Comprehensive CSS Lecture Notes

A Complete Guide to Cascading Style Sheets (2025 Edition)

Duration: 3 Hours
Level: Intermediate to Advanced
Instructor Notes: This document covers CSS from fundamentals to modern advanced techniques used in contemporary web development.

Table of Contents

Module 1: CSS Fundamentals

Duration: 45 minutes

1.1 Introduction to CSS

What is CSS?

CSS stands for **Cascading Style Sheets**. It is a style sheet language used to describe how HTML elements are displayed on screen, in print, or in other media formats. CSS provides the visual presentation layer of web applications, separating content (HTML) from presentation (CSS).

Why CSS Matters:

- Separates content from presentation
- Enables responsive design across devices
- Improves maintainability of code
- Enhances user experience through animations and transitions
- Reduces file size through code reusability

The Evolution of CSS:

Version	Year	Key Features
CSS 1	1996	Basic styling properties

Version	Year	Key Features
CSS 2	1998	Positioning, z-index, media types
CSS 2.1	2011	Bug fixes and clarifications
CSS 3+	2013+	Modules approach (Grid, Flexbox, Animations, etc.)
CSS 2025	2025	Container Queries, Cascade Layers, Advanced Functions

1.2 Adding CSS to HTML

There are three primary methods to add CSS to your HTML documents:

Method 1: Inline Styles

```
<p style="color: blue; font-size: 16px;">This is a styled paragraph.</p>
```

Advantages: Quick for testing
Disadvantages: Not maintainable, mixes presentation with content, hard to reuse

Method 2: Internal Stylesheet

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p {
            color: blue;
            font-size: 16px;
        }
    </style>
</head>
<body>
    <p>This is a styled paragraph.</p>
</body>
</html>
```

Advantages: Centralized styles, easier than inline
Disadvantages: Not reusable across multiple pages

Method 3: External Stylesheet (BEST PRACTICE)

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <p>This is a styled paragraph.</p>
</body>
</html>
```

File: styles.css

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

Advantages: Reusable, maintainable, clean separation of concerns
Disadvantages: Requires additional HTTP request (minor with modern optimization)

1.3 CSS Syntax and Structure

A CSS rule consists of a **selector** and a **declaration block**:

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

Example:

```
h1 {
    color: navy;
    font-size: 28px;
    margin-bottom: 20px;
}
```

Key Components:

- **Selector:** Targets HTML elements (h1 in example)
- **Property:** The aspect being styled (color, font-size)
- **Value:** The setting for that property (navy, 28px)
- **Declaration:** A complete property-value pair
- **Declaration Block:** All properties within curly braces

1.4 CSS Measurement Units

CSS supports various units for specifying sizes and distances:

Absolute Units

- **px (pixels):** Fixed size, 1/96th of an inch
- **pt (points):** Typography unit, 1/72nd of an inch
- **cm, mm, in:** Physical measurements

Relative Units (PREFERRED)

- **em:** Relative to parent element's font size

```
p { font-size: 16px; }
span { font-size: 1.5em; } /* 24px */
```

- **rem:** Relative to root element's font size

```
html { font-size: 16px; }
p { margin: 1rem; } /* 16px */
```

- **% (percentage):** Relative to parent element

```
.container { width: 1200px; }
.child { width: 50%; } /* 600px */
```

- **vh/vw:** Relative to viewport height/width

```
.fullscreen { height: 100vh; } /* Full viewport height */
```

Best Practice: Use `rem` for consistency, `em` for component-relative sizing, `%` for responsive layouts, and avoid fixed `px` except for borders and very small elements.

1.5 Colors in CSS

Color Representations

1. Named Colors:

```
p { color: blue; }
h1 { background-color: lightblue; }
```

2. Hexadecimal (Hex):

```
p { color: #0000FF; } /* Blue */
h1 { color: #FF5733; } /* Orange-red */
```

3. RGB (Red, Green, Blue):

```
p { color: rgb(0, 0, 255); } /* Blue */
h1 { color: rgb(255, 87, 51); } /* Orange-red */
```

4. RGBA (RGB with Alpha/Opacity):

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

5. HSL (Hue, Saturation, Lightness):

```
p { color: hsl(240, 100%, 50%); } /* Blue */
h1 { color: hsl(10, 100%, 60%); } /* Orange-red */
```

