



CSS 3



This is all about CSS

▼ Class 1: CSS Foundations

▼ What is CSS, Syntax, Inline/Internal/External



Class 1 (Deep Dive): What is CSS? + Syntax + Inline, Internal, External

1 What is CSS?

👉 Definition (Simple):

CSS stands for **Cascading Style Sheets**. It controls how HTML elements look.

👉 Definition (Deeper):

- **Cascading** → Styles can come from different places (browser default, inline, internal, external). The *cascade* decides which one wins.
- **Style** → The visual presentation: colors, fonts, spacing, layout, animations.
- **Sheet** → A file or collection of rules that the browser reads and applies.

👉 Why CSS?

- Without CSS, a website is plain (black text on white background).
- With CSS, we add **beauty + usability** (colors, alignment, responsiveness).

Analogy:

- HTML = Skeleton of the body 🦴
- CSS = Clothes, skin, and hairstyle 👤
- JS = Muscles and actions 💪

2 CSS Syntax (The Grammar of CSS)

👉 Every CSS rule looks like this:

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

- **Selector** → Which element to style (like "Hey `<p>` tags, listen!")
- **Property** → What feature of that element you want to change (like color, size, background).
- **Value** → The new style (like red, 20px, bold).

👉 Example:

```
p {
  color: red;
```

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

- Selector → `p` (all paragraphs)
- Property → `color` and `font-size`
- Values → `red` and `18px`

3 How CSS Works (Step by Step)

1. Browser loads HTML.
2. Browser loads CSS (inline, internal, or external).
3. CSS rules are matched with HTML elements.
4. Final look = HTML content + CSS style.

⚡ Behind the scenes: Browser uses a “rendering engine” (like Blink in Chrome, Gecko in Firefox) to apply CSS rules.

4 Ways to Apply CSS

There are **3 main methods**:

(A) Inline CSS

- Style is written **inside the HTML tag itself** using the `style` attribute.
- Fast, good for testing.
- But **not reusable** (if you want to change color for 100 paragraphs, you must edit all 100 tags 😓).

👉 Example:

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

⚡ Inline = “one-time use, quick, but messy for big projects.”

(B) Internal CSS

- CSS is written inside a `<style>` block in the `<head>`.
- Styles apply to the whole page.
- Easier than inline, but still not reusable across multiple pages.

👉 Example:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 { color: green; font-size: 40px; }
    p { color: gray; font-size: 20px; }
  </style>
</head>
<body>
  <h1>Welcome to My Page</h1>
  <p>This text is styled with Internal CSS.</p>
</body>
</html>
```

⚡ Internal = “page-level styling, neat but limited.”

(C) External CSS

- CSS is written in a **separate .css file**.
- Linked to HTML using `<link rel="stylesheet">`.
- **Best practice** → clean, reusable, scalable.
- Same stylesheet can be used for **multiple pages** → change once, all pages update.

👉 Example:

HTML file (`index.html`):

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Hello World</h1>
  <p>This text is styled with External CSS.</p>
</body>
</html>
```

CSS file (`style.css`):

```
h1 { color: purple; font-size: 40px; }
p { color: darkgray; font-size: 20px; }
```

⚡ External = “real-world standard, professional way.”

5 CSS Cascade & Priority

If all 3 are used at once, who wins?

- **Inline CSS** → Highest priority.
- **Internal CSS** → Next.
- **External CSS** → Lowest priority.
- (Unless we use `!important`, which overrides all — but don't overuse it.)

👉 Example:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="style.css">
  <style>
```

```

    p { color: blue; }
  </style>
</head>
<body>
  <p style="color: red;">This is a test.</p>
</body>
</html>

```

- External → maybe `p { color: green; }`
- Internal → `p { color: blue; }`
- Inline → `style="color: red;"`

👑 Final result = **Red** (because Inline wins).

6 Quick Comparison Table

Type	Where Written	Use Case	Disadvantage
Inline	Inside tag (<code>style=""</code>)	Quick test	Not reusable, messy
Internal	<code><style></code> in <code><head></code>	Small websites, single page	Not reusable across pages
External	Separate file <code>.css</code>	Professional projects	Needs extra file, but worth it

🎯 Activities for Students

1. **Inline Test** → Write `<h1>` with inline CSS to make text red and big.
2. **Internal Test** → Write a `<style>` block to make all `<p>` text blue.
3. **External Test** → Create `style.css`, link it, and make background yellow.
4. **Experiment with Cascade** → Use all three at once, see which wins.

▼ Selectors (basic + advanced)

CSS Selectors (Basic + Advanced)

1 What are CSS Selectors?

- A **selector** tells CSS *which HTML element(s)* you want to style.
- Example: If HTML is a classroom of students, selector is like “*All students wearing blue shirt stand up!*” → CSS applies to only those.

👉 Syntax Reminder:

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

2 Basic Selectors

(A) Universal Selector ()

- Selects **all elements** on the page.

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

👉 Often used to “reset” default browser styles.

(B) Element Selector

- Selects all elements of a given type.

```
p { color: blue; }  
h1 { font-size: 40px; }
```

👉 Styles all `<p>` or `<h1>` elements.

(C) Class Selector (`.classname`)

- Selects elements with a **class attribute**.

```
<p class="note">This is a note.</p>
```

```
.note {  
  color: green;  
  font-style: italic;  
}
```

👉 Multiple elements can share the same class.

(D) ID Selector (`#idname`)

- Selects element with a **unique id**.

```
<p id="special">This is special.</p>
```

```
#special {  
  color: red;  
  font-weight: bold;  
}
```

👉 IDs should be **unique** per page.

(E) Group Selector (`,`)

- Apply the same style to multiple selectors.

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

3 Advanced Selectors

(A) Descendant Selector (**A B**)

- Selects elements **inside** another element.

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

👉 "All `<p>` inside `<div>`."

(B) Child Selector (**A > B**)

- Selects **direct children only**.

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

👉 "Only `<p>` that are *directly inside* `<div>`."

(C) Attribute Selector

- Select elements based on attributes.

```
input[type="text"] {  
  border: 1px solid black;  
}  
a[target="_blank"] {  
  color: red;  
}
```

4 Pseudo-classes

A **pseudo-class** is a keyword added to a CSS selector that specifies a special state of the selected element. It allows you to style elements based on factors other than just their place in the document tree, such as user interaction, element position, or form states.

Common Examples

Pseudo-classes are very useful for creating dynamic and interactive web pages. Here are a few common examples:

- `:hover` : This is one of the most widely used pseudo-classes. It applies styles to an element when the user's mouse pointer hovers over it. This is great for creating visual feedback on buttons or links.
- `:first-child` : This selects the first element among a group of siblings. For example, you can use it to make the first item in a list bold.
- `:focus` : This applies styles to an element that has received focus, such as a text input field that has been clicked on or tabbed into by the user. It is essential for accessibility.
- `:active` : This selects an element while it is being activated by the user, such as when a mouse button is being held down on a link or button.
- Used to style elements in a **specific state**.

```
<!DOCTYPE html>
<html>
<head>
<title>CSS Selectors Example</title>
<style>
  /* This CSS rule makes the text color of a link turn orange when you h
over over it. */
  a:hover {
    color: orange;
  }

  /* This CSS rule makes the very first paragraph in its parent element b
old. */
  p:first-child {
    font-weight: bold;
  }
</style>
</head>
```

```
<body>

<div>
  <p>This is the first paragraph inside the div, so it will be bold.</p>
  <p>This is the second paragraph, so it will not be bold.</p>
</div>

<p>This paragraph is the first child of the body, so it will also be bold.
</p>

<a href="#">Hover over this link</a>

</body>
</html>
```

5 Pseudo-elements

A **pseudo-element** is a CSS keyword that allows you to style a specific, non-existent part of an element. It lets you create and style content that isn't actually in your HTML code.

The syntax for a pseudo-element uses a double colon (`::`). This is a modern convention to distinguish them from pseudo-classes, which use a single colon (`:`).

Common Pseudo-Elements

- `::before`: Creates a virtual element that is the **first child** of the selected element. It's often used to add decorative content, like an icon or a special character, before the element's actual content. To use `::before` or `::after`, you must include the `content` property in your CSS rule, even if it's empty.
- `::after`: Similar to `::before`, this creates a virtual element that is the **last child** of the selected element. It's used to add content after the element's main content.

- `::first-line` : Selects and styles the **first line of a block-level element** like a paragraph (`<p>`). The styling automatically adjusts if the user resizes the browser window.
- `::first-letter` : Selects and styles the **first letter of a block-level element**. This is commonly used to create a "drop cap" effect for a typographic style.
- `::selection` : Applies a style to the portion of an element that the user has **highlighted with their mouse**. It's perfect for customizing the look of selected text.

How They Differ from Pseudo-Classes

While both pseudo-classes and pseudo-elements add special effects to selectors, they do different things:

- **Pseudo-classes** (`:hover` , `:first-child`) target an element based on its **state** or its position relative to other elements. They style the existing element itself.
- **Pseudo-elements** (`::before` , `::first-line`) target a **part of an element** or even create a new, virtual element.

```
<!DOCTYPE html>
<html>
<head>
<title>Pseudo-elements Example</title>
<style>
p::first-line {
  font-size: 20px;
}

p::before {
  content: "👉 ";
}
</style>
</head>
```

```
<body>
```

```
<p>This is a paragraph. The first line of this text will be larger, and a pointing hand emoji will appear before the entire paragraph.</p>
```

```
</body>
```

```
</html>
```

6 CSS Specificity & Priority (Mini Intro)

- Inline styles > ID > Class > Element.
- More specific selector wins.

Example:

```
p { color: blue; } /* lowest */  
#para { color: red; } /* wins */
```

Student Activities

1. Create an HTML page with 3 paragraphs.
 - First one → style with **element selector**.
 - Second → style with **class selector**.
 - Third → style with **id selector**.
2. Create a `<div>` with 3 `<p>` inside it.
 - Use **descendant** and **child selector** to style them differently.
3. Create a form with `<input type="text">` and `<input type="password">`.
 - Style them using **attribute selector**.
4. Add an `<a>` link. Style `:hover` state to change color.

▼ Colors, Backgrounds, Gradients

CSS Colors, Backgrounds & Gradients — Deep Understanding

1 Colors in CSS — Why and How

CSS colors aren't just decoration — they **communicate meaning** and help **user experience**. Understanding different formats helps students control color precisely.

(A) Named Colors

- Simple: `red`, `blue`, `green`
- **Use:** Quick, easy to read code, good for beginners.
- **Limitation:** Only ~140 colors available → limited palette.

```
p { color: red; }
```

(B) HEX Colors

- Format: `#RRGGBB` (RR = red, GG = green, BB = blue, in hexadecimal 00–FF)
- **Why use HEX?**
 - Precise control of colors
 - Widely used in design tools
- **Example:**

```
p { color: #ff4500; } /* orange-red */
```

Explain:

- FF → 255, 45 → 69, 00 → 0 → RGB(255,69,0)
 - Helps students understand RGB mix in HEX.
-

(C) RGB / RGBA

- Format: `rgb(red, green, blue)`
- **RGBA** adds alpha channel → transparency (0 = invisible, 1 = opaque)

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

Why important:

- Helps with **overlapping content**
- Enables **layered UI design** (buttons, cards, modals)

Tip: Show **opacity vs. background** — demo overlapping divs with transparency.

(D) HSL / HSLA

HSL and HSLA are color models used in web development that are more intuitive for designers and developers to work with than RGB or hexadecimal codes. They are based on how humans perceive color.

HSL

HSL stands for **Hue, Saturation, and Lightness**. It uses a cylindrical color model, where colors are defined by three components:

- **Hue (H):** This is the actual color, like red, green, or blue. It is represented as a degree on a color wheel, ranging from 0 to 360. For example, 0 is red, 120 is green, and 240 is blue.
- **Saturation (S):** This controls the intensity or vividness of the color. It is a percentage from 0% to 100%. 0% is a shade of gray, and 100% is the full, most vibrant color.
- **Lightness (L):** This controls the brightness of the color. It is also a percentage from 0% to 100%. 0% is black, 100% is white, and 50% is the true hue of the color.

HSLA


```

/* HSL(Hue, Saturation, Lightness) */
.red-box {
  background-color: hsl(0, 100%, 50%); /* Pure Red */
}

.green-box {
  background-color: hsl(120, 100%, 50%); /* Pure Green */
}

.blue-box {
  background-color: hsl(240, 100%, 50%); /* Pure Blue */
}
</style>
</head>
<body>

<h1>HSL Colors</h1>

<div class="box red-box">Red</div>
<div class="box green-box">Green</div>
<div class="box blue-box">Blue</div>

</body>
</html>

```

2 Backgrounds — Why we care

Backgrounds set **context** and **visual hierarchy**. Students must understand how to combine **color, images, size, and position**.

(A) Background Color

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

- Gives **visual block separation**
- Helps **readability** of content

(B) Background Image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello, World!</title>
    <style>
      div {
        background-image: url('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR4eRjo2jggguCB0t2pGlboOI9xiv0UPOCFRkQ&s');
        background-size: cover;
        background-repeat: no-repeat;
        background-position: center;
        width: 100vw;
        height: 100vh;
      }
    </style>
  </head>
  <body>
    <div></div>
  </body>
</html>
```

Explain each property:

- **cover** → fills entire div without stretching
- **contain** → fits inside div, may leave gaps
- **repeat** → tiles image
- **position** → controls where image is anchored

Tip: Show bad examples (stretching, repeated images) → students understand why each property exists.

