CSS Notes

1. What is CSS?

CSS (Cascading Style Sheets) is a styling language used to describe the presentation of HTML documents. It controls layout, colors, fonts, spacing, and more.

□ **Key Features:**

- Separates content (HTML) from presentation (CSS)
- Reusability: One CSS file can style multiple pages
- Makes websites visually attractive and responsive

---

2. CSS Syntax / Format

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

Example:

```
p {
  color: blue;
  font-size: 16px;
}
```

- **Selector**: Specifies which HTML element to style (p)
- **Property**: What to style (e.g., color)
- **Value**: The value for that property (blue)

---

3. Ways to Include CSS in HTML

Type	Description	Syntax Example
Inline CSS	Style applied directly to an HTML element	<p style="color:red;">Hello</p>

Type	Description	Syntax Example
Internal CSS	CSS written inside <style> tag in <head>	<style> p {color: red;} </style>
External CSS	Separate .css file linked to HTML	<link rel="stylesheet" href="style.css">

---

4. color Property

Used to set the **text color** of an element.

```
h1 {
  color: green;
}
```

---

5. background-color Property

Sets the **background color** of an element.

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

---

6. Color Systems in CSS

1. Color Names

CSS has 140+ predefined color names like red, blue, lightgray.

```
p {
  color: crimson;
}
```

2. RGB (Red, Green, Blue)

Each color component has values from 0 to 255.

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

3. Hex Codes

A 6-digit hexadecimal code prefixed with #.

```
p {
  color: #ff0000; /* Red */
}
```

---

7. text-align Property

Aligns the text horizontally.

```
h2 {
  text-align: center; /* left | right | center | justify */
}
```

---

8. font-weight & text-decoration

- **font-weight**

Sets the boldness of the text.

```
p {
  font-weight: bold; /* normal | bold | 100–900 */
}
```

- **text-decoration**

Adds decoration to text like underline, overline, etc.

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

---

9. line-height and letter-spacing

- **line-height**

Adjusts the vertical spacing between lines of text.

```
p {
  line-height: 1.5;
}
```

- **letter-spacing**

Controls the space between letters.

```
h1 {
  letter-spacing: 2px;
}
```

---

10. Units in CSS

Unit	Description	Example
px	Pixels (fixed size)	font-size: 16px;
%	Relative to parent	width: 50%;
em	Relative to the parent element's font size	margin: 2em;
rem	Relative to the root (html) font size	padding: 1.5rem;
vh/vw	Viewport height/width	height: 50vh;

---

11. font-family Property

Specifies the font of the text.

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

- Always use a **fallback** system (Arial, sans-serif)
- Example: "Times New Roman", Georgia, serif

CSS Selectors

---

1. Universal Selector (*)

- **Syntax:** * { property: value; }
- **Description:** Selects **all elements** on the page.
- **Example:**

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

- **Use Case:** Applying a base style across all elements (reset styles).

---

2. Element Selector (Tag Selector)

- **Syntax:** element { property: value; }
- **Description:** Targets all elements of a **specific HTML tag**.
- **Example:**

```
p {
  font-size: 16px;
  color: black;
}
```

---

3. ID Selector

- **Syntax:** #idName { property: value; }
- **Description:** Targets the element with the **specific ID**.
- **Example:**

```
#main-header {
  background-color: navy;
  color: white;
}
```

- **Note:** An ID must be **unique** on a page.

---

4. Class Selector

- **Syntax:** .className { property: value; }
- **Description:** Targets **one or multiple elements** with a specific class.
- **Example:**

```
.card {
  border: 1px solid #ccc;
  padding: 10px;
  box-shadow: 2px 2px 5px gray;
}
```

---

5. Descendant Selector (Space)

- **Syntax:** parent descendant { }
- **Description:** Selects elements **inside** a specific parent element.
- **Example:**

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

- **Explanation:** Applies to all <p> tags inside a <div>, no matter how deeply nested.

