

CSS

Cascading Style Sheets

From HTML Structure to Professional Web Pages

STUDY GUIDE · Lecture 2 — Frontend Development

Midocean University

HTML → Structure CSS → Presentation

How to use this guide

This guide is the written companion to the Lecture 2 slides. The slides are short because the live demos matter most; here every idea is explained in full, with examples you can type and test yourself. Work through one chapter at a time: read, type the code into your own **index.html** and **style.css**, then change values and watch what happens.

Chapter	Topic	Study time
1	CSS introduction & motivation	8 min
2	CSS syntax & applying CSS	10 min
3	Selectors	18 min
4	Colors, backgrounds & typography	12 min
5	The Box Model	18 min
6	Cascade, inheritance & specificity	10 min
7	Display & basic layout	10 min
8	Pseudo-classes & states	7 min
9	Flexbox	15 min
10	DevTools & debugging	5 min
11	Final practical project	15–20 min
12	Review: cheat sheet, quiz & challenge	—

Study tip: Do not memorize properties. Understand *why* you use each one. Reading the guide once is not enough: type the examples, break them on purpose, and fix them with DevTools.

01

CSS Introduction & Motivation

Suggested study time: 8 min

HTML is structure

In Lecture 1 you learned to build the **structure** of a web page. HTML answers one question: *what is this element?* — a heading, a paragraph, a button, an image, a form, a navigation bar.

HTML

```
<h1>Mohamed Saied</h1>

<p>Frontend Developer</p>

<button>Contact Me</button>
```

Why HTML alone is not enough

With pure HTML the browser applies its own default appearance. The result works, but it has limited visual design, poor spacing, default colors and typography, and it is very hard to build professional layouts. The question is: how do we control the appearance? The answer is **CSS**.

What CSS controls

- Colors, fonts and sizes
- Spacing, borders and backgrounds
- Positioning and layout
- Responsive behavior and visual states (for example, what a button looks like on hover)

Why separate HTML and CSS?

Imagine styling every element individually:

Repeated inline styles

```
<h1 style="color: blue;">Hello</h1>
<h2 style="color: blue;">Welcome</h2>
<p style="color: blue;">Frontend</p>
```

To change blue to red you must edit every element. With CSS you write one rule and change one place:

One CSS rule

```
h1, h2, p {
  color: blue;
}
```

Key takeaway: Separation of concerns: HTML = structure, CSS = presentation (plus layout).

02

CSS Syntax & Applying CSS

Suggested study time: 10 min

Anatomy of a CSS rule

Pattern and example

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

```
p {  
  color: blue;  
  font-size: 20px;  
}
```

- **p** is the **selector** — it says which elements to style.
- **color** is the **property** — what to change.
- **blue** is the **value** — what to change it to.

A single **property: value;** pair is called a **declaration**. A rule can hold many declarations, and **each one ends with a semicolon.**

```
p {  
  color: blue;  
  font-size: 20px;  
  font-weight: bold;  
}
```

Your first demo

index.html

```
<h1>Hello CSS</h1>  
<p>My first styled paragraph.</p>
```

style.css

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

Link the two files together (see external CSS below) and open **index.html** in the browser.

Three ways to add CSS

Method	Example	When to use
Inline	<code><p style="color: red;">Hello</p></code>	Quick tests only. Hard to maintain, repeats code, mixes HTML and CSS.
Internal	<code><style> p { color: red; } </style></code> inside <code><head></code>	Small pages, quick demos, single-page experiments.
External	<code><link rel="stylesheet" href="style.css"></code>	Professional standard: reusable, maintainable, shared by many pages.

