Introduction to CSS

What is CSS?

- **CSS (Cascading Style Sheets)** is used to style and format the layout of HTML elements.
- It separates **content (HTML)** from **presentation (CSS)**.
- Helps make web pages **attractive, responsive, and consistent**.

Why use CSS?

1. Improves design (colors, fonts, layout).
2. Makes websites responsive across devices.
3. Reusable styles (write once, use multiple times).
4. Faster page loading (less inline styling).

Types of CSS

1. **Inline CSS** – written inside an HTML element using the `style` attribute.

```
<p style="color: blue; font-size: 18px;">This is inline CSS</p>
```

2. **Internal CSS** – written inside `<style>` tag in the HTML `<head>`.

```
<head>
  <style>
    p {
      color: green;
      font-size: 20px;
    }
  </style>
</head>
<body>
```

```
    <p>This is internal CSS</p>
  </body>
```

3. **External CSS** – written in a separate `.css` file and linked with `<link>`.

```
  <!-- index.html -->
  <head>
    <link rel="stylesheet" href="style.css">
  </head>

  <!-- style.css -->
  p {
    color: red;
    font-size: 22px;
  }
```

Best Practice: Always use **External CSS** for large projects.

CSS Syntax

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

Example:

```
h1 {
  color: blue;
  font-size: 30px;
}
```

- **Selector** → selects HTML element (`h1`)
- **Property** → defines what you want to change (`color`, `font-size`)
- **Value** → actual styling value (`blue`, `30px`)

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 {
      color: purple;
      text-align: center;
    }
    p {
      color: gray;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <h1>Hello CSS</h1>
  <p>This is styled with CSS.</p>
</body>
</html>
```

CSS Selectors

What are Selectors?

- A **selector** is used to target HTML elements so that we can apply styles to them.
- They define **"which elements"** the CSS rules will apply to.

Types of CSS Selectors

1. Universal Selector (*)

Selects **all elements** on the page.

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

Useful for resetting default browser styles.

2. Element Selector

Selects all elements of a given type.

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

All <p> elements will be blue.

3. ID Selector (#)

Selects an element with a specific **id**. (Unique for each element)

```
#main-heading {
  color: red;
}
<h1 id="main-heading">Welcome</h1>
```

4. Class Selector (.)

Selects elements with a specific **class**. (Can be reused multiple times)

```
.highlight {
  background-color: yellow;
}
<p class="highlight">This is highlighted</p>
```

5. Grouping Selector (,)

Apply same style to multiple elements.

```
h1, h2, p {
  font-family: Arial;
}
```

6. Combinators

Used to define relationships between elements.

- **Descendant (`space`)** → selects all elements inside another element

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

All `<p>` inside `<div>` will be green.

- **Child (>)** → selects direct children only

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

- **Adjacent sibling (+)** → selects element immediately after another

```
h1 + p {
  font-weight: bold;
}
```

- **General sibling (~)** → selects all siblings after an element

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

7. Attribute Selectors

Select elements based on attributes.

```
input[type="text"] {
  border: 2px solid blue;
}
a[href^="https"] {
  color: green;
}
a[href$=".pdf"] {
  color: red;
}
```

Note: `^=` = starts with, `$=` = ends with, `*=` = contains

8. Pseudo-classes

Select elements based on **state**.

```
a:hover {
  color: red;
}
input:focus {
  border-color: green;
}
li:nth-child(2) {
  color: orange;
```

```
}
```

9. Pseudo-elements

Select **parts of elements**.

```
p::first-letter {
  font-size: 30px;
  color: blue;
}
p::before {
  content: "Before Elment";
}
p::after {
  content: "After Element";
}
```

Colors and Background in CSS:

CSS supports multiple ways to define colors:

1. **Color Names**

```
h1 {
  color: red;
}
```

2. **Hexadecimal (`#RRGGBB`)**

```
p{
 color:#008000; /* green */
}
```

3. **RGB (Red Green Blue)**

```
p{
 color:rgb(255,0,0); /* red */
}
```

4. **RGBA (Red Green Blue Alpha)** → last value = transparency (0 to 1)

```
p{
 color:rgba(0,0,255,0.5); /* semi-transparent blue */
}
```

5. **HSL (Hue Saturation Lightness)**

```
p{
 color:hsl(120,100%,40%); /* green shade */
}
```

6. **HSLA (Hue Saturation Lightness Alpha)**

```
p{
 color:hsla(200,100%,50%,0.6);
```

}

Background Properties

1. Background Color

```
body {
  background-color: lightblue;
}
```

2. Background Image

```
body{
 background-image: url("bg.jpg");
}
```

3. Background Repeat

```
body{
 background-repeat: no-repeat; /* default is repeat */
}
```

4. Background Position