---

6. Child Combinator (>)

- **Syntax:** parent > child { }
- **Description:** Selects **direct children** of a parent element.
- **Example:**

```
ul > li {
  list-style-type: square;
}
```

---

7. Adjacent Sibling Combinator (+)

- **Syntax:** element1 + element2 { }
- **Description:** Targets the **immediate next sibling**.
- **Example:**

```
h2 + p {
  color: blue;
}
```

---

8. General Sibling Combinator (~)

- **Syntax:** element1 ~ element2 { }
- **Description:** Targets **all siblings** after a specific element.
- **Example:**

```
h2 ~ p {
  color: purple;
}
```

---

9. Attribute Selector

- **Syntax Examples:**

```
input[type="text"] {
  border: 1px solid gray;
}

a[target="_blank"] {
  color: red;
}
```

- **Description:** Selects elements based on the **presence or value** of an attribute.

---

10. Pseudo Classes

- **Syntax:** selector:pseudo-class
- **Description:** Style an element **based on its state or user interaction**.
- **Common Pseudo Classes:**

```
a:hover {
  text-decoration: underline;
}

input:focus {
  border-color: blue;
}

li:first-child {
  color: green;
}

li:last-child {
  color: red;
}

p:nth-child(2n) {
  background: #eee;
}
```

---

11. Pseudo Elements

- **Syntax:** selector::pseudo-element
- **Description:** Style specific **parts of an element**.
- **Common Pseudo Elements:**

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

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

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

---

12. Cascading and Specificity

Cascading

- CSS stands for **Cascading Style Sheets**.
- If multiple rules apply, the browser decides which one to apply using **cascading order**:
    1. **Inline styles** (highest priority)
    2. **Internal/Embedded styles**
    3. **External styles**
    4. **Browser defaults**

Specificity

- Determines which rule wins when multiple rules target the same element.
- **Specificity Score Order:**

  Inline Styles > ID Selector > Class/Attribute/Pseudo-class > Element/Pseudo-element

- Example:

  ```
  #box     -> Specificity: 100
  .card    -> Specificity: 010
  div      -> Specificity: 001
  ```

---

13. !important Rule

- **Syntax:** property: value !important;
- **Description:** Overrides **all other rules**, even inline styles.
- **Example:**

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

- □ **Avoid overuse** – it breaks the natural cascade and makes debugging harder.

---

14. Inheritance in CSS

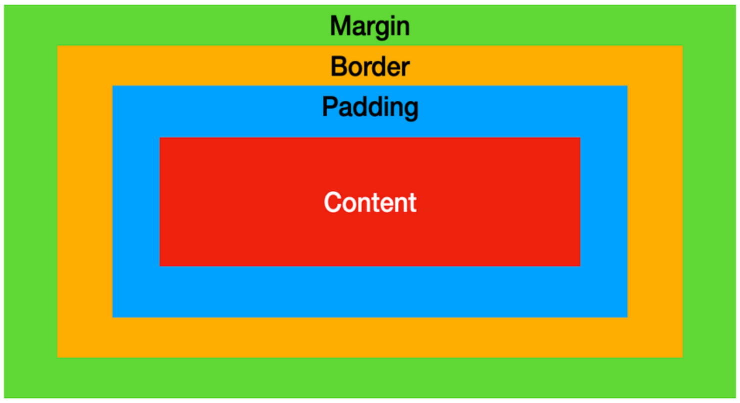
- Some CSS properties are **inherited by default** (e.g., color, font-family, line-height).
- **Non-inheritable** properties like margin, padding, border must be explicitly set.
- **Force inheritance** using inherit:

```
div {
  color: inherit;
}
```

1. Box Model

Definition:

The **CSS Box Model** is the foundation of layout and design in CSS. Every HTML element is considered as a box, composed of:



Parts of the Box Model:

- **Content**: Actual content (text, image, etc.)
- **Padding**: Space between content and border
- **Border**: Surrounds padding (optional)
- **Margin**: Space outside the border

Example:

```
div {
  margin: 10px;
  padding: 20px;
  border: 2px solid black;
}
```

---

2. Height and Width

Properties:

- width: Sets the width of an element
- height: Sets the height of an element

Units:

- px, em, rem, %

Example:

```
.box {
  width: 300px;
  height: 200px;
}
```

If box-sizing: border-box is not set, padding and border are **added to** the width/height.

