CSS Mastery for Svelte Developers

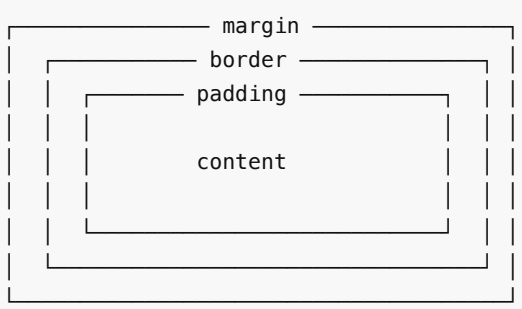
> *A thorough course on modern CSS — from layout fundamentals to building a full design system — all within the context of Svelte's scoped styling model.*

Table of Contents

Chapter 1 — CSS Fundamentals Refresher

1.1 The Box Model

Every element is a box with four layers:



Always use `box-sizing: border-box`:

```css
*,
*::before,
*::after {
  box-sizing: border-box;
}
```

1.2 Selectors and Specificity

Specificity hierarchy (lowest to highest):

1. Type selectors: `p`, `div`, `h1` — (0, 0, 1)
2. Class selectors: `.card`, `.active` — (0, 1, 0)
3. ID selectors: `#main` — (1, 0, 0)
4. Inline styles — (1, 0, 0, 0)
5. `!important` — overrides everything (avoid!)

1.3 The Cascade and Inheritance

```css
/* The cascade determines which rule wins */
.card { color: blue; }          /* specificity: 0,1,0 */
div.card { color: red; }        /* specificity: 0,1,1 — wins */

/* Inheritance flows from parent to child */
body { font-family: system-ui; }  /* all children inherit */
```

1.4 Display and Positioning

```css
/* Display */
.block    { display: block; }
.inline   { display: inline; }
.flex     { display: flex; }
.grid     { display: grid; }
.none     { display: none; }

/* Position */
.relative { position: relative; }
.absolute { position: absolute; }  /* relative to nearest positioned ancestor */
.fixed    { position: fixed; }     /* relative to viewport */
.sticky   { position: sticky; top: 0; }
```

1.5 Modern Reset

```css
/* Minimal modern reset */
*,
*::before,
*::after {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

html {
  -moz-text-size-adjust: none;
  -webkit-text-size-adjust: none;
  text-size-adjust: none;
}

body {
  min-height: 100vh;
  line-height: 1.6;
}

img, picture, video, canvas, svg {
  display: block;
  max-width: 100%;
}

input, button, textarea, select {
  font: inherit;
}
```

Exercise 1

1. Create an HTML page with the modern reset applied.
2. Build a simple card component using only box-model properties.
3. Experiment with `position: sticky` on a header.

Chapter 2 — Flexbox

2.1 Flex Container

```
.container {
  display: flex;
  flex-direction: row;        /* row | row-reverse | column | column-reverse */
  flex-wrap: wrap;             /* nowrap | wrap | wrap-reverse */
  justify-content: center;    /* flex-start | flex-end | center | space-between | space-around |
space-evenly */
  align-items: stretch;       /* flex-start | flex-end | center | stretch | baseline */
  gap: 1rem;                  /* row and column gap */
}
```

2.2 Flex Items

```
.item {
  flex-grow: 1;     /* how much to grow relative to siblings */
  flex-shrink: 1;   /* how much to shrink relative to siblings */
  flex-basis: auto;  /* initial size before growing/shrinking */

  /* Shorthand */
  flex: 1;          /* flex: 1 1 0% */
  flex: 0 0 300px;  /* fixed width, no grow, no shrink */

  align-self: center;  /* override align-items for this item */
  order: 2;            /* visual order (default: 0) */
}
```

2.3 Common Flexbox Patterns

Navbar:

```
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 2rem;
}
```

Centering anything:

```
.center {
  display: flex;
  justify-content: center;
  align-items: center;
```

```
  min-height: 100vh;
}
```

Holy Grail Footer:

```
.page {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}

.main { flex: 1; }
/* footer is pushed to the bottom automatically */
```

Card row with equal heights:

```
.card-row {
  display: flex;
  gap: 1.5rem;
}

.card {
  flex: 1;
  display: flex;
  flex-direction: column;
}

.card-body { flex: 1; }
```

2.4 Flexbox in Svelte

```
<div class="stack">
  <div>Item 1</div>
  <div>Item 2</div>
  <div>Item 3</div>
</div>

<style>
  .stack {
    display: flex;
    flex-direction: column;
    gap: 1rem;
  }
</style>
```

Exercise 2

1. Build a responsive navbar with a logo on the left and links on the right.
2. Create a card layout where three cards are always equal height.
3. Build a sticky footer layout using `flex-direction: column`.

