or more files from their device. |
| <code>type="submit"</code> | 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` .

Fieldset

Can be used to group related input together

use <legend> tag inside fieldset tag to give title

Textarea

Can be used to get large text input

attributes: rows, cols

GET vs POST

| Feature | method="GET" | method="POST" |
|------------|--|--|
| Visibility | Form data is appended to the URL (e.g., <code>.../search?query=html</code>). | 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 <code>https</code> to encrypt the data in transit). |

Semantic tags

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

Meta tag

The type of metadata provided by the `<meta>` element can be one of the following:

- If the `name` attribute is set, the `<meta>` element provides *document-level metadata* that applies to the whole page.

- If the `http-equiv` attribute is set, the `<meta>` element acts as a *pragma directive* to simulate directives that could otherwise be given by an HTTP header.
- If the `charset` attribute is set, the `<meta>` element is a *charset declaration*, giving the character encoding in which the document is encoded.
- If the `itemprop` attribute is set, the `<meta>` element provides *user-defined metadata*.

To set charset

```
<meta charset="UTF-8" />
```

To refresh webpage after delay (If url is provided it will redirect)

```
<meta http-equiv="refresh" content="2" />
```

To define information about page

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
<meta name="author" content="dakshil" />
<meta name="application-name" content="HTML Meta Tags Demo" />
<meta
  name="description"
  content="The HTML reference describes all elements and attributes of
  HTML, including global attributes that apply to all elements."
/>
```

To give keywords

```
<meta
  name="keywords"
```

```
content="HTML, tags, attributes, elements, reference, global attributes"
/>
```

To define instruction for web crawlers

- follow → allows following links
- noarchive → prevents storing a cached copy
- noindex → prevents indexing
- nofollow → prevents following links
- nosnippet → prevents showing a snippet in search results

```
<meta name="robots" content="index, follow" />
```

<a> tag

```
<a href="https://www.google.com" target="_blank">Open in New Tab (_blank)</a>
```

href: url

target:

 _blank: open in new tab

 _self: open in same tab

title: to give tooltip on hover

To open mail app use href="mailto:test@example.com"

<audio> tag

```
<audio controls controlslist="nodownload">
  <source src="./audio/sample.mp3" type="audio/mpeg" />
  Your browser does not support the audio element.
</audio>
```

Other attributes for the `audio` tag:

- `autoplay` : Starts playing the audio as soon as it's ready (default is no autoplay).
- `loop` : Restarts the audio from the beginning every time it finishes (default is no loop).
- `muted` : Mutes the audio by default (default is unmuted).
- `controls` : Displays audio controls like play, pause, and volume.
- `preload` : Specifies if and how the audio should be loaded when the page loads (`auto` , `metadata` , `none`).

<video> tag

```
<video
  controls
  poster="./img/antelope-canyon-9391418_1280.jpg"
  controlslist="nodownload"
  width="500px"
  height="300px"
>
  <source src="./video/sample.mp4" type="video/mp4" />
  Your browser does not support the video element.
</video>
```

Attributes

- `autoplay` : Starts playing the video as soon as it's ready (default is no autoplay).

- `loop` : Restarts the video from the beginning every time it finishes (default is no loop).
- `muted` : Mutes the video by default (default is unmuted).
- `controls` : Displays video controls like play, pause, and volume.
- `preload` : Specifies if and how the video should be loaded when the page loads (`auto` , `metadata` , `none`).
- `poster` : Specifies an image to be shown while the video is downloading or until the user plays the video.
- `controlslist` : Provides a hint to the browser about which controls to show or hide (`nodownload` , `nofullscreen` , `noremotep playback`).
- `disablepictureinpicture` : Prevents the video from being played in picture-in-picture mode.
- `disableremotep playback` : Prevents the video from being played on remote playback devices.

<dd>, <dt>, <dl> tags

