CSS Training Material - 2 Days

Day 1: CSS Fundamentals & Layout Basics

Session 1: Getting Started with CSS

1.1 What is CSS and How to Include It

What: CSS (Cascading Style Sheets) is a language used to describe the visual presentation of HTML elements.

Why: HTML defines structure, but CSS makes it beautiful. Without CSS, websites would look like plain text documents.

How: Three ways to include CSS:

```
html

<!-- 1. Inline CSS - Style directly in HTML tag -->
<p style="color: blue; font-size: 16px;">This is inline CSS</p>

<!-- 2. Internal CSS - Style in <head> section -->
<head>
  <style>
    p {
      color: blue;
      font-size: 16px;
    }
  </style>
</head>

<!-- 3. External CSS - Separate .css file (RECOMMENDED) -->
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

Exercise 1.1: Create an HTML file with a heading and paragraph. Style them using all three methods:

- Inline: Make heading red
- Internal: Make paragraph green
- External: Create `styles.css` and make both elements bold

1.2 CSS Selectors, Combinators, Pseudo-classes

What: Selectors target HTML elements to apply styles.

Why: You need precise control over which elements receive which styles.

How:

```css
/* Basic Selectors */
p { color: black; }            /* Element selector */
.className { color: blue; }    /* Class selector */
#idName { color: red; }        /* ID selector */
* { margin: 0; }               /* Universal selector */

/* Combinators */
div p { color: green; }        /* Descendant (all p inside div) */
div > p { color: purple; }     /* Child (direct p children of div) */
h1 + p { margin-top: 0; }      /* Adjacent sibling */
h1 ~ p { color: gray; }        /* General sibling */

/* Pseudo-classes */
a:hover { color: red; }        /* Mouse over */
a:active { color: purple; }    /* Being clicked */
a:visited { color: gray; }     /* Already visited */
input:focus { border: 2px solid blue; } /* Element has focus */
li:first-child { font-weight: bold; }   /* First item */
li:nth-child(2) { color: red; }         /* Second item */
```

Exercise 1.2: Create a navigation menu with 5 links:

- Style all links blue
- On hover, make them red
- Make the first link bold
- Add a bottom border to the last link

1.3 BEM Naming Convention

What: BEM stands for Block, Element, Modifier - a methodology for naming CSS classes.

Why: Creates clear, maintainable, and scalable CSS code that's easy for teams to understand.

How:

```css
/* Block: Standalone component */
.card { }

/* Element: Part of block (use __) */
.card__title { }
.card__image { }
.card__button { }

/* Modifier: Variation of block or element (use --) */
.card--featured { }
.card__button--disabled { }
```

```html
<div class="card card--featured">
  <img class="card__image" src="photo.jpg" alt="Photo">
  <h3 class="card__title">Title Here</h3>
  <button class="card__button card__button--disabled">Click Me</button>
</div>
```

Exercise 1.3: Create a profile card using BEM:

- Block: `profile`
- Elements: `profile__avatar`, `profile__name`, `profile__bio`
- Modifier: `profile--premium` (add gold border)

Session 2: The Box Model & Units

2.1 The Box Model

What: Every HTML element is a rectangular box with content, padding, border, and margin.

Why: Understanding the box model is crucial for controlling spacing and layout.

How:

```css
```

```css
.box {
  width: 200px;          /* Content width */
  height: 100px;         /* Content height */
  padding: 20px;         /* Space inside border */
  border: 5px solid black; /* Border around padding */
  margin: 15px;          /* Space outside border */
}

/* Total width = width + padding-left + padding-right + border-left + border-right
   Total width = 200 + 20 + 20 + 5 + 5 = 250px */

/* Use box-sizing to include padding/border in width */
.box-better {
  box-sizing: border-box; /* Now width includes padding and border */
  width: 200px;          /* Total width = 200px */
  padding: 20px;
  border: 5px solid black;
}
```

Exercise 2.1: Create three boxes:

- Box 1: 300px wide, 20px padding, 3px border, 10px margin
- Box 2: Same but with `box-sizing: border-box`
- Compare their actual rendered widths using browser DevTools

---

2.2 CSS Units

What: Units define measurements for sizes, spacing, and positioning.

Why: Different units serve different purposes - some are fixed, others are flexible and responsive.

How:

```
css
```

```css
/* Absolute Units */
.fixed {
  width: 200px;          /* Pixels - fixed size */
}

/* Relative Units */
.relative-to-parent {
  width: 50%;            /* 50% of parent element */
  height: 80vh;          /* 80% of viewport height */
}

.relative-to-font {
  font-size: 16px;       /* Base size */
  padding: 1em;          /* 16px (1 × parent font-size) */
  margin: 2rem;          /* 32px (2 × root font-size) */
}