3 Gradients — Why they exist

Gradients = **smooth transition between colors**.

- Why better than images?
 - No extra HTTP request
 - Scales to any size (responsive-friendly)
 - Dynamic design → combine colors easily

(A) Linear Gradient

```
<!DOCTYPE html>
<html>
<head>
<style>
  div { background: linear-gradient(to right, red, yellow); }
</style>
</head>
<body>
  <div>HSL Colors</div>
</body>
</html>
```

- Direction matters: `to right`, `to bottom`, `45deg`
- Can combine multiple stops:

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

(B) Radial Gradient

```
div { background: radial-gradient(circle, blue, lightblue); }
```

- Center → edges color transition
- **Use case:** Buttons, badges, background highlights

(C) Transparency with Gradients

```
div {  
  background: linear-gradient(to right, rgba(255,0,0,0.5), rgba(0,0,255,  
  0.5));  
}
```

- Enables **overlay effects** without separate images
- Good for **hero sections with text over gradient**

4 Important Notes

1. Why multiple color systems?

- Beginner friendly → named colors
- Designer precision → HEX / RGB
- Logical adjustments → HSL

2. Why backgrounds matter?

- Sets visual hierarchy → important sections stand out
- Good background improves **UX and accessibility**

3. Why gradients?

- Modern UI uses gradients instead of flat color → depth & realism
- Students can see **dynamic effects** with simple CSS

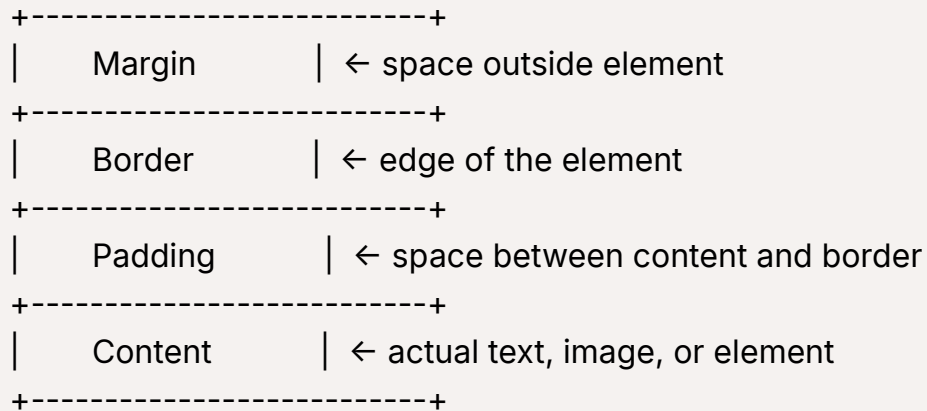
▼ Box Model (Margin, Padding, Border, box-sizing)

CSS Box Model — Complete Deep Dive

1 Overview

Every HTML element is a **rectangular box**. The CSS box model defines **how the size of an element is calculated** and how **spacing around it works**.

Box model components (from inside out):



2 Content

- This is the **actual content** of the element: text, image, or other HTML content.
- Width and height CSS properties affect this area by default.

```
div {  
  width: 200px;  
  height: 100px;  
  background-color: lightblue;  
}
```

3 Padding

- **Padding** is the space between **content** and **border**.
- Adds **internal spacing**.

```
div {  
  padding: 20px; /* 20px space inside the box */  
  background-color: lightgreen;  
}
```

- Padding **affects the total box size** when using `content-box` sizing.
-

4 Border

- **Border** is the edge of the box.
- Can have **width, style, and color**.

```
div {  
  border: 5px solid red;  
}
```

- Border styles: `solid`, `dashed`, `dotted`, `double`, `groove`, `ridge`, `none`.
 - Border adds to the **total element size** when using `content-box`.
-

5 Margin

- **Margin** is the space **outside the border**, pushing **other elements away**.

```
div {  
  margin: 30px; /* space outside the box */  
}
```

- Margins can be individual: `margin-top`, `margin-right`, `margin-bottom`, `margin-left`.
 - **Vertical margins collapse** if adjacent elements have margin.
-

6 Box-Sizing

By default, width and height are applied to **content only**.

width = content width
total width = content + padding + border

Example:

```
div {  
  width: 200px;  
  padding: 20px;  
  border: 5px solid black;  
  box-sizing: content-box;  
}
```

- Total width = 200 + 202 + 52 = 250px

border-box option:

```
div {  
  box-sizing: border-box;  
  width: 200px;  
  padding: 20px;  
  border: 5px solid black;  
}
```

- Total width = 200px
- Padding and border **do not increase total width**
- Simplifies layout calculations for complex designs

7 Background and Padding

- Background color or image **applies to content + padding**, but **not to margin**.
 - Borders **appear around padding**, visually separating the element from surroundings.
-

8 Collapsing Margins

- **Vertical margins** between two adjacent elements **merge into one**, taking the larger value.
- Horizontal margins **do not collapse**.

```
p { margin-bottom: 20px; }  
p + p { margin-top: 30px; }  
/* Actual space between paragraphs = 30px (larger margin) */
```

9 Practical Examples

Example 1 — Content-box

```
<div class="content-box-example">Content-box div</div>
```

```
.content-box-example {  
  width: 200px;  
  height: 100px;  
  padding: 20px;  
  border: 5px solid red;  
  margin: 30px;  
  background-color: lightblue;  
  box-sizing: content-box;  
}
```

- Total width = 200 + 40 + 10 = 250px
- Total height = 100 + 40 + 10 = 150px

Example 2 — Border-box

```
<div class="border-box-example">Border-box div</div>
```



```
.border-box-example {  
  width: 200px;  
  height: 100px;  
  padding: 20px;  
  border: 5px solid green;  
  margin: 30px;  
  background-color: lightyellow;  
  box-sizing: border-box;  
}
```

- Total width = 200px (content shrinks automatically)
- Total height = 100px
- Easier to **align multiple boxes** in layouts

10 Key Insights

1. **Content-box** vs **Border-box**: `border-box` simplifies layout calculations, preventing overflow when padding and border are added.
2. **Padding** is internal spacing, **margin** is external spacing.
3. **Borders** add thickness but are part of the visual box; combined with padding and content, they determine total element size in `content-box`.
4. **Backgrounds** cover **content + padding** but never margin.
5. **Collapsing margins** reduce extra vertical spacing between elements.

▼ Text & Fonts



CSS Text & Fonts — In-Depth

1 Font Family

- **Font-family** defines which font is used for an element.
- Can list **multiple fonts as fallback** in case first one is unavailable.

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

Explanation:

- "Arial" → first choice
- "Helvetica" → fallback if Arial not available
- sans-serif → generic fallback
- Generic families: serif, sans-serif, monospace, cursive, fantasy, system-ui

2 Font Size

- Defines the size of text. Can use **absolute units** or **relative units**.

```
p {  
  font-size: 16px; /* absolute */  
  font-size: 1.5em; /* relative to parent element */  
  font-size: 120%; /* relative percentage */  
}
```

Unit notes:

- px → pixels, fixed
- em → relative to parent font size
- rem → relative to root <html> font size
- % → relative to parent font size

3 Font Weight

- Controls **boldness** of text.

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

- 100 → Thin
- 400 → Normal
- 700 → Bold
- 900 → Extra bold

4 Font Style

- Controls **slant of text**

```
p { font-style: normal; }  
p.italic { font-style: italic; }  
p.oblique { font-style: oblique; }
```

5 Text Transform

- Controls **uppercase, lowercase, capitalization**

```
p.uppercase { text-transform: uppercase; }  
p.lowercase { text-transform: lowercase; }  
p.capitalize { text-transform: capitalize; }
```

6 Text Decoration

- Adds **underline, line-through, overline, or none**

```
a { text-decoration: none; }  
p.strike { text-decoration: line-through; }
```

```
p.underline { text-decoration: underline; }  
p.overline { text-decoration: overline; }
```

7 Text Align

- Controls **horizontal alignment** of text

```
p { text-align: left; }  
p.center { text-align: center; }  
p.right { text-align: right; }  
p.justify { text-align: justify; }
```

- `justify` → aligns text evenly along left & right edges

8 Line Height & Letter Spacing

- Line-height** → space between lines

```
p { line-height: 1.5; } /* 1.5x default line spacing */
```

- Letter-spacing** → space between characters

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

- Both improve readability and aesthetics.

9 Word Spacing & Text Indent

```
p { word-spacing: 5px; } /* space between words */  
p { text-indent: 30px; } /* first line indentation */
```

10 Font Variants & Advanced Properties

- **Small caps:**

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

- **Google Fonts / External fonts:**

```
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
```

```
body {  
  font-family: 'Roboto', sans-serif;
```

1 1 Demo HTML

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>CSS Text & Fonts</title>  
  <style>  
    body { font-family: Arial, sans-serif; padding: 1rem; }  
  
    h1 { font-size: 36px; font-weight: 700; text-align: center; }  
  
    p.normal { font-weight: normal; font-style: normal; }  
    p.bold { font-weight: bold; }  
    p.italic { font-style: italic; }  
    p.oblique { font-style: oblique; }  
    p.uppercase { text-transform: uppercase; }  
    p.capitalize { text-transform: capitalize; }  
    p.underline { text-decoration: underline; }  
    p.strike { text-decoration: line-through; }  
    p.center { text-align: center; }  
    p.justify { text-align: justify; }
```

```

p.spacing { letter-spacing: 2px; word-spacing: 5px; line-height: 1.8; }

.google-font {
  font-family: 'Roboto', sans-serif;
  font-weight: 700;
}
</style>
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@
400;700&display=swap" rel="stylesheet">
</head>
<body>

<h1>CSS Text & Fonts Demo</h1>

<p class="normal">Normal text</p>
<p class="bold">Bold text</p>
<p class="italic">Italic text</p>
<p class="oblique">Oblique text</p>

<p class="uppercase">Uppercase text</p>
<p class="capitalize">capitalize each word</p>
<p class="underline">Underline text</p>
<p class="strike">Strike-through text</p>

<p class="center">Centered text</p>
<p class="justify">Justified text for paragraph. Lorem ipsum dolor sit a
met, consectetur adipiscing elit. Vivamus laoreet.</p>

<p class="spacing">Letter-spacing, word-spacing, line-height demons
tration.</p>

<p class="google-font">Google Font: Roboto Bold</p>

```

```
</body>  
</html>
```

1 2 Key Insights

1. **Font-family order** ensures fallback if a font isn't available.
2. **Relative units** (`em` , `rem`) make text scalable and responsive.
3. **Line-height & spacing** improve readability, crucial for design.
4. **Text-transform & text-decoration** are visual tools, not semantic changes.
5. **Google Fonts** or custom fonts allow unique typography beyond system defaults.

▼ ● Mini Project



Mini Project: "Profile Card Dashboard"

Project Question / Task

Goal:

Create a **Profile Card Dashboard** displaying multiple user profiles. Each card should include:

1. **User Image** – circular photo
2. **Name & Designation** – styled text
3. **Short Bio** – styled paragraph
4. **Skills / Tags** – small badges
5. **Links / Buttons** – styled anchor tags

Requirements:

1. Use **HTML elements** and **CSS selectors** to style each card.

2. Apply **colors, backgrounds, and gradients** to page and cards.
3. Use **box model properties**: margin, padding, border, border-radius, box-sizing.
4. Style **text and fonts**: font-family, font-size, font-weight, text-align, line-height.
5. Arrange cards **side by side** using `inline-block` or `block` layout.
6. Include **hover effects** on cards or links.

Optional Enhancements:

- Change background gradients for cards or page
- Add multiple cards with different content
- Adjust padding/margin/border to see box model in action

▼ Answer

▼ HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Profile Card Dashboard</title>
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <h1>Profile Card Dashboard</h1>

  <div class="dashboard">
    <div class="card">
      <img src="https://via.placeholder.com/100" alt="User Photo" cl
```



```

    ass="profile-img">
      <h2 class="name">John Doe</h2>
      <p class="designation">Frontend Developer</p>
      <p class="bio">Passionate about UI/UX design and modern we
b technologies. Loves creating interactive web experiences.</p>
      <div class="skills">
        <span>HTML</span>
        <span>CSS</span>
        <span>JavaScript</span>
      </div>
      <div class="links">
        <a href="#">LinkedIn</a>
        <a href="#">GitHub</a>
      </div>
    </div>

    <div class="card">
      
      <h2 class="name">Jane Smith</h2>
      <p class="designation">UI Designer</p>
      <p class="bio">Focused on clean and creative web designs. En
joys turning ideas into beautiful interfaces.</p>
      <div class="skills">
        <span>Photoshop</span>
        <span>Figma</span>
        <span>CSS</span>
      </div>
      <div class="links">
        <a href="#">LinkedIn</a>
        <a href="#">Portfolio</a>
      </div>
    </div>
  </div>