```
<dl>
  <dt>HTML</dt>
  <dd>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sapien-
    te, laborum
    a, eveniet quos aspernatur, nostrum fugiat quaerat unde neque vero cum
    que.
    Perferendis quos dicta, totam quod earum eaque amet nulla?
  </dd>
  <dd>Lorem, ipsum dolor.</dd>

  <dt>CSS</dt>
  <dd>Lorem, ipsum dolor.</dd>
</dl>
```

HTML

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sapiente, laborum a, eveniet quos aspernatur, nostrum fugiat quaerat unde neque vero cumque. Perferendis quos dicta, totam quod earum eaque amet nulla?

Lorem, ipsum dolor.

CSS

Lorem, ipsum dolor.

dl: description list

dt: description title

dd: description data

CSS

How to use CSS

1. inline css

```
<div class="container" style="border: 2px solid black" />
```

2. Internal css

Write css in `<style></style>` tag in head

3. External css

Make a css file and then link it

```
<link rel="stylesheet" href="index.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 {}
```

7. Sibling selector

```
/* select div that follows another div */  
#thirdSection > div ~ div {  
  font-style: italic;  
  border: 1px dashed gray;  
}
```

8. Adjacent sibling selector

```
/* select div that immediately follows another div */
#thirdSection > div + div {
  background-color: lightyellow;
  padding: 5px;
}
```

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.

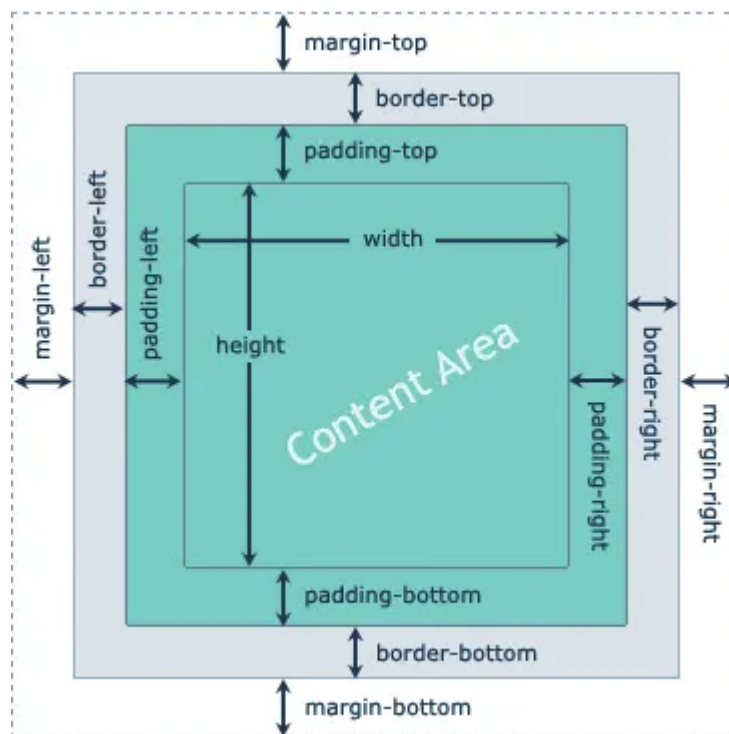
3. Class (`.highlight`), Attribute (`[type="text"]`), and Pseudo-class (`:hover`) Selectors

- 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