.viewport-units {
  width: 100vw;          /* 100% of viewport width */
  height: 100vh;         /* 100% of viewport height */
}
```

Key Differences:

- **px**: Fixed size, doesn't scale
- **em**: Relative to parent element's font-size
- **rem**: Relative to root element's font-size (more predictable)
- **%**: Relative to parent element
- **vh/vw**: Relative to viewport (browser window)

Exercise 2.2: Create a responsive container:

- Width: 90% on mobile (max-width: 1200px)
- Padding: 2rem
- Font-size: 1.2rem
- Child heading: 1.5em (relative to parent)

Session 3: Visual Styling

3.1 Colors, Backgrounds, Borders, Shadows

What: Properties that control the visual appearance of elements.

Why: These create the look and feel of your website.

How:

```css
.styled-box {
  /* Colors */
  color: #333333;                /* Text color (hex) */
  background-color: rgb(255, 200, 100); /* Background (rgb) */

  /* Backgrounds */
  background-image: url('pattern.png');
  background-size: cover;          /* Cover entire element */
  background-position: center;
  background-repeat: no-repeat;

  /* Shorthand */
  background: #f0f0f0 url('bg.jpg') no-repeat center/cover;

  /* Borders */
  border: 2px solid #333;          /* Shorthand */
  border-radius: 10px;             /* Rounded corners */
  border-top-left-radius: 20px;    /* Individual corner */

  /* Shadows */
  box-shadow: 5px 5px 15px rgba(0, 0, 0, 0.3); /* x y blur color */
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}

/* Multiple shadows */
.multi-shadow {
  box-shadow:
    0 2px 4px rgba(0,0,0,0.1),
    0 8px 16px rgba(0,0,0,0.1);
}
```

Exercise 3.1: Create a card with:

- Light gray background
- 1px gray border
- 8px rounded corners
- Subtle box shadow
- Hover effect: deeper shadow

Session 4: Display & Positioning

4.1 Display Property

What: Controls how an element is displayed in the document flow.

Why: Different elements need different layout behaviors.

How:

```
css
```

```css
/* Block Elements - Take full width, stack vertically */
.block {
  display: block;
  width: 100%;            /* Takes full width by default */
  margin: 10px 0;         /* Top and bottom margin work */
}

/* Inline Elements - Flow with text, only take needed width */
.inline {
  display: inline;
  /* width and height DON'T WORK */
  /* Top/bottom margin DON'T WORK */
  padding: 0 10px;        /* Left/right padding work */
}

/* Inline-block - Best of both worlds */
.inline-block {
  display: inline-block;
  width: 200px;           /* Width WORKS */
  height: 100px;          /* Height WORKS */
  margin: 10px;           /* All margins WORK */
}

/* None - Completely removed from document */
.hidden {
  display: none;          /* Element not rendered */
}
```

Exercise 4.1: Create a navigation bar with inline-block items:

- Each menu item should be inline-block
- Set width: 120px, height: 40px
- Add padding and margins
- Center text vertically and horizontally

4.2 Positioning

What: Controls how and where an element is positioned on the page.

Why: Needed for overlays, fixed headers, sticky navigation, and precise layouts.

How:

```css
/* Static - Default, follows normal document flow */
.static {
  position: static;
}

/* Relative - Positioned relative to its normal position */
.relative {
  position: relative;
  top: 20px;              /* Moves down 20px from normal position */
  left: 10px;             /* Moves right 10px from normal position */
  /* Original space is preserved */
}

/* Absolute - Positioned relative to nearest positioned ancestor */
.absolute {
  position: absolute;
  top: 0;                 /* Relative to positioned parent */
  right: 0;
  /* Removed from document flow, no space preserved */
}

/* Fixed - Positioned relative to viewport, stays during scroll */
.fixed {
  position: fixed;
  top: 0;                 /* Sticks to top of screen */
  width: 100%;
}

/* Sticky - Switches between relative and fixed */
.sticky {
  position: sticky;
  top: 0;                 /* Becomes fixed when scrolling past it */
}
```

Exercise 4.2: Create:

1. A fixed header that stays at top during scroll
2. A card with absolute-positioned badge in top-right corner
3. A sticky sidebar that fixes when scrolling past it

Hands-on Project (Day 1):

Build a Product Card

Requirements:

1. Use BEM naming convention
2. Include: image, title, description, price, button
3. Implement proper box model (padding, margins, borders)
4. Use rem units for fonts, px for borders
5. Add background colors and shadows
6. Position a "Sale" badge absolutely in the corner
7. Make the button change on hover

Case Study (Day 1):

Problem: A blog post has an image that needs a caption overlay at the bottom.

Solution Approach:

```
css
```

```css
.blog-post__image-container {
  position: relative;       /* Parent must be positioned */
  display: inline-block;
}