```

```
</body>  
</html>
```

▼ CSS

```
/* Global Styles */  
body {  
  font-family: 'Roboto', sans-serif;  
  background: linear-gradient(to right, #74ebd5, #ACB6E5);  
  margin: 0;  
  padding: 20px;  
  text-align: center;  
}  
  
h1 {  
  margin-bottom: 30px;  
}  
  
/* Dashboard Layout */  
.dashboard {  
  text-align: center;  
}  
  
.card {  
  display: inline-block;  
  vertical-align: top;  
  width: 250px;  
  padding: 20px;  
  margin: 15px;  
  background-color: white;  
  border-radius: 15px;  
  box-shadow: 0 6px 15px rgba(0,0,0,0.2);  
  box-sizing: border-box;  
  text-align: center;  
  transition: transform 0.3s, box-shadow 0.3s;
```

```

}

.card:hover {
  transform: translateY(-5px);
  box-shadow: 0 10px 20px rgba(0,0,0,0.3);
}

/* Profile Image */
.profile-img {
  width: 100px;
  height: 100px;
  border-radius: 50%;
  margin-bottom: 10px;
}

/* Text Styling */
.name {
  font-size: 1.5rem;
  font-weight: 700;
  margin: 5px 0;
}

.designation {
  font-style: italic;
  color: #555;
  margin-bottom: 10px;
}

.bio {
  font-size: 0.9rem;
  line-height: 1.5;
  text-align: justify;
  margin-bottom: 15px;
}

/* Skills */

```

```

.skills span {
  display: inline-block;
  background-color: #ACB6E5;
  color: white;
  padding: 5px 10px;
  border-radius: 12px;
  margin: 2px;
  font-size: 0.8rem;
}

/* Links / Buttons */
.links a {
  display: inline-block;
  margin: 5px;
  text-decoration: none;
  color: white;
  background: #74ebd5;
  padding: 5px 10px;
  border-radius: 8px;
  transition: background 0.3s;
}

.links a:hover {
  background: #ACB6E5;
}

```

▼ Class 2: Layout & Positioning

▼ Display: block, inline, inline-block, none



1. display: block

👉 Concept:

- A block element **takes the full width available** (100% of its parent).
- Always **starts on a new line**.

- You can set **width, height, margin, padding**.

👉 **Examples of block elements by default:**

`<div>` , `<p>` , `<h1>` - `<h6>` , `<section>` , `<article>` .

👉 **Analogy:**

Imagine a **sofa** in a room – it will **take the whole wall space** and push the next furniture (element) below it.

👉 **Code:**

```
<div style="background: lightblue; display: block;">
  I am block - I take the full width
</div>
<p style="background: pink;">I am also block</p>
```

2. **display: inline**

👉 **Concept:**

- Inline elements **do not start on a new line**.
- They take only the **space needed by content**.
- You **cannot set width & height** (only left/right padding, margin works).

👉 **Examples of inline elements by default:**

`` , `<a>` , `` , `` .

👉 **Analogy:**

Think of **words in a sentence** – they just line up next to each other, no forced breaks.

👉 **Code:**

```
<span style="background: yellow;">Inline 1</span>
<span style="background: orange;">Inline 2</span>
<a href="#">Inline link</a>
```

3. `display: inline-block`

👉 Concept:

- Behaves like **inline** (sits next to each other).
- But also behaves like **block** (you can set width, height, margin, padding fully).
- Best of both worlds.

👉 Examples:

Buttons, navigation links, product cards often use `inline-block`.

👉 Analogy:

Think of **books on a shelf** – they sit next to each other (inline), but each book has its **own height/width**.

👉 Code:

```
<span style="display: inline-block; width: 100px; height: 50px; background: lightgreen;">
  Box 1
</span>
<span style="display: inline-block; width: 120px; height: 50px; background: lightcoral;">
  Box 2
</span>
```

4. `display: none`

👉 Concept:

- Completely removes the element from the **document flow**.
- It's like the element **doesn't exist** (not visible, no space reserved).

👉 Important:

This is **different from** `visibility: hidden` → hidden keeps the space, `none` removes it.

👉 Analogy:

Like **removing a chair from the room** (none). While `visibility: hidden` = the chair is invisible but still blocking the space.

👉 Code:

```
<p>This is visible</p>
<p style="display: none;">I am hidden (no space taken)</p>
<p>This comes right after</p>
```

🔥 Quick Comparison Table

Display Type	Starts New Line?	Can Set Width/Height?	Space Taken
block	✅ Yes	✅ Yes	Full width
inline	❌ No	❌ No	Content only
inline-block	❌ No	✅ Yes	Content size
none	❌ No	❌ No	No space



Student Task

👉 Make a small **Profile Card UI** that uses all 4 display types.

Requirements:

1. A **heading** (`block`)
2. Some **inline text** (`inline`) like *tags* or *hashtags*
3. A **button** (`inline-block`) with fixed size
4. A hidden **note** (`display: none`)

▼ Answer

```
<!DOCTYPE html>
<html>
<head>
  <title>Profile Card</title>
```

```
<style>
.card {
  width: 300px;
  padding: 20px;
  border: 2px solid #ccc;
  border-radius: 10px;
  font-family: Arial, sans-serif;
  background: #f9f9f9;
}

/* 1. Heading → block */
.card h2 {
  display: block;
  text-align: center;
  margin-bottom: 10px;
  background: lightblue;
  padding: 5px;
}

/* 2. Inline text → hashtags */
.tags span {
  display: inline;
  background: #ffe680;
  margin-right: 5px;
}

/* 3. Button → inline-block */
.btn {
  display: inline-block;
  background: #4CAF50;
  color: white;
  padding: 8px 15px;
  border-radius: 5px;
  margin-top: 15px;
  text-decoration: none;
}
```



```

/* 4. Hidden note → display none */
.hidden-note {
  display: none;
  margin-top: 10px;
  color: red;
  font-size: 14px;
}
</style>
</head>
<body>
  <div class="card">
    <!-- Block →
    <h2>John Doe</h2>

    <!-- Inline →
    <p class="tags">
      <span>#Developer</span>
      <span>#Designer</span>
      <span>#Learner</span>
    </p>

    <!-- Inline-block →
    <a href="#" class="btn">Follow</a>

    <!-- None →
    <p class="hidden-note">This is a hidden note (students won't see it).</p>
  </div>
</body>
</html>

```

▼ Overflow & visibility

Overflow & Visibility

These control **how content is displayed** when it doesn't fit inside its container.

1. Overflow

👉 Definition:

Decides **what happens when content is bigger than the box** (container) size.

Values of **overflow** :

1. visible (default)

- Content **overflows outside** the box.
- No clipping.

```
<div style="width:150px; height:50px; border:2px solid; overflow:visible;">  
  This text is very long and will overflow outside the box.  
</div>
```

2. hidden

- Extra content is **cut off** (not visible).
- No scrollbars.

```
<div style="width:150px; height:50px; border:2px solid; overflow:hidden;">  
  This text is too long and will be cut off.  
  Hi my name is something that will cut off shitt  
</div>
```

3. scroll

- Always adds **scrollbars** (both horizontal & vertical).
- Even if content fits, scrollbar will still show.

```
<div style="width:150px; height:50px; border:2px solid; overflow:scroll;">
```

This text is long and you can scroll to see it.

How to see scroll man oh come on

```
</div>
```

4. auto

- Scrollbars appear **only if needed**.
- Best choice in most real cases.

```
<div style="width:150px; height:50px; border:2px solid; overflow:auto;">
```

This text is long and scrollbars will appear only if needed.

```
</div>
```

2. Visibility

👉 Definition:

Controls whether an element is **visible or hidden**, but **still takes up space**.

Values of **visibility** :

1. visible (default)

- Element is shown normally.

```
<p style="visibility:visible;">I am visible</p>
```

2. hidden

- Element is invisible but **still occupies space**.

```
<p style="visibility:hidden;">I am hidden but my space is reserved</p>
```

Difference Between `display: none` vs `visibility: hidden`

- `display: none` → removes the element completely (no space).
- `visibility: hidden` → makes it invisible but keeps the space.

👉 Example:

```
<p style="display:none;">Gone completely (no space)</p>
<p style="visibility:hidden;">Invisible but still here (space taken)</p>
```

Student Task

👉 Create a **Text Box UI** with these requirements:

1. A fixed-size `div` (200px × 100px) with **overflow: visible** → text should spill out.
2. Another with **overflow: hidden** → extra text is cut.
3. Another with **overflow: scroll** → scrollbars always appear.
4. A button below the text → when clicked, it toggles between **visibility: visible** and **visibility: hidden**.

▼ **Position: static, relative, absolute, fixed, sticky**

CSS `position` – Complete Beginner-Friendly Deep Dive

Think of a **webpage as a big room** 🏠, and elements (`div`, `p`, `img`) as **furniture** inside that room.

The `position` property tells **where each piece of furniture should sit**.

1. `static` – Default, “Normal Seating”

- Every element is **static by default**.

- Elements appear **one after another** in the normal document flow.
- Properties like `top`, `left`, `bottom`, `right` → **don't work** here.

📖 Analogy:

Students in a classroom sit in their **roll number order**. No shifting, no jumping.

✅ Use case:

When you don't need any special positioning.

```
<p style="position: static; border: 2px solid blue;">
  I am static. I sit where I am naturally supposed to.
</p>
```

2. **relative** – “Still on the Seat, Just Shifted a Bit”

- The element stays in the **normal flow** (space is reserved).
- But you can **move it slightly** using `top`, `left`, `right`, `bottom`.
- The original space is still kept as if the element was not moved.

📖 Analogy:

A student is sitting in their seat, but **slides a little** to the corner.

The seat is still reserved for them.

✅ Use case:

- When you just want to nudge something a bit.
- To make the parent an “anchor” for absolute children.

```
<div style="position: relative; top: 20px; left: 30px; border: 2px solid green;">
  I moved 20px down & 30px right, but my original seat is still reserved.
</div>
```

3. **absolute** – “Left the Seat and Standing Anywhere”

- The element is **removed from the normal flow** (takes no space).
- Its position is calculated **relative to the nearest positioned ancestor** (`relative`, `absolute`, `fixed`, or `sticky`).
- If no ancestor is positioned → it uses the entire `html` page as reference.

📖 Analogy:

A student **leaves their seat** and stands anywhere in the hall.

Their seat is now empty.

✅ Use case:

- Dropdown menus
- Tooltips
- Badges/icons inside containers

```
<div style="position: relative; width: 200px; height: 100px; border: 2px solid black;">
  Parent (relative)
  <div style="position: absolute; top: 10px; right: 10px; background: lightcoral;">
    I am absolute inside parent
  </div>
</div>
```

4. **fixed** – “Glued to the Wall”

- The element is positioned **relative to the viewport** (the browser window).
- Even when you **scroll**, the element stays fixed.
- It's removed from normal flow.

📖 Analogy:

A student leaves their seat and **sticks to the classroom wall**.

No matter how the benches move, they stay on the wall.

✅ Use case:

- Sticky chat buttons
- Always-visible navigation bar
- "Back to Top" buttons

```
<div style="position: fixed; bottom: 20px; right: 20px; background: yellow; padding: 10px;">
```

I stay fixed at bottom-right even when scrolling

```
</div>
```

5. **sticky** – "Normal Until You Scroll, Then Sticks"

- Acts like **relative** at first (in normal flow).
- When the scroll reaches a threshold (e.g., **top: 0**), the element **sticks** like **fixed**.
- BUT it only sticks **within its parent container**.

📖 Analogy:

A student sits normally, but when the **teacher walks in**, they glue themselves to their desk and stay there.

✅ Use case:

- Sticky navigation menus
- Table headers that stay visible
- Scrollable sidebars

```
<div style="height: 100px; background: lightgreen; position: sticky; top: 0;">
```

```
I stick to the top while scrolling inside parent
</div>
<p style="height: 1000px;">Scroll to test sticky...</p>
```

✓ Quick Comparison Table

Position	What it does	Analogy	Use Case
static	Normal flow	Student sitting in seat	Default layout
relative	Keeps seat, shifts slightly	Student sliding on their bench	Small adjustments, anchors
absolute	Removed from flow, anchored	Student standing in the hall	Dropdowns, tooltips
fixed	Sticks to viewport	Student glued to the wall	Fixed headers, chat button
sticky	Relative → then fixed	Student glued to desk on scroll	Sticky menus, table headers



Student Task

👉 Build a **"Classroom Layout Page"** with 5 boxes:

1. One **static** box → stays in natural order.
2. One **relative** box → slightly shifted.
3. One **absolute** box → positioned inside a parent.
4. One **fixed** box → always visible in a corner while scrolling.
5. One **sticky** box → sticks to the top when scrolling.