Chapter 3 — CSS Grid

3.1 Grid Container

```css
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: auto 1fr auto;
  gap: 1.5rem;
}
```

3.2 Defining Tracks

```css
/* Fixed widths */
grid-template-columns: 200px 1fr 200px;

/* Fractional units */
grid-template-columns: 1fr 2fr 1fr;

/* Repeat */
grid-template-columns: repeat(4, 1fr);

/* Auto-fill and auto-fit */
grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

/* Mixed */
grid-template-columns: 250px repeat(3, 1fr) 250px;
```

3.3 Placing Items

```css
.item {
  grid-column: 1 / 3;       /* span columns 1 through 2 */
  grid-row: 1 / 2;          /* span row 1 */
}

/* Shorthand */
.hero {
  grid-column: 1 / -1;      /* full width */
}

/* Span syntax */
.wide {
  grid-column: span 2;
}
```

3.4 Named Grid Areas

```css
.layout {
  display: grid;
  grid-template-areas:
    "header  header  header"
    "sidebar content aside"
    "footer  footer  footer";
```

```
  grid-template-columns: 250px 1fr 200px;
  grid-template-rows: auto 1fr auto;
  min-height: 100vh;
}

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

3.5 Alignment in Grid

```
.grid {
  justify-items: center;    /* horizontal alignment of items within cells */
  align-items: center;      /* vertical alignment of items within cells */
  justify-content: center;  /* horizontal alignment of the grid within container */
  align-content: center;    /* vertical alignment of the grid within container */
}

.item {
  justify-self: end;        /* override for single item */
  align-self: start;
}
```

3.6 Responsive Grid Without Media Queries

```
.auto-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(min(300px, 100%), 1fr));
  gap: 1.5rem;
}
```

3.7 Subgrid

```
.parent {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 1rem;
}

.child {
  display: grid;
  grid-column: span 3;
  grid-template-columns: subgrid; /* inherit parent column tracks */
}
```

Exercise 3

1. Build a full-page layout using named grid areas (header, sidebar, main, footer).
2. Create a responsive card grid using `auto-fit` and `minmax`.
3. Build a dashboard layout with cards that span multiple columns and rows.

Chapter 4 — Custom Properties (CSS Variables)

4.1 Defining and Using Variables

```
:root {
  --color-primary: #3b82f6;
  --color-secondary: #10b981;
  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --spacing-lg: 2rem;
  --radius: 8px;
  --shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

.card {
  padding: var(--spacing-md);
  border-radius: var(--radius);
  box-shadow: var(--shadow);
}

.btn-primary {
  background: var(--color-primary);
}
```

4.2 Fallback Values

```
.element {
  color: var(--text-color, #333);
  padding: var(--custom-padding, var(--spacing-md));
}
```

4.3 Scoped Variables

Variables cascade and can be overridden in any scope:

```
:root {
  --btn-bg: #3b82f6;
  --btn-color: white;
}

.btn-danger {
  --btn-bg: #ef4444;
}

.btn {
  background: var(--btn-bg);
  color: var(--btn-color);
}
```

4.4 Dark Mode with Variables

```css
:root {
  --bg: #ffffff;
  --text: #1a1a1a;
  --surface: #f5f5f5;
  --border: #e0e0e0;
}

:root[data-theme="dark"] {
  --bg: #0a0a0a;
  --text: #e0e0e0;
  --surface: #1a1a1a;
  --border: #333333;
}

body {
  background: var(--bg);
  color: var(--text);
}
```

4.5 CSS Variables in Svelte Components

```svelte
<Card --card-bg="#f0f9ff" --card-radius="16px" />

<!-- Card.svelte -->
<div class="card">
  {@render children()}
</div>

<style>
  .card {
    background: var(--card-bg, white);
    border-radius: var(--card-radius, 8px);
    padding: 1.5rem;
  }
</style>
```

4.6 Dynamic Variables with JavaScript

```svelte
<script>
  let hue = $state(200);
</script>

<div class="theme-preview" style:--hue={hue}>
  <input type="range" min="0" max="360" bind:value={hue} />
  <div class="color-box">Hue: {hue}</div>
</div>

