# CSS

- CSS stands for Cascading Style Sheets
- CSS is the language we use to style an HTML document.
- CSS describes how HTML elements should be displayed

## 🎯 Importance of CSS in Web Design

- Enhances **visual appeal** of websites.
- Enables **responsive design** for different devices.
- Allows for **reusable styles**, making maintenance easier.

## 🕰️ CSS Versions

| Version | Description |
|---|---|
| **CSS1** | Introduced in 1996. Basic styling for fonts, colors, and spacing. |
| **CSS2** | Released in 1998. Added positioning, z-index, media types, etc. |
| **CSS3** | Modular and still evolving. Introduced media queries, flexbox, animations, transitions, gradients, etc. |

## 🧩 How to Add CSS to HTML

1. **Inline CSS**

Applied directly to an HTML element using the style attribute.

Hello World

2. **Internal CSS**

Written inside a <style> tag within the <head> of the HTML document.

```
<head>
 <style>
  p {
   color: blue;
```

```
    font-size: 18px;
   }
  </style>
</head>
```

3. **External CSS**

Stored in a separate .css file and linked via the <link> tag.

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

## 📝 Basic Syntax of CSS

```
selector {
  property: value;
}
```

- **Selector**: The HTML element to style (e.g., h1)
- **Property**: The style attribute (e.g., color)
- **Value**: The value of the property (e.g., navy)

# 🎯 CSS Selectors

**CSS selectors** are used to "select" HTML elements to apply styles to them.

1. **Universal Selector (*)**
2. **Type Selector (Element Selector)**
3. **Class Selector (.class)**
4. **ID Selector (#id)**
5. **Grouping Selectors**
6. **Attribute Selectors**
7. **Pseudo-Classes**
8. **Pseudo-elements**
9. **Combinators**

1. **Universal Selector (*)**

- Selects all elements on the page.

```
* {
  margin: 0;
```

```
  padding: 0;
}
```

2. **Type Selector (Element Selector)**

- Targets all elements of a specific type (tag name), like h1, p, div.

```
h1 {
  color: blue;
}
```

## 3. Class Selector (.class)

- Targets elements with a specific **class** attribute.
- Can be reused on multiple elements.

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

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

## 4. ID Selector (#id)

- Targets a single, **unique** element by its id.

```html
<h1 id="main-title">Welcome</h1>
```

```css
#main-title {
  font-size: 30px;
  color: green;
}
```

## 5. Grouping Selectors

- Apply the **same styles** to multiple selectors at once by separating them with commas.

```css
h1, h2, p {
  font-family: Arial, sans-serif;
  color: #333;
}
```

## 6. Attribute Selectors

- Attribute selectors target HTML elements **based on attributes or attribute values**

```css
input[required] {
  border: 2px solid red;
}
```

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

## 7. **Pseudo-Classes ( : )**

- Pseudo-classes let you apply styles to elements **based on their state or position** in the document.

### a. **:hover**

Styles an element when the mouse is over it.

```css
button:hover {
  background-color: blue;
  color: white;
}
```

### b. **:focus**

Applies when an element (like an input) is focused.

```css
input:focus {
  outline: 2px solid orange;
}
```

### c. **:first-child / :last-child**

Targets the first or last child of a parent.

```css
li:first-child {
  color: red;
}

li:last-child {
  color: green;
}
```

### d. **:nth-child(n)**

Targets the nth child (1-based index).

```css
li:nth-child(2) {
  font-weight: bold;
}
```

## 8. Pseudo-elements **( :: )**

- **Pseudo-elements** allow you to style or manipulate specific parts of an element .

### 1. ::before

Inserts content before an element’s actual content.

```css
p::before {
  content: "🔷 ";
  color: blue;
}
```

### 2. ::after

Inserts content after an element’s actual content.

```css
p::after {
  content: " ✓";
  color: green;
}
```

### 3. ::first-letter

Styles the first letter of a block of text.

```css
p::first-letter {
  font-size: 200%;
  color: crimson;
}
```

### 4. ::first-line

Styles the first line of text in a block.

```css
p::first-line {
  font-weight: bold;
}
```

### 5. ::selection

Styles the part of an element that is highlighted/selected by the user.

```css
::selection {
  background: yellow;
  color: black;
}
```

## 9. Combinators

- **Combinators** are used to define the relationship between selectors.
- They allow you to target elements based on their position or relationship to other elements in the HTML structure.
- **There are four main types of combinators in CSS:**

### 1. Descendant Combinator (space)

The descendant combinator selects all elements that are nested inside a specified element, regardless of depth.

```css
div p {
  color: blue;
}
```

