CSS Core Concepts to Responsive Design

- CSS insertion methods and selectors
- Box model, display, position, and styling decisions
- Typography, backgrounds, and responsive thinking

Today’s Learning Path

A full route from first CSS rules to responsive layouts

Core Ideas

- Start with a quick reference of the most useful CSS properties.
- Learn how CSS is added to a page and when each method is appropriate.
- Build selector confidence so styling targets feel predictable.
- Move into the box model, layout behavior, backgrounds, and type.
- Finish with responsive design and media queries.

Why This Order Works

Students first need a visual vocabulary. Then they need control over targeting and layout. Responsive design becomes easier once those ideas are stable.

Classroom Rhythm

- Explain the idea
- Read the code
- Predict the output
- Run and verify
- Refine the design

Learning Outcomes

What students should be able to do after this lesson

Core Ideas

- Explain the difference between inline, internal, and external CSS.
- Choose the correct selector for a styling task.
- Control size, spacing, borders, layout flow, and position.
- Improve readability with typography and background choices.
- Write a basic media query and adapt a design for different screens.

Success Check

By the end, a student should be able to create a small HTML page and defend each CSS decision in simple English.

Output Expectation

A good result is not “more CSS.” A good result is a page that feels intentional, readable, and stable across devices.

How a CSS Rule Works

Selector, property, and value form the basic grammar of styling

Core Ideas

- A selector chooses which HTML elements will be styled.
- A property names the feature that will change.
- A value sets the chosen feature.
- Curly braces group all declarations that belong to the selector.
- Clear CSS starts with readable, predictable rule blocks.

Example Rule

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

Reading Tip

Ask students to read every rule aloud as a sentence:

“Select these elements, then change this property to this value.”

Table A. Typography Properties

CSS CODE	EXPLANATION	USAGE
`font-size`	Font size	`font-size: 16px;` font-size: 0.5em;
`font-style`	Font style	`font-style: italic;`
`font-weight`	Font weight	`font-weight: bold;`
`font-family`	Font family	`font-family: Verdana;`
`color`	Text color	`color: blue;`
`letter-spacing`	Space between letters	`letter-spacing: 0.5em;`
`word-spacing`	Space between words	`word-spacing: 0.5em;`
`word-wrap`	Wraps long words when needed	`word-wrap: break-word;`
`line-height`	Line height	`line-height: 12px;`
`text-transform`	Changes text case	`text-transform: uppercase;`
`text-decoration`	Decoration lines	`text-decoration: underline;`
`text-align`	Text alignment	`text-align: center;`
`text-indent`	First-line indent	`text-indent: 10px;`
`text-shadow`	Text shadow	`text-shadow: 2px gray;`
`text-overflow`	Clipped text behavior	`text-overflow: ellipsis;`

Table B. Box Properties

Size, spacing, borders, and rounded-corner control

CSS CODE	EXPLANATION	USAGE
`width`	Width	`width: 300px;`
`height`	Height	`height: 100px;`
`max-width`, max-height	Maximum allowed size	`max-width: 400px;` max-height: 250px;
`min-width`, min-height	Minimum allowed size	`min-width: 10px;` min-height: 10px;
`margin`	Outer space	`margin: 15px;` margin-left: 10px;
`padding`	Inner space	`padding: 10px;` padding-bottom: 5px;
`border-color`	Border color	`border-color: red;`
`border-style`	Border style	`border-style: solid;`
`border-width`	Border width	`border-width: 3px;`
`border-radius`	Rounded corners	`border-radius: 10px;`

Table C. Background Properties

The main rules used to control background color, image, size, and position

CSS CODE	EXPLANATION	USAGE
`background-color`	Background color	`background-color: purple;`
`background-image`	Background image	`background-image: url("image.jpg");`
`background-size`	Background size	`background-size: 300px 100px;`
`background-position`	Starting position of the background	`background-position: center center;`
`background-repeat`	Repeats the background image	`background-repeat: no-repeat;`
`background-attachment`	Relationship with page scroll	`background-attachment: fixed;`