```
.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;  
}
```

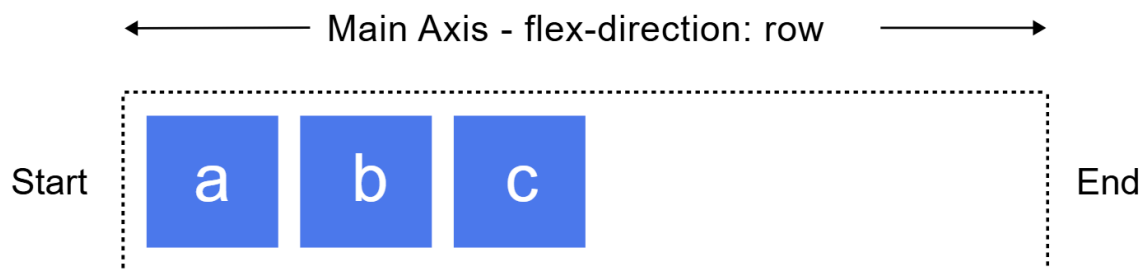
| Value | Meaning |
|---------------------------------------|--|
| <code>content-box</code> (default) | Width includes only content. Padding and border are added outside. |
| <code>border-box</code> | Width includes content + padding + border (easier for layouts). |

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`).



- Items display in a row (the `flex-direction` property's default value is `row`).
- The items start from the start edge of the main axis.
- The items do not stretch on the main dimension but can shrink (a flex-item's `flex-grow` property's default value is `0` and its `flex-shrink` property's default value is `1`).
- The items will stretch to fill the size of the cross-axis (the `align-items` property's default value is `stretch`).
- All the items will be in a single row (the `flex-wrap` property's default value is `nowrap`), overflowing their container if their combined `width / height` exceeds the containing element `width / height` .

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>

```

justify-content options

- flex-start (default): items align to the start of the container
- flex-end: items align to the end of the container
- space-between: items are evenly distributed, first item at start, last item at end
- space-around: items are evenly distributed with equal space around them (start and end have half space)

- space-evenly: items are evenly distributed with equal space between them (including start and end)

align-item options

- flex-start: items align to the start of the cross axis
- flex-end: items align to the end of the cross axis
- stretch (default): items stretch to fill the container along the cross axis

flex-direction options: row, column, row-reverse, column-reverse

Shorthand for flex-direction and flex-wrap

```
flex-flow: column wrap;
```

Flex property for individual items

Define minimum space required

```
flex-basis: 150px;
```

Define how fast it will shrink

```
flex-shrink: 1;
```

Item can take place if there is available

Item with higher value will grow faster

```
flex-grow: 1;
```

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>
    </section>
</body>
```

```

<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>

```

To repeat column size

```
grid-template-columns: repeat(2, 1fr 2fr);
```

To make flexible rows

```
grid-auto-rows: minmax(100px, auto); /* with minimum row size 100px */
```

For items to set position

```
grid-row-start: 2;
grid-column-start: 2;
```

To use grid area

Set template in parent

```

grid-template-areas:
  "hd hd hd hd  hd  hd  hd  hd  hd"
  "sd sd sd main main main main main"
  ". . ft ft ft ft ft ft ft";

```

To place child in specific position

```
grid-area: hd;
```

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 increase 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.

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

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

Before, after

`::before` and `::after` are **CSS pseudo-elements** used to insert **content before or after** an element's **actual content** — without adding extra HTML elements.

```
selector::before {  
  content: "Text or symbol";  
}  
  
selector::after {  
  content: "Text or symbol";  
}
```

Without `content`, the pseudo-element won't appear.

Background

To change background color

```
background-color: red;  
background-color: #ff0000ff;  
background-color: rgba(255, 0, 0, 0.5);  
background-color: hsl(0, 100%, 50%);  
background-color: hsla(115, 100%, 51%, 0.5);
```

To use background image

```
background-image: url("../html/img/antelope-canyon-9391418_1280.jpg");
```

other property to control image

```
background-repeat: no-repeat; /* repeat-x, repeat-y, no-repeat, round, space */  
  
/* position of image */
```



```
background-position: 25% 75%; /* x-offset y-offset */
background-position: 0 0;
background-position: center center;

background-position-x: 0;

background-size: cover; /* contain, auto, <length> <length> */
```

Shorthand

```
background: no-repeat
           url("../html/img/antelope-canyon-9391418_1280.jpg") center/cover;
```

Animation

To create keyframe (we can also use from, to instead of percentage)