### 2. Child Combinator (>)

The child combinator selects elements that are direct children of a specified parent element (only one level deep).

```css
ul > li {
  color: red;
}
```

### 3. Adjacent Sibling Combinator (+)

The adjacent sibling combinator selects the first element that is immediately after a specified element and has the same parent.

```css
h2 + p {
  font-size: 14px;
}
```

### 4. General Sibling Combinator (~)

The general sibling combinator selects all elements that are siblings of a specified element and appear after it.

```css
h2 ~ p {
  color: green;
}
```

## 🟦 Box Model

The Box Model is the foundation of layout in CSS. Every HTML element is treated as a box composed of four layers:

- Content
- Padding
- Border
- Margin

1. **Content**

- The **actual text or media** inside the element.

2. **Padding**

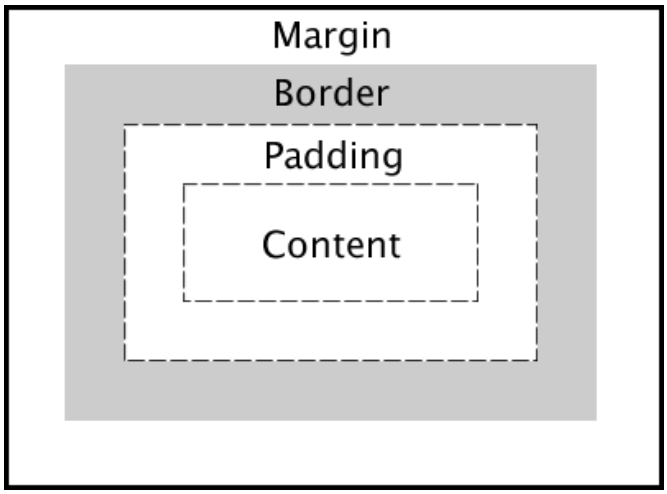
- The **space between the content and the border**.
- Transparent area inside the border.

3. **Border**

- A line around the padding (if any) and content.
- Can be styled, colored, and sized.

4. **Margin**

- The **space outside the border**, between this element and others.
- Used for spacing and layout.




## 🧩 box-sizing Property

1. content-box (default):

- Width/height only applies to content.
- Padding and border are added outside of it.

2. border-box:

- Width/height includes content, padding, and border.
- Makes layouts easier to manage.

## CSS reset rule

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

1. *** (Universal Selector)**
   - Applies the styles to **every element** on the page.
2. **box-sizing: border-box;**
   - Makes the declared width and height **include padding and border**, preventing unexpected layout issues.
   - Helps with easier layout calculations.
3. **margin: 0; & padding: 0;**
   - Removes the browser's **default spacing**, so all elements start with a **clean slate**.

## 🔤 Basic Text Properties

| Property | Description | Example |
| --- | --- | --- |
| color | Sets the text color. | color: red; |
| font-size | Sets the size of the font. | font-size: 16px; |
| font-family | Sets the typeface (font). | font-family: Arial, sans-serif; |
| font-weight | Sets the boldness. | font-weight: bold; |
| font-style | Sets the style (normal, italic). | font-style: italic; |
| text-align | Aligns text (left, center, right, justify). | text-align: center; |
| text-decoration | Adds decoration (underline, line-through). | text-decoration: underline; |
| text-transform | Controls letter casing. | text-transform: uppercase; |
| letter-spacing | Controls spacing between letters. | letter-spacing: 2px; |
| word-spacing | Controls spacing between words. | word-spacing: 4px; |
| line-height | Controls the vertical space between lines. | line-height: 1.5; |
| direction | Sets text direction (LTR or RTL). | direction: rtl; |
| text-shadow | Adds shadow to text. | text-shadow: 2px 2px 4px gray; |
| white-space | Controls how whitespace is handled. | white-space: nowrap; |
| overflow-wrap (or word- | Breaks long words. | overflow-wrap: break-word; |

| wrap) | | |
|---|---|---|

### 🎨 Basic Background Properties

| Property | Description |
|---|---|
| background-color | Sets the background color. |
| background-image | Sets one or more background images. |
| background-repeat | Sets how a background image repeats (e.g., repeat, no-repeat, repeat-x, repeat-y). |
| background-position | Sets the starting position of a background image (e.g., top left, center, 50% 50%). |
| background-size | Specifies the size of the background image (e.g., cover, contain, auto, 100px 200px). |
| background-attachment | Sets whether the background image scrolls with the page or is fixed (scroll, fixed, local). |
| background-clip | Specifies the painting area of the background (border-box, padding-box, content-box). |
| background-origin | Specifies the positioning area of the background image. |
| background-blend-mode | Defines blending between background layers (e.g., multiply, screen, overlay). |