6. HSLA (HSL with Alpha):

```
p { color: hsla(240, 100%, 50%, 0.5); } /* Semi-transparent blue */
```

7. Modern Color Spaces (2025):

```
/* HWB (Hue, Whiteness, Blackness) */
p { color: hwb(240 0% 0%); }

/* LAB Color Space */
p { color: lab(50% 20 -30); }

/* LCH Color Space */
p { color: lch(50% 30 240); }
```

1.6 Comments in CSS

CSS comments are used for documentation and explanation:

```
/* This is a single-line comment */

/*
 * This is a multi-line comment.
 * It spans multiple lines.
 * Useful for detailed explanations.
 */

h1 { color: blue; } /* Inline comment */
```

Best Practice: Use comments to explain the purpose of styles, especially for complex rules or non-obvious choices.

Module 2: Selectors, Cascade, and Specificity

Duration: 45 minutes

2.1 CSS Selectors

Selectors are patterns used to target HTML elements for styling. Mastering selectors is crucial for efficient CSS writing.

2.1.1 Element Selectors

```
/* Targets all <p> elements */
p {
    color: blue;
}

/* Targets all <h1> elements */
h1 {
    font-size: 28px;
}
```

2.1.2 Class Selectors

```
/* Targets elements with class="highlight" */
.highlight {
    background-color: yellow;
}

/* Multiple elements with same class */
<p class="highlight">This is highlighted</p>
<span class="highlight">This too</span>
```

2.1.3 ID Selectors

```
/* Targets element with id="header" */
#header {
    background-color: navy;
    color: white;
}

/* HTML */
<div id="header">Header Content</div>
```

Important: ID selectors are more specific than class selectors and should be used sparingly.

2.1.4 Attribute Selectors

```
/* Targets input elements with type="text" */
input[type="text"] {
    border: 1px solid black;
}

/* Targets links with href containing "example" */
a[href*="example"] {
```

```
    color: red;
}

/* Targets images with alt attribute */
img[alt] {
    border: 2px solid blue;
}

/* Targets links starting with "https" */
a[href^="https"] {
    color: green;
}

/* Targets links ending with ".pdf" */
a[href$=".pdf"] {
    color: orange;
}
```

2.1.5 Pseudo-Classes

Pseudo-classes represent a specific state of an element:

```
/* Link states */
a:link { color: blue; }      /* Unvisited link */
a:visited { color: purple; } /* Visited link */
a:hover { color: red; }      /* Mouse hovering */
a:active { color: orange; }  /* During click */

/* Form states */
input:focus { border-color: blue; }
input:disabled { opacity: 0.5; }
input:checked { background: green; }

/* Structural pseudo-classes */
li:first-child { font-weight: bold; }
li:last-child { border-bottom: none; }
li:nth-child(2n) { background-color: #f0f0f0; } /* Even items */
li:nth-of-type(3) { color: red; }
p:not(.special) { color: gray; }
```

2.1.6 Pseudo-Elements

Pseudo-elements create virtual elements that don't exist in the HTML:

```
/* First line of text */
p::first-line {
    font-weight: bold;
    text-transform: uppercase;
}

/* First letter */
```

```
p::first-letter {
    font-size: 2em;
    font-weight: bold;
}

/* Before and after content */
h2::before {
    content: "➤ ";
    color: blue;
}

h2::after {
    content: " ✓";
    color: green;
}

/* Selection styling */
p::selection {
    background-color: yellow;
    color: black;
}
```

2.1.7 Combinators

Combinators combine multiple selectors to target elements based on relationships:

```
/* Descendant combinator (space) */
div p {
    color: blue; /* All <p> inside <div>, at any depth */
}

/* Child combinator (>) */
div > p {
    color: blue; /* Only direct <p> children of <div> */
}

/* Adjacent sibling combinator (+) */
h1 + p {
    margin-top: 0; /* <p> immediately after <h1> */
}

/* General sibling combinator (~) */
h1 ~ p {
    color: gray; /* All <p> that are siblings of <h1> */
}
```

2.2 The Cascade

The cascade is the mechanism by which CSS resolves conflicting styles:

Rule 1: Source Order

When multiple rules have the same specificity, the **last one wins**:

```
p { color: blue; }
p { color: red; } /* This wins */
```