Table D. Other Essential CSS Rules

Layout behavior, motion, overflow, effects, and media queries

CSS CODE	EXPLANATION	USAGE
display	Display behavior	`display: inline;`
position	Positioning controls	`position: absolute;` left: 100px; top: 200px;
float	Floated layout	`float: left;` float: right;
clear	Clears float effects	`clear: left;` clear: both;
z-index	Layer order	`z-index: 2;`
opacity	Transparency level	`opacity: 0.4;`
box-shadow	Shadow effect	`box-shadow: 3px gray;`
cursor	Mouse cursor style	`cursor: pointer;`
overflow	Handles overflowing content	`overflow: scroll;`
list-style-type	List marker style	`list-style-type: square;`
filter	Filter effect	`filter: blur(5px);`
scroll-behavior	Smooth in-page scrolling	`html { scroll-behavior: smooth; }`
transition	Animated property change	`transition-property: color;` transition-duration: 0.5s;
@media	Media query rule	`@media screen and` (min-width: 600px) { }

Choosing a CSS Insertion Method

CSS can be added in three common ways

Core Ideas

- Inline CSS sits directly inside an HTML element.
- Internal CSS is written inside a <style> block in the same page.
- External CSS lives in a separate .css file and is linked to HTML.
- The best choice depends on scope, reuse, and maintenance needs.

Main Principle

Use the smallest tool that solves the problem, but keep future editing in mind.

Professional Habit

For real projects, external stylesheets are usually the cleanest and most scalable option.

Inline CSS

A fast way to style one element directly inside HTML

Core Ideas

- Inline CSS uses the style attribute.
- It only affects the element where it is written.
- It is useful for very small demos or one-off adjustments.
- It becomes hard to manage when repeated across a page.

Best Use Case

Short demonstrations, email templates, or a tiny experimental change during practice.

Avoid Overuse

If the same style appears on many elements, move it to internal or external CSS.

Inline CSS Example

One paragraph receives color, size, and background directly

HTML

```
<p style="color: white; background-color: #dc2626;
          font-size: 20px; padding: 10px;">
  CSS helps plain HTML become visually clear.
</p>
```

CSS

```
/* No separate stylesheet is needed here. */
/* The styles live inside the HTML tag. */
```

Practice Prompt

Change the background color, then change the font size, and predict the output before opening the browser.

Your Turn: Inline Styling

A short activity with visible results

Core Ideas

1. Create three paragraph lines.
2. Give each line a different background color.
3. Increase the first line to 20px, the second to 15px, and the third to 10px.
4. Use white text where contrast is needed.
5. Compare how text size changes visual emphasis.

Focus

The goal is not decoration alone. The goal is to connect a CSS value with a visible result.

Expected Result

Students should clearly see how background-color, color, and font-size work together on the same HTML element.

Internal CSS

Page-level styling inside a <style> block

Core Ideas

- Internal CSS sits in the <head> section of the HTML page.
- It can style many matching elements at once.
- It is useful when a single page needs its own styling rules.
- It is easier to manage than inline CSS but less reusable than an external file.

Good Match

A standalone demo page, quiz page, or small prototype with only one HTML file.

Reminder

Internal CSS still uses normal selectors such as p, .card, or #banner. Only its location changes.

Internal CSS Example

The page includes a <style> block that controls all matching paragraphs

HTML

```
<html>
<head>
  <style>
    p { background-color: #ef4444; color: white; }
  </style>
</head>
<body>
  <p>This paragraph is controlled by internal CSS.</p>
</body>
</html>
```

CSS

```
p {
  background-color: #ef4444;
  color: white;
  font-size: 20px;
  padding: 12px;
}
```

Practice Prompt

Add a second paragraph and observe that the same selector styles both elements immediately.

External CSS

The most reusable and maintainable method

Core Ideas

- External CSS lives in a separate file such as style.css.
- HTML connects to it with the <link> tag.
- One stylesheet can control multiple HTML pages.
- This method keeps structure and presentation separate.

Why Teams Prefer It

It reduces repetition and makes large visual updates much faster.