HTML – external CSS

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
```

Key takeaway: Avoid inline CSS for normal styling. Use an external stylesheet so the same file can style every page of a project.

Comments

```
/* Main page heading */
h1 {
  color: navy;
}
```

Comments are ignored by the browser. Use them to explain *why* a rule exists.

03 Selectors

Suggested study time: 18 min

A selector tells CSS **which HTML elements should receive these styles**. Selectors are one of the most important CSS concepts.

Selector	Syntax	Meaning
Element	<code>p { }</code>	Every element of that tag.
Class	<code>.important { }</code>	Every element with <code>class="important"</code> . Reusable.
ID	<code>#main-header { }</code>	The one element with <code>id="main-header"</code> .
Universal	<code>* { }</code>	All elements.
Grouping	<code>h1, h2, p { }</code>	Several selectors sharing one rule.
Descendant	<code>.card p { }</code>	Any <code>p</code> inside <code>.card</code> , at any depth.
Child	<code>.card > p { }</code>	Only <code>p</code> that are <i>direct</i> children of <code>.card</code> .
Attribute	<code>input[type="text"] { }</code>	Elements with a given attribute value.

Element selector

CSS – affects every `<p>`

```
p {  
  color: gray;  
}
```

Class selector

HTML

```
<p class="important">Important information</p>  
<p class="important">One</p>
```

CSS

```
.important {  
  color: red;  
}
```

ID selector

HTML

```
<header id="main-header">
```

CSS

```
#main-header {  
  background: black;  
}
```

Class vs ID

Class (.card)	ID (#main-card)
Reusable	Intended to identify one element
Many elements	Normally unique
Common for styling	Less commonly needed for styling

Key takeaway: Prefer classes for styling. Use IDs to identify a single, particular element.

Universal selector

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

Useful for resetting default spacing and global rules, but use it carefully because it affects **everything**.

Grouping selectors

```
/* instead of three identical rules */  
h1, h2, p {  
  color: blue;  
}
```

Descendant vs child

HTML

```
<div class="card">  
  <h2>Profile</h2>  
  <p>Hello</p>  
</div>
```

CSS

```
.card p { color: gray; } /* any p inside .card */  
.card > p { color: red; } /* only direct children */
```

The difference matters in larger applications, where elements are nested several levels deep.

Attribute selectors

CSS – targets only type="text" inputs

```
input[type="text"] {  
  border: 2px solid blue;  
}
```

Practice

HTML

```
<h1 class="title">Profile</h1>
<p class="description">Frontend Developer</p>
<button id="contact">Contact</button>
```

Which selector targets (1) the **h1**, (2) the description, (3) the button, (4) all paragraphs?

Answers: 1) **h1** (or **.title**) 2) **.description** 3) **#contact** 4) **p**

04

Colors, Backgrounds & Typography

Suggested study time: 12 min

Color formats

Format	Example	Notes
Named	<code>color: red;</code>	Easy to read; limited choice.
HEX	<code>color: #ff0000;</code>	#RRGGBB — the most common format.
RGB	<code>color: rgb(255, 0, 0);</code>	Red, green, blue from 0-255.
RGBA	<code>color: rgba(255, 0, 0, 0.5);</code>	RGB plus opacity from 0 to 1.

In HEX, the six digits are three pairs: **RR GG BB**.

```
#FF0000 → Red      #000000 → Black
#00FF00 → Green    #FFFFFF → White
#0000FF → Blue     #3498db → a nice sky blue
```

Backgrounds

```
body {
  background-color: #f5f5f5;
}

.card {
  background-color: white;
}

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

Text styling

```
p {
  color: #333;
  font-size: 18px;
  font-family: Arial, sans-serif;
  font-weight: bold;
  text-align: center;
}
```

Other common properties: `font-style`, `text-decoration`, `line-height`.

Font-size units

Unit	Meaning
px	Fixed CSS pixel unit.
%	Percentage relative to another value.
em	Relative to the element's font context.
rem	Relative to the root (<code>html</code>) font size.

For now: **px** = fixed size, **rem** = root-relative size, **%** = percentage. Lecture 3 returns to relative units.

Alignment and decoration

```
h1 { text-align: center; }      /* left | center | right | justify */  
a { text-decoration: none; }    /* removes the underline */
```

Borders

```
.card {  
  border: 2px solid black; /* width style color */  
}
```

Other border styles: **solid**, **dashed**, **dotted**, **double**.