▼ Flexbox

◆ 1. Flexbox Kya Hai?

Flexbox ek **one-dimensional layout system** hai jo CSS me aaya tha.

Iska kaam → **items ko row ya column me align karna**, chahe unka size dynamic ho.

👉 Pehle, hum normal CSS me `float`, `inline-block`, ya `position` use karte the layouts banane ke liye. Lekin unme **alignment, spacing, responsiveness** headache ban jaata tha.

Flexbox ne ise solve kar diya.

◆ 2. Flex Container & Flex Items

Sabse pehla rule:

- **Flexbox do cheezon ke beech ka rishta hai:**
 1. **Flex Container** → wo parent element jisme flex lagate hain.
 2. **Flex Items** → container ke andar ke direct child.

```
<div class="container">
  <div class="item">One</div>
  <div class="item">Two</div>
  <div class="item">Three</div>
</div>
```

```
.container {
  display: flex; /* yahi magic on button hai */
}
.item {
  background: lightblue;
  padding: 20px;
  margin: 10px;
}
```

👉 Ab `.container` ek **Flex Container** ban gaya, aur uske andar ke `div` s automatically **Flex Items** ban gaye.

◆ 3. Flex Direction

Default: `row` (items left → right jaate hain).

Options:

- `row` → horizontal left to right
- `row-reverse` → horizontal right to left
- `column` → vertical top to bottom
- `column-reverse` → vertical bottom to top

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Flex Direction Examples</title>
<style>
.container {
  display: flex;
  border: 2px solid black;
  margin-bottom: 20px;
  height: 100px;
  width: 300px;
}
.item {
  background-color: lightblue;
  border: 1px solid blue;
  width: 50px;
  text-align: center;
  line-height: 50px;
}
.row {
  flex-direction: row;
}
.row-reverse {
  flex-direction: row-reverse;
```

```

}
.column {
  flex-direction: column;
}
.column-reverse {
  flex-direction: column-reverse;
}
</style>
</head>
<body>

<h1>Flex Direction: row</h1>
<div class="container row">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>Flex Direction: row-reverse</h1>
<div class="container row-reverse">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>Flex Direction: column</h1>
<div class="container column">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>Flex Direction: column-reverse</h1>
<div class="container column-reverse">
  <div class="item">1</div>
  <div class="item">2</div>

```

```
<div class="item">3</div>
</div>

</body>
</html>
```

👉 Imagine railway track ki direction, aur uspar items train ke coaches hain.

◆ 4. Main Axis & Cross Axis

Ye sabse important concept hai.

- **Main Axis** → jis direction me `flex-direction` hai.
- **Cross Axis** → uske perpendicular.

flex-direction	Main Axis	Cross Axis
row	Left → Right	Top → Bottom
column	Top → Bottom	Left → Right

◆ 5. Justify Content (Main Axis Alignment)

Ye items ko main axis pe align karta hai.

Options:

- `flex-start` → sab left/top me chipak jayenge
- `flex-end` → sab right/bottom me chipak jayenge
- `center` → beech me aayenge
- `space-between` → first aur last edge pe, baaki evenly spaced
- `space-around` → items ke aas paas equal spacing
- `space-evenly` → equal spacing everywhere

Example:

```
<!DOCTYPE html>
<html>
```

```

<head>
<title>Justify Content Examples</title>
<style>
  .container {
    display: flex;
    border: 2px solid black;
    margin-bottom: 20px;
    height: 100px;
    width: 500px;
  }
  .item {
    background-color: lightgreen;
    border: 1px solid green;
    width: 60px;
    text-align: center;
    line-height: 50px;
  }
</style>
</head>
<body>

  <h1>justify-content: flex-start (Default)</h1>
  <div class="container" style="justify-content: flex-start;">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
  </div>

  <h1>justify-content: flex-end</h1>
  <div class="container" style="justify-content: flex-end;">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
  </div>

  <h1>justify-content: center</h1>

```

```

<div class="container" style="justify-content: center;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>justify-content: space-between</h1>
<div class="container" style="justify-content: space-between;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>justify-content: space-around</h1>
<div class="container" style="justify-content: space-around;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>justify-content: space-evenly</h1>
<div class="container" style="justify-content: space-evenly;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

</body>
</html>

```

◆ 6. Align Items (Cross Axis Alignment)

Ye items ko cross axis pe align karta hai.

Options:

- `flex-start` → cross axis ki starting pe
- `flex-end` → cross axis ki end pe
- `center` → bilkul center me
- `stretch` (default) → items stretch ho jaate hain container fill karne ke liye
- `baseline` → text ki baseline align hoti hai

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Align Items Examples</title>
<style>
.container {
  display: flex;
  border: 2px solid black;
  margin-bottom: 20px;
  height: 150px; /* Increased height to show alignment */
  width: 400px;
  align-items: center; /* Default for demonstration */
}
.item {
  background-color: lightcoral;
  border: 1px solid red;
  width: 60px;
  text-align: center;
  line-height: 50px;
}
.item-stretch {
  background-color: lightcoral;
  border: 1px solid red;
  width: 60px;
  text-align: center;
  /* Height is not set so it can stretch */
}
```

```

.item-baseline {
  background-color: lightcoral;
  border: 1px solid red;
  width: 60px;
  text-align: center;
}
</style>
</head>
<body>

<h1>align-items: flex-start</h1>
<p>Items align to the start of the cross axis (top in this case).</p>
<div class="container" style="align-items: flex-start;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>align-items: flex-end</h1>
<p>Items align to the end of the cross axis (bottom in this case).</p>
<div class="container" style="align-items: flex-end;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>align-items: center</h1>
<p>Items align to the center of the cross axis.</p>
<div class="container" style="align-items: center;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<h1>align-items: stretch (Default)</h1>
<p>Items stretch to fill the container's height (cross axis) because the

```


y have no specific height defined.</p>

```
<div class="container" style="align-items: stretch;">
  <div class="item-stretch">1</div>
  <div class="item-stretch">2</div>
  <div class="item-stretch">3</div>
</div>
```

<h1>align-items: baseline</h1>

<p>Items align based on their text baseline. Notice the '1' and '2' items have different font sizes to demonstrate.</p>

```
<div class="container" style="align-items: baseline;">
  <div class="item-baseline" style="font-size: 1.5em;">1</div>
  <div class="item-baseline" style="font-size: 1em;">2</div>
  <div class="item-baseline" style="font-size: 2em;">3</div>
</div>
```

</body>

</html>

◆ 7. Flex Wrap

By default flex items ek hi line me fit hone ki koshish karte hain. Agar jagah kam hai, wo shrink ho jaate hain.

Lekin agar hume chahiye items **next line me wrap ho**, to:

```
<!DOCTYPE html>
<html>
<head>
<title>Flex Wrap Examples</title>
<style>
.container {
  display: flex;
  border: 2px solid black;
  margin-bottom: 20px;
  width: 300px; /* Fixed width to demonstrate wrapping */
}
```

```

    height: 200px;
  }
  .item {
    background-color: lightyellow;
    border: 1px solid goldenrod;
    width: 100px; /* Each item is 100px wide */
    text-align: center;
    line-height: 50px;
    flex-shrink: 0; /* Prevents items from shrinking */
  }
</style>
</head>
<body>

  <h1>flex-wrap: nowrap (Default)</h1>
  <p>All items are forced onto a single line and overflow the container.
</p>
  <div class="container" style="flex-wrap: nowrap;">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
  </div>

  <h1>flex-wrap: wrap</h1>
  <p>Items wrap onto new lines below as needed.</p>
  <div class="container" style="flex-wrap: wrap;">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
  </div>

  <h1>flex-wrap: wrap-reverse</h1>
  <p>Items wrap onto new lines above as needed.</p>
  <div class="container" style="flex-wrap: wrap-reverse;">

```

```
<div class="item">1</div>
<div class="item">2</div>
<div class="item">3</div>
<div class="item">4</div>
</div>

</body>
</html>
```

Options:

- `nowrap` (default)
- `wrap`
- `wrap-reverse`

◆ 8. Align Content (Multiple Line Alignment)

Jab wrapping on ho aur multiple rows/columns ban jaayein, tab rows ko **cross-axis** pe align karne ke liye `align-content` use hota hai.

Options:

- `flex-start`
- `flex-end`
- `center`
- `space-between`
- `space-around`
- `stretch`

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Align Content Examples</title>
```

```

<style>
  .container {
    display: flex;
    flex-wrap: wrap; /* Essential for align-content to work */
    border: 2px solid black;
    margin-bottom: 20px;
    width: 300px;
    height: 300px; /* Fixed height to see alignment */
  }
  .item {
    background-color: lightcyan;
    border: 1px solid teal;
    width: 80px;
    height: 50px;
    text-align: center;
    line-height: 50px;
    font-size: 1.2em;
    flex-grow: 0;
  }
</style>
</head>
<body>

  <h1>align-content: flex-start</h1>
  <p>Rows are aligned to the top of the container.</p>
  <div class="container" style="align-content: flex-start;">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
  </div>

  <h1>align-content: flex-end</h1>
  <p>Rows are aligned to the bottom of the container.</p>

```

```
<div class="container" style="align-content: flex-end;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

```
<h1>align-content: center</h1>
<p>Rows are centered within the container.</p>
<div class="container" style="align-content: center;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

```
<h1>align-content: space-between</h1>
<p>First and last rows are at the edges, with equal space between the other rows.</p>
```

```
<div class="container" style="align-content: space-between;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>
```

```
<h1>align-content: space-around</h1>
<p>Rows have equal space around them.</p>
<div class="container" style="align-content: space-around;">
  <div class="item">1</div>
```

```

<div class="item">2</div>
<div class="item">3</div>
<div class="item">4</div>
<div class="item">5</div>
<div class="item">6</div>
</div>

<h1>align-content: stretch (Default)</h1>
<p>Rows stretch to fill the remaining space. This is the default value.
</p>
<div class="container" style="align-content: stretch;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
  <div class="item">4</div>
  <div class="item">5</div>
  <div class="item">6</div>
</div>

</body>
</html>

```

◆ 9. Flex Grow, Shrink, Basis (The Trio)

(a) **flex-grow**

flex-grow determines how a flex item will **grow** if there is extra space in the flex container. The value is a unitless number that serves as a proportion. The higher the number, the more of the extra space the item will take up.

- **Example:** If you have three items, and two have **flex-grow: 1** while the third has **flex-grow: 2**, the third item will take up twice as much of the available extra space as each of the other two items.
- **Default:** **0**. This means items won't grow to fill extra space.

(b) `flex-shrink`

`flex-shrink` determines how a flex item will **shrink** if there isn't enough space in the container. It's the inverse of `flex-grow`. The value is a unitless number that represents a proportion.

- **Example:** If you set `flex-shrink: 0`, the item will not shrink at all, even if it causes overflow. This is useful for items you want to keep at a minimum size.
- **Default:** `1`. This means items will shrink by default to fit the container.

(c) `flex-basis`

`flex-basis` sets the **initial size** of a flex item before any growing or shrinking occurs. You can set a fixed size using a value like `200px` or a percentage.

- **Example:** Setting `flex-basis: 150px` gives the item an initial size of 150 pixels. The remaining space (or lack thereof) will then be distributed based on `flex-grow` and `flex-shrink`.
- **Default:** `auto`. This means the initial size is based on the item's content.

`flex` Shorthand

The `flex` property is a shorthand for `flex-grow`, `flex-shrink`, and `flex-basis`, in that order.

- **Syntax:** `flex: <flex-grow> <flex-shrink> <flex-basis>;`
- **Example:** `flex: 1 1 200px;` is the same as writing `flex-grow: 1;`, `flex-shrink: 1;`, and `flex-basis: 200px;`.
- **Common Shortcuts:**
 - `flex: 1;` is shorthand for `flex: 1 1 0;` (grow, shrink, and basis of 0, meaning it will use the remaining space).
 - `flex: auto;` is shorthand for `flex: 1 1 auto;` (grow, shrink, and a basis based on content).
 - `flex: none;` is shorthand for `flex: 0 0 auto;` (no grow, no shrink, and a basis based on content).