Connection Rule

The HTML page must include:
<link rel="stylesheet" href="style.css">

External CSS Example

HTML and CSS stay in separate files but work together

HTML

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="message">External CSS scales well.</div>
</body>
```

CSS

```
.message {
  font-size: 28px;
  color: white;
  background-color: rebeccapurple;
  padding: 16px;
}
```

Practice Prompt

Ask students to edit only the CSS file and watch the HTML page update without changing the markup.

Which Method Should You Choose?

A simple decision guide

Core Ideas

- Inline CSS: one element, one quick change.
- Internal CSS: one page, several related rules.
- External CSS: multiple pages or a project that will grow.
- Reuse and clarity are usually stronger than speed alone.

Classroom Answer

For almost every serious assignment, external CSS is the target skill students should practice most.

Design Mindset

Structure belongs to HTML. Presentation belongs to CSS. Keeping them separate helps both stay cleaner.

Selectors: Finding the Right Target

CSS becomes powerful when students can aim at the right element

Core Ideas

- A selector tells CSS which HTML elements should receive a style.
- Different selectors solve different targeting problems.
- Good selectors reduce repetition and accidental styling.
- Students should connect selector choice to document structure.

Selector Families

Tag, ID, class, grouping, child, descendant, pseudo-class, and pseudo-element selectors cover most beginner work.

Teaching Tip

Draw the HTML hierarchy before writing CSS when the structure feels confusing.

Why HTML Hierarchy Matters

Selectors read structure, not only names

Core Ideas

- HTML elements exist inside a parent-child tree.
- A selector can match by tag name, class, ID, or relationship.
- Child and descendant selectors only make sense when hierarchy is understood.
- Strong CSS begins with clean HTML structure.

Mental Model

Think of the page as a tree. Parent, child, and descendant relationships guide targeting decisions.

Before Styling

Ask: Which element do I mean? Is it unique, reusable, or defined by where it lives in the page?

Tag Selectors

The simplest way to style every matching HTML element

Core Ideas

- A tag selector uses the HTML tag name directly.
- It affects all matching elements on the page.
- It is useful for base styling such as body, p, img, or li.
- It can become too broad when the page grows.

Common Pattern

Use tag selectors to set shared defaults before adding more specific classes.

Typical Examples

body for page-wide defaults
p for paragraph rhythm
img for consistent media sizing

Tag Selector Example

A small page uses tag names to style list items and images

HTML

```
<body>
  <ol>
    <li>HTML</li>
    <li>CSS</li>
    <li>JavaScript</li>
    <li>ASP.NET Core</li>
  </ol>
  <img src="photo-1.jpg">
  <img src="photo-2.jpg">
</body>
```

CSS

```
body {
  background-color: skyblue;
}
li {
  color: black;
  font-size: 16px;
  text-decoration: underline;
}
img {
  width: 220px;
  height: 130px;
}
```

Practice Prompt

Change the li selector and observe how one rule updates every list item at once.

ID Selectors

Use an ID when one element is intentionally unique

Core Ideas

- An ID selector starts with #.
- It should point to one unique element on the page.
- IDs are useful for a hero banner, a special panel, or an anchor target.
- Reusing the same ID on many elements is poor HTML practice.

Memory Rule

One page, one ID value. Treat it like a unique name, not a reusable label.

Good Examples

#mainHeader
#courseBanner
#siteFooter

ID Selector Example

Three boxes use unique IDs so each can get its own color

HTML

```
<div id="box1">H</div>
<div id="box2">T</div>
<div id="box3">M</div>
<div id="box4">L</div>
```

CSS

```
div {
  width: 90px;
  height: 90px;
  display: inline-block;
  text-align: center;
  font-size: 30px;
  line-height: 90px;
  color: white;
}
#box1 { background-color: tomato; }
#box2 { background-color: teal; }
#box3 { background-color: purple; }
#box4 { background-color: darkorange; }
```

Practice Prompt

Add one more box with a new ID and a new color, then discuss why class selectors become better when many boxes share the same structure.

Class Selectors

The most reusable selector for everyday component styling

Core Ideas

- A class selector starts with a dot.
- Classes can be reused on many elements.
- They are ideal for cards, buttons, badges, and repeated layouts.
- Modern CSS work relies heavily on well-named classes.

Why Classes Matter

A class describes a reusable role. It keeps styles modular without pretending every element is unique.

Naming Advice

Choose names based on purpose:
.card, .button, .warning, .hero-title

Class Selector Example

A reusable style is shared by multiple cards

HTML

```
<section>
  <div class="skill-card">HTML</div>
  <div class="skill-card">CSS</div>
  <div class="skill-card">JavaScript</div>
</section>
```

CSS