Rounded corners

```
.card { border-radius: 10px; }  
.profile-image { border-radius: 50%; } /* a circle */
```

Try it: A square image becomes a circular profile picture with **border-radius: 50%**.

05 The Box Model

Suggested study time: 18 min

The most important concept in CSS: every HTML element is treated as a rectangular box made of four layers.



Layer	Meaning
Content	The actual text, image, button or form.
Padding	Space inside the element, between content and border.
Border	The line around the padding.
Margin	Space outside the element, separating it from others.

```
.card {
  width: 300px;      /* content size */
  height: 200px;
  padding: 20px;     /* inside */
  margin: 20px;      /* outside */
}
```

Remember: Padding pushes the content inward. Margin separates elements from each other.

Shorthand

```
padding: 20px;          /* all four sides */
padding: 10px 20px 30px 40px; /* top right bottom left (clockwise) */

margin: 10px 20px;      /* top/bottom 10px, left/right 20px */
margin: 0 auto;         /* centers a fixed-width block horizontally */
```

The width + padding problem

```
.card {
  width: 300px;
  padding: 20px;
  border: 5px solid black;
}
```

With the default box model the real outer width is **300 + 40 (padding) + 10 (border) = 350px**. This surprises almost every beginner. The fix is **box-sizing**:

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

Now the declared width *includes* content, padding and border, so the card stays exactly **300px** wide.

Key takeaway: Start almost every stylesheet with the `box-sizing: border-box` rule.

06

Cascade, Inheritance & Specificity

Suggested study time: 10 min

The cascade

CSS is called *Cascading* Style Sheets because several rules can apply to the same element. The cascade decides which one wins.

```
p { color: blue; }  
p { color: red; }      /* later rule wins → red */
```

Specificity

All three match – result: red

```
p          { color: blue; }  
.text      { color: green; }  
#main-text { color: red; }  
  
<p id="main-text" class="text">Hello</p>
```

More specific selectors beat less specific ones. The mental model:

```
Element < Class / Attribute / Pseudo-class < ID
```

!important

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

It overrides normal cascade rules, but avoid it: it makes CSS harder to debug, creates specificity problems and makes future overrides harder.

Professional rule: Understand the cascade instead of fighting it.

Inheritance

Some properties (such as text color and font) are passed from parent to child.

```
body {  
  color: #333;  
}  
  
<body>  
  <p>Hello</p>      /* inherits the #333 color */  
</body>
```

07

Display & Basic Layout

Suggested study time: 10 min

The **display** property controls how an element takes part in layout — the bridge between basic CSS and modern layout.

Value	Behaviour
block	Starts on a new line and takes the full width. Examples: div, p, h1, section.
inline	Stays in the flow of text; only as wide as its content. Examples: span, a, strong.
inline-block	Sits inline but accepts width, height, padding and margin.
none	Removed from the layout completely.
flex	Makes the element a flex container (Chapter 9).

Inline elements share a line

Hello world and <a>link <!-- inline -->

CSS

```
.card { display: inline-block; }  
.menu { display: none; }
```

Compare: **display: none** removes the element and its space; **visibility: hidden** hides it but **leaves the space**.

08

Pseudo-classes & States

Suggested study time: 7 min

A pseudo-class describes a **state** of an element. Common ones: **:hover**, **:focus**, **:active**, **:checked**, **:first-child**, **:last-child**.

```
button {  
  background: black;  
  color: white;  
}  
  
button:hover {  
  background: blue; /* when the mouse is over it */  
}
```

CSS can react to user interaction **without JavaScript**.

:focus and accessibility

```
input:focus {  
  border-color: blue;  
  outline: none;  
}
```

Accessibility: Focus styles are important for keyboard users. Do not blindly remove focus indicators — if you remove the outline, give users another visible focus style.

Exercise — style the Lecture 1 registration form

Take the form from Lecture 1 (name, email, password, Register button) and style it: a centered card, padded inputs, a colored button with a hover state and a visible focus style.

09 Flexbox

Suggested study time: 15 min

Flexbox is designed for arranging elements along one axis. Make the parent a **flex container** and its children become **flex items**.