<style>
  .color-box {
    background: hsl(var(--hue), 70%, 50%);
    color: white;
    padding: 2rem;
    border-radius: 8px;
```

```
  }
</style>
```

Exercise 4

1. Create a complete colour and spacing token system using CSS custom properties.
2. Implement a dark/light theme toggle.
3. Build a Svelte component that accepts CSS variables as style props.

Chapter 5 — Responsive Design

5.1 The Viewport Meta Tag

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

5.2 Media Queries

```
/* Mobile first — styles apply to all sizes, then override for larger */
.container {
  padding: 1rem;
}

@media (min-width: 640px) {
  .container { padding: 2rem; }
}

@media (min-width: 1024px) {
  .container { padding: 3rem; max-width: 1200px; margin: 0 auto; }
}
```

5.3 Common Breakpoints

```
:root {
  --bp-sm: 640px;
  --bp-md: 768px;
  --bp-lg: 1024px;
  --bp-xl: 1280px;
  --bp-2xl: 1536px;
}

/* Note: CSS variables cannot be used directly in media queries.
   Use the raw values or a preprocessor. */
@media (min-width: 640px)  { /* sm  */ }
@media (min-width: 768px)  { /* md  */ }
@media (min-width: 1024px) { /* lg  */ }
@media (min-width: 1280px) { /* xl  */ }
@media (min-width: 1536px) { /* 2xl */ }
```

5.4 Container Queries

Style based on the container size, not the viewport:

```css
.card-container {
  container-type: inline-size;
  container-name: card;
}

@container card (min-width: 400px) {
  .card {
    display: flex;
    gap: 1.5rem;
  }
}

@container card (min-width: 700px) {
  .card {
    grid-template-columns: 1fr 2fr;
  }
}
```

5.5 Responsive Images

```html
<img
  srcset="image-400.jpg 400w, image-800.jpg 800w, image-1200.jpg 1200w"
  sizes="(max-width: 640px) 100vw, (max-width: 1024px) 50vw, 33vw"
  src="image-800.jpg"
  alt="Descriptive alt text"
/>
```

```css
img {
  max-width: 100%;
  height: auto;
}
```

5.6 Responsive Patterns

Stack to horizontal:

```css
.layout {
  display: flex;
  flex-direction: column;
  gap: 1rem;
}

@media (min-width: 768px) {
  .layout {
    flex-direction: row;
  }
}
```

Hide/Show elements:

```css
.desktop-only { display: none; }
.mobile-only  { display: block; }
```

```css
@media (min-width: 768px) {
  .desktop-only { display: block; }
  .mobile-only  { display: none; }
}
```

Exercise 5

1. Build a responsive page layout that stacks on mobile and shows a sidebar on desktop.
2. Implement a responsive navigation that collapses into a hamburger menu on mobile.
3. Use container queries to make a card component adapt to its container width.

Chapter 6 — Fluid Typography

6.1 The Problem with Fixed Sizes

Fixed font sizes create jarring jumps at breakpoints:

```css
/* Bad — jumpy */
h1 { font-size: 1.5rem; }
@media (min-width: 768px) { h1 { font-size: 2.5rem; } }
@media (min-width: 1200px) { h1 { font-size: 3.5rem; } }
```

6.2 The `clamp()` Function

`clamp(min, preferred, max)` enables smooth scaling:

```css
h1 {
  font-size: clamp(1.5rem, 4vw + 0.5rem, 3.5rem);
}
```

- **min:** Never smaller than `1.5rem`
- **preferred:** Scales with viewport (`4vw + 0.5rem`)
- **max:** Never larger than `3.5rem`

6.3 A Complete Fluid Type Scale

```css
:root {
  --text-xs:   clamp(0.75rem,  0.7rem  + 0.25vw, 0.875rem);
  --text-sm:   clamp(0.875rem, 0.8rem  + 0.35vw, 1rem);
  --text-base: clamp(1rem,     0.9rem  + 0.5vw,  1.125rem);
  --text-lg:   clamp(1.125rem, 1rem    + 0.6vw,  1.25rem);
  --text-xl:   clamp(1.25rem,  1rem    + 1.2vw,  1.75rem);
  --text-2xl:  clamp(1.5rem,   1.1rem  + 1.8vw,  2.25rem);
  --text-3xl:  clamp(1.875rem, 1.2rem  + 2.8vw,  3rem);
  --text-4xl:  clamp(2.25rem,  1.3rem  + 4vw,    4rem);
}
```

6.4 Fluid Spacing

Apply the same principle to spacing:

```css
:root {
  --space-xs:  clamp(0.25rem, 0.2rem + 0.25vw, 0.5rem);
  --space-sm:  clamp(0.5rem,  0.4rem + 0.5vw,  0.75rem);
```

```
  --space-md:  clamp(1rem,    0.8rem + 1vw,     1.5rem);
  --space-lg:  clamp(1.5rem,  1rem   + 2vw,     2.5rem);
  --space-xl:  clamp(2rem,    1.2rem + 3vw,     4rem);
  --space-2xl: clamp(3rem,    1.5rem + 5vw,     6rem);
}
```

6.5 Line Length for Readability

```
.prose {
  max-width: 65ch;  /* ~65 characters per line */
  margin: 0 auto;
  font-size: var(--text-base);
  line-height: 1.7;
}
```

6.6 Fluid Typography in Svelte

```
<article class="prose">
  <h1>{data.title}</h1>
  <p>{data.content}</p>
</article>

<style>
  .prose {
    max-width: 65ch;
    font-size: clamp(1rem, 0.9rem + 0.5vw, 1.125rem);
    line-height: 1.7;
  }

  .prose h1 {
    font-size: clamp(1.875rem, 1.2rem + 2.8vw, 3rem);
    line-height: 1.2;
    margin-bottom: clamp(1rem, 0.8rem + 1vw, 1.5rem);
  }
</style>
```

Exercise 6

1. Create a fluid type scale with at least 6 sizes using `clamp()`.
2. Apply fluid spacing to a page layout.
3. Build a blog-style prose layout with optimal line length and fluid sizes.

Chapter 7 — Scoped Styles in Svelte

7.1 How Scoped Styles Work

Svelte generates unique class attributes (e.g., `svelte-1abc2de`) and appends them to selectors, ensuring styles do not leak between components:

```
<!-- Input -->
<p>Hello</p>
<style>
  p { color: teal; }
```

```
</style>

<!-- Output (simplified) -->
<p class="svelte-1abc2de">Hello</p>
<style>
  p.svelte-1abc2de { color: teal; }
</style>
```

7.2 The `:global()` Modifier

Escape scoping for specific selectors:

```
<style>
  /* Scoped to this component */
  p { color: teal; }

  /* Applies globally */
  :global(body) { margin: 0; }

  /* Global within a scoped context */
  .wrapper :global(a) { color: var(--link-color); }
</style>
```

7.3 CSS Modules Alternative

Svelte's scoping is effectively CSS Modules built in. There is no need for separate CSS Module configuration.

7.4 Passing Styles to Children

Use CSS custom properties to pass styling information from parent to child:

```
<!-- Parent -->
<ChildComponent --accent-color="tomato" --padding="2rem" />

<!-- ChildComponent.svelte -->
<div class="child">
  {@render children()}
</div>

<style>
  .child {
    color: var(--accent-color, black);
    padding: var(--padding, 1rem);
  }
</style>
```

7.5 Dynamic Styles

```
<script>
  let color = $state('#3b82f6');
  let size = $state(16);
</script>

<p style:color style:font-size="{size}px">
```

```
  Dynamic styling
</p>
```

7.6 Class Directives

```
<script>
  let active = $state(false);
  let status = $state('success');
</script>

<div class:active class:highlighted={status === 'success'}>
  Content
</div>

<style>
  .active { font-weight: bold; }
  .highlighted { background: #d1fae5; }
</style>
```

7.7 Global Stylesheets

For truly global styles, import a CSS file in your layout:

```
<!-- src/routes/+layout.svelte -->
<script>
  import '../styles/global.css';
  let { children } = $props();
</script>

{@render children()}
```

Exercise 7

1. Create two components with the same element selectors and verify styles do not leak.
2. Build a theme provider component that passes CSS variables to children.
3. Use the `class:` directive to toggle visual states.

Chapter 8 — CSS Animation

8.1 Transitions

```
.button {
  background: var(--color-primary);
  color: white;
  padding: 0.75rem 1.5rem;
  border: none;
  border-radius: var(--radius);
  cursor: pointer;
  transition: background 200ms ease, transform 150ms ease;
}