```
body{
 background-position: center top;
}
```

5. Background Size

```
body{
 background-size: cover; /* cover entire screen */
 /* other values: auto, contain, specific px/% */
}
```

6. Background Attachment

```
body{
 background-attachment: fixed; /* background stays fixed while scrolling */
}
```

7. Background Shorthand

Instead of writing separately, you can write in one line:

```
body{
 background: url("bg.jpg") no-repeat center center/cover;
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background: url("https://picsum.photos/800/400") no-repeat center
center/cover;
      color: white;
      font-family: Arial;
      text-align: center;
      height: 100vh;
    }
    h1 {
      background-color: rgba(0, 0, 0, 0.6);
      padding: 20px;
    }
    p {
```

```
      color: hsl(50, 100%, 50%);
    }
  </style>
</head>
<body>
  <h1>CSS Background Example</h1>
  <p>This text has color using HSL.</p>
</body>
</html>
```

Text and Fonts in CSS

➢ Text Properties:

1. Text Color

```
p {
  color: darkblue;
}
```

2. Text Alignment

```
h1 {
  text-align: center;   /* left, right, center, justify */
}
```

3. Text Decoration

```
a {
  text-decoration: none;   /* none, underline, overline, line-through */
}
```

4. Text Transform

```
p {
  text-transform: uppercase;   /* uppercase, lowercase, capitalize */
}
```

5. Line Height

```
p {
  line-height: 1.6;   /* spacing between lines */
}
```

6. Letter Spacing & Word Spacing

```
p {
  letter-spacing: 2px;
  word-spacing: 5px;
}
```

7. Text Shadow

```
h1 {
  text-shadow: 2px 2px 5px gray;
}
```

➢ Font Properties

1. **Font Family**

```
p {
  font-family: Arial, Helvetica, sans-serif;
}
```

Note: Always provide a **fallback font**.

2. **Font Size**

```
p {
  font-size: 18px;
}
```

3. **Font Style**

```
p {
  font-style: italic;   /* normal, italic, oblique */
}
```

4. **Font Weight**

```
p {
  font-weight: bold;    /* normal, bold, bolder, lighter, 100-900 */
}
```

5. **Font Variant**

```
p {
  font-variant: small-caps;
}
```

➢ Units for Fonts

- **Absolute Units**: px, pt
- **Relative Units**: em (relative to parent), rem (relative to root), %, vh, vw

Example:

```
h1 {
  font-size: 2em;   /* 2 times parent's font size */
}
p {
  font-size: 1.5rem;  /* relative to root <html> font size */
}
```

➢ Using Google Fonts

```
<head>
  <link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
  <style>
    p {
      font-family: 'Roboto', sans-serif;
    }
  </style>
</head>
```

Example:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f4f4f4;
      padding: 20px;
    }
    h1 {
      text-align: center;
      text-transform: uppercase;
      text-shadow: 2px 2px 5px gray;
    }
    p {
      font-size: 18px;
      line-height: 1.6;
      letter-spacing: 1px;
      color: #333;
    }
    a {
      text-decoration: none;
      color: darkblue;
    }
    a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <h1>Text and Fonts Example</h1>
  <p>This paragraph demonstrates text properties and fonts in CSS.</p>
  <a href="#">Hover over this link</a>
</body>
</html>
```

CSS Box Model

What is the Box Model?

- Every HTML element is treated as a **rectangular box** in CSS.
- The box consists of **4 layers**:
    1. **Content** → The actual text, image, or content inside.
    2. **Padding** → Space between content and border (inside the box).
    3. **Border** → Surrounds padding and content.
    4. **Margin** → Space between this element and other elements (outside the box).

Box Model Structure

```
+-----------------------------+
|          Margin             |
|  +-----------------------+  |
|  |       Border          |  |
|  |  +-----------------+  |  |
|  |  |    Padding      |  |  |
|  |  | +-------------+ |  |  |
|  |  | |   Content   | |  |  |
|  |  | +-------------+ |  |  |
|  |  +-----------------+  |  |
```

```
|  +------------------------+  |
+-----------------------------+
```

Properties of Box Model

1. Content

```
p {
  width: 200px;
  height: 100px;
}
```

2. Padding

```
p {
  padding: 20px;          /* all sides */
  padding: 10px 15px;     /* top-bottom | left-right */
  padding: 5px 10px 15px 20px; /* top | right | bottom | left */
}
```

3. Border

```
p {
  border: 2px solid black;
  border-radius: 10px;  /* rounded corners */
}
```

4. Margin

```
p {
  margin: 20px;           /* all sides */
  margin: 10px auto;      /* top-bottom | left-right (auto = center) */
}
```

Outline vs Border

- **Border** → part of the box (affects layout).
- **Outline** → drawn outside border (does not affect layout).

```
p {
  border: 2px solid red;
  outline: 2px dashed blue;
}
```

Box-sizing

- Controls how width and height are calculated.

```
p {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: content-box; /* default (width = content only) */
}
p {
  box-sizing: border-box; /* width includes padding + border */
}
```

Display and Positioning

Display Property

The `display` property defines how an element is shown on the page.