HTML

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

CSS

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

Main axis and direction

By default **flex-direction: row** lays items out horizontally (main axis →). Use **column** to stack them vertically.

```
.container { flex-direction: row; } /* 1 2 3 across */
.container { flex-direction: column; } /* 1 / 2 / 3 down */
```

Aligning items

Property	Axis	Common values
justify-content	Main axis	flex-start, center, flex-end, space-between, space-around, space-evenly
align-items	Cross axis	stretch, flex-start, center, flex-end

justify-content

```
.container {
  display: flex;
  justify-content: space-between;
}

/* result:  [1]          [2]          [3] */
```

Centering anything

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

The child is now centered horizontally *and* vertically.

gap

```
.container {  
  display: flex;  
  gap: 20px;  
}
```

gap puts even spacing between items, so you do not have to add margins manually.

Demo 1 — navigation bar

HTML

```
<nav>  
  <a href="#">Home</a>  
  <a href="#">About</a>  
  <a href="#">Projects</a>  
  <a href="#">Contact</a>  
</nav>
```

CSS

```
nav {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}
```

Demo 2 — cards

HTML

```
<div class="cards">  
  <div class="card"><h3>HTML</h3><p>Structure</p></div>  
  <div class="card"><h3>CSS</h3><p>Presentation</p></div>  
  <div class="card"><h3>JavaScript</h3><p>Behavior</p></div>  
</div>
```

```
.cards {  
  display: flex;  
  gap: 20px;  
  flex-wrap: wrap; /* items move to a new line when space runs out */  
}  
  
.card {  
  flex: 1; /* items share the free space equally */  
}
```

Key takeaway: **justify-content** → main axis, **align-items** → cross axis. Use Flexbox for one-dimensional layouts (a row or a column).

10

DevTools & Debugging

Suggested study time: 5 min

When something looks wrong, **do not randomly change CSS**. Open the browser DevTools: right-click → **Inspect**, or press **F12**.

- **Elements panel:** see the HTML structure and the selected element.
- **Styles:** see applied rules and which ones are overridden (crossed out).
- **Box model diagram:** inspect and edit margin, border, padding and content size live.
- Changes in DevTools are temporary — copy what works back into your CSS file.

Three common mistakes

1. Wrong file name

```
<link rel="stylesheet" href="styles.css">  <!-- but the file is style.css -->
```

2. Missing semicolon

```
p {  
  color: red          /* ← missing semicolon */  
  font-size: 20px;  
}
```

3. Selector does not match the HTML

```
.card-title { color: red; }
```

```
<h2 class="title">          <!-- class names do not match -->
```

Debugging checklist: If a style does not apply: (1) is the CSS file linked? (2) does the selector match the HTML? (3) is a more specific rule overriding it? DevTools answers all three.

11

Final Practical Project

Suggested study time: 15–20 min

Transform your Lecture 1 profile page into a styled page: navigation bar, hero section with a circular photo, skill cards and a contact button.

Workflow

1) Build the HTML structure → 2) give important elements classes → 3) create the CSS → 4) typography and colors → 5) spacing → 6) layout → 7) interaction states → 8) inspect with DevTools → 9) refactor.

Step 1 — semantic HTML

```
<header>
  <nav> ... </nav>
</header>

<main>
  <section> ... </section>
  <section> ... </section>
</main>

<footer> ... </footer>
```

CSS should style a good HTML structure; it does not replace semantic HTML.

Step 2 — global CSS

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

body {
  margin: 0;
  font-family: Arial, sans-serif;
  color: #333;
}
```

Step 3 — navigation

```
nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 20px 40px;
}
nav a { text-decoration: none; }
nav a:hover { text-decoration: underline; }
```

Step 4 — profile section

```
.profile {  
  text-align: center;  
  padding: 60px 20px;  
}  
.profile img {  
  width: 150px;  
  height: 150px;  
  border-radius: 50%;  
}
```

Step 5 — skills

HTML

```
<div class="skills">  
  <div class="skill">HTML</div>  
  <div class="skill">CSS</div>  
  <div class="skill">JavaScript</div>  
</div>
```

CSS