```

<!DOCTYPE html>
<html>
<head>
<title>Flex Grow, Shrink, Basis Examples</title>
<style>
.container {
  display: flex;
  border: 2px solid black;
  margin-bottom: 20px;
  height: 100px;
  width: 600px;
}
.item {
  background-color: lightsalmon;
  border: 1px solid brown;
  text-align: center;
  line-height: 50px;
  font-weight: bold;
  color: white;
}
</style>
</head>
<body>

<h1>Flex Grow</h1>
<p>Item 2 has a `flex-grow` of 2, so it takes up twice as much of the
extra space as the others.</p>
<div class="container">
  <div class="item" style="flex-grow: 1;">Item 1</div>
  <div class="item" style="flex-grow: 2;">Item 2</div>
  <div class="item" style="flex-grow: 1;">Item 3</div>
</div>

<h1>Flex Shrink</h1>

```


<p>Item 2 has a `flex-shrink` of 0, so it will not shrink and might cause overflow.</p>

```
<div class="container" style="width: 300px;">
  <div class="item" style="width: 150px; flex-shrink: 1;">Item 1 (150px)</div>
  <div class="item" style="width: 150px; flex-shrink: 0;">Item 2 (150px, No Shrink)</div>
  <div class="item" style="width: 150px; flex-shrink: 1;">Item 3 (150px)</div>
</div>
```

<h1>Flex Basis</h1>

<p>Item 2 has `flex-basis: 200px`, setting its initial size before any growth or shrinking.</p>

```
<div class="container">
  <div class="item" style="flex-basis: 100px;">Item 1 (100px)</div>
  <div class="item" style="flex-basis: 200px;">Item 2 (200px)</div>
  <div class="item" style="flex-basis: 100px;">Item 3 (100px)</div>
</div>
```

<h1>Flex Shorthand</h1>

<p>Using the `flex` shorthand property for different behaviors.</p>

```
<div class="container">
  <div class="item" style="flex: 1;">flex: 1</div>
  <div class="item" style="flex: 2 1 100px;">flex: 2 1 100px</div>
  <div class="item" style="flex: 0 0 200px;">flex: 0 0 200px</div>
</div>
```

```
</body>
</html>
```

◆ 10. Order

Flex items ko visually reorder karne ke liye:

```
<!DOCTYPE html>
<html>
<head>
<title>Flex Order Example</title>
<style>
.container {
  display: flex;
  border: 2px solid black;
  margin-bottom: 20px;
  height: 100px;
  width: 400px;
}
.item {
  background-color: lightgoldenrodyellow;
  border: 1px solid goldenrod;
  width: 80px;
  text-align: center;
  line-height: 50px;
  font-weight: bold;
  font-size: 1.2em;
}
</style>
</head>
<body>

  <h1>Original Order (Based on HTML Source)</h1>
  <p>All items have the default `order: 0`.</p>
  <div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
```

```

    <div class="item">4</div>
  </div>

  <h1>Reordered with `order`</h1>
  <p>Item 2 is moved to the end, Item 4 is moved to the beginning.</p>
  <div class="container">
    <div class="item">1</div>
    <div class="item" style="order: 2;">2</div>
    <div class="item">3</div>
    <div class="item" style="order: -1;">4</div>
  </div>

  <p>In the second example, the visual order is **4, 1, 3, 2**.</p>
  <ul>
    <li>Item 4 has `order: -1`, so it comes first.</li>
    <li>Items 1 and 3 have the default `order: 0`, so they follow their source order.</li>
    <li>Item 2 has `order: 2`, so it comes last.</li>
  </ul>

</body>
</html>

```

👉 Ye HTML order change nahi karta, sirf display order change hota hai.

◆ 11. Gap (Modern Spacing)

Flex items ke beech me gap dene ka easy way:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Gap Example</title>

```

```

<style>
  body {
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background-color: #f0f0f0;
    margin: 0;
  }

  .flex-container {
    display: flex;
    background-color: #e0e0e0;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    /* Flex items ke beech mein space dene ke liye gap property */
    gap: 20px; /* Horizontally aur vertically 20px ka gap dega */
  }

  .flex-item {
    background-color: #4CAF50;
    color: white;
    padding: 30px;
    border-radius: 5px;
    font-size: 18px;
    font-weight: bold;
    text-align: center;
  }
</style>
</head>
<body>

  <div class="flex-container">
    <div class="flex-item">Box 1</div>

```

```
<div class="flex-item">Box 2</div>
<div class="flex-item">Box 3</div>
</div>

</body>
</html>
```

Flexbox Ka Mantra

1. Container ko `display: flex;` do.
2. Decide karo `flex-direction` .
3. Main axis alignment ke liye → `justify-content` .
4. Cross axis alignment ke liye → `align-items` ya `align-self` .
5. Multiple lines ke liye → `flex-wrap` + `align-content` .
6. Size control ke liye → `flex-grow` , `flex-shrink` , `flex-basis` .
7. Spacing ke liye → `gap` .
8. Order ke liye → `order` .

▼ Grid

◆ 1. What is CSS Grid?

- **Grid** = A **two-dimensional layout system** in CSS.
- It allows you to arrange items in **rows and columns simultaneously**.
- Perfect for complex layouts (dashboards, galleries, page structures).

👉 Think of Grid as an **Excel sheet** (rows × columns), where you can place elements exactly where you want.

◆ 2. Grid Container & Grid Items

- **Grid Container** → parent element with `display: grid;` .

- **Grid Items** → direct children of the container.

```
<div class="grid-container">
  <div class="grid-item">One</div>
  <div class="grid-item">Two</div>
  <div class="grid-item">Three</div>
</div>
```

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

◆ 3. Defining Rows & Columns

We define grid **rows and columns** using:

- `grid-template-columns`
- `grid-template-rows`

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
=1.0">
  <title>CSS Grid Rows & Columns</title>
  <style>
    body {
      font-family: sans-serif;
      padding: 20px;
      background-color: #f0f0f0;
    }
  </style>
</head>
<body>
```

```

.grid-container {
    display: grid;
    /* Columns: 100px, 100px, 100px */
    grid-template-columns: 100px 100px 100px;

    /* Rows: 50px, 50px */
    grid-template-rows: 50px 50px;

    gap: 10px; /* Grid items ke beech mein gap */
    background-color: #ddd;
    padding: 10px;
}

.grid-item {
    background-color: #4CAF50;
    color: white;
    padding: 10px;
    text-align: center;
    border-radius: 5px;
}
</style>
</head>
<body>

<div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
</div>

</body>
</html>

```

👉 First column = 100px, second = 200px, third = flexible (`auto`).

◆ 4. The `fr` Unit (Fractional Space)

Fractional unit (`fr`) is a flexible unit in CSS Grid that represents a portion of the available free space in a grid container. It allows you to create fluid and responsive column and row layouts without using fixed pixel values. The `fr` unit is especially useful for distributing space proportionally among grid items.

Explanation

In the CSS, the `grid-template-columns: 1fr 2fr 1fr;` declaration tells the browser to create three columns. The total number of fractions is $1 + 2 + 1 = 4$. The available space is divided into these four fractions.

- The **first column** gets `1` fraction ($1/4$ of the total space).
- The **second column** gets `2` fractions ($2/4$ or $1/2$ of the total space).
- The **third column** gets `1` fraction ($1/4$ of the total space).

This setup ensures that the middle column is always twice as wide as the other two, regardless of the screen size. If you resize the browser window, the columns will adjust their width proportionally, maintaining the same ratio. This makes `fr` units excellent for building responsive and adaptable grid layouts.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Grid fr Unit Example</title>
  <style>
    body {
      font-family: sans-serif;
```



```

padding: 20px;
background-color: #f0f0f0;
}

.grid-container {
display: grid;
/* Defines three columns with widths in a 1:2:1 ratio */
grid-template-columns: 1fr 2fr 1fr;

gap: 10px;
background-color: #ddd;
padding: 10px;
max-width: 600px;
margin: auto;
}

.grid-item {
background-color: #4CAF50;
color: white;
padding: 20px;
text-align: center;
border-radius: 5px;
font-size: 1.2em;
font-weight: bold;
}
</style>
</head>
<body>

<h2>CSS Grid with `fr` Units</h2>
<p>This grid has three columns with a 1:2:1 ratio. The middle column
is twice as wide as the other two.</p>

<div class="grid-container">
  <div class="grid-item">1fr</div>
  <div class="grid-item">2fr</div>

```

```
<div class="grid-item">1fr</div>
</div>

</body>
</html>
```

👉 Means → 1:2:1 ratio (like flexible slices of a pie).

◆ 5. Repeat Function

The `repeat()` function in CSS Grid is a powerful tool for defining multiple columns or rows that have the same size and structure. It simplifies the process, so you don't have to write out each track individually.

How it Works

The `repeat()` function takes two arguments:

1. **The number of times to repeat:** This is an integer specifying how many columns or rows you want to create.
2. **The value of each track:** This is the size of each column or row (e.g., `1fr`, `100px`, `auto`).

```
.grid-container {
  grid-template-columns: 1fr 1fr 1fr 1fr 1fr;
}
```

You can use the `repeat()` function to achieve the same result more concisely:

```
.grid-container {
  grid-template-columns: repeat(5, 1fr);
}
```

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
=1.0">
  <title>CSS Grid Repeat Function Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      background-color: #f4f4f4;
    }

    .grid-container {
      display: grid;
      /* Creates four columns, each taking an equal fraction of the av
ailable space */
      grid-template-columns: repeat(4, 1fr);

      gap: 15px; /* Adds space between grid items */
      background-color: #e0e0e0;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    }

    .grid-item {
      background-color: #2c3e50;
      color: white;
      padding: 30px;
      text-align: center;
      border-radius: 5px;
      font-size: 1.2em;
      font-weight: bold;
    }

    /* For smaller screens, change to two columns */

```

```

    @media (max-width: 768px) {
      .grid-container {
        grid-template-columns: repeat(2, 1fr);
      }
    }
  </style>
</head>
<body>

```

<h2>CSS Grid `repeat()` Function</h2>

<p>This grid uses `repeat(4, 1fr)` to create four equal-width columns. On smaller screens, the layout changes to two columns.</p>

```

<div class="grid-container">
  <div class="grid-item">Box 1</div>
  <div class="grid-item">Box 2</div>
  <div class="grid-item">Box 3</div>
  <div class="grid-item">Box 4</div>
  <div class="grid-item">Box 5</div>
  <div class="grid-item">Box 6</div>
</div>

</body>
</html>

```

◆ 6. Gap

Adds spacing between rows and columns.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

```

<title>CSS Gap Property Example</title>
<style>
  body {
    font-family: Arial, sans-serif;
    padding: 20px;
    background-color: #f4f4f4;
  }

  .grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);

    /* Adds a 20px gap between all rows and columns */
    gap: 20px;

    background-color: #e0e0e0;
    padding: 10px;
    border-radius: 8px;
  }

  .grid-item {
    background-color: #3498db;
    color: white;
    padding: 30px;
    text-align: center;
    border-radius: 5px;
    font-size: 1.2em;
    font-weight: bold;
  }
</style>
</head>
<body>

  <h2>CSS `gap` Property</h2>
  <p>This grid uses `gap: 20px;` to add consistent spacing between al
l boxes.</p>

```

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>

</body>
</html>
```

◆ 7. Placing Items

Grid items can be placed precisely using **line numbers**.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
=1.0">
  <title>CSS Grid Placing Items</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      background-color: #f4f4f4;
    }

    .grid-container {
      display: grid;
      grid-template-columns: repeat(3, 100px);
      grid-template-rows: repeat(3, 100px);
```

```

    gap: 10px;
    background-color: #e0e0e0;
    padding: 10px;
    border-radius: 8px;
}

.grid-item {
    background-color: #3498db;
    color: white;
    padding: 15px;
    text-align: center;
    border-radius: 5px;
    font-size: 1.2em;
}

/* This is the item we will place precisely */
.special-item {
    background-color: #e74c3c;
    /* Starts at column line 2 and ends at column line 4 */
    grid-column-start: 2;
    grid-column-end: 4;
    /* Starts at row line 1 and ends at row line 3 */
    grid-row-start: 1;
    grid-row-end: 3;
}

</style>
</head>
<body>

    <h2>Placing Grid Items with Line Numbers</h2>
    <p>The red box is placed precisely to span from column line 2 to 4 and row line 1 to 3.</p>

    <div class="grid-container">
        <div class="grid-item">1</div>

```

```

<div class="special-item">Special</div>
<div class="grid-item">3</div>
<div class="grid-item">4</div>
<div class="grid-item">5</div>
<div class="grid-item">6</div>
<div class="grid-item">7</div>
<div class="grid-item">8</div>
<div class="grid-item">9</div>
</div>

</body>
</html>

```

👉 Grid lines are numbered starting from **1**.

◆ 8. Shorthand → **grid-area**

Combine row and column placement:

```

.item1 {
  grid-area: 1 / 1 / 2 / 3;
  /* row-start / col-start / row-end / col-end */
}

```

◆ 9. Auto-placement

If no position is defined, Grid **automatically flows items** into available cells.

Flow direction:

- Default: row-wise (**grid-auto-flow: row**)
- Column-wise: **grid-auto-flow: column**

```

<!DOCTYPE html>
<html lang="en">
<head>

```



```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale
=1.0">
<title>CSS Grid Auto-Placement</title>
<style>
  body {
    font-family: Arial, sans-serif;
    padding: 20px;
    background-color: #f4f4f4;
  }

  .grid-container {
    display: grid;
    grid-template-columns: repeat(3, 100px);
    grid-template-rows: 100px 100px;
    gap: 10px;
    background-color: #e0e0e0;
    padding: 10px;
    border-radius: 8px;
    margin-bottom: 20px;
  }