Rule 2: Importance

The `!important` declaration overrides normal cascade (use sparingly):

```
p { color: blue; }
p { color: red !important; } /* This wins */
```

Rule 3: Specificity

More specific selectors override less specific ones:

```
p { color: blue; }           /* Specificity: 1 */
.highlight { color: red; }   /* Specificity: 10 */
#main p { color: green; }    /* Specificity: 101 */

/* In this case, #main p has highest specificity and wins */
```

2.3 Specificity

Specificity is a weight system that determines which CSS rule applies:

Specificity Calculation

```
Specificity = (ID selectors) (Class selectors) (Element selectors)
```

Examples:

Selector	Specificity	Calculation
`p`	0-0-1	1 element selector
`.highlight`	0-1-0	1 class selector
`#header`	1-0-0	1 ID selector
`div p.highlight`	0-1-2	1 class, 2 elements
`#header .nav li:hover`	1-2-1	1 ID, 2 classes, 1 element

Specificity War Example

```
/* Specificity: 0-0-1 */
p { color: blue; }

/* Specificity: 0-1-0 - This wins */
.text { color: red; }
```

```
/* Specificity: 1-0-0 - This would win over both */
#paragraph { color: green; }
```

Best Practices for Specificity

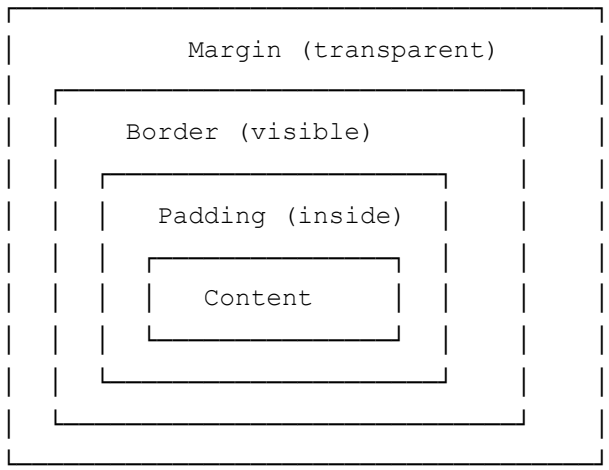
1. **Avoid ID selectors** in CSS for styling (use for JavaScript)
2. **Use class selectors** for most styling
3. **Minimize nesting** in preprocessors
4. **Never use** `!important` unless absolutely necessary
5. **Keep selectors simple** and readable

Module 3: Box Model and Layout

Duration: 45 minutes

3.1 The CSS Box Model

Every element in CSS is a box consisting of four layers:



3.1.1 Content

The actual content of the element (text, images, etc.):

```
.box {
    width: 300px;      /* Width of content */
    height: 200px;     /* Height of content */
}
```

3.1.2 Padding

Internal space between content and border:

```
.box {
    padding: 20px;              /* All sides */
    padding: 20px 30px;         /* Top/Bottom 20px, Left/Right 30px */
    padding: 10px 20px 30px 40px; /* Top, Right, Bottom, Left */
    padding-top: 10px;
    padding-right: 20px;
    padding-bottom: 30px;
    padding-left: 40px;
}
```

3.1.3 Border

The border surrounding the padding:

```
.box {
    border: 2px solid black;        /* Width, Style, Color */
    border-width: 2px;
    border-style: solid;
    border-color: black;

    /* Individual sides */
    border-top: 2px solid black;
    border-right: 3px dashed blue;
    border-bottom: 1px dotted red;
    border-left: 4px double green;

    /* Border radius for rounded corners */
    border-radius: 10px;                /* All corners */
    border-radius: 10px 20px 30px 40px;  /* TL, TR, BR, BL */
}
```

Border Styles:

```
border-style: solid;    /* Continuous line */
border-style: dashed;   /* Dashed line */
border-style: dotted;   /* Dotted line */
border-style: double;   /* Double line */
border-style: groove;   /* 3D groove effect */
```

3.1.4 Margin

External space outside the border:

```
.box {
    margin: 20px;                   /* All sides */
    margin: 20px 30px;              /* Top/Bottom 20px, Left/Right 30px */
    margin: 10px 20px 30px 40px;    /* T, R, B, L */
    margin-top: 10px;
```

```
    margin-bottom: 10px;
}
```