```
body { background-color: lightcyan; }
.skill-card {
  width: 160px;
  height: 90px;
  display: inline-block;
  margin-right: 12px;
  text-align: center;
  line-height: 90px;
  background-color: #2563eb;
  color: white;
  border-radius: 10px;
}
```

Practice Prompt

Edit one class rule and notice how every matching card updates together.

Grouping Selectors

One rule can target multiple selectors at the same time

Core Ideas

- Grouping uses commas between selectors.
- It helps apply a shared style to different elements.
- It reduces repetition when the same visual rule fits several targets.
- It should be used when the shared style is genuinely the same.

Example

```
h1, h2, h3 {
  color: #1d4ed8;
}
```

When to Avoid It

Do not group selectors that only look similar for one moment. Group them only when the shared meaning is stable.

Child and Descendant Selectors

Relationship selectors depend on HTML nesting

Core Ideas

- A child selector uses > and matches only direct children.
- A descendant selector uses a space and matches nested elements at any depth.
- These selectors are useful inside menus, articles, and card content blocks.
- They reward clean structure and punish messy nesting.

Quick Difference

nav > a means direct links inside nav.

article a means any link that appears anywhere inside article.

Debug Habit

When a relationship selector fails, inspect the HTML structure before blaming the CSS.

Child vs Descendant Example

A menu and an article use relationship-based targeting

HTML

```
<nav>
  <a href="#">Home</a>
  <a href="#">Projects</a>
</nav>
<article>
  <p>Read the <a href="#">full guide</a>.</p>
</article>
```

CSS

```
nav > a {
  color: white;
  background-color: #0f766e;
  padding: 8px 12px;
}
article a {
  color: #1d4ed8;
  text-decoration: underline;
}
```

Practice Prompt

Add a nested wrapper inside nav and test which links still match the child selector.

Pseudo-Classes

Style an element based on state or interaction

Core Ideas

- Pseudo-classes begin with a colon, such as :hover or :focus.
- They respond to states like hovering, clicking, or visiting.
- They are essential for interactive links and buttons.
- Good pseudo-class styling should support clarity, not surprise.

Useful Starting Set

:link, :visited, :hover, :active, :focus

Accessibility Reminder

Hover alone is not enough. Keyboard users also need focus styles they can see clearly.

Pseudo-Class Example

Link styles change across interaction states

HTML

```
<body>
  <a href="#">Course notes</a>
  <a href="#">Exercises</a>
</body>
```

CSS

```
body { background-color: indigo; }
a:link { color: orange; font-size: 22px; }
a:visited { color: pink; }
a:hover { background-color: white; cursor: pointer; }
a:active { font-size: 30px; }
```

Practice Prompt

Hover over the links, then click them, and describe exactly which rule changed the visual state.

Pseudo-Elements

Style a piece of an element instead of the full element

Core Ideas

- Pseudo-elements begin with a double colon, such as ::first-letter.
- They target part of the element content.
- They help create editorial touches or emphasis.
- They should support readability rather than feel decorative for no reason.

Common Examples

::first-letter
::first-line
::before
::after

Best Use

Article introductions, callout labels, or small UI accents where content structure stays clear.

Pseudo-Element Example

A paragraph gets a styled opening letter and first line

HTML

```
<div class="story-box">
  <p>Responsive design improves comfort across many
screens and makes content easier to read.</p>
</div>
```

CSS

```
.story-box {
  width: 300px;
  background-color: indigo;
  color: white;
  padding: 16px;
}
p::first-letter {
  font-size: 24px;
  font-weight: bold;
}
p::first-line {
  background-color: orange;
  color: black;
}
```

Practice Prompt

Change the paragraph length and see how the first-line style responds to the actual rendered line.

Selector Strategy

Choosing selectors well makes future styling easier

Core Ideas

- Start with broad defaults using tag selectors.
- Use classes for reusable components.
- Keep IDs for unique cases.
- Use relationship selectors only when structure is meaningful.
- Prefer clarity over extreme specificity.

Simple Rule

If a selector feels hard to read, it will probably be hard to maintain.

Team Benefit

Predictable selectors make debugging faster and help multiple students work inside the same project without confusion.

The Box Model

Every visual element on a page behaves like a layered box

Core Ideas

- CSS boxes contain content, padding, border, and margin.
- Layout bugs often come from mixing up these layers.
- Understanding the box model turns random styling into controlled spacing.
- Even the body element acts as a large outer box.

Layer Order

content → padding → border → margin

Practical Result

When spacing looks wrong, ask whether the missing or excessive space belongs inside the element or outside it.

CSS Units for Size

Pick units based on what should stay fixed and what should scale

Core Ideas

- px gives a fixed value in pixels.
- % responds to the size of a related container.
- em depends on the font size of the current context.
- rem depends on the root font size and creates more stable scaling.

Beginner Baseline

Use px for first experiments, then introduce %, em, and rem when students start building responsive layouts.

Quick Comparison

Fixed feeling: px

Container-relative: %

Text-relative: em / rem

Width and Height

Size rules define the content box unless other rules change the calculation

Core Ideas

- width and height control the size of the content area.
- max-width and min-width create safer responsive behavior.
- Fixed heights can break when content grows unexpectedly.
- A readable layout often needs limits, not only large sizes.

Strong Default

max-width is often more helpful than a rigid width for text-heavy areas.

Example Pattern