```
@keyframes moveBox {
  0% {
    transform: translateX(0);
    background: red;
  }
  50% {
    transform: translateX(100px);
    background: orange;
  }
  100% {
    transform: translateX(200px);
    background: yellow;
  }
}
```

To set animation

```
/* name duration timing-function delay siteration-count direction fill-mode */
animation: moveBox 2s ease-in-out 1s 1 alternate forwards;
```

other property

```
animation-name: moveBox2;

animation-delay: 250ms;

/*
    normal: plays forward
    reverse: plays backward
    alternate: plays forward then backward
    alternate-reverse: plays backward then forward
*/
animation-direction: normal;
animation-duration: 1s;

animation-fill-mode: both;

animation-iteration-count: 1;
```

Border

To set bottom border

```
border-bottom: 2px dashed red; /* shorthand */
border-bottom-color: blue;
border-bottom-left-radius: 20px;
border-bottom-style: dotted;
```

To set border style

```
border-style: dotted;
```

To set border-radius

```
border-radius: 10px 20px 30px 40px; /* top-left, top-right, bottom-right, bottom-left */
```

Shorthand

```
border: 2px solid black;
```

Display

```
display: inline;
```

- inline: element does not start on a new line and only takes up as much width as necessary
- block: element starts on a new line and takes up the full width available
- inline-block: element does not start on a new line and width and height can be set
- none: element is not displayed
- flex: element becomes a flex container
- grid: element becomes a grid container

Position

```
position: sticky;
```

- static: default value, element is positioned according to the normal flow of the document
- relative: element is positioned relative to its normal position
- absolute: element is positioned relative to its nearest positioned ancestor (not static)
- fixed: element is positioned relative to the viewport, stays in the same place even during scrolling
- sticky: element is treated as relative positioned until it crosses a specified threshold, then it is treated as fixed positioned

Font family

```
font-variant: small-caps;
```

```
font-family: sans-serif;
```

```
font-size: large;
```

```
font-size: 20px;
```

```
font-style: italic;
```

```
font-weight: bold;
```

```
font-weight: 100;
```

Short hand

```
font: italic small-caps bold 16px;
```

Text family

```
text-decoration: underline;
text-decoration: overline;
text-decoration: line-through red;
text-decoration-line: underline overline;
text-decoration-color: black;
text-decoration-style: dashed; /* solid, wavy, dotted, double */

text-indent: 30px;

text-wrap: stable; /* wrap, nowrap, balance, pretty, stable */
```

Overflow

controls overflowing items

```
overflow: auto;
```

Other values

- visible: default value, content is not clipped
- hidden: content is clipped, no scrollbars
- scroll: content is clipped, always show scrollbars
- auto: content is clipped, show scrollbars when necessary
- clip: content is clipped, no scrollbars (same as hidden)

Shadow

```
/* box-shadow: h-offset v-offset blur spread color; */
box-shadow: 10px 5px 10px 10px red;
```

Transition

```
transition-property: background-color, transform;  
transition-duration: 0.5s, 0.3s;  
transition-timing-function: ease, ease;  
transition-delay: 0s, 0s;
```

timing function

- ease: starts slow, speeds up, ends slow
- ease-in: starts slow, ends fast
- ease-out: starts fast, ends slow
- ease-in-out: starts slow, speeds up, ends slow
- linear: constant speed
- step-start: jumps to end value at start of duration
- step-end: jumps to end value at end of duration

Variables

To define it