Margin Collapsing: Vertical margins between adjacent elements collapse to the larger value:

```
h1 { margin-bottom: 20px; }
p { margin-top: 30px; }
/* Resulting gap is 30px (not 50px) */
```

3.1.5 Box-Sizing

Controls how element width and height are calculated:

```
/* Default: width = content width */
.box {
    box-sizing: content-box;
    width: 300px;
    padding: 20px;
    border: 2px solid black;
    /* Total width: 300px + 40px(padding) + 4px(border) = 344px */
}

/* Better: width = content + padding + border */
.box {
    box-sizing: border-box;
    width: 300px;
    padding: 20px;
    border: 2px solid black;
    /* Total width: 300px */
}
```

Best Practice: Always use `border-box`:

```
* {
    box-sizing: border-box;
}
```

3.2 Display Property

The `display` property determines how an element is rendered:

3.2.1 Block Display

Block elements take up full width and create new lines:

```
div { display: block; }

/* Block elements: <div>, <p>, <h1>, <section>, <header>, <footer> */
```

```
<p>First paragraph</p>
<p>Second paragraph</p>
<!-- These appear on separate lines -->
```

3.2.2 Inline Display

Inline elements flow within text and don't create new lines:

```
span { display: inline; }

/* Inline elements: <span>, <a>, <strong>, <em>, <img> */
```

```
<p>This is <span>inline</span> content.</p>
<!-- Appears on same line -->
```

3.2.3 Inline-Block Display

Combines properties of both:

```
.icon { display: inline-block; }

/* Properties: */
.icon {
    display: inline-block;
    width: 50px;        /* Can set width/height */
    height: 50px;
    margin: 10px;       /* Can set margins */
    vertical-align: middle;
}
```

3.2.4 None Display

Removes element from document flow:

```
.hidden { display: none; }

/* Note: This is different from visibility: hidden; */
/* display: none - element takes no space */
/* visibility: hidden; - element takes space but is invisible */
```

3.3 Position Property

Controls how elements are positioned in the document:

3.3.1 Static (Default)

Elements follow normal document flow:

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

3.3.2 Relative

Positioned relative to its normal position:

```
.box {
    position: relative;
    top: 20px;      /* Move down 20px from normal position */
    left: 30px;     /* Move right 30px from normal position */
}

/* Element still takes up space in document flow */
```

3.3.3 Absolute

Positioned relative to nearest positioned ancestor:

```
.container {
    position: relative;  /* Create positioning context */
}

.box {
    position: absolute;
    top: 50px;
    left: 100px;
    width: 200px;
    height: 150px;
}

/* Element is removed from normal flow */
```

3.3.4 Fixed

Positioned relative to viewport (stays in place when scrolling):

```
.navbar {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    height: 60px;
    background: navy;
    color: white;
}

/* Element doesn't scroll with page */
```

3.3.5 Sticky

Hybrid of relative and fixed positioning:

```
.header {
    position: sticky;
    top: 0;
    background: white;
    z-index: 100;
}

/* Scrolls with content until reaching top, then stays fixed */
```

3.4 Flexbox Layout

Flexbox is a modern, powerful layout system for arranging items in rows or columns:

3.4.1 Flex Container

```
.container {
    display: flex;
    flex-direction: row;        /* row (default), column, row-reverse, column-reverse */
    justify-content: center;    /* Horizontal alignment */
    align-items: center;        /* Vertical alignment */
    gap: 20px;                  /* Space between items */
    flex-wrap: wrap;            /* Wrap items if needed */
}
```

3.4.2 Justify-Content (Main Axis Alignment)

```
.container {
    display: flex;
    justify-content: flex-start;   /* Items at start (default) */
    justify-content: flex-end;     /* Items at end */
    justify-content: center;       /* Items centered */
    justify-content: space-between; /* Equal space between items */
    justify-content: space-around;  /* Equal space around items */
    justify-content: space-evenly;  /* Equal space everywhere */
}
```

3.4.3 Align-Items (Cross Axis Alignment)

```
.container {
    display: flex;
    align-items: flex-start;    /* Items at start */
    align-items: flex-end;      /* Items at end */
    align-items: center;        /* Items centered */
```

```
    align-items: stretch;       /* Items stretch to fill (default) */
    align-items: baseline;      /* Items align by baseline */
}
```