---

3. Border

Syntax:

border: [width] [style] [color];

Border Styles:

- solid, dashed, dotted, double, groove, none

Shorthand:

```
div {
  border: 2px solid blue;
}
```

Individual sides:

```
border-top: 2px solid red;
border-right: 1px dashed green;
```

---

4. Border Radius

Purpose:

To create **rounded corners** on elements.

Syntax:

```
css
border-radius: [value];
```

Example:

```
button {
  border-radius: 10px;
}
```

Circular:

```
img {
  border-radius: 50%;
}
```

---

5. Padding

Purpose:

Space between content and border.

Syntax:

```
padding: 10px;              /* all sides */
padding: 10px 20px;         /* top-bottom | left-right */
padding: 10px 20px 30px 40px; /* top | right | bottom | left */
```

Example:

```
div {
  padding: 15px;
}
```

---

6. Margin

Purpose:

Space outside the border — between elements.

Syntax (same as padding):

```
margin: 10px;
margin: 10px 20px;
margin: 10px 20px 30px 40px;
```

Auto-Centering:

```
margin: 0 auto; /* center horizontally */
```

---

7. Display Property

Common Values:

Value	Description
block	Takes full width, starts on a new line
inline	Takes only needed width, no line break
inline-block	Like inline but supports height/width
none	Hides the element (not rendered)
flex	Enables flexbox layout
grid	Enables grid layout

Example:

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

---

8. Inline-block

Behavior:

- Combines features of inline and block
- Can set height and width
- Respects inline layout (flows like text)

Example:

```
span {
  display: inline-block;
  width: 100px;
  height: 50px;
}
```

---

9. Relative Units – %, em, rem

1. Percentage (%)

- Relative to **parent** element

```
div {
  width: 50%; /* 50% of parent */
}
```

2. em

- Relative to **current element's font-size**

```
p {
  font-size: 16px;
  padding: 2em; /* 32px (2 × 16px) */
}
```

3. rem

- Relative to the **root element's font-size** (usually <html>)

```
html {
  font-size: 16px;
}
h1 {
  font-size: 2rem; /* 32px */
}
```

Comparison:

Unit	Relative To	Example Use
%	Parent element	Width, height
em	Current element	Padding, font
rem	Root (html) element	Font sizing

1. Alpha & Opacity

 Opacity Property

- The opacity property sets the transparency level of an element.
- Ranges from 0 (fully transparent) to 1 (fully opaque).

```
.transparent-box {
  opacity: 0.5; /* 50% transparent */
}
```

 RGBA Color with Alpha

- RGBA = Red, Green, Blue, Alpha
- Alpha value also ranges from 0 to 1.

```
.semi-transparent-bg {
  background-color: rgba(255, 0, 0, 0.5); /* Red with 50% opacity */
}
```

---

2. Transitions in CSS

 Definition

- Transitions allow property changes in CSS values to occur smoothly over a duration.

 Basic Syntax

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

 Example

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

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

 Multiple Properties

```
.box {
  transition: width 1s, height 0.5s;
```

}

---

3. Transform in CSS

Transforms allow elements to be **visually manipulated** (rotate, scale, translate, skew).

 Syntax:

```
transform: function(value);
```

---

a) Rotate

```
.rotate-box {
  transform: rotate(45deg); /* Rotates element 45 degrees clockwise */
}
```

---

b) Scale

```
.scale-box {
  transform: scale(1.5); /* Increases size by 1.5 times */
}
```

---

c) Translate

```
.translate-box {
  transform: translate(50px, 100px); /* Moves 50px right and 100px down */
}
```

---

d) Skew

```
.skew-box {
  transform: skew(20deg, 10deg); /* Skews horizontally and vertically */
}
```

Combine Transforms:

```
.combined-box {
  transform: rotate(15deg) scale(1.2) translateX(50px);
}
```

---

4. Box Shadow

Purpose

- Adds shadow effects to elements.