1. Block

- Takes full width available.
- Starts on a new line.
- Examples: `<div>`, `<p>`, `<h1>`

```
div {
  display: block;
}
```

2. Inline

- Takes only required width.
- Does not start on a new line.
- Examples: `<span>`, `<a>`, `<strong>`

```
span {
  display: inline;
}
```

3. Inline-block

- Behaves like inline (same line) but allows **width & height**.

```
button {
  display: inline-block;
  width: 100px;
  height: 40px;
}
```

4. None

- Hides the element (not displayed at all).

```
p {
  display: none;
}
```

Visibility vs Display

- `display: none;` → element is removed from layout.
- `visibility: hidden;` → element is invisible but still occupies space.

Position Property

The `position` property defines how an element is placed in the document.

1. **Static** (default)

- Normal flow of the page.

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

2. Relative

- Positioned relative to its **normal position**.
- Can be moved using `top, left, right, bottom`.

```
div {
  position: relative;
  top: 20px;
  left: 30px;
}
```

3. Absolute

- Positioned relative to the **nearest positioned ancestor**.
- If no ancestor, then relative to `<html>`.

```
div {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

4. Fixed

- Positioned relative to the **viewport** (browser window).
- Does not move when scrolling.

```
nav {
  position: fixed;
  top: 0;
  width: 100%;
}
```

5. Sticky

- Acts like `relative` until a scroll point, then becomes `fixed`.

```
h1 {
  position: sticky;
  top: 10px;
}
```

Z-Index

Controls the **stack order** (which element is on top).

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

```
  z-index: 2; /* on top of box1 */
}
```

Overflow

Defines what happens when content is larger than the container.

```
div {
  width: 200px;
  height: 100px;
  overflow: hidden; /* hidden, scroll, auto, visible */
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .container {
      width: 300px;
      height: 200px;
      border: 2px solid black;
      position: relative;
    }
    .box1 {
      width: 100px;
      height: 100px;
      background: lightblue;
      position: absolute;
      top: 20px;
      left: 30px;
      z-index: 1;
    }
    .box2 {
      width: 100px;
      height: 100px;
      background: lightgreen;
      position: absolute;
      top: 40px;
      left: 50px;
      z-index: 2;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="box1">Box 1</div>
    <div class="box2">Box 2</div>
  </div>
</body>
</html>
```


Lists and Tables in CSS

Styling Lists

Lists in HTML are of two main types:

- **Ordered List (`<ol>`)** → numbered list.

- **Unordered List (`<ul>`)** → bullet list.

1. List Style Type

Defines the type of marker.

```
ul {
  list-style-type: square;  /* disc, circle, square, none */
}
ol {
  list-style-type: upper-roman; /* decimal, lower-alpha, upper-roman */
}
```

2. List Style Image

Use a custom image as a bullet.

```
ul {
  list-style-image: url("icon.png");
}
```

3. List Style Position

- `inside` → bullets appear inside content box.
- `outside` (default) → bullets outside content.

```
ul {
  list-style-position: inside;
}
```

4. Shorthand

```
ul {
  list-style: square inside url("icon.png");
}
```

Styling Tables

HTML tables use `<table>`, `<tr>`, `<td>`, `<th>` tags.

1. Borders

```
table, th, td {
  border: 1px solid black;
}
```

2. Border Collapse

```
table {
  border-collapse: collapse;  /* collapse or separate */
}
```

3. Cell Padding & Spacing

```
td {
  padding: 10px;
}
table {
  border-spacing: 15px; /* works only if border-collapse: separate */
}
```

4. Text Alignment

```
th {
  text-align: left;  /* left, right, center */
}
td {
  vertical-align: top;  /* top, middle, bottom */
}
```

5. Table Width and Height

```
table {
  width: 100%;
}
td {
  height: 50px;
}
```

6. Table Background

```
th {
  background-color: lightgray;
}
td {
  background-color: #f9f9f9;
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    /* List Example */
    ul {
      list-style-type: square;
      padding-left: 20px;
    }
    ol {
      list-style-type: upper-roman;
    }

    /* Table Example */
    table {
      width: 80%;
      margin: 20px auto;
      border-collapse: collapse;
    }
    th, td {
      border: 1px solid black;
      padding: 10px;
      text-align: center;
    }
    th {
      background: lightblue;
    }
    tr:nth-child(even) {
      background: #f2f2f2;
    }
  </style>
</head>
<body>
  <h2>Styled List</h2>
  <ul>
    <li>Apple</li>
```