```
--main-color: cyan;
```

To use it

```
border: 2px solid var(--main-color);
```

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-sRII4kxILFvY47J16cr9ZwB07vP4J8+LH7qKQnuqkuAvNWLzeN8tE5YBujZqJLB"
  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-FKyoEForCGlyvw9Hj09JcYn3nv7wiPVIz7YYwJrWVcXK/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

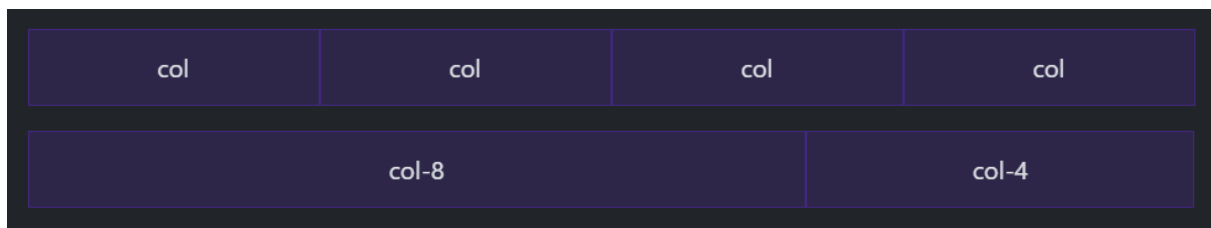
Bootstrap's **grid system** makes it easy to **create responsive page layouts**. It divides the page into **12 equal columns** that adjust automatically based on screen size.

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:



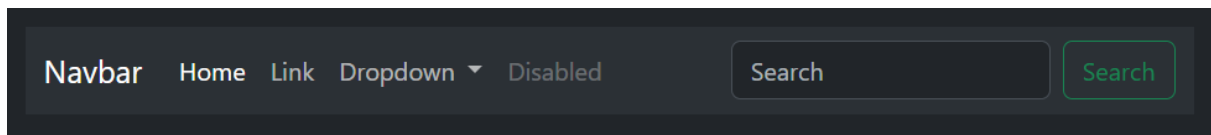
```
<div class="container text-center">
  <div class="row">
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
    <div class="col">col</div>
  </div>
  <div class="row">
    <div class="col-8">col-8</div>
    <div class="col-4">col-4</div>
  </div>
</div>
```

Components

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

Ex:

Navbar



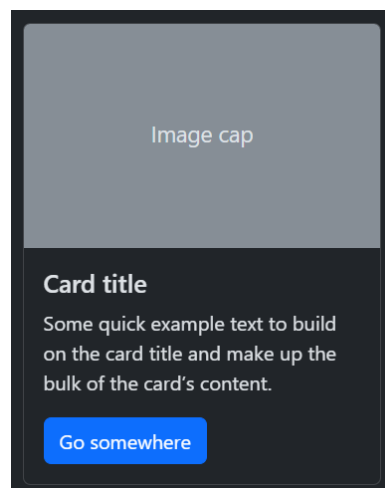
```
<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" 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>
    </div>
  </div>
</nav>
```

```

    </li>
  </ul>
  <form class="d-flex" role="search">
    <input class="form-control me-2" type="search" placeholder="Search"
    aria-label="Search"/>
    <button class="btn btn-outline-success" type="submit">Search</button>
  </form>
</div>
</div>
</nav>

```

Card



```

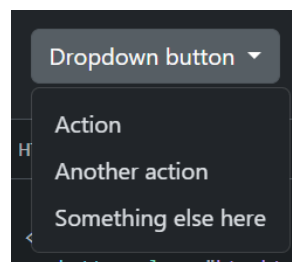
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title
    and make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  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

Other classes

position

position-stick, position-fixed

border

```
<div  
  class="container border border-start-0 border-black"  
  style="height: 100px; width: 100px"  
></div>
```

text-color

text-success, text-primary, ..

display

d-flex, d-block, d-inline

links(<a> tag)

link-danger, link-underline-dark, ..

Overflow

overflow-hidden, overflow-scroll, over-auto

Shadow

shadow-lg, shadow-md

Text

text-start, text-center, text-end

Breakpoints

| Breakpoint | Class infix | Dimensions |
|-------------------|-------------|------------|
| Extra small | <i>None</i> | <576px |
| Small | <i>sm</i> | ≥576px |
| Medium | <i>md</i> | ≥768px |
| Large | <i>lg</i> | ≥992px |
| Extra large | <i>xl</i> | ≥1200px |
| Extra extra large | <i>xxl</i> | ≥1400px |