.button:hover {
  background: var(--color-primary-dark);
```

```
  transform: translateY(-1px);
}

.button:active {
  transform: translateY(0);
}
```

8.2 Keyframe Animations

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

.card {
  animation: fadeIn 600ms ease-out both;
}

/* Staggered animation */
.card:nth-child(1) { animation-delay: 0ms; }
.card:nth-child(2) { animation-delay: 100ms; }
.card:nth-child(3) { animation-delay: 200ms; }
```

8.3 Animation Properties

```
.element {
  animation-name: slideIn;
  animation-duration: 500ms;
  animation-timing-function: cubic-bezier(0.16, 1, 0.3, 1);
  animation-delay: 100ms;
  animation-iteration-count: 1;       /* or infinite */
  animation-direction: normal;        /* or reverse, alternate */
  animation-fill-mode: both;          /* or forwards, backwards, none */
  animation-play-state: running;      /* or paused */

  /* Shorthand */
  animation: slideIn 500ms cubic-bezier(0.16, 1, 0.3, 1) 100ms both;
}
```

8.4 Svelte Transitions

Svelte has a built-in transition system:

```
<script>
  import { fade, fly, slide, scale } from 'svelte/transition';
  import { quintOut } from 'svelte/easing';
```

```
  let visible = $state(true);
</script>

<button onclick={() => visible = !visible}>Toggle</button>

{#if visible}
  <div transition:fly={{ y: 50, duration: 400, easing: quintOut }}>
    I fly in and out!
  </div>
{/if}
```

8.5 In and Out Transitions

```
{#if visible}
  <div
    in:fly={{ y: 50, duration: 400 }}
    out:fade={{ duration: 200 }}
  >
    Different enter/exit animations
  </div>
{/if}
```

8.6 Custom Transitions

```
<script>
  function typewriter(node, { speed = 50 }) {
    const text = node.textContent;
    const duration = text.length * speed;

    return {
      duration,
      tick: (t) => {
        const i = Math.trunc(text.length * t);
        node.textContent = text.slice(0, i);
      }
    };
  }
</script>

{#if visible}
  <p transition:typewriter={{ speed: 30 }}>
    This text types itself out letter by letter.
  </p>
{/if}
```

8.7 Respecting `prefers-reduced-motion`

```
@media (prefers-reduced-motion: reduce) {
  *,
  *::before,
  *::after {
    animation-duration: 0.01ms !important;
    animation-iteration-count: 1 !important;
```

```
    transition-duration: 0.01ms !important;
  }
}
```

Exercise 8

1. Create a button with smooth hover and active transitions.
2. Build a card list with staggered fade-in animations using `@keyframes`.
3. Implement a Svelte `fly` transition on a notification toast.
4. Write a custom Svelte transition that scales and rotates.

Chapter 9 — Accessibility and CSS

9.1 Focus Styles

Never remove focus outlines without providing an alternative:

```
/* Bad */
*:focus { outline: none; }

/* Good — custom focus style */
:focus-visible {
  outline: 3px solid var(--color-primary);
  outline-offset: 2px;
  border-radius: 2px;
}

/* Hide outline for mouse clicks, show for keyboard */
:focus:not(:focus-visible) {
  outline: none;
}
```

9.2 Visually Hidden Content

Content for screen readers that is visually hidden:

```
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}
```

```
<button>
  <svg aria-hidden="true"><!-- icon --></svg>
  <span class="sr-only">Close menu</span>
</button>
```

9.3 Colour Contrast

WCAG 2.1 requires:

- **AA:** 4.5:1 for normal text, 3:1 for large text
- **AAA:** 7:1 for normal text, 4.5:1 for large text

```
/* Example: good contrast */
:root {
  --text-primary: #1a1a2e;   /* dark on light: ~15:1 */
  --bg-primary: #ffffff;
}
```

9.4 Touch Targets

Minimum 44x44 CSS pixels for touch targets:

```
button,
a,
input,
select {
  min-height: 44px;
  min-width: 44px;
}
```

9.5 Reduced Motion

```
@media (prefers-reduced-motion: reduce) {
  *, *::before, *::after {
    animation-duration: 0.01ms !important;
    animation-iteration-count: 1 !important;
    transition-duration: 0.01ms !important;
    scroll-behavior: auto !important;
  }
}
```

9.6 Colour Scheme Preferences

```
@media (prefers-color-scheme: dark) {
  :root {
    --bg: #0a0a0a;
    --text: #e0e0e0;
  }
}
```