```
    <li>Banana</li>
    <li>Mango</li>
  </ul>

  <h2>Styled Table</h2>
  <table>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>City</th>
    </tr>
    <tr>
      <td>John</td>
      <td>25</td>
      <td>New York</td>
    </tr>
    <tr>
      <td>Alice</td>
      <td>30</td>
      <td>London</td>
    </tr>
  </table>
</body>
</html>
```

CSS Units & Measurements

CSS units define the **size** of elements (width, height, font-size, margin, padding, etc.).

They are mainly of **two types**:

1. **Absolute Units** → fixed size (do not change with screen).
2. **Relative Units** → size depends on another value (like parent element or viewport).

Absolute Units

These units are fixed and do not change with screen size.

Unit	Description	Example
`px`	Pixels (most common, screen-based)	`font-size: 16px;`
`cm`	Centimeters	`width: 5cm;`
`mm`	Millimeters	`height: 30mm;`
`in`	Inches (1in = 96px)	`width: 2in;`
`pt`	Points (1pt = 1/72 inch)	`font-size: 12pt;`
`pc`	Picas (1pc = 12pt)	`width: 3pc;`

Note: Absolute units are rarely used in web design (except `px`), as they are not responsive.

Relative Units

These units change depending on the context (viewport, parent, or font).

Unit	Description	Example
`%`	Percentage of parent element	`width: 80%;`

Unit	Description	Example
em	Relative to element's own font-size	padding: 2em;
rem	Relative to root (html) font-size	font-size: 1.5rem;
vw	% of viewport width (1vw = 1% width)	width: 50vw;
vh	% of viewport height (1vh = 1% height)	height: 100vh;
vmin	% of smaller viewport side	font-size: 5vmin;
vmax	% of larger viewport side	font-size: 5vmax;
ch	Width of "0" character	width: 40ch;
ex	Height of lowercase "x"	line-height: 2ex;

Key Differences

- **em** → relative to **parent element** font size.
- **rem** → relative to **root element (html)** font size.
- **vw/vh** → relative to **viewport size** (responsive).
- **%** → relative to **parent container**.

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .absolute {
      font-size: 20px;  /* fixed size */
    }
    .relative-em {
      font-size: 2em;   /* twice the parent font-size */
    }
    .relative-rem {
      font-size: 2rem;  /* twice the root (html) font-size */
    }
    .viewport {
      width: 50vw;      /* 50% of viewport width */
      height: 20vh;     /* 20% of viewport height */
      background: lightblue;
    }
  </style>
</head>
<body>
  <p class="absolute">This is 20px text.</p>
  <div style="font-size: 16px;">
    <p class="relative-em">This is 2em (32px) text.</p>
  </div>
  <p class="relative-rem">This is 2rem (based on root font-size).</p>
  <div class="viewport">This box is responsive with viewport size.</div>
</body>
</html>
```

Float & Clear (Old Layout Method)

What is Float?

- The **float** property is used to move elements **to the left or right** of their container.

- Other elements will wrap around it.
- Originally created for text wrapping around images, later used for layouts (before Flexbox/Grid).

Syntax

```
selector {
  float: left | right | none | inherit;
}
```

- `left` → floats element to the left.
- `right` → floats element to the right.
- `none` → default (no floating).

Example (Image with Text)

```
<!DOCTYPE html>
<html>
<head>
  <style>
    img {
      float: left;
      margin: 10px;
      width: 150px;
    }
  </style>
</head>
<body>
  <img src="image.jpg" alt="Sample">
  <p>This text will wrap around the image because the image is floated to
the left.</p>
</body>
</html>
```

Problem with Float

- Floated elements are **removed from normal document flow**.
- Parent container may collapse if it only contains floated children.

The Clear Property

- **clear** is used to prevent elements from flowing around floated elements.
- Syntax:

```
selector {
  clear: left | right | both | none;
}
```

Example with Clear

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .box1 {
      float: left;
      width: 100px;
      height: 100px;
```

```
      background: lightblue;
    }
    .box2 {
      float: right;
      width: 100px;
      height: 100px;
      background: lightgreen;
    }
    .clear {
      clear: both;
    }
  </style>
</head>
<body>
  <div class="box1">Left</div>
  <div class="box2">Right</div>
  <div class="clear"></div>
  <p>This paragraph is below both floated boxes.</p>
</body>
</html>
```

Clearing Techniques

1. **Using `clear` element** (like `<div class="clear">`).
2. **CSS clearfix hack** (most common):