.blog-post__image {
  display: block;
  width: 100%;
}

.blog-post__caption {
  position: absolute;
  bottom: 0;                /* Stick to bottom */
  left: 0;
  right: 0;
  background: rgba(0,0,0,0.7);
  color: white;
  padding: 1rem;
}
```

Day 1 Assignment:

Create a landing page header with:

1. Fixed navigation bar at top
2. Logo on the left (inline-block)
3. Menu items on the right (inline-block)
4. Hero section with background image
5. Centered heading with text shadow
6. Call-to-action button with hover effect
7. Use BEM naming throughout
8. Proper box model implementation

Day 2: Modern CSS Layouts & Advanced Features

Session 5: Flexbox

5.1 Introduction to Flexbox

What: A one-dimensional layout system for arranging items in rows or columns.

Why: Makes creating flexible, responsive layouts much easier than traditional methods.

How:

```css
/* Container (Parent) Properties */
.flex-container {
  display: flex;                /* Enable flexbox */
  flex-direction: row;          /* row, row-reverse, column, column-reverse */
  justify-content: center;      /* Align on main axis: flex-start, flex-end, center, space-between, space-around */
  align-items: center;          /* Align on cross axis: stretch, flex-start, flex-end, center */
  flex-wrap: wrap;              /* nowrap, wrap, wrap-reverse */
  gap: 20px;                    /* Space between items */
}

/* Item (Child) Properties */
.flex-item {
  flex-grow: 1;                 /* Ability to grow (0 = no grow) */
  flex-shrink: 1;               /* Ability to shrink (0 = no shrink) */
  flex-basis: 200px;            /* Initial size */
  flex: 1 1 200px;              /* Shorthand: grow shrink basis */
  align-self: flex-end;         /* Override container alignment */
  order: 2;                     /* Change visual order */
}
```

Common Patterns:

```css
```

```css
/* Centered content */
.center-everything {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
}

/* Three equal columns */
.three-columns {
  display: flex;
  gap: 20px;
}
.three-columns > * {
  flex: 1;                    /* Each item takes equal space */
}

/* Responsive cards that wrap */
.card-container {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
}
.card {
  flex: 1 1 300px;            /* Grow, shrink, min-width 300px */
}
```

Exercise 5.1: Create a navigation bar:

- Logo on the left
- Menu items on the right
- Centered vertically
- Space between logo and menu
- Make it responsive (wrap on small screens)

Session 6: CSS Grid

6.1 Introduction to CSS Grid

What: A two-dimensional layout system for creating complex layouts with rows and columns.

Why: Perfect for page layouts, galleries, and any content requiring precise 2D positioning.

How:

```css
/* Container Properties */
.grid-container {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;    /* 3 columns: fixed, flex, flex */
  grid-template-rows: auto 1fr auto;       /* 3 rows */
  gap: 20px;                               /* Space between cells */

  /* Named areas (optional but powerful) */
  grid-template-areas:
    "header header header"
    "sidebar main main"
    "footer footer footer";
}

/* Item Properties */
.grid-item {
  grid-column: 1 / 3;                      /* Span from column 1 to 3 */
  grid-row: 2 / 4;                         /* Span from row 2 to 4 */

  /* Or use named areas */
  grid-area: header;
}

/* Modern responsive grid */
.auto-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
}
```

Common Patterns:

```css
```

```css
/* Simple equal columns */
.three-columns {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 20px;
}

/* Responsive without media queries! */
.responsive-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 20px;
}

/* Page layout with named areas */
.page-layout {
  display: grid;
  grid-template-columns: 250px 1fr;
  grid-template-rows: 80px 1fr 60px;
  grid-template-areas:
    "header header"
    "sidebar content"
    "footer footer";
  min-height: 100vh;
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.content { grid-area: content; }
.footer { grid-area: footer; }
```

Exercise 6.1: Create a dashboard layout:

- Header across the top (full width)
- Sidebar on the left (250px)
- Main content area (takes remaining space)
- Footer across the bottom (full width)

Session 7: Responsive Design

7.1 Media Queries

What: CSS rules that apply only when certain conditions (like screen width) are met.

Why: Makes your site look good on all devices - phones, tablets, desktops.

How:

```css
```

```css
/* Base styles (mobile or desktop, depending on approach) */
.container {
  width: 100%;
  padding: 15px;
}

/* Media query for tablets and up */
@media (min-width: 768px) {
  .container {
    width: 750px;
    margin: 0 auto;
    padding: 20px;
  }
}

/* Media query for desktops */
@media (min-width: 1200px) {
  .container {
    width: 1170px;
    padding: 30px;
  }
}

/* Common breakpoints */
/* Phone: < 576px */
/* Tablet: 576px - 768px */
/* Desktop: 768px - 1200px */
/* Large desktop: > 1200px */