3.4.4 Flex Items

```
.item {
    flex-grow: 1;           /* Growth factor when space available */
    flex-shrink: 1;         /* Shrink factor when space limited */
    flex-basis: 200px;      /* Base size before growing/shrinking */
    flex: 1 1 200px;        /* Shorthand: grow shrink basis */
}
```

Practical Example:

```
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 20px;
    background: #333;
}

.logo {
    color: white;
    font-size: 24px;
    font-weight: bold;
}

.nav-items {
    display: flex;
    gap: 30px;
    list-style: none;
}

.nav-items li {
    color: white;
}

.nav-items a {
    color: white;
    text-decoration: none;
}
```

3.5 CSS Grid Layout

CSS Grid is a two-dimensional layout system for creating complex layouts:

3.5.1 Grid Container

```
.container {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;  /* 3 equal columns */
    grid-template-rows: auto 200px auto;  /* 3 rows */
    gap: 20px;                            /* Space between items */
}
```

3.5.2 Grid Template Columns

```
.container {
    /* Fixed widths */
    grid-template-columns: 200px 300px 100px;

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

    /* Mixed */
    grid-template-columns: 200px 1fr 200px;

    /* Repeat function */
    grid-template-columns: repeat(3, 1fr);  /* Same as 1fr 1fr 1fr */
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}
```

3.5.3 Positioning on Grid

```
.item {
    grid-column: 1 / 3;  /* Start column 1, end column 3 */
    grid-row: 1 / 3;     /* Start row 1, end row 3 */
}

.item-span {
    grid-column: span 2;  /* Span 2 columns */
    grid-row: span 2;     /* Span 2 rows */
}
```

3.5.4 Named Areas

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    grid-template-rows: auto 1fr auto;
    grid-template-areas:
        "header header header"
        "sidebar main aside"
        "footer footer footer";
}

.header { grid-area: header; }
```

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

Module 4: Modern CSS Features and Best Practices

Duration: 45 minutes

4.1 CSS Variables (Custom Properties)

CSS Variables allow you to store and reuse values throughout your stylesheet:

4.1.1 Defining Variables

```
/* Define at root level for global scope */
:root {
    --primary-color: #3498db;
    --secondary-color: #2ecc71;
    --font-size-base: 16px;
    --font-size-lg: 20px;
    --spacing-unit: 8px;
    --border-radius: 4px;
}

/* Define at component level for scoped access */
.button {
    --button-padding: 12px 20px;
    --button-border-color: blue;
}
```

4.1.2 Using Variables

```
/* Using variables */
h1 {
    color: var(--primary-color);
    font-size: var(--font-size-lg);
    margin-bottom: calc(var(--spacing-unit) * 2);
    border-radius: var(--border-radius);
}

button {
    background-color: var(--primary-color);
    padding: var(--button-padding);
    border: 2px solid var(--button-border-color);
}
```

4.1.3 Fallback Values

```
/* Provide fallback if variable not defined */
p {
    color: var(--text-color, #333);  /* Uses #333 if --text-color not found */
}
```

4.1.4 Dynamic Theme Switching

```
:root {
    --bg-color: #ffffff;
    --text-color: #000000;
}

/* Dark mode */
@media (prefers-color-scheme: dark) {
    :root {
        --bg-color: #1a1a1a;
        --text-color: #ffffff;
    }
}

body {
    background-color: var(--bg-color);
    color: var(--text-color);
    transition: background-color 0.3s, color 0.3s;
}
```

JavaScript Integration:

```
// Get variable value
const primaryColor = getComputedStyle(document.documentElement)
    .getPropertyValue('--primary-color');

// Set variable value
document.documentElement.style.setProperty('--primary-color', '#ff0000');
```

4.2 Transitions and Animations

Transitions and animations add motion and interactivity to web pages.

4.2.1 CSS Transitions

Smooth changes from one state to another:

```
.button {
    background-color: blue;
    color: white;
```

```
    transition: background-color 0.3s ease-in-out;
}

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

Transition Properties:

```
.element {
    transition-property: background-color;     /* What to transition */
    transition-duration: 0.3s;                 /* How long */
    transition-timing-function: ease-in-out;  /* Animation curve */
    transition-delay: 0s;                      /* When to start */

    /* Shorthand */
    transition: background-color 0.3s ease-in-out 0s;

    /* Transition multiple properties */
    transition: background-color 0.3s, color 0.2s, transform 0.5s;

    /* Transition all properties */
    transition: all 0.3s ease-in-out;
}
```

Timing Functions:

```
transition-timing-function: linear;
transition-timing-function: ease;       /* Default */
transition-timing-function: ease-in;
transition-timing-function: ease-out;
transition-timing-function: ease-in-out;
transition-timing-function: cubic-bezier(0.25, 0.46, 0.45, 0.94);
```

4.2.2 CSS Animations

More complex motion with keyframes:

```
@keyframes slideIn {
    from {
        opacity: 0;
        transform: translateX(-100%);
    }
    to {
        opacity: 1;
        transform: translateX(0);
    }
}

.element {
```

```
    animation-name: slideIn;
    animation-duration: 0.5s;
    animation-timing-function: ease-in-out;
    animation-iteration-count: 1;
    animation-direction: normal;
    animation-fill-mode: forwards;
    animation-delay: 0s;

    /* Shorthand */
    animation: slideIn 0.5s ease-in-out 0s 1 normal forwards;
}
```

Keyframe Syntax:

```
@keyframes complex-animation {
    0% {
        opacity: 0;
        transform: scale(0.8) rotate(0deg);
    }
    50% {
        opacity: 1;
        transform: scale(1.1) rotate(180deg);
    }
    100% {
        opacity: 1;
        transform: scale(1) rotate(360deg);
    }
}
```

Multiple Animations:

```
.element {
    animation: spin 2s linear infinite,
               pulse 1s ease-in-out infinite;
    animation-composition: add;  /* Combine animations */
}

@keyframes spin {
    to { transform: rotate(360deg); }
}

@keyframes pulse {
    0%, 100% { opacity: 1; }
    50% { opacity: 0.5; }
}
```

4.3 Container Queries (2025 Feature)

Container queries allow styling components based on their parent container size:

```
/* Define container context */
.container {
    container-type: inline-size;
    container-name: card-container;
}

/* Style based on container size */
@container (min-width: 400px) {
    .card {
        display: flex;
        flex-direction: row;
    }

    .card-content {
        flex: 1;
    }
}

@container (max-width: 400px) {
    .card {
        display: flex;
        flex-direction: column;
    }
}

/* Named container queries */
@container card-container (min-width: 500px) {
    .card-image {
        width: 200px;
    }
}
```

Advantages:

- Components adapt based on their container, not viewport
- Highly reusable and modular design
- Perfect for responsive cards and components

4.4 Cascade Layers (@layer)

Organize CSS into predictable zones and control specificity:

```
/* Define layers */
@layer reset, base, components, utilities;

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

```
}

@layer base {
    body {
        font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-serif;
        font-size: 16px;
        line-height: 1.5;
    }

    h1 { font-size: 2em; }
    p { margin-bottom: 1em; }
}

@layer components {
    .button {
        padding: 10px 20px;
        border-radius: 4px;
        background-color: blue;
        color: white;
    }

    .card {
        border: 1px solid #ddd;
        border-radius: 8px;
        padding: 20px;
    }
}

@layer utilities {
    .text-center { text-align: center; }
    .mt-1 { margin-top: 8px; }
    .p-1 { padding: 8px; }
}

/* Specificity order: reset < base < components < utilities */
/* Later layers override earlier ones regardless of selector specificity */
```

4.5 Modern Color Spaces (2025)

CSS supports advanced color spaces for better color control:

```
/* LCH (Lightness, Chroma, Hue) - best for modern web */
.primary-button {
    background-color: lch(50% 100 240);
}

/* LAB (Perceptually uniform) */
.text {
    color: lab(50% 50 -50);
}

/* HWB (Hue, Whiteness, Blackness) */
```

```
.accent {
    color: hwb(240 0% 0%);
}

/* oklch (Perceptually uniform, modern) */
.element {
    background-color: oklch(0.5 0.2 240);
}
```

Color Function Examples:

```
h1 {
    /* Mix two colors */
    color: color-mix(in lch, blue 80%, red 20%);