```
.container::after {
  content: "";
  display: table;
  clear: both;
}
```

When to Use Float Today?

- For **text wrapping around images**.
- Not recommended for page layouts (use **Flexbox or Grid** instead).

CSS Media Queries (Responsive Design)

What are Media Queries?

- **Media Queries** allow CSS to adapt styles based on device characteristics like **screen size, orientation, or resolution**.
- They are the foundation of **responsive web design**.

Syntax

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

- **media-type** → e.g., `screen`, `print`, `all`.
- **condition** → usually screen width (`max-width`, `min-width`).

Common Conditions

1. **Max-width**

```
@media (max-width: 600px) {
  body {
    background: lightblue;
  }
}
```

Applied only if screen width ≤ 600px.

2. Min-width

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

Applied only if screen width ≥ 768px.

3. Range (Min + Max)

```
@media (min-width: 600px) and (max-width: 1200px) {
  body {
    background: lightgreen;
  }
}
```

4. Orientation

```
@media (orientation: landscape) {
  body {
    background: yellow;
  }
}
```

5. Print Styles

```
@media print {
  body {
    color: black;
    background: white;
  }
}
```

Mobile-First Approach

Start with **small screens** first, then use `min-width` for larger devices.

Example:

```
/* Base styles (mobile first) */
body {
  font-size: 14px;
}

/* Tablet (≥ 768px) */
@media (min-width: 768px) {
  body {
    font-size: 16px;
  }
```

```
}

/* Desktop (≥ 1024px) */
@media (min-width: 1024px) {
  body {
    font-size: 18px;
  }
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background: lightgray;
      font-family: Arial;
    }
    .box {
      width: 100%;
      padding: 20px;
      text-align: center;
      background: coral;
    }

    /* Tablet */
    @media (min-width: 600px) {
      .box {
        background: lightblue;
        font-size: 20px;
      }
    }

    /* Desktop */
    @media (min-width: 1024px) {
      .box {
        background: lightgreen;
        font-size: 24px;
      }
    }
  </style>
</head>
<body>
  <div class="box">Resize the window to see changes</div>
</body>
</html>
```


Key Points

- Use **`max-width`** → desktop-first design.
- Use **`min-width`** → mobile-first design (recommended).
- Media queries make layouts responsive across devices.
- Common breakpoints:
  - `576px` → Mobile
  - `768px` → Tablet
  - `1024px` → Small Desktop
  - `1200px+` → Large Desktop

CSS Transitions & Animations

Difference

- **Transition** → Smoothly changes a property **from one value to another** (triggered by an event like hover, focus, or click).
- **Animation** → Allows multiple style changes using **keyframes**, can loop infinitely without user interaction.

CSS Transitions

Syntax

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

- **property** → which CSS property to animate (e.g., `color`, `width`, `all`).
- **duration** → how long animation takes (e.g., `1s`, `500ms`).
- **timing-function** → speed curve (`linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out`).
- **delay** → wait time before transition starts.

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .box {
      width: 100px;
      height: 100px;
      background: blue;
      transition: width 1s ease-in-out, background 0.5s;
    }
    .box:hover {
      width: 200px;
      background: red;
    }
  </style>
</head>
<body>
  <div class="box"></div>
</body>
</html>
```

On hover, width expands smoothly and color changes.

CSS Animations

Syntax

```
@keyframes animationName {
  0%   { property: value; }
  50%  { property: value; }
  100% { property: value; }
}
```

```
selector {
  animation: name duration timing-function delay iteration-count direction
fill-mode;
}
```

Properties

- **name** → keyframe name.
- **duration** → how long 1 cycle runs.
- **timing-function** → speed curve (`linear`, `ease`, etc.).
- **delay** → wait before animation starts.
- **iteration-count** → number of repeats (`1`, `infinite`).
- **direction** → play direction (`normal`, `reverse`, `alternate`).
- **fill-mode** → final state (`none`, `forwards`, `backwards`, `both`).

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .circle {
      width: 100px;
      height: 100px;
      border-radius: 50%;
      background: orange;
      position: relative;
      animation: moveCircle 3s linear infinite alternate;
    }

    @keyframes moveCircle {
      0%   { left: 0;   background: orange; }
      50%  { left: 150px; background: yellow; }
      100% { left: 300px; background: red; }
    }
  </style>
</head>
<body>
  <div class="circle"></div>
</body>
</html>
```

Circle moves left to right, changing color smoothly.

Transition vs Animation Quick Compare

Feature	Transition	Animation
Trigger	Needs event (hover, click)	Can auto-play
Keyframes	Not needed	Required
Complexity	Simple	Complex
Repetition	No loop	Can loop infinitely

Common Uses

- **Transitions** → hover effects, button animations, smooth UI changes.

- **Animations** → banners, loaders, moving objects, attention-grabbing effects.

Transform

- Transform changes an element's **shape, size, rotation, or position**.

Common Functions:

- `translate(x, y)` → move
- `rotate(deg)` → rotate
- `scale(x, y)` → resize
- `skew(x, y)` → tilt
- `matrix()` → combination of all

Example:

```
.box {
  transform: translateX(50px) rotate(45deg) scale(1.2);
}
```

CSS Variables & Custom Properties

What are CSS Variables?

- CSS Variables (also called **Custom Properties**) are **reusable values** defined once and used throughout the stylesheet.
- They make CSS **cleaner, consistent, and easier to maintain**.

Syntax

1. Defining a variable

```
:root {
  --main-color: #3498db;
  --font-size: 16px;
}
```

- Variables must start with `--`.
- Usually defined in `:root` so they are global.

2. Using a variable

```
h1 {
  color: var(--main-color);
  font-size: var(--font-size);
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    :root {
```

```
      --primary-bg: lightblue;
      --secondary-bg: coral;
      --text-color: #333;
      --padding-size: 20px;
    }

    body {
      background: var(--primary-bg);
      color: var(--text-color);
      font-family: Arial, sans-serif;
    }

    .box {
      background: var(--secondary-bg);
      padding: var(--padding-size);
      margin: 20px;
      border-radius: 8px;
    }
  </style>
</head>
<body>
  <h1>Welcome to CSS Variables</h1>
  <div class="box">This is a reusable box using CSS variables.</div>
</body>
</html>
```

Variable Scope

- **Global Scope** → Defined inside `:root`, can be used anywhere.
- **Local Scope** → Defined inside a selector, can only be used in that selector.

Example:

```
:root {
  --main-color: blue;   /* Global */
}

.card {
  --main-color: red;    /* Local (only inside .card) */
  color: var(--main-color);
}
```

Fallback Values

You can set a **default value** if the variable is not defined.

```
h1 {
  color: var(--non-existing, black);
}
```

Changing Variables with JavaScript

CSS variables can be updated dynamically using JS.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    :root {
      --main-bg: lightgreen;
```

```
    }
    body {
      background: var(--main-bg);
      transition: background 0.5s;
    }
  </style>
</head>
<body>
  <button onclick="changeTheme()">Change Theme</button>

  <script>
    function changeTheme() {
      document.documentElement.style.setProperty('--main-bg',
'lightcoral');
    }
  </script>
</body>
</html>
```

Clicking the button changes the background color.

Advantages of CSS Variables

Easy to **update and maintain**.
Helps in creating **themes** (dark mode/light mode).
Reduces code duplication.
Works well with JavaScript.

CSS Pseudo-classes & Pseudo-elements

A. Pseudo-classes

- A **pseudo-class** defines a special state of an element.
- Written using a **colon (:)**.

Common Pseudo-classes:

1. :hover → When the mouse is over an element.

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

2. :active → When an element is clicked.

```
button:active {
  transform: scale(0.95);
}
```

3. **:focus** → When an input field is focused.

```
input:focus {
  border: 2px solid green;
}
```

4. **:first-child / :last-child**

```
p:first-child { color: red; }
p:last-child { color: blue; }
```

5. **:nth-child(n)**

```
li:nth-child(2) { color: orange; } /* 2nd element */
li:nth-child(odd) { background: lightgray; }
```

B. Pseudo-elements

- A **pseudo-element** allows you to style **specific parts of an element**.
- **Written using** double colons (`::`).

Common Pseudo-elements:

1. **::first-line** → Styles the first line of text.

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

2. **::first-letter** → Styles the first letter.

```
p::first-letter {
  font-size: 30px;
  color: blue;
}
```

3. **::before** → Insert content **before** an element.

```
h1::before {
  content: "Before";
  color: green;
}
```

4. **::after** → Insert content **after** an element.

```
h1::after {
  content: "After";
  color: orange;
}
```

5. **::selection** → Styles text when highlighted.

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

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    a:hover {
      color: red;
```

```
    }

    p::first-letter {
      font-size: 24px;
      color: blue;
    }

    h2::before {
      content: "* ";
      color: gold;
    }

    h2::after {
      content: " *";
      color: gold;
    }
  </style>
</head>
<body>
  <a href="#">Hover over me</a>
  <p>This is an example of pseudo-elements.</p>
  <h2>Heading</h2>
</body>
</html>
```

Key Points

- **Pseudo-class (`:`)** → styles element states (hover, active, focus, etc.).
- **Pseudo-element (`::`)** → styles parts of an element (before, after, first-line, etc.).
- They make styling **interactive and powerful**.

Advanced CSS

CSS Functions

1. **calc()** → Do calculations directly in CSS.

```
.box {
  width: calc(100% - 50px);
}
```

2. **min()** → Chooses the **smallest value**.