/* Other useful media queries */
@media (orientation: landscape) {
  /* Styles for landscape mode */
}

@media (prefers-color-scheme: dark) {
  /* Styles for users who prefer dark mode */
}

@media print {
  /* Styles for printing */
}
```

Exercise 7.1: Create a grid of cards:

- 1 column on mobile (< 576px)
- 2 columns on tablet (576px - 992px)
- 3 columns on desktop (> 992px)
- Different padding at each breakpoint

7.2 Mobile-First vs Desktop-First

What: Two approaches to responsive design - starting from mobile or desktop.

Why: Mobile-first is generally preferred because it's easier to add complexity than remove it.

How:

```
css
```

```css
/* MOBILE-FIRST (Recommended) */
/* Start with mobile, add complexity for larger screens */

/* Base styles = mobile */
.nav {
  flex-direction: column;
}

/* Enhance for larger screens */
@media (min-width: 768px) {
  .nav {
    flex-direction: row;
  }
}



/* DESKTOP-FIRST */
/* Start with desktop, simplify for smaller screens */

/* Base styles = desktop */
.nav {
  flex-direction: row;
}

/* Simplify for smaller screens */
@media (max-width: 767px) {
  .nav {
    flex-direction: column;
  }
}
```

Benefits of Mobile-First:

- Forces you to prioritize content
- Better performance (mobile loads less CSS)
- Easier to progressively enhance
- Aligns with "mobile-first" development mindset

Exercise 7.2: Refactor a desktop navigation to mobile-first:

- Start: Vertical hamburger menu

- Tablet: Horizontal menu
- Desktop: Horizontal with dropdowns

Session 8: CSS Variables & Themes

8.1 CSS Variables (Custom Properties)

What: Reusable values that can be defined once and used throughout your CSS.

Why: Makes maintaining and updating styles much easier, enables theme switching.

How:

```css
/* Define variables in :root (global) */
:root {
  --primary-color: #007bff;
  --secondary-color: #6c757d;
  --font-main: 'Arial', sans-serif;
  --spacing-unit: 8px;
  --border-radius: 4px;
}

/* Use variables with var() */
.button {
  background-color: var(--primary-color);
  font-family: var(--font-main);
  padding: calc(var(--spacing-unit) * 2);
  border-radius: var(--border-radius);
}

/* Local scope variables */
.card {
  --card-padding: 20px;
  padding: var(--card-padding);
}

/* Fallback values */
.element {
  color: var(--text-color, #333); /* Use #333 if --text-color not defined */
}
```

Exercise 8.1: Create a design system with CSS variables:

- Define colors (primary, secondary, success, danger)
- Define spacing scale (xs, sm, md, lg, xl)
- Define typography (font families, sizes)
- Use them throughout a component

8.2 Multiple Themes

What: Using CSS variables to create switchable color schemes.

Why: Users expect dark mode and theme customization options.

How:

```css
/* Light theme (default) */
:root {
  --bg-color: #ffffff;
  --text-color: #333333;
  --border-color: #dddddd;
}

/* Dark theme */
[data-theme="dark"] {
  --bg-color: #1a1a1a;
  --text-color: #ffffff;
  --border-color: #444444;
}

/* Apply variables */
body {
  background-color: var(--bg-color);
  color: var(--text-color);
  transition: background-color 0.3s, color 0.3s;
}

.card {
  border: 1px solid var(--border-color);
}
```

```html
<!-- Switch themes with JavaScript -->
<button onclick="toggleTheme()">Toggle Theme</button>

<script>
function toggleTheme() {
  const root = document.documentElement;
  const current = root.getAttribute('data-theme');
  root.setAttribute('data-theme', current === 'dark' ? 'light' : 'dark');
}
</script>
```

Exercise 8.2: Create a theme switcher:

- Light and dark themes
- Variables for all colors
- Smooth transition between themes
- Save preference to localStorage

Session 9: Animations & Transitions

9.1 CSS Transitions

What: Smooth changes between CSS property values over a duration.

Why: Makes interactions feel smooth and polished, provides visual feedback.

How:

```css
```

```css
/* Transition single property */
.button {
  background-color: blue;
  transition: background-color 0.3s ease;
}

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

/* Transition multiple properties */
.card {
  transform: scale(1);
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.card:hover {
  transform: scale(1.05);
  box-shadow: 0 8px 16px rgba(0,0,0,0.2);
}

/* Transition all properties (use carefully) */
.element {
  transition: all 0.3s ease;
}