9.7 Skip Links

```
<a href="#main-content" class="skip-link">Skip to main content</a>

<main id="main-content">
  <!-- Page content -->
</main>
```

```
<style>
  .skip-link {
    position: absolute;
    top: -100%;
    left: 1rem;
    padding: 0.5rem 1rem;
    background: var(--color-primary);
    color: white;
    border-radius: 0 0 4px 4px;
    z-index: 999;
    transition: top 200ms ease;
  }

  .skip-link:focus {
    top: 0;
  }
</style>
```

9.8 Accessible Form Styling

```
<div class="field">
  <label for="email">Email</label>
  <input
    id="email"
    type="email"
    aria-describedby="email-error"
    aria-invalid={hasError}
  />
  {#if hasError}
    <p id="email-error" class="error" role="alert">
      Please enter a valid email.
    </p>
  {/if}
</div>

<style>
  .field { display: flex; flex-direction: column; gap: 0.25rem; }

  label { font-weight: 600; font-size: var(--text-sm); }

  input {
    padding: 0.75rem 1rem;
    border: 2px solid var(--border);
    border-radius: var(--radius);
    font: inherit;
  }

  input:focus-visible {
    outline: 3px solid var(--color-primary);
    outline-offset: 1px;
  }

  input[aria-invalid="true"] {
    border-color: var(--color-danger);
```

```
  }

  .error {
    color: var(--color-danger);
    font-size: var(--text-sm);
  }
</style>
```

Exercise 9

1. Add `:focus-visible` styles to all interactive elements in a page.
2. Create a `.sr-only` utility and use it for icon-only buttons.
3. Implement a skip link that becomes visible on focus.
4. Audit your colour palette for WCAG AA contrast compliance.

Chapter 10 — Building a Design System

10.1 Design Tokens

Define your entire visual language as CSS custom properties:

```
/* src/styles/tokens.css */

:root {
  /* Colors */
  --color-gray-50:  #f9fafb;
  --color-gray-100: #f3f4f6;
  --color-gray-200: #e5e7eb;
  --color-gray-300: #d1d5db;
  --color-gray-400: #9ca3af;
  --color-gray-500: #6b7280;
  --color-gray-600: #4b5563;
  --color-gray-700: #374151;
  --color-gray-800: #1f2937;
  --color-gray-900: #111827;

  --color-primary-50:  #eff6ff;
  --color-primary-100: #dbeafe;
  --color-primary-500: #3b82f6;
  --color-primary-600: #2563eb;
  --color-primary-700: #1d4ed8;

  --color-success-500: #22c55e;
  --color-warning-500: #f59e0b;
  --color-danger-500:  #ef4444;

  /* Semantic tokens */
  --bg-primary:   var(--color-gray-50);
  --bg-surface:   #ffffff;
  --text-primary: var(--color-gray-900);
  --text-muted:   var(--color-gray-500);
  --border:       var(--color-gray-200);

  /* Typography */
  --font-sans:  'Inter', system-ui, -apple-system, sans-serif;
  --font-mono:  'JetBrains Mono', 'Fira Code', monospace;
```

```
  --text-xs:   clamp(0.75rem,  0.7rem  + 0.25vw, 0.875rem);
  --text-sm:   clamp(0.875rem, 0.8rem  + 0.35vw, 1rem);
  --text-base: clamp(1rem,     0.9rem  + 0.5vw,  1.125rem);
  --text-lg:   clamp(1.125rem, 1rem    + 0.6vw,  1.25rem);
  --text-xl:   clamp(1.25rem,  1rem    + 1.2vw,  1.75rem);
  --text-2xl:  clamp(1.5rem,   1.1rem  + 1.8vw,  2.25rem);
  --text-3xl:  clamp(1.875rem, 1.2rem  + 2.8vw,  3rem);

  /* Spacing */
  --space-1:  0.25rem;
  --space-2:  0.5rem;
  --space-3:  0.75rem;
  --space-4:  1rem;
  --space-6:  1.5rem;
  --space-8:  2rem;
  --space-12: 3rem;
  --space-16: 4rem;

  /* Border Radius */
  --radius-sm:   4px;
  --radius-md:   8px;
  --radius-lg:   12px;
  --radius-xl:   16px;
  --radius-full: 9999px;

  /* Shadows */
  --shadow-sm:  0 1px 2px rgba(0, 0, 0, 0.05);
  --shadow-md:  0 4px 6px -1px rgba(0, 0, 0, 0.1);
  --shadow-lg:  0 10px 15px -3px rgba(0, 0, 0, 0.1);
  --shadow-xl:  0 20px 25px -5px rgba(0, 0, 0, 0.1);

  /* Transitions */
  --duration-fast:   150ms;
  --duration-normal: 250ms;
  --duration-slow:   400ms;
  --ease-out:        cubic-bezier(0.16, 1, 0.3, 1);
}

/* Dark mode */
:root[data-theme="dark"] {
  --bg-primary:   var(--color-gray-900);
  --bg-surface:   var(--color-gray-800);
  --text-primary: var(--color-gray-50);
  --text-muted:   var(--color-gray-400);
  --border:       var(--color-gray-700);
}
```