```
.container {
  width: 100%;
  max-width: 960px;
}
```

Margin

Margin creates space outside the border

Core Ideas

- Margin separates one element from surrounding elements.
- It controls outer rhythm and visual breathing room.
- Vertical margin behavior can feel strange because margins may collapse.
- Consistent spacing scales are more readable than random values.

Memory Rule

Margin controls the distance between boxes, not the comfort inside the box.

Useful Pattern

```
.card {
  margin-bottom: 24px;
}
```

Border

Borders define the edge of a box

Core Ideas

- A border sits between padding and margin.
- It can change width, style, color, and radius.
- Thin borders usually guide the eye without overpowering the content.
- Rounded corners soften rigid boxes and help grouping.

Common Border Rule

```
border: 1px solid #d1d5db;
```

Design Advice

Use borders to clarify structure, not to add visual noise around every element.

Padding

Padding creates room inside the border

Core Ideas

- Padding keeps text and media from touching the edge of a box.
- It improves comfort and clarity.
- Buttons, cards, alerts, and banners all rely on good padding.
- Too little padding feels cramped; too much can make a component weak and oversized.

Memory Rule

Padding is inner comfort. Margin is outer rhythm.

Simple Example

```
.button {
  padding: 10px 16px;
}
```

Content Area and box-sizing

A more predictable box model improves component sizing

Core Ideas

- By default, width describes only the content area.
- Padding and border can increase the final visual size.
- box-sizing: border-box makes width include padding and border.
- Most modern projects use border-box as a baseline reset.

Recommended Global Rule

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

Why It Helps

Students spend less time fighting width calculations and more time understanding layout decisions.

Display Values

Display changes how an element participates in layout flow

Core Ideas

- display: block starts on a new line and can grow full width.
- display: inline stays inside text flow.
- display: inline-block mixes line flow with box sizing.
- display: none hides the element from layout completely.

Important Difference

display: none removes the element from layout. visibility: hidden would keep its space.

When Students Struggle

If width or height seems ignored, the display value is often the reason.

Block, Inline, and Inline-Block

The same boxes behave differently when the display value changes

HTML

```
<body>
  <div>H</div>
  <div>T</div>
  <div>M</div>
  <div>L</div>
</body>
```

CSS

```
body { background-color: skyblue; }
div {
  border: 2px solid gray;
  width: 70px;
  height: 70px;
  background-color: orange;
  text-align: center;
  line-height: 70px;
  display: inline-block;
}
```

Practice Prompt

Switch inline-block to block and inline, then describe how width, height, and line breaks respond.

Float and Clear

Classic layout tools that still matter for reading legacy CSS

Core Ideas

- float moves an element to the left or right of its container.
- Other content can wrap around the floated element.
- clear stops following elements from wrapping around floats.
- Floats are older than Flexbox and Grid, but students still meet them in real code.

Key Pair

float changes flow.

clear restores clean flow after floating.

Modern Perspective

Use float to understand older layouts. Use Flexbox and Grid for most new interface layout work.

A Classic Multi-Column Layout

Header, columns, content area, and footer with floats

Core Ideas

• A simple float layout can create a header, two sidebars, a main content area, and a footer.

• Column widths must be planned carefully.

• The footer usually needs clear: both.

• This pattern helps students understand why newer layout systems became popular.

Essential Footer Rule