/* Timing functions */
.timing-demo {
  transition-timing-function: ease;       /* Slow start and end */
  transition-timing-function: linear;     /* Constant speed */
  transition-timing-function: ease-in;    /* Slow start */
  transition-timing-function: ease-out;   /* Slow end */
  transition-timing-function: ease-in-out; /* Slow start and end */
}
```

Exercise 9.1: Create interactive buttons:

- Smooth color change on hover
- Scale up slightly on hover
- Add shadow that grows on hover

- Use 0.3s duration with ease-in-out

9.2 CSS Animations

What: More complex, multi-step animations defined with keyframes.

Why: Create engaging, attention-grabbing effects and loading indicators.

How:

```css
```

```css
/* Define keyframes */
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Apply animation */
.fade-in {
  animation: fadeIn 0.6s ease-out;
}

/* Multiple steps */
@keyframes pulse {
  0% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.1);
  }
  100% {
    transform: scale(1);
  }
}

.pulsing {
  animation: pulse 2s infinite;
}

/* Animation shorthand */
.animated {
  animation-name: slideIn;
  animation-duration: 1s;
  animation-timing-function: ease;
  animation-delay: 0.5s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
```

```css
  /* Shorthand */
  animation: slideIn 1s ease 0.5s infinite alternate;
}

/* Useful animations */
@keyframes spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}

@keyframes bounce {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-20px); }
}
```

Exercise 9.2: Create:

- A loading spinner (rotating circle)
- A bouncing ball animation
- A fade-in effect for page elements
- A sliding notification that enters from the right

Session 10: Typography & Pseudo-elements

10.1 Typography

What: Styling text for readability and visual hierarchy.

Why: Good typography makes content readable and establishes visual hierarchy.

How:

```css
```

```css
/* Font properties */
.text {
  font-family: 'Helvetica', Arial, sans-serif;
  font-size: 16px;
  font-weight: 400;            /* 100-900, or normal, bold */
  font-style: italic;
  line-height: 1.6;            /* 1.6 × font-size */
  letter-spacing: 0.5px;
  word-spacing: 2px;
  text-transform: uppercase;     /* uppercase, lowercase, capitalize */
  text-decoration: underline;
  text-align: center;          /* left, right, center, justify */
}

/* Typography scale */
:root {
  --font-size-xs: 0.75rem;     /* 12px */
  --font-size-sm: 0.875rem;    /* 14px */
  --font-size-base: 1rem;      /* 16px */
  --font-size-lg: 1.125rem;    /* 18px */
  --font-size-xl: 1.25rem;     /* 20px */
  --font-size-2xl: 1.5rem;     /* 24px */
  --font-size-3xl: 1.875rem;   /* 30px */
  --font-size-4xl: 2.25rem;    /* 36px */
}

/* Responsive typography */
.heading {
  font-size: 1.5rem;
}

@media (min-width: 768px) {
  .heading {
    font-size: 2rem;
  }
}

/* Fluid typography (scales with viewport) */
.fluid-text {
  font-size: clamp(1rem, 2vw, 2rem); /* min, preferred, max */
}
```

Exercise 10.1: Create a blog post with proper typography:

- Define a type scale
- Set appropriate line-height (1.6 for body)
- Create heading hierarchy (h1-h6)
- Add proper spacing between elements
- Make font-size responsive

10.2 Pseudo-elements

What: Virtual elements that can be styled without adding HTML.

Why: Add decorative content, icons, or effects without cluttering HTML.

How:

```
css
```

```css
/* ::before - Insert content before element */
.icon::before {
  content: "★ ";
  color: gold;
}

/* ::after - Insert content after element */
.required::after {
  content: " *";
  color: red;
}

/* Create decorative elements */
.fancy-heading::before {
  content: "";
  display: block;
  width: 50px;
  height: 3px;
  background: blue;
  margin-bottom: 10px;
}

/* Quote marks */
.quote::before {
  content: """;
  font-size: 3em;
  color: #ccc;
}

.quote::after {
  content: """;
  font-size: 3em;
  color: #ccc;
}

/* Clearfix (old but classic use) */
.clearfix::after {
  content: "";
  display: table;
  clear: both;
}

/* Tooltip */
```

```css
.tooltip {
  position: relative;
}

.tooltip::after {
  content: attr(data-tooltip);
  position: absolute;
  bottom: 100%;
  left: 50%;
  transform: translateX(-50%);
  background: black;
  color: white;
  padding: 5px 10px;
  border-radius: 4px;
  opacity: 0;
  transition: opacity 0.3s;
}

.tooltip:hover::after {
  opacity: 1;
}
```

Exercise 10.2: Create:

- A list where each item has a custom bullet (using ::before)
- A link with an external link icon after it (::after)
- A heading with a decorative underline (::after)
- A tooltip that appears on hover

Session 11: CSS Frameworks

11.1 Bootstrap Basics

What: A popular CSS framework with pre-built components and utility classes.

Why: Speeds up development with ready-made, responsive components.

How:

```html
```

```html
<!-- Include Bootstrap -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