## Important CSS properties

### 1. Layout & Positioning

| Property | Description |
| --- | --- |
| display | Defines the display type of an element (block, flex, etc.) |
| position | Specifies positioning method (static, relative, etc.) |
| top | Offset from top edge (used with positioning) |
| right | Offset from right edge |
| bottom | Offset from bottom edge |
| left | Offset from left edge |
| z-index | Defines stack order of overlapping elements |
| float | Floats the element to left or right |
| clear | Controls behavior around floated elements |

## 2. Box Model

| Property | Description |
| --- | --- |
| display | Defines the display type of an element (block, flex, etc.) |
| position | Specifies positioning method (static, relative, etc.) |
| top | Offset from top edge (used with positioning) |
| right | Offset from right edge |
| bottom | Offset from bottom edge |
| left | Offset from left edge |
| z-index | Defines stack order of overlapping elements |
| float | Floats the element to left or right |
| clear | Controls behavior around floated elements |

**3. Visual Effects**

| Property | Description |
| --- | --- |
| box-shadow | Adds shadow around the element |
| opacity | Sets the transparency level of the element |
| filter | Applies visual effects (blur, brightness, contrast, etc.) |
| visibility | Controls whether element is visible without removing it |

4. Animation & Transition

| Property | Description |
| --- | --- |
| transition | Smooth change of property values |
| transform | Applies transformations (rotate, scale, move, skew, etc.) |
| animation | Assigns animation to element |
| @keyframes | Defines keyframe animation steps |

5. Border & Outline

| Property | Description |
| --- | --- |
| border-radius | Rounds the corners of the element |
| outline | Draws a line outside the border (not part of box model) |
| outline-offset | Distance between outline and border |

## 6. Overflow & Scroll

| Property | Description |
| --- | --- |
| overflow | Handles overflow (visible, hidden, scroll, auto) |
| scroll-behavior | Sets scroll animation behavior (auto, smooth) |

## 7. Visibility & Interaction

| Property | Description |
| --- | --- |
| cursor | Changes mouse cursor on hover |
| pointer-events | Controls mouse interactions with the element |
| user-select | Allows or disables text selection |

## 🧱 CSS Layout Techniques

Modern layout in CSS is handled using **Display**, **Positioning**, **Float**, **Flexbox**, and **Grid**.

### 🔳 Display Property

Controls how an element behaves in the layout flow.

1. **display: block**

- Takes up **full width**.
- Starts on a **new line**.

2. **display: inline**

- Respects content width.
- Flows **with text**, no line break.

3. **display: inline-block**

- Like inline, but allows **width/height** control.

## 📍 Positioning Elements

Controls how elements are placed relative to normal flow or other elements.

1. **static (default)**

Normal flow positioning (no special behavior).

2. **relative**

Moves element **relative to its normal position**.

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

3. **absolute**

Positions **relative to nearest positioned ancestor**.

```css
div {
  position: absolute;
  top: 0;
  right: 0;
}
```

4. **fixed**

Positions **relative to viewport**, stays fixed when scrolling.

```css
header {
  position: fixed;
  top: 0;
}
```

5. **sticky**

Scrolls with the page but **sticks** when it reaches a position.

```css
nav {
  position: sticky;
  top: 0;
}
```

## 🌀 Float Properties

**float**

- Moves elements **left or right**, allowing text to wrap around.

# 📦 Flexbox Layout

**Flexbox** (Flexible Box Module) is a CSS layout model for arranging elements in **one dimension** (either row or column).

## 1. **Creating a Flex Container**

Apply display: flex to a parent element to enable Flexbox.

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

All direct children of .container become **flex items**.

## 2. **Main Axis vs Cross Axis**

- **Main Axis**: Defined by flex-direction (row by default).
- **Cross Axis**: Perpendicular to the main axis.