10.2 Base Component: Button

```
<!-- src/lib/components/Button.svelte -->
<script lang="ts">
  import type { Snippet } from 'svelte';
  import type { HTMLButtonAttributes } from 'svelte/elements';
```

```
  interface Props extends HTMLButtonAttributes {
    variant?: 'primary' | 'secondary' | 'ghost' | 'danger';
    size?: 'sm' | 'md' | 'lg';
    children: Snippet;
  }

  let { variant = 'primary', size = 'md', children, ...rest }: Props = $props();
</script>

<button class="btn btn-{variant} btn-{size}" {...rest}>
  {@render children()}
</button>

<style>
  .btn {
    display: inline-flex;
    align-items: center;
    justify-content: center;
    gap: var(--space-2);
    border: 2px solid transparent;
    border-radius: var(--radius-md);
    font-weight: 600;
    cursor: pointer;
    transition:
      background var(--duration-fast) var(--ease-out),
      transform var(--duration-fast) var(--ease-out);
  }

  .btn:active { transform: translateY(1px); }

  /* Sizes */
  .btn-sm { padding: var(--space-1) var(--space-3); font-size: var(--text-sm); }
  .btn-md { padding: var(--space-2) var(--space-4); font-size: var(--text-base); }
  .btn-lg { padding: var(--space-3) var(--space-6); font-size: var(--text-lg); }

  /* Variants */
  .btn-primary {
    background: var(--color-primary-500);
    color: white;
  }
  .btn-primary:hover { background: var(--color-primary-600); }

  .btn-secondary {
    background: transparent;
    color: var(--text-primary);
    border-color: var(--border);
  }
  .btn-secondary:hover { background: var(--color-gray-100); }

  .btn-ghost {
    background: transparent;
    color: var(--text-primary);
  }
  .btn-ghost:hover { background: var(--color-gray-100); }

  .btn-danger {
```

```
    background: var(--color-danger-500);
    color: white;
  }
  .btn-danger:hover { background: #dc2626; }

  /* Focus */
  .btn:focus-visible {
    outline: 3px solid var(--color-primary-500);
    outline-offset: 2px;
  }
</style>
```

10.3 Input Component

```
<!-- src/lib/components/Input.svelte -->
<script lang="ts">
  import type { HTMLInputAttributes } from 'svelte/elements';

  interface Props extends HTMLInputAttributes {
    label: string;
    error?: string;
  }

  let { label, error, id, ...rest }: Props = $props();

  const inputId = id ?? label.toLowerCase().replace(/\s+/g, '-');
</script>

<div class="field">
  <label for={inputId}>{label}</label>
  <input
    {id}
    aria-invalid={!!error}
    aria-describedby={error ? `${inputId}-error` : undefined}
    {...rest}
  />
  {#if error}
    <p id="{inputId}-error" class="error" role="alert">{error}</p>
  {/if}
</div>

<style>
  .field {
    display: flex;
    flex-direction: column;
    gap: var(--space-1);
  }

  label {
    font-size: var(--text-sm);
    font-weight: 600;
    color: var(--text-primary);
  }

  input {
```

```
    padding: var(--space-2) var(--space-3);
    border: 2px solid var(--border);
    border-radius: var(--radius-md);
    font: inherit;
    font-size: var(--text-base);
    background: var(--bg-surface);
    color: var(--text-primary);
    transition: border-color var(--duration-fast) var(--ease-out);
  }

  input:focus-visible {
    outline: 3px solid var(--color-primary-500);
    outline-offset: 1px;
  }

  input[aria-invalid="true"] {
    border-color: var(--color-danger-500);
  }

  .error {
    color: var(--color-danger-500);
    font-size: var(--text-sm);
  }
</style>
```