```
#footer {
  clear: both;
}
```

Teaching Use

This is a good comparison slide before introducing students to more modern layout systems in later weeks.

Overflow, Opacity, Shadow, and Radius

Small visual rules that strongly affect component feel

Core Ideas

- overflow controls what happens when content exceeds the box.
- opacity changes transparency.
- box-shadow adds depth.
- border-radius softens corners and changes the personality of a component.

Best Practice

Use these properties with restraint. A calm, readable interface usually beats an overloaded one.

Useful Starter Values

```
overflow: auto;
opacity: 0.95;
box-shadow: 0 6px 18px rgba(0,0,0,.12);
border-radius: 8px;
```

Positioning Basics

Position changes where a box sits relative to flow or a reference area

Core Ideas

- static is the normal default position.
- relative offsets the element while keeping its original space.
- absolute removes the element from normal flow and positions it from a reference ancestor.
- fixed attaches the element to the viewport.

Coordinate Helpers

top, right, bottom, and left work with positioned elements.

First Debug Question

If absolute positioning looks strange, ask: which parent is acting as the reference box?

Relative vs Absolute Position

One keeps space in flow, the other leaves flow entirely

Core Ideas

- position: relative moves the element but preserves its original place.
- position: absolute lets the element overlap or escape normal layout flow.
- Absolute positioning is useful for badges, icons, and layered details.
- It can break quickly when used for large page structure.

Reliable Pairing

Give the parent position: relative when a child needs absolute positioning inside that area.

Mini Example

```
.card { position: relative; }
.badge { position: absolute; top: 12px; right: 12px; }
```

Fixed Position and z-index

Useful for sticky tools, quick actions, and layer control

Core Ideas

- position: fixed locks an element to the viewport.
- z-index controls which positioned element appears on top.
- Floating action buttons and back-to-top controls often use these rules.
- Layer control should stay deliberate to avoid confusing overlap.

Good Use Cases

Back-to-top button
Chat bubble
Floating help shortcut

Caution

A high z-index is not a design solution by itself. It should reflect a clear visual priority.

Building a Card Component

A small component can combine size, spacing, typography, and position

Core Ideas

- Start with a wrapper box.
- Add padding for comfort.
- Use a gentle border and a short shadow.
- Place a label or icon with absolute positioning if needed.
- Control the hierarchy with font size, weight, and color.

Why Cards Matter

Cards are small enough for practice and rich enough to demonstrate many CSS rules in one place.

Student Goal

Students should be able to explain every property in the card without calling anything “random styling.”

Card Component Code

A compact example that mixes layout and visual styling

HTML

```
<article class="card">
  <span class="badge">New</span>
  <h3>CSS Basics Lab</h3>
  <p>Practice selectors, spacing, and responsive
thinking.</p>
</article>
```

CSS

```
.card {
  position: relative;
  max-width: 320px;
  padding: 24px;
  border: 1px solid #d8dee6;
  border-radius: 8px;
  background: white;
  box-shadow: 0 10px 24px rgba(0,0,0,.08);
}
.badge {
  position: absolute;
  top: 12px;
  right: 12px;
  background: #1d4ed8;
  color: white;
  padding: 4px 8px;
}
```

Practice Prompt

Ask students to restyle the badge and increase the card readability without changing the HTML structure.

Color Foundations

Color changes meaning, hierarchy, and emotional tone

Core Ideas

- CSS can style text, borders, backgrounds, and shadows with color.
- Strong contrast improves readability.
- Accent colors should guide attention, not compete with every element.
- A small, consistent palette often looks more professional than many random colors.

Starter Palette Advice

Choose one primary color, one support color, one surface color, and one text color before styling a full page.

First Readability Rule

Dark text on light backgrounds and light text on dark backgrounds usually create the safest baseline.

RGB, HEX, and HSL

CSS offers several ways to write color values

Core Ideas

- HEX uses values such as #1d4ed8.
- RGB uses rgb(29, 78, 216).
- HSL uses hue, saturation, and lightness.
- Students should learn to recognize all three formats in real code.

Same Color, Different Syntax

#1d4ed8
rgb(29, 78, 216)
hsl(221, 76%, 48%)

Teaching Benefit

HSL is often the easiest format for explaining how a color becomes lighter, darker, calmer, or more intense.

Gradients and Layered Backgrounds

Backgrounds can do more than fill a flat color

Core Ideas

- background-image can place a photo or a gradient behind content.
- Gradients help create depth without extra image files.
- Background choices should still protect readability.
- Layered backgrounds work best when text contrast is tested carefully.

Useful Gradient Pattern