```
.box {
  width: min(80%, 600px); /* whichever is smaller */
}
```

3. **max()** → Chooses the **largest value**.

```
.box {
  width: max(300px, 50%); /* whichever is larger */
}
```

4. **clamp()** → Restrict value between min and max.

```
.box {
  font-size: clamp(14px, 2vw, 20px); /* min:14, preferred:2vw, max:20 */
}
```

CSS Filters

Used to apply visual effects to images or elements.

```
img {
  filter: blur(5px) brightness(120%) contrast(150%);
}
```

- **blur(px)** → adds blur.
- **brightness(%)** → adjusts brightness.
- **contrast(%)** → adjusts contrast.
- **grayscale(%)**, **sepia(%)**, **invert(%)**, etc.

Shadows

1. Box Shadow

```
.box {
  width: 150px;
  height: 150px;
  background: lightblue;
  box-shadow: 5px 5px 15px rgba(0,0,0,0.3);
}
```

2. Text Shadow

```
h1 {
  text-shadow: 2px 2px 5px gray;
}
```

Gradients

Gradients are backgrounds that smoothly transition between colors.

1. **Linear Gradient**

```
div {
  background: linear-gradient(to right, red, yellow);
}
```

2. **Radial Gradient**

```
div {
  background: radial-gradient(circle, red, yellow, green);
}
```

3. **Conic Gradient**

```
div {
  background: conic-gradient(red, yellow, green, blue);
}
```

CSS Flexbox (Flexible Box Layout)

What is Flexbox?

- **Flexbox** is a CSS layout model that makes it easy to design **responsive layouts**.
- It arranges items in a **row** or **column**, and distributes space automatically.
- Useful for alignment, spacing, and ordering of elements.

Enable Flexbox

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

All direct child elements of `.container` become **flex items**.

Flex Container Properties

These properties are applied on the **parent (container)**.

1. **flex-direction**
   Defines main axis (row or column).

   ```
   flex-direction: row;        /* default, left → right */
   flex-direction: row-reverse; /* right → left */
   flex-direction: column;     /* top → bottom */
   flex-direction: column-reverse; /* bottom → top */
   ```

2. **justify-content**
   Aligns items along the **main axis**.

   ```
   justify-content: flex-start;   /* default */
   justify-content: flex-end;     /* items at end */
   justify-content: center;       /* center items */
   justify-content: space-between; /* equal space between */
   justify-content: space-around; /* space around each */
   justify-content: space-evenly; /* equal space around */
   ```

3. **align-items**
   Aligns items along the **cross axis**.

   ```
   align-items: stretch;   /* default */
   align-items: flex-start;
   align-items: flex-end;
   align-items: center;
   align-items: baseline;  /* aligns text baselines */
   ```

4. **align-content**
   Used when there are **multiple rows** of flex items.

   ```
   align-content: flex-start;
   align-content: flex-end;
   align-content: center;
   align-content: space-between;
   align-content: space-around;
   align-content: stretch; /* default */
   ```

5. **flex-wrap**
   Controls whether items stay in one line or wrap.

   ```
   flex-wrap: nowrap;   /* default */
   ```

```
flex-wrap: wrap;     /* items move to next line */
flex-wrap: wrap-reverse;
```

6. **gap**
   Sets space between flex items.

```
gap: 20px;
```

Flex Item Properties

These are applied on the **children (items)**.

1. **order** → changes order of items.

```
.item1 { order: 2; }
.item2 { order: 1; }
```

2. **flex-grow** → how much item grows (relative).

```
.item1 { flex-grow: 1; } /* grow twice as much as item2 */
.item2 { flex-grow: 2; }
```

3. **flex-shrink** → how much item shrinks.

```
.item1 { flex-shrink: 1; }
```

4. **flex-basis** → initial size of item before space distribution.

```
.item1 { flex-basis: 200px; }
```

5. **align-self** → overrides `align-items` for a single item.

```
.item1 { align-self: flex-start; }
```

6. **Shorthand: flex**

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

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .container {
      display: flex;
      flex-direction: row;
      justify-content: space-around;
      align-items: center;
      gap: 15px;
      border: 2px solid black;
      height: 200px;
    }
    .item {
      background: lightblue;
      padding: 20px;
```

```
      border: 1px solid blue;
    }
    .grow {
      flex-grow: 2;
    }
  </style>
</head>
<body>
  <h2>Flexbox Example</h2>
  <div class="container">
    <div class="item">Item 1</div>
    <div class="item grow">Item 2 (grows more)</div>
    <div class="item">Item 3</div>
  </div>
</body>
</html>
```

Key Points

- Flexbox works in **one dimension** (row OR column).
- Use **container properties** (`flex-direction`, `justify-content`, `align-items`) to control layout.
- Use **item properties** (`flex-grow`, `order`, `align-self`) to control individual items.
- For **2D layouts (rows + columns)** → use **CSS Grid**.