Syntax

```
box-shadow: h-offset v-offset blur spread color;
```

Example

```
.shadow-box {
  box-shadow: 5px 5px 15px 0px rgba(0, 0, 0, 0.5);
}
```

Inset Shadow

```
.inset-box {
  box-shadow: inset 4px 4px 8px gray;
}
```

---

5. Background Image

Syntax

```
background-image: url("image.jpg");
```

Additional Properties

```
background-size: cover;       /* Resize image to cover entire element */
background-repeat: no-repeat; /* Prevents image repetition */
background-position: center;  /* Center the image */
```

 Example

```
.bg-container {
  background-image: url("nature.jpg");
  background-size: cover;
  background-repeat: no-repeat;
  background-position: center;
}
```

---

6. Position Property

 Types of Positioning:

Value	Description
static	Default. Element is positioned normally.
relative	Positioned relative to its normal position.
absolute	Positioned relative to the nearest positioned ancestor.
fixed	Positioned relative to the viewport.
sticky	Scrolls with the page until a threshold, then becomes fixed.

 Example

```
.static-box {
  position: static;
}

.relative-box {
  position: relative;
  top: 10px;
  left: 20px;
}

.absolute-box {
  position: absolute;
  top: 0;
  right: 0;
}

.fixed-box {
  position: fixed;
  bottom: 10px;
  right: 10px;
}

.sticky-header {
  position: sticky;
  top: 0;
}
```

CSS Flexbox

 What is Flexbox?

Flexbox (Flexible Box Layout) is a **CSS layout module** that provides a more efficient way to **align, distribute, and space elements** in a container, even when their size is unknown or dynamic.

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

1.display: flex

 This makes the container a **flex container** and its children become **flex items**.

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

Result: Items are aligned horizontally (default direction is row).

2.flex-direction

 Defines the **main axis** direction (horizontal/vertical).

Value	Description
row	Default. Left to right
row-reverse	Right to left
column	Top to bottom
column-reverse	Bottom to top

```
.container {
  flex-direction: row; /* or column, row-reverse, etc. */
}
```

---

3. justify-content

 Aligns items **along the main axis** (horizontal in row, vertical in column).

Value	Description
flex-start	Items at the start
flex-end	Items at the end
center	Items at the center
space-between	Space between items
space-around	Space around each item
space-evenly	Equal space between all items

```
.container {
  justify-content: center;
}
```

---

4. flex-wrap

 Controls whether items should **wrap** to the next line if there isn’t enough space.

Value	Description
nowrap	Default. No wrapping.
wrap	Wrap to next line.
wrap-reverse	Wrap to next line in reverse order.

```
.container {
  flex-wrap: wrap;
}
```

---

5. align-items

 Aligns items **along the cross axis** (perpendicular to main axis).

Value	Description
stretch	Default. Items stretch to fill
flex-start	Align to start of cross axis
flex-end	Align to end of cross axis
center	Align at the center
baseline	Align based on text baseline

```
.container {
  align-items: center;
}
```

6.align-content

 Aligns **multiple lines** of items (used with flex-wrap) along cross axis.

Value	Description
flex-start	Packs lines to start of cross axis
flex-end	Packs lines to end
center	Lines in the center
space-between	Even spacing between lines
space-around	Space around lines
stretch	Default. Lines stretch to fill

```
.container {
  align-content: space-between;
}
```

---

7. align-self

 Overrides align-items **for a single item**.

```
.item {
  align-self: flex-end;
}
```

 Used per item, not on the container.

8.Flex Item Sizing (flex-grow, flex-shrink, flex-basis)

 Controls how items grow/shrink.

Property	Description
flex-grow	How much an item grows (default 0)
flex-shrink	How much an item shrinks (default 1)
flex-basis	Initial size before space distribution

```
.item {
  flex-grow: 1;
  flex-shrink: 1;
  flex-basis: 100px;
}
```

---

9. flex Shorthand

 Shorthand for: flex-grow, flex-shrink, and flex-basis.

```
.item {
  flex: 1 1 100px; /* grow shrink basis */
}
```

Common Shorthands:

- flex: 1; → 1 1 0 (grow and shrink)
- flex: 0 0 auto; → Don't grow or shrink, natural size