## 3. **flex-direction**

Defines the **direction** of the main axis.

```css
.container {
  flex-direction: row;       /* default */
  flex-direction: row-reverse;
  flex-direction: column;
  flex-direction: column-reverse;
}
```

## 4. **justify-content** (Main Axis Alignment)

Aligns items **horizontally** (for row).

```css
.container {
  justify-content: flex-start;    /* default */
  justify-content: flex-end;
  justify-content: center;
  justify-content: space-between;
  justify-content: space-around;
  justify-content: space-evenly;
}
```

## 5. **align-items** (Cross Axis Alignment)

Aligns items **vertically** (for row).

```css
.container {
  align-items: stretch;      /* default */
  align-items: flex-start;
  align-items: flex-end;
  align-items: center;
  align-items: baseline;
}
```

### 6. **flex-wrap**

Allows flex items to wrap onto multiple lines.

```css
.container {
  flex-wrap: nowrap;         /* default */
  flex-wrap: wrap;
  flex-wrap: wrap-reverse;
}
```

### 7. **align-content** (Multi-line Cross-Axis Alignment)

Used **only when items wrap**.

```css
.container {
  align-content: flex-start;
  align-content: flex-end;
  align-content: center;
  align-content: space-between;
  align-content: space-around;
  align-content: stretch;
}
```

🧩 Flex Item Properties

### 1. **flex-grow**

Defines how much a flex item **grows** relative to the rest.

```css
.item {
  flex-grow: 1;
}
```

0 means it won't grow, 1 means it will take up remaining space.

### 2. **flex-shrink**

Defines how a flex item **shrinks** when space is tight.

```css
.item {
  flex-shrink: 1; /* default */
}
```

### 3. **flex-basis**

Specifies the **initial size** of the flex item before growing/shrinking.

```css
.item {
  flex-basis: 200px;
}
```

#### 4. **Shorthand: flex**

Combines the three properties above.

```css
.item {
  flex: 1 1 200px; /* grow shrink basis */
}
```

#### **5. align-self**

Overrides align-items for individual items.

```css
.item {
  align-self: center;
}
```

🧪 **CSS Variables** (Custom Properties)

Define reusable values:

```css
:root {
  --main-color: #007bff;
  --padding: 1rem;
}

button {
  background-color: var(--main-color);
  padding: var(--padding);
}
```

## Transition

The transition property in CSS is used to create smooth animations between property changes.

```
selector {
  transition: [property] [duration] [timing-function] [delay];
}
```

- **property**: The CSS property you want to animate
- **duration**: How long the transition should take
- **timing-function** *(optional)*: Defines the speed curve (e.g., ease, linear, ease-in, ease-out, ease-in-out)
- **delay** *(optional)*: How long to wait before starting the transition.

```
.button {
  background-color: blue;
  color: white;
  transition: background-color 0.3s ease-in-out;
}

.button:hover {
  background-color: darkblue;
}
```

## Animation

The animation property allows you to create complex animations by gradually changing CSS properties over time using **keyframes**.

```
selector {
  animation: [name] [duration] [timing-function] [delay] [iteration-count] [direction]
}
```

| Property | Description |
|---|---|
| name | The name of the keyframes you defined (@keyframes) |
| duration | How long the animation lasts (2s, 500ms, etc.) |
| timing-function | Speed curve (linear, ease, ease-in, ease-out, etc.) |
| delay | Delay before animation starts |
| iteration-count | How many times to run (1, infinite, etc.) |
| direction | Direction of animation (normal, reverse, alternate, etc.) |
| fill-mode | What styles apply before/after the animation (none, forwards, etc.) |
| play-state | Play or pause the animation (running, paused) |

```css
@keyframes growAndColor {
  from {
    width: 100px;
    height: 100px;
    background-color: red;
  }
  to {
    width: 150px;
    height: 150px;
    background-color: green;
  }
}

.box {
  animation: growAndColor 2s ease-in-out infinite alternate;
}
```

## 🗒️ Media Query

A media query in CSS is used to apply styles based on the characteristics of the user's device, such as screen width, height, orientation, resolution, etc.

It's a key part of responsive web design, allowing websites to adapt to different screen sizes (like mobile, tablet, desktop).

## 🧠 Why Use Media Queries?

- To make your website **responsive**.
- To ensure optimal viewing experience across **mobile**, **tablet**, and **desktop**.

### 📌 Syntax:

```css
@media media-type and (condition) {
  /* CSS Rules */
}
```

  - media-type (optional): all, screen, print, etc.
  - condition: Query condition like max-width, min-width, orientation, etc.

```html
<!DOCTYPE html>
<html>
<head>
  <style>
    .box {
      background-color: blue;
      color: white;
      padding: 20px;
      text-align: center;
      font-size: 24px;
    }

    /* Media query for small screens */
    @media (max-width: 600px) {
      .box {
        background-color: green;
        font-size: 18px;
      }
    }
  </style>
</head>
<body>

  <div class="box">
    Resize the screen to see the effect!
  </div>

</body>
</html>
```