    /* Adjust lightness */
    background: hwb(240 0% 20%);  /* 20% darker than hue */
}
```

4.6 Advanced Layout Functions

4.6.1 clamp() Function

Responsive sizing with limits:

```
h1 {
    /* Min 20px, Preferred 5vw, Max 60px */
    font-size: clamp(20px, 5vw, 60px);
}

.container {
    /* Min 300px, Preferred 90%, Max 1200px */
    width: clamp(300px, 90%, 1200px);
}

.padding {
    /* Responsive padding */
    padding: clamp(1rem, 5%, 3rem);
}
```

4.6.2 min() and max() Functions

```
/* Use smallest value */
.container {
    width: min(100%, 1200px);  /* Never exceed 1200px */
}

/* Use largest value */
.section {
```

```
    min-height: max(100vh, 500px);  /* At least 500px */
}

/* Combining functions */
h1 {
    font-size: clamp(
        1.5rem,
        2vw,
        3rem
    );
}
```

4.7 CSS Best Practices

4.7.1 Naming Conventions: BEM (Block Element Modifier)

```
/* Block - standalone entity */
.button { ... }

/* Element - part of a block */
.button__text { ... }
.button__icon { ... }

/* Modifier - variation of a block or element */
.button--primary { ... }
.button--secondary { ... }
.button__icon--small { ... }
```

HTML:

```
<button class="button button--primary">
    <span class="button__icon button__icon--small"></span>
    <span class="button__text">Click Me</span>
</button>
```

4.7.2 Organizing Stylesheets

```
/* 1. Reset and Global Styles */
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    padding: 0;
    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', sans-serif;
}

/* 2. Base Element Styles */
```

```
h1, h2, h3 { line-height: 1.2; }
p { margin-bottom: 1em; }
a { color: blue; text-decoration: none; }

/* 3. Layout Styles */
.container { max-width: 1200px; margin: 0 auto; }
.grid { display: grid; }
.flex { display: flex; }

/* 4. Component Styles */
.button { ... }
.card { ... }
.navbar { ... }

/* 5. Utility Styles */
.text-center { text-align: center; }
.mb-1 { margin-bottom: 8px; }
.p-1 { padding: 8px; }

/* 6. Responsive Media Queries */
@media (max-width: 768px) {
    .container { padding: 0 16px; }
    .grid { grid-template-columns: 1fr; }
}
```

4.7.3 Responsive Design Principles

```
/* Mobile-first approach (BEST PRACTICE) */

/* Default styles for mobile */
.container {
    width: 100%;
    padding: 16px;
}

.grid {
    display: grid;
    grid-template-columns: 1fr;  /* 1 column on mobile */
}

/* Small screens */
@media (min-width: 640px) {
    .grid {
        grid-template-columns: 1fr 1fr;  /* 2 columns */
    }
}

/* Medium screens */
@media (min-width: 1024px) {
    .container {
        max-width: 960px;
        margin: 0 auto;
```

```
    }

    .grid {
        grid-template-columns: repeat(3, 1fr);  /* 3 columns */
    }
}

/* Large screens */
@media (min-width: 1280px) {
    .container {
        max-width: 1200px;
    }
}
```

Viewport Meta Tag (Required in HTML):

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

4.7.4 Avoiding Common Mistakes

1. Avoid Deep Nesting (if using preprocessors):

```
/*   Bad - Too deep */
.header {
    .navbar {
        .nav-item {
            .link {
                color: blue;
            }
        }
    }
}

/*   Good - Use classes for styling */
.header { ... }
.navbar { ... }
.nav-item { ... }
.link { ... }
```

2. Avoid `!important` (use specificity instead):

```
/*   Bad */
.button { background-color: blue !important; }

/*   Good - Use more specific selector */
.button.button--primary { background-color: blue; }
```

3. Minimize ID Selectors:

```
/*   Not ideal for styling */
#header { background: navy; }

/*   Better - Use classes */
.header { background: navy; }
```

4. Group Related Properties:

```
/*   Scattered properties */
.card {
    color: black;
    margin: 10px;
    font-size: 16px;
    padding: 20px;
    border: 1px solid gray;
    background: white;
}