<!-- Grid System (12 columns) -->
<div class="container">
  <div class="row">
    <div class="col-md-4">Column 1</div>
    <div class="col-md-4">Column 2</div>
    <div class="col-md-4">Column 3</div>
  </div>
</div>

<!-- Utility Classes -->
<div class="mt-3 p-4 bg-primary text-white rounded">
  <!--
    mt-3: margin-top
    p-4: padding all sides
    bg-primary: blue background
    text-white: white text
    rounded: border-radius
  -->
  Content here
</div>

<!-- Components -->
<button class="btn btn-primary">Click Me</button>
<div class="alert alert-success">Success message!</div>
<div class="card">
  <div class="card-body">
    <h5 class="card-title">Card Title</h5>
    <p class="card-text">Card content here.</p>
  </div>
</div>
```

Common Bootstrap Classes:

- **Spacing**: `m-{size}` (margin), `p-{size}` (padding), `mt-3`, `pb-2`
- **Colors**: `bg-primary`, `text-danger`, `border-success`
- **Display**: `d-flex`, `d-none`, `d-md-block`
- **Flexbox**: `justify-content-center`, `align-items-center`
- **Grid**: `col-{breakpoint}-{size}`, `col-md-6`

Exercise 11.1: Using Bootstrap, create:

- A responsive navbar that collapses on mobile
- A 3-column card layout on desktop, 1 column on mobile
- A form with styled inputs and buttons
- Use utility classes for spacing and colors

11.2 Tailwind CSS Basics

What: A utility-first CSS framework where you build designs by combining small utility classes.

Why: Highly customizable, no pre-designed components to override, faster development.

How:

```
html
```

```
<!-- Include Tailwind (for development) -->
<script src="https://cdn.tailwindcss.com"></script>

<!-- Utility Classes Approach -->
<div class="flex items-center justify-between p-4 bg-blue-500 text-white rounded-lg shadow-lg">
  <h1 class="text-2xl font-bold">Title</h1>
  <button class="px-4 py-2 bg-white text-blue-500 rounded hover:bg-gray-100 transition">
    Click Me
  </button>
</div>

<!-- Responsive Design -->
<div class="w-full md:w-1/2 lg:w-1/3">
  <!-- Full width on mobile, half on tablet, third on desktop -->
</div>

<!-- Card Example -->
<div class="max-w-sm mx-auto bg-white rounded-xl shadow-md overflow-hidden">
  <img class="w-full h-48 object-cover" src="image.jpg" alt="Card">
  <div class="p-6">
    <h2 class="text-xl font-bold mb-2">Card Title</h2>
    <p class="text-gray-600">Card description goes here.</p>
    <button class="mt-4 px-4 py-2 bg-blue-500 text-white rounded hover:bg-blue-600">
      Learn More
    </button>
  </div>
</div>
```

Common Tailwind Classes:

- **Spacing**: `p-4`, `m-2`, `px-6`, `mt-8`, `space-x-4`
- **Layout**: `flex`, `grid`, `block`, `inline-block`, `hidden`
- **Sizing**: `w-full`, `h-64`, `max-w-md`, `min-h-screen`
- **Colors**: `bg-blue-500`, `text-gray-700`, `border-red-300`
- **Typography**: `text-xl`, `font-bold`, `leading-relaxed`
- **Effects**: `shadow-lg`, `rounded-md`, `hover:opacity-80`

Bootstrap vs Tailwind:

Bootstrap	Tailwind
Component-based	Utility-based
Pre-designed look	Build custom designs
Less HTML clutter	More classes in HTML
Easier for beginners	More flexible

Exercise 11.2: Using Tailwind, create:

- A hero section with centered text and background image
- A grid of 3 feature cards with icons
- A contact form with styled inputs
- Hover effects on buttons and cards

Hands-on Project (Day 2):

Build a Complete Responsive Landing Page

Requirements:

1. **Header**: Fixed navigation with logo and menu (Flexbox)
2. **Hero Section**: Full-height section with centered content
3. **Features**: 3-column grid (CSS Grid) that stacks on mobile
4. **Testimonials**: Flexbox card layout
5. **Contact Form**: Styled form with focus states
6. **Footer**: Multi-column footer (Grid)

Technical Requirements:

- Mobile-first responsive design
- CSS variables for theming (light/dark toggle)
- Smooth transitions on interactive elements
- Proper typography hierarchy
- BEM naming convention
- At least 3 media queries

- Use both Flexbox and Grid appropriately
- Add loading animation for page elements

Case Study (Day 2):

Problem: Building a Dashboard Layout

Requirements:

- Fixed header across top
- Fixed sidebar on left
- Scrollable main content area
- Card-based widgets in the main area
- Responsive: sidebar collapses on mobile

Solution:

```
css
```

```css
/* Layout Structure */
.dashboard {
  display: grid;
  grid-template-columns: 250px 1fr;
  grid-template-rows: 60px 1fr;
  grid-template-areas:
    "header header"
    "sidebar main";
  height: 100vh;
}

.dashboard__header {
  grid-area: header;
  position: sticky;
  top: 0;
  background: white;
  border-bottom: 1px solid #ddd;
  z-index: 100;
}

.dashboard__sidebar {
  grid-area: sidebar;
  overflow-y: auto;
  background: #f5f5f5;
  border-right: 1px solid #ddd;
}

.dashboard__main {
  grid-area: main;
  overflow-y: auto;
  padding: 20px;
}

/* Widget Grid */
.dashboard__widgets {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  gap: 20px;
}

/* Mobile Responsive */
@media (max-width: 768px) {
  .dashboard {
```

```
    grid-template-columns: 1fr;
    grid-template-areas:
      "header"
      "main";
  }

  .dashboard__sidebar {
    position: fixed;
    left: -250px;
    transition: left 0.3s;
  }

  .dashboard__sidebar--open {
    left: 0;
  }
}
```

Key Concepts Applied:

- CSS Grid for overall layout
- Sticky positioning for header
- Flexbox for widgets
- Responsive design with mobile hamburger menu
- CSS variables for theme colors

Use Cases & Real-World Applications:

Use Case 1: E-commerce Product Gallery

Challenge: Display products in a responsive grid that looks good on all devices.

Solution:

```
css
```

```css
.product-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
  gap: 24px;
  padding: 20px;
}

.product-card {
  background: white;
  border-radius: 8px;
  overflow: hidden;
  transition: transform 0.3s, box-shadow 0.3s;
}

.product-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 10px 20px rgba(0,0,0,0.1);
}
```

Use Case 2: Pricing Table

Challenge: Create a pricing table where one plan is highlighted.

Solution:

```
css
```

```css
.pricing-table {
  display: flex;
  gap: 20px;
  flex-wrap: wrap;
  justify-content: center;
}

.pricing-card {
  flex: 1 1 300px;
  max-width: 350px;
  padding: 30px;
  border: 2px solid #ddd;
  border-radius: 8px;
}

.pricing-card--featured {
  border-color: var(--primary-color);
  transform: scale(1.05);
  box-shadow: 0 10px 30px rgba(0,0,0,0.2);
}
```

Use Case 3: Modal/Overlay

Challenge: Create a modal that appears over content.

Solution:

```
css
```

```css
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: rgba(0,0,0,0.7);
  display: flex;
  align-items: center;
  justify-content: center;
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.3s;
}

.modal-overlay--active {
  opacity: 1;
  pointer-events: auto;
}

.modal {
  background: white;
  padding: 30px;
  border-radius: 8px;
  max-width: 500px;
  transform: scale(0.9);
  transition: transform 0.3s;
}