Summary Table

Property	Axis	Applied On	Description
flex-direction	Main	Container	Direction of flex items
justify-content	Main	Container	Align items along main axis
flex-wrap	Main	Container	Wrap items when no space
align-items	Cross	Container	Align items on cross axis
align-content	Cross	Container	Align multiple lines
align-self	Cross	Flex Item	Self-alignment override
flex	Main/Cross	Flex Item	Grow, shrink, and basis combined

CSS Grid, Animations, Media Queries & Z-Index –

---

1. What is Grid in CSS?

- **CSS Grid** is a powerful layout system used to design web pages in a 2D space (rows and columns).
- Unlike Flexbox (which is 1D – row or column), Grid can handle both rows and columns simultaneously.
- Ideal for creating complex layouts like photo galleries, dashboards, and cards.

---

2. Grid Model

The **Grid Model** includes:

- A **container** (display: grid) and
- **Items** (child elements placed inside the grid).

Terminology:

Term	Description
Grid Container	The parent element with display: grid.
Grid Items	Direct children of the grid container.
Grid Lines	Dividing lines between rows and columns.
Grid Tracks	Rows and columns created between grid lines.
Grid Cells	Intersection of a row and a column.
Grid Area	A rectangular area covering one or more cells.

---

3. Grid Template

- Defines how rows and columns are created in the grid.

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

- **grid-template-columns**: Sets width for each column.
- **grid-template-rows**: Sets height for each row.

---

4. Grid Template (repeat)

- repeat() function makes template writing easier.

```
grid-template-columns: repeat(3, 1fr);
```

This creates 3 columns of equal width using the fraction unit (fr = part of available space).

You can also use:

```
grid-template-rows: repeat(2, 100px);
```

---

5. Grid Gaps

- Controls spacing **between** rows and columns.

```
grid-gap: 10px; /* shorthand for both */
row-gap: 10px;
column-gap: 15px;
```

---

6. Grid Columns

- Controls placement/size of columns.

```
grid-template-columns: 1fr 2fr;
```

- Grid item can span columns:

```
.item {
  grid-column: 1 / 3; /* spans from column line 1 to 3 */
}
```

---

7. Grid Rows

- Works like columns but vertically.

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

- Grid item spanning rows:

```
.item {
  grid-row: 1 / 3; /* spans two rows */
}
```

---

8. Grid Properties (Common)

Property	Description
display: grid	Turns container into a grid.
grid-template-rows	Defines rows’ heights.
grid-template-columns	Defines columns’ widths.
grid-gap	Space between grid items.
justify-items	Horizontal alignment of items.
align-items	Vertical alignment of items.
grid-row, grid-column	Item placement and span.
grid-area	Assigns an item to a named grid area.

---

9. Animation in CSS

CSS animations allow you to animate transitions between styles using @keyframes.

```
@keyframes slideRight {
  from {
    transform: translateX(0);
  }
  to {
    transform: translateX(100px);
  }
}
```

```
.box {
  animation-name: slideRight;
  animation-duration: 2s;
  animation-timing-function: ease-in-out;
  animation-delay: 0s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

---

10. Animation Shorthand

```
animation: slideRight 2s ease-in-out 0s infinite alternate;
```

Shorthand Order:
animation: name duration timing-function delay iteration-count direction;

---

11. Percentage (%) in Animation

- Percentages define **keyframes** stages (instead of from/to):

```
@keyframes colorChange {
  0%   { background-color: red; }
  50%  { background-color: yellow; }
  100% { background-color: green; }
}
```

Used for more fine-tuned transitions and multi-step animations.

---

12. Media Queries in CSS

Used for making **responsive designs** based on screen size or device type.

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

Common Media Features	Description
min-width / max-width	Based on screen/device width.
min-height / max-height	Based on screen height.
orientation	Detects portrait or landscape.
aspect-ratio	Based on width/height ratio.

---

13. Media Queries (Orientation)

```
@media (orientation: landscape) {
  body {
    background-color: lightgreen;
  }
}

@media (orientation: portrait) {
  body {
    background-color: lightpink;
  }
}
```

Useful for tablets and mobile views.

---

14. Z-Index

- Determines **stacking order** of overlapping elements.
- Higher z-index = placed above others.

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

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

Only works with **positioned elements** (relative, absolute, fixed, sticky).