10.4 Card Component

```
<!-- src/lib/components/Card.svelte -->
<script lang="ts">
  import type { Snippet } from 'svelte';

  interface Props {
    variant?: 'elevated' | 'outlined' | 'filled';
    padding?: 'sm' | 'md' | 'lg';
    children: Snippet;
  }

  let { variant = 'elevated', padding = 'md', children }: Props = $props();
</script>

<div class="card card-{variant} pad-{padding}">
  {@render children()}
</div>

<style>
  .card {
    border-radius: var(--radius-lg);
    background: var(--bg-surface);
  }

  .card-elevated { box-shadow: var(--shadow-md); }
  .card-outlined { border: 2px solid var(--border); }
  .card-filled   { background: var(--color-gray-100); }

  .pad-sm { padding: var(--space-3); }
```

```
  .pad-md { padding: var(--space-6); }
  .pad-lg { padding: var(--space-8); }
</style>
```

10.5 Utility Classes

```
/* src/styles/utilities.css */

/* Layout */
.flex        { display: flex; }
.flex-col    { flex-direction: column; }
.items-center { align-items: center; }
.justify-between { justify-content: space-between; }
.gap-2       { gap: var(--space-2); }
.gap-4       { gap: var(--space-4); }
.gap-6       { gap: var(--space-6); }

/* Spacing */
.p-2  { padding: var(--space-2); }
.p-4  { padding: var(--space-4); }
.p-6  { padding: var(--space-6); }
.mt-4 { margin-top: var(--space-4); }
.mb-4 { margin-bottom: var(--space-4); }

/* Typography */
.text-sm   { font-size: var(--text-sm); }
.text-base { font-size: var(--text-base); }
.text-lg   { font-size: var(--text-lg); }
.text-xl   { font-size: var(--text-xl); }
.font-bold { font-weight: 700; }
.text-muted { color: var(--text-muted); }

/* Width */
.w-full    { width: 100%; }
.max-w-md  { max-width: 28rem; }
.max-w-lg  { max-width: 32rem; }
.max-w-xl  { max-width: 36rem; }
.max-w-prose { max-width: 65ch; }
.mx-auto   { margin-left: auto; margin-right: auto; }

/* Visibility */
.sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}
```

10.6 Putting It All Together

```
<!-- Example page using the design system -->
<script lang="ts">
  import Button from '$lib/components/Button.svelte';
  import Input from '$lib/components/Input.svelte';
  import Card from '$lib/components/Card.svelte';
</script>

<div class="page">
  <Card>
    <h2>Create Account</h2>
    <form class="form">
      <Input label="Full Name" placeholder="Ada Lovelace" />
      <Input label="Email" type="email" placeholder="ada@example.com" />
      <Input label="Password" type="password" />
      <Button variant="primary" size="lg" type="submit">
        Sign Up
      </Button>
    </form>
  </Card>
</div>

<style>
  .page {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: var(--space-4);
    background: var(--bg-primary);
  }

  h2 {
    font-size: var(--text-2xl);
    margin-bottom: var(--space-6);
  }

  .form {
    display: flex;
    flex-direction: column;
    gap: var(--space-4);
    min-width: 320px;
  }
</style>
```

Exercise 10

1. Create a complete design tokens file with colours, typography, spacing, and shadows.
2. Build a component library with at least: Button, Input, Card, Badge, and Modal.
3. Implement dark mode using semantic tokens.
4. Document your design system with a Storybook-style showcase page that displays all components in all variants.
5. Ensure every component passes a WCAG AA accessibility audit.

Appendix — CSS Properties Quick Reference

Layout

Property	Common Values
`display`	`flex`, `grid`, `block`, `inline-flex`, `none`
`position`	`relative`, `absolute`, `fixed`, `sticky`
`gap`	`1rem`, `var(--space-4)`
`justify-content`	`center`, `space-between`, `flex-start`
`align-items`	`center`, `stretch`, `flex-start`

Typography

Property	Common Values
`font-size`	`clamp(...)`, `var(--text-base)`
`line-height`	`1.2` (headings), `1.6` (body)
`font-weight`	`400`, `600`, `700`
`letter-spacing`	`-0.02em`, `0.05em`

Colour and Effects

Property	Common Values
`color`	`var(--text-primary)`
`background`	`var(--bg-surface)`
`box-shadow`	`var(--shadow-md)`
`opacity`	`0` to `1`
`border-radius`	`var(--radius-md)`

Congratulations! You now have a complete CSS toolkit for building beautiful, accessible, and maintainable Svelte applications.