/*   Organized logically */
.card {
    /* Display & Box Model */
    display: block;
    margin: 10px;
    padding: 20px;
    border: 1px solid gray;

    /* Colors */
    background: white;
    color: black;

    /* Typography */
    font-size: 16px;
}
```

4.8 Performance Optimization

4.8.1 CSS Minification

Minimize CSS file size for production:

```
/* Before minification */
.header {
    background-color: navy;
    color: white;
    padding: 20px;
}

/* After minification */
.header{background-color:navy;color:white;padding:20px}
```

Tools: Sass, PostCSS, CSSnano

4.8.2 Critical CSS

Load essential styles inline, defer non-critical CSS:

```
<head>
    <!-- Critical styles for above-the-fold content -->
    <style>
        body { margin: 0; padding: 0; font-family: sans-serif; }
        .header { background: navy; color: white; }
    </style>

    <!-- Defer non-critical styles -->
    <link rel="preload" href="styles.css" as="style">
    <link rel="stylesheet" href="styles.css">
</head>
```

4.8.3 CSS-in-JS (Optional for Modern Frameworks)

```
// Example with styled-components or emotion
const buttonStyles = `
    background-color: blue;
    color: white;
    padding: 10px 20px;
    border-radius: 4px;
    cursor: pointer;

    &:hover {
        background-color: darkblue;
    }
`;
```

4.9 Accessibility in CSS

4.9.1 Color Contrast

Ensure text is readable:

```
/* WCAG AA: 4.5:1 contrast ratio for normal text */
.text {
    color: #000000;  /* Black */
    background-color: #ffffff;  /* White = 21:1 contrast ✓ */
}

/* Check contrast: https://webaim.org/resources/contrastchecker/ */
```

4.9.2 Focus Indicators

Never remove focus outline without replacement:

```
/*   Bad - removes accessibility */
button:focus {
    outline: none;
}

/*   Good - custom focus indicator */
button:focus {
    outline: 2px solid blue;
    outline-offset: 2px;
}
```

4.9.3 Reduced Motion

Respect user preferences for animations:

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

Summary and Key Takeaways

Module 1: Fundamentals

- CSS is the presentation layer of web pages
- Use external stylesheets for maintainability
- Learn CSS units (rem, em, %, vh/vw)
- Understand color formats (hex, rgb, hsl, modern spaces)

Module 2: Selectors and Cascade

- Master different selector types for efficient targeting
- Understand the cascade mechanism
- Calculate and manage specificity properly
- Avoid specificity wars with good practices

Module 3: Layout

- Box Model: content, padding, border, margin
- Display property: block, inline, inline-block

- Position: static, relative, absolute, fixed, sticky
- Modern layouts: Flexbox for 1D, Grid for 2D
- Container Queries for responsive components

Module 4: Modern Features

- CSS Variables for maintainability and theming
- Transitions for smooth state changes
- Animations with keyframes for complex motion
- Cascade Layers for predictable specificity
- Best practices: BEM naming, mobile-first, accessibility

Advanced Topics to Explore Further

- CSS Preprocessors (Sass, Less)
- CSS Frameworks (Tailwind CSS, Bootstrap)
- CSS-in-JS solutions
- Performance optimization and critical CSS
- Accessibility and WCAG guidelines
- Design systems and component libraries

Resources for Further Learning

1. **MDN Web Docs:** https://developer.mozilla.org/en-US/docs/Web/CSS
2. **CSS Tricks:** https://css-tricks.com
3. **Web.dev Learn CSS:** https://web.dev/learn/css
4. **Can I Use:** https://caniuse.com (browser support)
5. **Color Contrast Checker:** https://webaim.org/resources/contrastchecker/

Practical Exercises for Students

Exercise 1: Build a Personal Portfolio Page

Create a responsive portfolio using HTML and CSS with:

- Flexbox navigation bar
- CSS Grid for project showcase
- CSS Variables for color theming
- Smooth transitions on hover

Exercise 2: Create an Interactive Component Library

Build reusable components with:

- BEM naming convention
- Responsive design with media queries
- Animations and transitions
- Accessible focus states

Exercise 3: Responsive Web Design Project

Build a complete responsive website with:

- Mobile-first approach
- Container Queries for adaptable components
- CSS Grid for complex layouts
- Dark mode toggle using CSS Variables

End of Lecture Notes