```
background: linear-gradient(135deg, #0f172a, #1d4ed8);
```

Common Mistake

An exciting background can become a problem if it fights the text instead of supporting it.

Background Control in Practice

Size, position, repeat, and attachment shape the final result

Core Ideas

- background-size controls scaling.
- background-position controls focus area.
- background-repeat controls tiling.
- background-attachment changes how the background behaves during scroll.

Safe Photo Settings

```
background-size: cover;
background-position: center;
background-repeat: no-repeat;
```

Design Habit

When using an image background, test the layout on narrow and wide screens so the important area stays visible.

Typography Foundations

Good typography makes content easier to follow and remember

Core Ideas

- font-family shapes tone.
- font-size builds hierarchy.
- font-weight adds emphasis.
- line-height improves reading comfort.
- letter-spacing and text-transform can create labels and navigation styles.

Priority Order

Readability first, hierarchy second, personality third.

Strong Beginner Rule

One clear body font and a calm, repeatable size scale usually outperform overdesigned typography.

Type Scale and Spacing

Consistent size relationships make a page feel designed

Core Ideas

- Headings should clearly stand apart from body text.
- Paragraphs need comfortable line height.
- Labels and metadata can use smaller sizes and more muted colors.
- Inconsistent font sizes make a page feel accidental.

Reusable Scale

H1: 36px
H2: 28px
Body: 16px
Meta: 13px

Spacing Companion

Typography is not only font choice. It also includes line-height, paragraph gaps, and alignment decisions.

Article Example: HTML and CSS Together

A short article block combines type, spacing, and background choices

HTML

```
<article class="article-card">
  <p class="eyebrow">Front-End Basics</p>
  <h1>Readable CSS starts with structure.</h1>
  <p class="meta">Week 9 • 15 April</p>
  <p class="summary">Small typography choices create
big improvements in clarity.</p>
</article>
```

CSS

```
.article-card {
  max-width: 560px;
  padding: 28px;
  background: white;
  border-radius: 8px;
}
.eyebrow {
  text-transform: uppercase;
  letter-spacing: 1.6px;
  color: #1d4ed8;
  font-weight: 700;
}
h1 { font-size: 36px; line-height: 1.15; }
.meta { color: #64748b; font-size: 13px; }
.summary { line-height: 1.7; }
```

Practice Prompt

Adjust only typography-related rules and explain how the mood changes without touching layout dimensions.

Small Motion and Micro-Interaction

Transition, cursor, filter, and scroll behavior add polish when used carefully

Core Ideas

- transition softens property changes.
- cursor helps communicate clickability.
- filter can adjust image appearance.
- scroll-behavior can smooth in-page navigation.
- These rules should support usability rather than distract from it.

Subtle Transition Example

```
a {
  transition: color 0.3s ease;
}
```

Use Restraint

Micro-interactions feel best when they are quick, intentional, and not competing with the main content.

What Responsiveness Means

A responsive design adapts to different screen sizes and contexts

Core Ideas

• The same page may be viewed on a phone, tablet, laptop, or large monitor.
• Responsive design keeps content readable and usable across those environments.
• A layout should adapt without breaking hierarchy or accessibility.
• CSS plays the main role in these adjustments.

Core Goal

The design should feel natural on each device, not merely “shrunk down.”

First Student Insight

Responsiveness is not only about width. It also affects spacing, type size, media behavior, and interaction comfort.

Fluid Layout Rules

Responsive pages rely on flexibility more than fixed dimensions

Core Ideas

- Use width: 100% when an element should fill available space.
- Use max-width to stop lines from becoming too long.
- Prefer flexible containers over rigid page widths.
- Let spacing and type adjust in a controlled way.

Useful Container Pattern

```
.page {
  width: 100%;
  max-width: 1100px;
  margin: 0 auto;
}
```

Design Effect

A fluid layout creates comfort on small screens without making large screens feel empty or chaotic.

Viewport Meta and Flexible Media

Responsiveness begins in HTML and continues in CSS

Core Ideas

- The viewport meta tag helps mobile browsers render layouts correctly.
- Images should not overflow narrow screens.
- Videos and embeds also need flexible sizing.
- Responsive thinking starts before media queries are written.

Important HTML Line

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

Safe Media Rule

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

Media Query Syntax

CSS can apply different rules when a condition becomes true

Core Ideas

- @media introduces a conditional block.
- Conditions often check screen width.
- Different ranges can activate different visual rules.
- Media queries work best when the base design is already clean.

Basic Pattern