  .grid-item {
    background-color: #3498db;
    color: white;
    padding: 15px;
    text-align: center;
    border-radius: 5px;
    font-size: 1.2em;
  }

  /* Example of default row-wise auto-placement */
  .row-flow {
    /* grid-auto-flow: row; (This is the default, so it's optional) */
  }

```

```

/* Example of column-wise auto-placement */
.column-flow {
    grid-auto-flow: column;
}

</style>
</head>
<body>

<h2>Auto-Placement: Default (Row-wise)</h2>
<p>Items flow from left to right, filling rows.</p>
<div class="grid-container row-flow">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
</div>

<hr>

<h2>Auto-Placement: Column-wise</h2>
<p>Items flow from top to bottom, filling columns.</p>
<div class="grid-container column-flow">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
</div>

```

```
</body>
</html>
```

◆ 11. Justify & Align in Grid

- **justify-items** → align inside each cell (horizontally).
- **align-items** → align inside each cell (vertically).
- **justify-content** → aligns the whole grid in container (horizontally).
- **align-content** → aligns the whole grid in container (vertically).

Values: `start`, `end`, `center`, `stretch`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Grid Justify & Align</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      background-color: #f0f0f0;
      text-align: center;
    }

    .grid-container {
      display: grid;
      grid-template-columns: 100px 100px;
      grid-template-rows: 100px 100px;
      gap: 10px;
      background-color: #ddd;
      padding: 10px;
```

```

    margin: 20px auto;
    border: 2px solid #333;
    max-width: 400px;
    height: 300px; /* To show align-content effect */
}

.grid-item {
    background-color: #4CAF50;
    color: white;
    display: flex; /* To easily center the text */
    justify-content: center;
    align-items: center;
    font-size: 1.2em;
    border-radius: 5px;
}

/* ----- Item Alignment within each cell ----- */
.items-example {
    justify-items: center; /* Horizontally centers each item */
    align-items: center; /* Vertically centers each item */
}

/* ----- Grid Alignment within the container ----- */
.content-example {
    justify-content: center; /* Horizontally centers the whole grid */
    align-content: center; /* Vertically centers the whole grid */
    height: 300px; /* Important for align-content to be visible */
}

.content-start {
    justify-content: start;
    align-content: start;
}

.content-end {
    justify-content: end;
}

```

```

        align-content: end;
    }
</style>
</head>
<body>

    <h1>CSS Grid Alignment</h1>

    <h2>1. Justifying & Aligning Items (Inside a Cell)</h2>
    <p>This grid uses <code>justify-items: center</code> and <code>align-items: center</code>.</p>
    <div class="grid-container items-example">
        <div class="grid-item">Box 1</div>
        <div class="grid-item">Box 2</div>
        <div class="grid-item">Box 3</div>
        <div class="grid-item">Box 4</div>
    </div>
    <hr>

    <h2>2. Justifying & Aligning Content (The Whole Grid)</h2>
    <p>The entire grid is centered horizontally and vertically using <code>justify-content: center</code> and <code>align-content: center</code>.</p>
    <div class="grid-container content-example">
        <div class="grid-item">1</div>
        <div class="grid-item">2</div>
        <div class="grid-item">3</div>
        <div class="grid-item">4</div>
    </div>
    <hr>

    <h2>3. Aligning Content at `start` and `end`</h2>
    <p><code>justify-content: start</code> and <code>align-content: start</code></p>
    <div class="grid-container content-start">
        <div class="grid-item">1</div>

```

```

    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
  </div>
  <p><code>justify-content: end</code> and <code>align-content: e
nd</code></p>
  <div class="grid-container content-end">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
  </div>

</body>
</html>

```

Complete HTML + CSS Example (All Grid Concepts)

👉 Copy and run this to see everything in action:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSS Grid Master Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }

    .grid-container {
      display: grid;

```

```

grid-template-columns: 1fr 2fr 1fr; /* Three columns */
grid-template-rows: 100px 150px auto; /* Three rows */
gap: 15px; /* spacing */
border: 2px solid black;
padding: 20px;
height: 500px;
}

.grid-item {
background: lightblue;
border: 2px solid navy;
text-align: center;
padding: 20px;
font-weight: bold;
}

/* Spanning columns */
.item1 {
grid-column: 1 / 3; /* spans column 1 and 2 */
grid-row: 1; /* row 1 only */
background: lightcoral;
}

/* Spanning rows */
.item2 {
grid-column: 3; /* column 3 */
grid-row: 1 / 3; /* spans row 1 and 2 */
background: lightgreen;
}

/* Auto placement */
.item3 {
grid-column: 1;
grid-row: 2;
}

```

```

.item4 {
  grid-column: 2;
  grid-row: 2;
}

/* Full width footer */
.item5 {
  grid-column: 1 / 4; /* spans all columns */
  grid-row: 3;
  background: lightgoldenrodyellow;
}
</style>
</head>
<body>

<h1>CSS Grid Master Example</h1>
<div class="grid-container">
  <div class="grid-item item1">Item 1 (Spans 2 Columns)</div>
  <div class="grid-item item2">Item 2 (Spans 2 Rows)</div>
  <div class="grid-item item3">Item 3</div>
  <div class="grid-item item4">Item 4</div>
  <div class="grid-item item5">Item 5 (Footer Full Width)</div>
</div>

</body>
</html>

```

CSS Grid Master Formula

1. Turn on grid → `display: grid;`.
2. Define rows/columns → `grid-template-columns`, `grid-template-rows`.
3. Place items → `grid-column`, `grid-row`, or `grid-area`.

4. Use `fr`, `repeat()`, `minmax()` for flexible layouts.
5. Use `auto-fill` / `auto-fit` for responsiveness.
6. Control spacing → `gap`.
7. Align → `justify-items`, `align-items`, `justify-content`, `align-content`.
8. Use named `grid-template-areas` for semantic layouts.

▼ Mini Project



Practice

Project: "My Personal Portfolio"



"Portfolio Project" → Sub Mini-Projects

Sub Mini-Project 1: Navbar & Hero Section

Focus Concepts:

- CSS Syntax (internal/external)
- Selectors (class, element)
- Flexbox
- Position: fixed
- Colors, background, gradients

Task:

1. Build a fixed navbar with logo + links.
2. Create a hero section with text on the left and image on the right using flexbox.

Hints:

- Navbar → `position: fixed; top: 0; width: 100%;`

- Hero → `display: flex; justify-content: space-between; align-items: center;`
-

Sub Mini-Project 2: About Section

Focus Concepts:

- Box Model: margin, padding, border, border-radius
- Background color
- Text & Fonts
- Display: block

Task:

1. Create an about section below the hero.
2. Include heading + paragraph describing yourself.
3. Style it with padding, margin, background, and box shadow.

Hints:

- Wrap content in `<div class="about">`
 - Add `border-radius` and `box-shadow` for style
-

Sub Mini-Project 3: Skills Section

Focus Concepts:

- Flexbox or inline-block
- Colors, gradients
- Hover effects

Task:

1. Create boxes representing skills (HTML, CSS, JS, React).
2. Align them horizontally and evenly spaced.
3. Add hover effects (scale or color change).

Hints:

- Use `display: flex; justify-content: space-around;` or `inline-block` for boxes
 - `transition: transform 0.3s;` for smooth hover effect
-

Sub Mini-Project 4: Projects Section

Focus Concepts:

- CSS Grid
- Box Model (padding, margin, border)
- Responsive design (`minmax` , `auto-fit`)

Task:

1. Display multiple project cards in a responsive grid layout.
2. Each card has a title and short description.

Hints:

- Parent: `display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 20px;`
 - Cards → `padding` , `border` , `border-radius` , `background-color`
-

Sub Mini-Project 5: Sidebar & Main Content

Focus Concepts:

- Flexbox for layout
- Position: sticky
- Margin/padding

Task:

1. Create a sidebar on the right with quick links.
2. Make it sticky when scrolling.
3. Main content area displays projects and testimonials.

Hints:

- Container → `display: flex;`

- Sidebar → `position: sticky; top: 80px;`
 - Main content → `flex: 3`
-

Sub Mini-Project 6: Overflow Section (Testimonials)

Focus Concepts:

- Overflow (`auto` , `scroll`)
- Box height & padding

Task:

1. Add a testimonials box inside main content.
2. Add multiple `<p>` to make it scrollable.

Hints:

- `.testimonials { height: 120px; overflow: auto; border: 2px solid red; }`
-

Sub Mini-Project 7: Footer

Focus Concepts:

- Display: block
- Text alignment
- Colors, background

Task:

1. Create a footer at the bottom with centered text.

▼ Answer

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```

<meta charset="UTF-8">
<title>My Portfolio</title>
<style>
  /* ----- General Styles ----- */
  body {
    margin: 0;
    font-family: Arial, sans-serif;
    line-height: 1.6;
    background: #f5f5f5;
    color: #333;
  }

  h1, h2, h3, h4 {
    margin: 0;
    padding: 0;
  }

  a {
    text-decoration: none;
    color: white;
  }

  /* ----- Navbar (Fixed) ----- */
  .navbar {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    background: linear-gradient(to right, #4caf50, #81c784);
    color: white;
    padding: 15px 20px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    z-index: 1000;
  }

```

```
.navbar .logo {  
  font-weight: bold;  
  font-size: 20px;  
}  
  
.navbar .nav-links a {  
  margin-left: 20px;  
}  
  
/* ----- Hero Section (Flexbox) ----- */  
.hero {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
  padding: 120px 50px 50px; /* top padding for navbar */  
  background: #e8f5e9;  
}  
  
.hero-text {  
  flex: 1;  
  padding-right: 20px;  
}  
  
.hero-text h1 {  
  font-size: 40px;  
  color: #2e7d32;  
}  
  
.hero-text p {  
  font-size: 18px;  
  margin-top: 10px;  
}  
  
.hero-img {  
  flex: 1;
```

```

    text-align: center;
}

.hero-img img {
    width: 250px;
    height: 250px;
    border-radius: 50%;
    border: 5px solid #4caf50;
}

/* ----- About Section ----- */
.about {
    padding: 50px;
    margin: 20px;
    background: #ffffff;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

.about h2 {
    color: #388e3c;
    margin-bottom: 15px;
}

/* ----- Skills Section ----- */
.skills {
    display: flex;
    justify-content: space-around;
    padding: 50px;
    background: linear-gradient(to right, #aed581, #c5e1a5);
}

.skill-box {
    width: 150px;
    height: 150px;
    display: flex;

```

```

    align-items: center;
    justify-content: center;
    background: #ffffff;
    border-radius: 10px;
    font-weight: bold;
    font-size: 18px;
    transition: transform 0.3s;
  }

  .skill-box:hover {
    transform: scale(1.1);
    background: #aed581;
  }

  /* ----- Projects Section (Grid) ----- */
  .projects {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 20px;
    padding: 50px;
  }

  .project-card {
    background: #ffffff;
    padding: 20px;
    border-radius: 10px;
    border: 1px solid #c8e6c9;
    text-align: center;
  }

  /* ----- Sticky Sidebar ----- */
  .main-content {
    display: flex;
    padding: 20px;
  }

```



```

.content {
  flex: 3;
  margin-right: 20px;
}

.sidebar {
  flex: 1;
  position: sticky;
  top: 80px;
  background: #c8e6c9;
  padding: 20px;
  border-radius: 10px;
}

/* ----- Overflow Example ----- */
.testimonials {
  margin-top: 20px;
  height: 120px;
  border: 2px solid #f44336;
  overflow: auto;
  padding: 10px;
  background: #ffcdd2;
}

/* ----- Footer ----- */
.footer {
  display: block;
  background: #4caf50;
  color: white;
  text-align: center;
  padding: 20px;
  margin-top: 50px;
}
</style>
</head>
<body>

```

```

<!-- Navbar →
<div class="navbar">
  <div class="logo">My Portfolio</div>
  <div class="nav-links">
    <a href="#">Home</a>
    <a href="#">About</a>
    <a href="#">Projects</a>
    <a href="#">Contact</a>
  </div>
</div>

<!-- Hero Section →
<div class="hero">
  <div class="hero-text">
    <h1>Hello, I'm Karan</h1>
    <p>Welcome to my personal portfolio. I create modern web layouts
using CSS, Flexbox, and Grid.</p>
  </div>
  <div class="hero-img">
    
  </div>
</div>

<!-- About Section →
<div class="about">
  <h2>About Me</h2>
  <p>I am a web developer learning CSS layouts, positions, flexbox, gr
id, and responsive design. I enjoy creating clean and functional web de
signs.</p>
</div>

<!-- Skills Section →
<div class="skills">
  <div class="skill-box">HTML</div>
  <div class="skill-box">CSS</div>

```

```

<div class="skill-box">JS</div>
<div class="skill-box">React</div>
</div>

<!-- Main Content with Sidebar →
<div class="main-content">
  <div class="content">
    <!-- Projects Section →
    <div class="projects">
      <div class="project-card">Project 1</div>
      <div class="project-card">Project 2</div>
      <div class="project-card">Project 3</div>
      <div class="project-card">Project 4</div>
    </div>

    <!-- Overflow / Testimonials →
    <div class="testimonials">
      <strong>Testimonials:</strong>
      <p>User1: Great portfolio!</p>
      <p>User2: Very creative!</p>
      <p>User3: Easy to navigate.</p>
      <p>User4: Well organized.</p>
      <p>User5: Clean design!</p>
    </div>
  </div>

  <!-- Sticky Sidebar →
  <div class="sidebar">
    <h3>Quick Links</h3>
    <ul>
      <li>Contact Me</li>
      <li>Resume</li>
      <li>Blog</li>
      <li>Follow Me</li>
    </ul>
  </div>

```

```
</div>

<!-- Footer -->
<div class="footer">© 2025 My Portfolio</div>

</body>
</html>
```

▼ Class 3: Responsive Design & Advanced Styling

▼ Media Queries (mobile-first)

Media Queries (Mobile-First)

What are Media Queries?