CSS Grid Layout

What is CSS Grid?

- **CSS Grid** is a 2D layout system (rows + columns).
- Unlike Flexbox (1D), Grid can handle **complex layouts** easily.
- Perfect for web page structures, dashboards, and responsive designs.

Enable Grid

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

All direct children of `.container` become **grid items**.

Grid Container Properties

These are applied on the **parent (container)**.

1. **grid-template-columns** → defines column structure.

```
grid-template-columns: 200px 200px 200px;   /* 3 fixed columns */
grid-template-columns: 1fr 2fr;             /* flexible fractions */
grid-template-columns: repeat(3, 1fr);      /* shorthand */
```

2. **grid-template-rows** → defines row structure.

```
grid-template-rows: 100px 200px;
```

3. **gap** → space between rows/columns.

```
gap: 20px;        /* row and column gap */
row-gap: 10px;
column-gap: 30px;
```

4. **justify-items** → aligns items horizontally (inside their cells).

```
justify-items: start | end | center | stretch;
```

5. **align-items** → aligns items vertically (inside their cells).

```
align-items: start | end | center | stretch;
```

6. **justify-content & align-content** → align whole grid within container.

```
justify-content: center;
align-content: space-around;
```

7. **grid-template-areas** → define layout by naming areas.

```
grid-template-areas:
  "header header"
  "sidebar main"
  "footer footer";
```

Grid Item Properties

These are applied on the **children (items)**.

1. **grid-column & grid-row** → define item position.

```
.item1 {
  grid-column: 1 / 3;  /* spans column 1 to 2 */
  grid-row: 1 / 2;
}
```

2. **grid-area** → shorthand for `row-start / column-start / row-end / column-end`.

```
.item2 {
  grid-area: 2 / 1 / 4 / 3;
}
```

3. **justify-self & align-self** → align single item inside its cell.

```
.item3 {
  justify-self: center;
  align-self: end;
}
```

Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .container {
      display: grid;
      grid-template-columns: 1fr 2fr 1fr;
      grid-template-rows: 80px 200px 80px;
```

```
      gap: 10px;
      height: 400px;
    }
    .item {
      background: lightblue;
      border: 1px solid blue;
      text-align: center;
      padding: 20px;
    }
    .header { grid-column: 1 / 4; }
    .sidebar { grid-column: 1 / 2; grid-row: 2 / 3; }
    .main { grid-column: 2 / 4; grid-row: 2 / 3; }
    .footer { grid-column: 1 / 4; }
  </style>
</head>
<body>
  <h2>Grid Example</h2>
  <div class="container">
    <div class="item header">Header</div>
    <div class="item sidebar">Sidebar</div>
    <div class="item main">Main Content</div>
    <div class="item footer">Footer</div>
  </div>
</body>
</html>
```

Key Points

- **Grid** = 2D layout (rows + columns).
- Use `fr` (fraction unit) for flexible layouts.
- Use `grid-template-areas` for named layouts.
- Use **Grid** for page-level layouts, **Flexbox** for content alignment inside.

Practical Styling

Navbar Design

```
nav {
  background: #333;
  padding: 10px;
}
nav a {
  color: white;
  text-decoration: none;
  margin: 10px;
}
nav a:hover {
  color: yellow;
}
```

Card Design

```
.card {
  width: 250px;
  padding: 20px;
  background: white;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.2);
}
```

Buttons & Hover Effects

```
.btn {
  background: blue;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  transition: 0.3s;
}
.btn:hover {
  background: darkblue;
  transform: scale(1.05);
}
```

Form Styling

```
input, textarea {
  width: 100%;
  padding: 10px;
  margin: 8px 0;
  border: 1px solid #ccc;
  border-radius: 5px;
}
input:focus {
  border-color: blue;
  outline: none;
}
```

Layouts

2-column layout

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px;
}
```

3-column layout

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 20px;
}
```

Header-Footer Layout

```
.container {
  display: grid;
  grid-template-rows: auto 1fr auto;
  height: 100vh;
}
header, footer {
  background: #333;
  color: white;
  padding: 10px;
}
```

Best Practices

Writing Clean CSS

- Use meaningful class names (`.navbar`, `.btn-primary`) instead of `.box1`, `.abc`.
- Keep indentation consistent.

Avoiding Redundancy

Bad:

```
h1 { font-size: 20px; }
h2 { font-size: 20px; }
```

Good:

```
h1, h2 { font-size: 20px; }
```

Commenting in CSS

```
/* This styles the navigation bar */
nav {
  background: #333;
}
```

External Stylesheet for Reusability

Instead of inline or internal CSS, use an external file:

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