.modal-overlay--active .modal {
  transform: scale(1);
}
```

Day 2 Comprehensive Assignment:

Build a Portfolio Website

Create a complete portfolio website with the following sections:

Requirements:

1. Navigation (20 points)

- Fixed header that becomes smaller on scroll
- Logo on left, menu on right (Flexbox)
- Smooth scroll to sections
- Mobile hamburger menu
- Active state for current section

2. Hero Section (15 points)

- Full viewport height
- Centered content (Flexbox)
- Background image with overlay
- Animated heading (fade in and slide up)
- Call-to-action button with hover effect

3. About Section (10 points)

- Two-column layout (Grid on desktop, stack on mobile)
- Profile image with shadow
- Proper typography hierarchy

4. Projects Section (20 points)

- Responsive grid of project cards (CSS Grid)
- Auto-fit with minimum 300px width
- Hover effects (scale + shadow)
- ::before pseudo-element for overlay on hover
- Project tags using inline-block

5. Skills Section (10 points)

- Progress bars with animation
- Use CSS Grid or Flexbox for layout
- Icon + label for each skill

6. Contact Form (15 points)

- Styled input fields with focus states

- Validation styling (::after for error messages)
- Submit button with loading animation
- Use pseudo-classes for validation

7. Footer (5 points)

- Multi-column layout (Grid)
- Social media icons
- Copyright notice

8. Theme Switcher (5 points)

- Light and dark mode toggle
- CSS variables for all colors
- Smooth transitions between themes
- LocalStorage persistence (JavaScript)

Technical Requirements:

- Mobile-first responsive design
- At least 3 breakpoints (576px, 768px, 1200px)
- BEM naming convention throughout
- CSS variables for colors, spacing, typography
- Use both Flexbox and Grid appropriately
- Semantic HTML5 elements
- Proper use of pseudo-classes and pseudo-elements
- Transitions on interactive elements
- At least 2 CSS animations
- Proper box model usage (box-sizing: border-box)
- Typography scale with rem units
- Accessible (proper contrast, focus states)

Bonus Challenges:

- Parallax scrolling effect on hero section
- Animated menu icon (hamburger to X)
- Scroll progress indicator
- Lazy loading animation for sections
- Implement with either Bootstrap OR Tailwind framework
- Custom cursor effect
- Smooth page transitions

Additional Exercises & Mini-Projects:

Exercise Set 1: Flexbox Challenges

1. Create a holy grail layout (header, 3 columns, footer)
2. Build a navigation menu with dropdowns
3. Create a card layout that wraps responsively
4. Build a photo gallery with varying image sizes

Exercise Set 2: Grid Challenges

1. Create a magazine-style layout with overlapping elements
2. Build a calendar grid (7 columns × 5 rows)
3. Create a masonry-style Pinterest layout
4. Build a responsive dashboard with widgets

Exercise Set 3: Animation Challenges

1. Create a loading spinner with 3 dots bouncing
2. Build an animated hamburger menu
3. Create a typing effect with CSS
4. Build a sliding carousel with CSS only

Exercise Set 4: Real-World Components

1. Build a dropdown menu with smooth animations
2. Create an accordion component

3. Build a tabs component
4. Create a tooltip component
5. Build a notification toast
6. Create an image modal/lightbox

Assessment Checklist:

Day 1 Skills:

- [ ] Can include CSS using all three methods
- [ ] Understands and uses selectors correctly
- [ ] Implements BEM naming convention
- [ ] Understands the box model
- [ ] Uses appropriate CSS units
- [ ] Can style colors, backgrounds, and shadows
- [ ] Understands display property differences
- [ ] Can use positioning correctly

Day 2 Skills:

- [ ] Can create layouts with Flexbox
- [ ] Can create layouts with CSS Grid
- [ ] Implements responsive design with media queries
- [ ] Understands mobile-first approach
- [ ] Uses CSS variables effectively
- [ ] Creates themes with CSS variables
- [ ] Implements smooth transitions
- [ ] Creates keyframe animations
- [ ] Applies proper typography
- [ ] Uses pseudo-elements creatively
- [ ] Can work with Bootstrap OR Tailwind

Final Project Evaluation Rubric:

Code Quality (30%)

- Clean, organized CSS
- Proper BEM naming

- Efficient selectors
- Good commenting

Responsive Design (25%)

- Works on all screen sizes
- Proper breakpoints
- Mobile-first approach
- Fluid typography

Visual Design (20%)

- Good use of color
- Proper spacing
- Typography hierarchy
- Visual consistency

Functionality (15%)

- Interactive elements work
- Smooth transitions
- No broken layouts
- Cross-browser compatible

Advanced Techniques (10%)

- CSS variables usage
- Animations/transitions
- Pseudo-elements
- Modern CSS features

Resources for Continued Learning:

Documentation:

- MDN Web Docs (CSS Reference)
- CSS-Tricks (Articles & Guides)

- Can I Use (Browser Compatibility)

Practice:

- Frontend Mentor (Real-world projects)
- CSS Battle (CSS challenges)
- Codepen (Inspiration & practice)

Tools:

- Chrome DevTools
- Flexbox Froggy (Learn Flexbox)
- Grid Garden (Learn Grid)
- CSS Grid Generator
- Gradient generators
- Box shadow generators

Best Practices:

- Use CSS reset or normalize.css
- Implement box-sizing: border-box globally
- Use relative units for responsive design
- Organize CSS with clear sections/comments
- Test in multiple browsers
- Optimize for performance (minimize repaints)
- Use CSS variables for maintainability
- Follow accessibility guidelines

Quick Reference Guide:

Flexbox Cheatsheet:

```
css
```

```
/* Container */
display: flex;
flex-direction: row | column;
justify-content: flex-start | center | space-between;
align-items: stretch | center | flex-start;
flex-wrap: nowrap | wrap;
gap: 20px;

/* Items */
flex: 1; /* grow, shrink, basis */
align-self: auto | center;
order: 0;
```

Grid Cheatsheet:

```
css

/* Container */
display: grid;
grid-template-columns: 1fr 1fr 1fr;
grid-template-rows: auto;
gap: 20px;
grid-template-areas: "header header" "sidebar main";

/* Items */
grid-column: 1 / 3;
grid-row: 1 / 2;
grid-area: header;
```

Media Query Cheatsheet:

```
css

/* Mobile First */
@media (min-width: 576px) { /* tablet */ }
@media (min-width: 768px) { /* desktop */ }
@media (min-width: 1200px) { /* large desktop */ }
```

End of Training Material

Good luck with your CSS journey! Remember: Practice is key. Build projects, experiment, and don't be afraid to break things!