Media Queries are like **rules** that tell the website:

👉 "If the screen size is small, show this style. If the screen is big, show another style."

They help us make **responsive websites** – websites that look good on **mobile, tablet, and desktop**.

Syntax (How to Write It)

```
@media (condition) {
  /* CSS code here */
}
```

Example:

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


```
}  
}
```

👉 This means: if the screen is **600px or smaller**, the background will be **yellow**.

Mobile-First Idea

Mobile-First means:

1. Write CSS for **mobile** first (smallest screen).
2. Then add styles for **bigger screens** step by step.

Why? → Most people use the web on mobile, so we make sure mobile looks good first 

Example (Mobile-First)

```
/* Default: Mobile (small screens) */  
.container {  
  display: flex;  
  flex-direction: column;  
}  
  
/* Tablet (screen 768px or bigger) */  
@media (min-width: 768px) {  
  .container {  
    flex-direction: row;  
  }  
}  
  
/* Desktop (screen 1024px or bigger) */  
@media (min-width: 1024px) {  
  .container {  
    justify-content: space-between;  
  }  
}
```

```
}  
}
```

Common Screen Sizes (Breakpoints)

- 📱 Mobile → up to 600px
- 📺 Tablet → 601px to 768px
- 💻 Small Laptop → 769px to 1024px
- 🖥️ Desktop → 1025px and above

👉 Simple way to remember:

- **Mobile is default**
- **Tablet adds changes**
- **Laptop/Desktop adds more changes**

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale  
=1.0">  
  <title>Mobile-First Media Queries Example</title>  
  <style>  
    /* Mobile (default styles) */  
    body {  
      font-family: Arial, sans-serif;  
      margin: 0;  
      padding: 20px;  
      background-color: #f0f0f0;  
      text-align: center;  
    }  
  
    .container {
```

```

border: 2px solid #333;
padding: 10px;
display: flex;
flex-direction: column;
gap: 10px;
background-color: #fff;
}

.item {
padding: 20px;
background-color: #4CAF50;
color: white;
border-radius: 5px;
}

/* Tablet (601px to 768px) */
@media (min-width: 601px) {
body {
background-color: #e0e0ff;
}
.container {
flex-direction: row;
flex-wrap: wrap;
justify-content: center;
}
.item {
flex: 1 1 45%; /* Items will take up about half the container's
width */
}
}

/* Small Laptop (769px to 1024px) */
@media (min-width: 769px) {
body {
background-color: #ffe0e0;
}
}

```

```

        .container {
            flex-direction: row;
            justify-content: space-between;
        }
        .item {
            flex: 1 1 30%; /* Items will take up about a third of the contain
er's width */
        }
    }

    /* Desktop (1025px and above) */
    @media (min-width: 1025px) {
        body {
            background-color: #e0fff0;
        }
        .container {
            max-width: 1200px;
            margin: 0 auto;
        }
        .item {
            flex: 1 1 20%; /* Items will take up about a fifth of the containe
r's width */
            font-size: 1.2em;
        }
    }
</style>
</head>
<body>
    <h1>Responsive Layout Example</h1>
    <p>Resize your browser window to see the layout change. The back
ground color and item arrangement will adapt to the screen size.</p>

    <div class="container">
        <div class="item">Item 1</div>
        <div class="item">Item 2</div>
        <div class="item">Item 3</div>

```



```
<div class="item">Item 4</div>
<div class="item">Item 5</div>
</div>
</body>
</html>
```

▼ Units (px, %, em, rem, vh, vw)

px (Pixels)

Pixels are an absolute unit of measurement, meaning they're fixed in size. A pixel is a single dot on a screen. Because they don't scale with the user's settings, they are not ideal for creating responsive layouts. However, they are useful for elements that need a precise, unchanging size, like borders or small icons.

Example Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>px Example</title>
  <style>
    .box {
      width: 200px;
      height: 100px;
      border: 2px solid black;
      background-color: lightblue;
      margin: 20px;
    }
    .text {
```

```

        font-size: 16px;
        padding: 10px;
    }
</style>
</head>
<body>
    <h1>px Example</h1>
    <div class="box">
        <p class="text">This box has a fixed width of 200px and the text i
s 16px.</p>
    </div>
</body>
</html>

```

% (Percentage)

The percentage unit is relative to the parent element. For example, if a child element has a width of `50%`, it will take up half the width of its parent container. This makes it a great unit for creating fluid, responsive designs that adapt to different screen sizes.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
=1.0">
    <title>% Example</title>
    <style>
        .container {
            width: 80%;
            height: 200px;
            border: 2px solid black;
            background-color: lightgray;

```

```

        margin: 20px auto;
    }
    .child-box {
        width: 50%;
        height: 50%;
        background-color: lightcoral;
    }
</style>
</head>
<body>
    <h1>% Example</h1>
    <div class="container">
        <div class="child-box">This box is 50% of its parent container's
width and height.</div>
    </div>
</body>
</html>

```

em (Element's Font Size)

The `em` unit is relative to the font size of its **direct parent element**. This is often used for text, where an `em` value of `1.5` would be 1.5 times the parent's font size. It's a useful unit for scaling elements in relation to each other, but it can be tricky to manage since the size of an element is dependent on its entire chain of parent elements.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
=1.0">
    <title>em Example</title>

```

```

<style>
  .parent {
    font-size: 20px;
    background-color: lightgray;
    padding: 1em; /* padding will be 20px */
  }
  .child {
    font-size: 1.5em; /* font-size will be 1.5 * 20px = 30px */
    margin-top: 1em; /* margin will be 20px */
    background-color: lightgreen;
  }
</style>
</head>
<body>
  <h1>em Example</h1>
  <div class="parent">
    <p>This is the parent element with a font-size of 20px.</p>
    <p class="child">This child element's font-size is 1.5em, which is
30px.</p>
  </div>
</body>
</html>

```

rem (Root em)

Similar to `em`, the `rem` unit is also a relative unit, but it's always relative to the font size of the **root** `<html>` **element**. This makes it much easier to manage than `em` because you only have one value to keep track of—the root font size. `rem` is excellent for creating a consistent and scalable typography system. By changing the root font size, you can scale the entire layout.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
=1.0">
  <title>rem Example</title>
  <style>
    html {
      font-size: 16px; /* Base font size */
    }
    body {
      font-family: Arial, sans-serif;
      margin: 2rem; /* margin will be 2 * 16px = 32px */
    }
    .title {
      font-size: 3rem; /* font-size will be 3 * 16px = 48px */
    }
    .text {
      font-size: 1.25rem; /* font-size will be 1.25 * 16px = 20px */
    }
  </style>
</head>
<body>
  <h1 class="title">rem Example</h1>
  <p class="text">This text's font size is 1.25rem, based on the root ht
ml element's font size.</p>
</body>
</html>

```

vh (Viewport Height)

The **vh** unit is relative to the **height of the viewport** (the browser window). A value of **1vh** is equal to 1% of the viewport's height. This unit is especially

useful for creating layouts that are a specific proportion of the screen height, such as full-screen sections.

Example Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
=1.0">
  <title>vh Example</title>
  <style>
    .full-screen-section {
      height: 100vh;
      width: 100%;
      background-color: #f0f0f0;
      display: flex;
      justify-content: center;
      align-items: center;
      text-align: center;
    }
  </style>
</head>
<body>
  <div class="full-screen-section">
    <h1>This section takes up 100% of the viewport height.</h1>
  </div>
</body>
</html>
```

vw (Viewport Width)

Similar to `vh`, the `vw` unit is relative to the **width of the viewport**. `1vw` is equal to 1% of the viewport's width. This unit is perfect for creating

elements that scale proportionally with the browser window's width, like a font size that gets larger as the screen widens.

Example Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vw Example</title>
  <style>
    .responsive-heading {
      font-size: 8vw;
      text-align: center;
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <h1 class="responsive-heading">Resize me! I get bigger with the window.</h1>
  <p style="text-align: center;">This heading's font size is 8% of the viewport's width.</p>
</body>
</html>
```

▼ Forms & Buttons styling

Forms & Buttons Styling in CSS

Forms and buttons are the **main way users interact** with a website. Agar tum inhe stylish aur user-friendly bana do → website professional lagti hai.

● Step 1: Basic Form Elements

HTML me form ke andar hota hai:

- **Input fields** (text, email, password, etc.)
- **Textarea** (for longer text)
- **Select dropdowns**
- **Buttons** (submit, reset, normal)

Example:

```
<form>
  <label for="name">Name:</label>
  <input type="text" id="name" placeholder="Enter your name">

  <label for="email">Email:</label>
  <input type="email" id="email" placeholder="Enter your email">

  <textarea placeholder="Your message"></textarea>

  <button type="submit">Submit</button>
</form>
```

● Step 2: Styling Inputs

```
input, textarea, select {
  width: 100%;
  padding: 10px;
  margin: 8px 0;
  border: 1px solid #ccc;
  border-radius: 6px;
  font-size: 16px;
}

input:focus, textarea:focus {
```



```
border-color: #4CAF50; /* green border */
outline: none; /* removes default blue outline */
}
```

👉 Makes inputs neat, with padding and rounded corners.

🟢 Step 3: Styling Buttons

```
button {
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  font-size: 16px;
}

/* Hover Effect */
button:hover {
  background-color: #45a049;
}
```

👉 This gives a **modern green button** with hover effect.

🟢 Step 4: Advanced Touch (Optional for Students)

- Different button styles:
 - `button.primary` (main action)
 - `button.secondary` (less important action)
- Add **transition** for smooth hover:

```
button {  
  transition: background-color 0.3s ease;  
}
```

Beginner-Friendly Summary

- **Inputs** → use padding, border, rounded corners.
- **Focus effect** → change border color to guide users.
- **Buttons** → add color, hover, and cursor pointer.
- Keep it **consistent** (all inputs same style).

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=  
1.0">  
  <title>Styled Form Example</title>  
  <style>  
    body {  
      font-family: Arial, sans-serif;  
      background: #f2f2f2;  
      display: flex;  
      justify-content: center;  
      align-items: center;  
      min-height: 100vh;  
    }  
  
    .form-container {  
      background: white;  
      padding: 20px 30px;  
      border-radius: 10px;  
      box-shadow: 0 4px 8px rgba(0,0,0,0.1);
```

```
width: 100%;  
max-width: 400px;  
}  
  
.form-container h2 {  
text-align: center;  
margin-bottom: 20px;  
color: #333;  
}  
  
label {  
display: block;  
margin-bottom: 6px;  
font-weight: bold;  
color: #444;  
}  
  
input, textarea, select {  
width: 100%;  
padding: 10px;  
margin-bottom: 15px;  
border: 1px solid #ccc;  
border-radius: 6px;  
font-size: 16px;  
box-sizing: border-box;  
}  
  
input:focus, textarea:focus, select:focus {  
border-color: #4CAF50;  
outline: none;  
}  
  
button {  
background-color: #4CAF50;  
color: white;  
padding: 12px;
```

```

width: 100%;
border: none;
border-radius: 6px;
font-size: 16px;
cursor: pointer;
transition: background-color 0.3s ease;
}

button:hover {
background-color: #45a049;
}
</style>
</head>
<body>

<div class="form-container">
  <h2>Contact Us</h2>
  <form>
    <label for="name">Name:</label>
    <input type="text" id="name" placeholder="Enter your name" required>

    <label for="email">Email:</label>
    <input type="email" id="email" placeholder="Enter your email" required>

    <label for="subject">Subject:</label>
    <select id="subject">
      <option>General Inquiry</option>
      <option>Support</option>
      <option>Feedback</option>
    </select>

    <label for="message">Message:</label>
    <textarea id="message" rows="4" placeholder="Write your message"></textarea>
  </form>
</div>

```

```
<button type="submit">Submit</button>
</form>
</div>

</body>
</html>
```

▼ Z-index & stacking context

CSS Z-index & Stacking Context

1. What is Z-index?

Imagine your web page as **layers of paper stacked on a table**.