```
@media screen and (min-width: 600px) {
  body { background-color: #e76f51; }
}
```

Teaching Reminder

Students should learn to read a media query as a sentence: “When the screen is at least this wide, apply these rules.”

Mobile-First Breakpoints

Start with the smallest layout, then add room as the screen grows

Core Ideas

- Write the simplest layout for narrow screens first.
- Add @media rules for wider breakpoints.
- This approach keeps the core experience focused.
- It also reduces the risk of hiding mobile problems under desktop assumptions.

Common Breakpoints

600px for small tablets

800px for larger tablets

1200px for wide desktop layouts

Why Mobile-First Helps

It forces students to prioritize content and spacing before they have extra screen space to depend on.

Responsive Background Example

A simple practice page changes background color at wider widths

HTML

```
<body>
  <h1>Responsive Practice</h1>
  <p>Resize the browser to test the breakpoints.</p>
</body>
```

CSS

```
body {
  background-color: #a8dadc;
  color: white;
}
@media screen and (min-width: 600px) {
  body { background-color: #e76f51; }
}
@media screen and (min-width: 800px) {
  body { background-color: #457b9d; }
}
@media screen and (min-width: 1200px) {
  body { background-color: #e9c46a; color: #111827; }
}
```

Practice Prompt

Resize slowly and ask students to say exactly which rule became active at each breakpoint.

Responsive Banner Swap

A small banner and a large banner can appear at different device widths

HTML

```
<div id="smallAd">Small-screen banner</div>
<div id="largeAd">Large-screen banner</div>
```

CSS

```
#smallAd, #largeAd {
  padding: 18px;
  background: #1d4ed8;
  color: white;
}
#largeAd { display: none; }
@media screen and (min-width: 750px) {
  #smallAd { display: none; }
  #largeAd { display: block; }
}
```

Practice Prompt

Discuss when hiding and showing content is helpful and when a flexible shared component would be the better design choice.

Responsive Card Grid

A repeated component layout can expand gracefully on wider screens

Core Ideas

- One-column layouts are often best on narrow screens.
- Wider screens can support two, three, or more columns.
- Grid or Flexbox will usually handle this more cleanly than floats.
- The content should remain readable at every stage.

Modern Pattern

```
.grid {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(auto-fit,
minmax(220px, 1fr));
}
```

Bridge to Later Weeks

This is where responsive thinking connects naturally to Flexbox and Grid lessons that follow in the course sequence.

Guided Lab

A classroom activity that combines methods, selectors, spacing, and responsiveness

Core Ideas

1. Create a small announcement card in HTML.
2. Style it with an external CSS file.
3. Use at least one class selector and one pseudo-class.
4. Add padding, border-radius, and a subtle shadow.
5. Write one media query that improves the layout at 800px and above.

Lab Goal

Students should be able to explain both the visual result and the logic behind each rule they used.

Minimum Feature Set

Readable typography
Clear spacing
Safe contrast
A visible hover state
A layout change at a breakpoint

Common Mistakes and Debugging

Most CSS problems become easier when the right question is asked first

Core Ideas

- If spacing looks wrong, inspect the box model.
- If a style does not apply, inspect the selector.
- If size feels broken, inspect display and box-sizing.
- If the layout fails on mobile, inspect fixed widths and overflow.
- If overlap feels random, inspect position and z-index.

Best Debug Habit

Change one thing at a time and verify the visible effect before adding a second fix.

Student Mindset

Confusion usually shrinks when the problem is named clearly: targeting, spacing, sizing, flow, or responsiveness.

Summary and Homework

A strong CSS beginner controls both clarity and behavior

Core Ideas

- CSS methods decide where style lives.
- Selectors decide what gets styled.
- The box model decides how space works.
- Typography and backgrounds decide how the page feels.
- Media queries help the design adapt across devices.

Homework

Create one original student-facing component such as a course card, event notice, or profile teaser.

Use external CSS, at least three selectors, one pseudo-class, and one media query.

Quality Standard

The final work should look calm, readable, and intentional on both a narrow and a wide screen.