```
.skills {  
  display: flex;  
  justify-content: center;  
  gap: 20px;  
}
```

Step 6 — skill cards

```
.skill {  
  padding: 20px;  
  border: 1px solid #ddd;  
  border-radius: 10px;  
  background: white;  
}  
.skill:hover {  
  transform: translateY(-3px); /* lifts the card slightly */  
}
```

Step 7 — contact button

```
.contact-btn {  
  display: inline-block;  
  padding: 12px 24px;  
  border-radius: 8px;  
  text-decoration: none;  
  background: #222;  
  color: white;  
}  
.contact-btn:hover { background: #444; }
```

Think in components

As projects grow, do not write random CSS everywhere. Think in components — Navbar, Hero, Card, Button, Form, Footer — and name classes accordingly:

```
.card { }  
.card-title { }  
.card-description { }  
.card-button { }
```

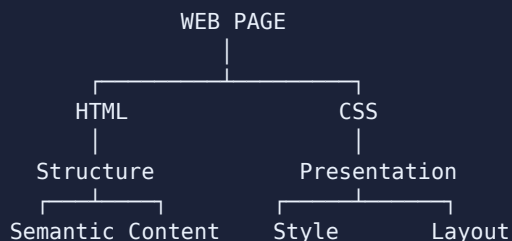
This prepares you for Bootstrap, Angular, React and other component-based UI.

12

Review: Cheat Sheet, Quiz & Challenge

Suggested study time: 30 min

HTML + CSS mental model



Seven rules to remember

- HTML describes **what** something is.
- CSS describes **how** it looks.
- Use classes for reusable styling.
- Understand the Box Model.
- Understand the cascade before using **!important**.
- Use Flexbox for one-dimensional layouts.
- Use DevTools instead of guessing.

Property cheat sheet

Goal	Property / example
Text color	<code>color: #333;</code>
Background	<code>background-color: #f5f5f5;</code>
Font	<code>font-family: Arial, sans-serif; font-size: 18px; font-weight: bold;</code>
Align text	<code>text-align: center;</code>
Remove underline	<code>text-decoration: none;</code>
Border	<code>border: 2px solid #333;</code>
Rounded corners	<code>border-radius: 10px;</code>
Space inside / outside	<code>padding: 20px; margin: 20px;</code>
Include padding in width	<code>box-sizing: border-box;</code>
Center a block	<code>margin: 0 auto;</code>
Hide element	<code>display: none;</code>
Hover state	<code>button:hover { ... }</code>
Flex row with gap	<code>display: flex; gap: 20px;</code>
Center with flex	<code>justify-content: center; align-items: center;</code>

Self-check quiz

Question 1. What does CSS stand for?

- A. Computer Style Sheets
- B. Cascading Style Sheets
- C. Creative Style Syntax
- D. Colorful Style Sheets

Question 2. Which selector selects a class?

- A. #card
- B. .card
- C. card
- D. *card

Question 3. Which property creates space *inside* an element?

- A. margin
- B. padding
- C. border
- D. spacing

Question 4. Which declaration enables Flexbox?

- A. position: flex;
- B. layout: flex;
- C. display: flex;
- D. flex: display;

Question 5. What does `button:hover { background: blue; }` do?

- A. Changes the button permanently when clicked
- B. Changes the button while the pointer is over it
- C. Hides the button
- D. Changes the HTML

Answers: 1-B, 2-B, 3-B, 4-C, 5-B

Final challenge — build your own profile card

```
Profile Image
Student Name
Frontend Student
HTML CSS JavaScript
[ Contact ]
```

Requirements: external CSS, class selectors, colors, typography, padding, margin, border, border-radius, Flexbox and a `:hover` state.

Rules of the challenge: Build it yourself — no copy/paste. Work in this order: HTML structure → typography → colors → box model → layout → interaction.

Coming next: Lecture 3

CSS Layout & Responsive Design — advanced Flexbox, CSS Grid, media queries, mobile-first development, responsive navigation and cards, max-width, relative and viewport units, and a real responsive website.