- Some elements are **on top**
- Some are **below**

 `z-index` tells the browser **which layer should be on top**.

2. Default Behavior

- By default, elements appear in the order they are written in HTML.
- Later elements can overlap earlier ones if positioned.

Example (without z-index):

```
.box1 {
  position: absolute;
  top: 50px;
  left: 50px;
  background: red;
}

.box2 {
```

```
position: absolute;
top: 80px;
left: 80px;
background: blue;
}
```

👉 The **blue box** will cover the red box if they overlap.

3. Using Z-index

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

.box2 {
  z-index: 1; /* below */
}
```

👉 Now red box (z-index 2) will appear **above** blue box.

4. Important Rule

- **z-index** **only works** on elements that have **position** set as: **relative**, **absolute**, **fixed**, or **sticky**.
 - It **does not work** on **position: static** (default).
-

5. Stacking Context

Sometimes, even if you give a **higher z-index**, element still doesn't come on top 😞.

That's because of **stacking context**.

👉 A new stacking context is created when:

- You use **position** with **z-index**

- CSS properties like `opacity < 1` , `transform` , `filter`

Example:

```
.parent {  
  position: relative;  
  z-index: 1;  
}  
  
.child {  
  position: absolute;  
  z-index: 999; /* Still cannot escape parent's stacking */  
}
```

👉 Even though child has `z-index: 999` , it will stay under other elements if its **parent** is below in stacking.

6. Quick Analogy

Think of **folders**:

- Each folder = stacking context.
- Files inside can have high numbers, but they **can't jump out of their folder**.

7. Visual Example (Practice)

```
<div class="box red">Red</div>  
<div class="box blue">Blue</div>  
<div class="box green">Green</div>
```

```
.box {  
  width: 150px;  
  height: 150px;  
  position: absolute;
```

👉 Order: **Blue > Green > Red**

- **z-index** decides which element is **on top**.
- Works only with positioned elements.
- Bigger **z-index** → closer to the user.
- **Stacking context** = a box where z-index works, but can't escape parent.

CSS 3


```
display: flex;
justify-content: center;
align-items: center;
min-height: 100vh;
flex-direction: column;
}

h2 {
margin-bottom: 20px;
color: #333;
}

.box {
width: 150px;
height: 150px;
position: absolute; /* important for z-index */
top: 100px;
left: 100px;
color: white;
font-size: 20px;
font-weight: bold;
display: flex;
justify-content: center;
align-items: center;
}

.red {
background: red;
z-index: 1;
}

.blue {
background: blue;
left: 130px;
top: 130px;
z-index: 3; /* highest */
}
```

```

}

.green {
  background: green;
  left: 160px;
  top: 160px;
  z-index: 2;
}

/* just for clarity */
.note {
  margin-top: 350px;
  background: white;
  padding: 10px;
  border-radius: 6px;
  box-shadow: 0 2px 6px rgba(0,0,0,0.1);
  font-size: 14px;
}
</style>
</head>
<body>

<h2>Z-index Demo</h2>

<div class="box red">Red (z=1)</div>
<div class="box blue">Blue (z=3)</div>
<div class="box green">Green (z=2)</div>

<div class="note">
  👉 In this example: Blue (z=3) is on top, Green (z=2) is in the middle,
  Red (z=1) is at the bottom.
  <br>Try changing the <b>z-index</b> values in CSS and see what happens!
</div>

```

```
</body>
</html>
```

▼ Class 4: Animations, Variables

▼ Transforms (scale, rotate, skew, translate)

Transforms are CSS properties that let you change the shape, position, and size of an element without affecting the layout of other elements around it. Think of them as special effects that you can apply to an object on a webpage. They are great for creating animations and interactive designs.

1. `translate()` (Move) ➡

The `translate()` function moves an element from its original position. It's like dragging an object to a new spot on the screen.

- `translate(x, y)` : Moves the element horizontally by `x` and vertically by `y`.
- `translateX(x)` : Only moves the element horizontally.
- `translateY(y)` : Only moves the element vertically.

Example Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Translate Example</title>
  <style>
    .box {
      width: 100px;
      height: 100px;
      background-color: lightblue;
      margin: 50px;
      text-align: center;
      line-height: 100px;
      font-weight: bold;
      transition: transform 0.5s ease;
```

```

    }
    .box:hover {
        transform: translate(50px, 20px);
    }
</style>
</head>
<body>
    <div class="box">Move Me</div>
</body>
</html>

```

2. `scale()` (Resize)

The `scale()` function changes the size of an element. It's like using a magnifying glass to either enlarge or shrink an object. A value greater than 1 makes the element bigger, while a value between 0 and 1 makes it smaller.

- `scale(x, y)` : Resizes the element horizontally by `x` and vertically by `y`.
- `scaleX(x)` : Only changes the width.
- `scaleY(y)` : Only changes the height.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Scale Example</title>
    <style>
        .box {
            width: 100px;
            height: 100px;
            background-color: lightgreen;
            margin: 50px;
            text-align: center;

```

```

        line-height: 100px;
        font-weight: bold;
        transition: transform 0.5s ease;
    }
    .box:hover {
        transform: scale(1.5);
    }
</style>
</head>
<body>
    <div class="box">Resize Me</div>
</body>
</html>

```

3. `rotate()` (Spin)

The `rotate()` function spins an element around its center. The rotation is measured in **degrees (deg)**. A positive value rotates the element clockwise, while a negative value rotates it counter-clockwise.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Rotate Example</title>
    <style>
        .box {
            width: 100px;
            height: 100px;
            background-color: lightcoral;
            margin: 50px;
            text-align: center;
            line-height: 100px;
            font-weight: bold;

```

```

        transition: transform 0.5s ease;
    }
    .box:hover {
        transform: rotate(45deg);
    }
</style>
</head>
<body>
    <div class="box">Rotate Me</div>
</body>
</html>

```

4. `skew()` (Slant)

The `skew()` function slants an element along its horizontal and vertical axes. It's like taking a rectangle and pushing one of its corners, turning it into a parallelogram. The values are also measured in degrees.

- `skew(x, y)` : Skews the element horizontally by `x` and vertically by `y`.
- `skewX(x)` : Only slants the element horizontally.
- `skewY(y)` : Only slants the element vertically.

Example Code:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Skew Example</title>
    <style>
        .box {
            width: 100px;
            height: 100px;
            background-color: lightsalmon;
            margin: 50px;
            text-align: center;

```

```
    line-height: 100px;
    font-weight: bold;
    transition: transform 0.5s ease;
  }
  .box:hover {
    transform: skew(20deg, 10deg);
  }
</style>
</head>
<body>
  <div class="box">Skew Me</div>
</body>
</html>
```

▼ 🎯 Final Project

Project Requirement Document (PRD)

Project Title: Mini Restaurant Website

Difficulty Level: Beginner – Intermediate

Tech Stack: HTML, CSS only

🎯 Project Goal

Create a **responsive restaurant landing page** that looks modern and clean.

Students will practice **layouts, styling, forms, transforms, media queries**.

🧱 Project Sections

1. Navbar

- Logo text → *Foodie's Hub*
- Menu links: Home, Menu, Specials, Contact

- Navbar should:
 - Have dark background
 - Use **Flexbox** for alignment
 - Stick to top while scrolling (**sticky + z-index**)
-

2. Hero Section

- Big background image of food
 - Text: **"Welcome to Foodie's Hub"** + tagline
 - One **CTA button**: *Order Now*
 - Text should be **centered** in the middle of the hero
-

3. Popular Dishes Section

- Title: **"🍽️ Popular Dishes"**
 - A **grid** with at least 4 dishes
 - Each dish card should have:
 - Image
 - Dish name
 - Price
 - On hover → card should slightly **scale up**
-

4. Special Offer Section

- A banner with gradient background
 - Text: **"🔥 Special Weekend Offer 🔥"**
 - Subtitle: *Get 20% off on orders above \$50*
-

5. Contact / Reservation Form

- Title: **"Book a Table"**

- Inputs:
 - Name
 - Email
 - Number of People
 - Special Requests (textarea)
 - Submit button: *Reserve Now*
 - Form should be centered and styled with a card-like background
-

6. Footer

- Dark background
- Centered text: © 2025 Foodie's Hub | All Rights Reserved

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Foodie's Hub</title>
  <style>
    * { margin: 0; padding: 0; box-sizing: border-box; }
    body { font-family: Arial, sans-serif; background: #f9f9f9; }

    /* Navbar */
    .navbar {
      background: #2c3e50;
      color: white;
      padding: 15px 30px;
      display: flex;
      justify-content: space-between;
      align-items: center;
```

```

    position: sticky;
    top: 0;
    z-index: 100;
}
.navbar .logo { font-size: 1.5rem; font-weight: bold; color: #e67e22; }
.navbar ul {
    list-style: none;
    display: flex;
    gap: 20px;
}
.navbar ul li a {
    color: white;
    text-decoration: none;
    font-weight: bold;
}
.navbar ul li a:hover { color: #e67e22; }

/* Hero */
.hero {
    height: 400px;
    background: url("https://images.pexels.com/photos/1640777/pexels-photo-1640777.jpeg") center/cover no-repeat;
    display: flex;
    justify-content: center;
    align-items: center;
    text-align: center;
    color: white;
}
.hero h1 { font-size: 3rem; margin-bottom: 10px; }
.hero p { font-size: 1.2rem; margin-bottom: 20px; }
.hero button {
    padding: 10px 20px;
    font-size: 1rem;
    border: none;
    background: #e67e22;
    color: white;
}

```

```

    border-radius: 5px;
    cursor: pointer;
}
.hero button:hover { background: #d35400; }

/* Popular Dishes */
.dishes {
    padding: 40px 20px;
    text-align: center;
}
.dishes h2 { margin-bottom: 20px; }
.grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 20px;
}
.card {
    background: white;
    padding: 15px;
    border-radius: 8px;
    box-shadow: 0 2px 6px rgba(0,0,0,0.1);
    transition: transform 0.3s;
}
.card img {
    width: 100%;
    height: 180px;
    object-fit: cover;
    border-radius: 6px;
}
.card h3 { margin: 10px 0; }
.card p { color: green; font-weight: bold; }
.card:hover { transform: scale(1.05); }

/* Special Offer */
.offer {
    background: linear-gradient(90deg, #e67e22, #d35400);

```

```

    color: white;
    text-align: center;
    padding: 40px 20px;
    margin: 40px 0;
}
.offer h2 { margin-bottom: 10px; }

/* Contact Form */
.contact {
    padding: 40px 20px;
    background: #fff;
    max-width: 600px;
    margin: auto;
    border-radius: 10px;
    box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}
.contact h2 { text-align: center; margin-bottom: 20px; }
.contact form { display: flex; flex-direction: column; gap: 15px; }
.contact input, .contact textarea {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
.contact button {
    padding: 10px;
    border: none;
    background: #e67e22;
    color: white;
    cursor: pointer;
    border-radius: 5px;
}
.contact button:hover { background: #d35400; }

/* Footer */
footer {
    background: #2c3e50;

```

```

    color: white;
    text-align: center;
    padding: 15px;
    margin-top: 20px;
  }

  /* Responsive */
  @media (max-width: 768px) {
    .hero h1 { font-size: 2rem; }
    .hero p { font-size: 1rem; }
  }
</style>
</head>
<body>

<!-- Navbar →
<nav class="navbar">
  <div class="logo">Foodie's Hub</div>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">Menu</a></li>
    <li><a href="#">Specials</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>

<!-- Hero →
<section class="hero">
  <div>
    <h1>Welcome to Foodie's Hub</h1>
    <p>Delicious meals made fresh just for you</p>
    <button>Order Now</button>
  </div>
</section>

<!-- Popular Dishes →

```

```

<section class="dishes">
  <h2>🍴 Popular Dishes</h2>
  <div class="grid">
    <div class="card">
      
      <h3>Grilled Burger</h3>
      <p>$12.99</p>
    </div>
    <div class="card">
      
      <h3>Pasta Alfredo</h3>
      <p>$10.99</p>
    </div>
    <div class="card">
      
      <h3>Cheese Pizza</h3>
      <p>$8.99</p>
    </div>
    <div class="card">
      
      <h3>Fresh Salad</h3>
      <p>$6.99</p>
    </div>
  </div>
</section>

<!-- Offer →
<section class="offer">
  <h2>🔥 Special Weekend Offer 🔥</h2>
  <p>Get 20% off on all orders above $50</p>
</section>

```

```
<!-- Contact →  
<section class="contact">  
  <h2>Book a Table</h2>  
  <form>  
    <input type="text" placeholder="Your Name" required>  
    <input type="email" placeholder="Your Email" required>  
    <input type="number" placeholder="Number of People" required>  
    <textarea rows="4" placeholder="Special Requests"></textarea>  
    <button type="submit">Reserve Now</button>  
  </form>  
</section>  
  
<!-- Footer →  
<footer>  
  <p>&copy; 2025 Foodie's Hub | All Rights Reserved</p>  
</footer>  
  
</body>  
</html>
```