HTML & CSS Basics — Selectors, Cascade, Specificity & Accessibility

Web Technologies — Lecture 4

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

Minimal HTML5 Document

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First Page</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Hello, Zanzibar!</h1>
  <p>Welcome to the web.</p>
  <script src="app.js"></script>
</body>
</html>
```

Element Anatomy

- Elements have an **opening tag**, content, and **closing tag**: `<p>hi</p>`
- Some are **void** (no closing tag): `<img>`, `<br>`, `<input>`, `<meta>`
- Elements have **attributes**: `<a href="...">link</a>`
- Elements nest — they form a tree (the **DOM**)

Semantic HTML5 Elements

```
<header>   ... site header ... </header>
<nav>      ... navigation ... </nav>
<main>
  <article>
    <header><h1>Article title</h1></header>
    <section> ... content ... </section>
    <footer> ... article footer ... </footer>
  </article>
  <aside>   ... sidebar ... </aside>
</main>
<footer>   ... site footer ... </footer>
```

Semantic tags help **accessibility**, **SEO**, and code readability.

CSS Rule Anatomy

```
p {                          /* selector  */
  color: navy;               /* declaration: property + value */
  font-size: 16px;           /* declaration */
}                            /* declaration block */
```

Vocabulary you must know

- **Rule** = selector + declaration block
- **Selector** — which elements to style
- **Property** — what to change (`color`, `margin`, `font-size`)
- **Value** — the new value
- **Declaration** — one property/value pair, ends with `;`

Three Ways to Apply CSS

```
<!-- 1. Inline (avoid -- mixes content and presentation) -->
<p style="color: red;">Red text</p>

<!-- 2. <style> block in <head> (OK for tiny pages) -->
<style>
  p { color: red; }
</style>

<!-- 3. External file (preferred, cacheable, reusable) -->
<link rel="stylesheet" href="styles.css">
```

Basic Selectors

```
/* element (type) selector */
p               { color: navy; }

/* class selector -- many elements may share one class */
.warning        { color: red; font-weight: bold; }

/* id selector -- exactly one element */
#site-logo      { width: 120px; }

/* universal */
*               { box-sizing: border-box; }

/* attribute */
input[type="email"] { border-color: blue; }
a[href^="https://"] { color: green; }   /* starts with */
```

Pseudo-Classes & Pseudo-Elements

```css
/* Pseudo-classes (state) */
a:hover            { text-decoration: underline; }
a:focus            { outline: 2px solid blue; }
button:disabled    { opacity: 0.5; }
li:first-child     { font-weight: bold; }
li:nth-child(odd)  { background: #f8f8f8; }

/* Pseudo-elements (parts of an element) */
p::first-letter    { font-size: 2em; }
p::before          { content: "* "; }
```

Pseudo-classes use `:`, pseudo-elements use `::`.

The Four Combinators

```css
/* Descendant (space) -- any depth inside */
nav a               { color: white; }

/* Child (>) -- direct child only */
ul > li             { list-style: square; }

/* Adjacent sibling (+) -- the next element only */
h2 + p              { font-size: 1.1em; }

/* General sibling (~) -- any later sibling */
h2 ~ p              { color: gray; }
```

Why this matters: you can target precise spots in the DOM tree without adding extra classes.

Grouping Selectors

```css
/* same rule for several selectors */
h1, h2, h3 {
  font-family: 'Inter', sans-serif;
  line-height: 1.2;
}

/* combine them */
nav a:hover,
.btn:hover { text-decoration: underline; }
```

The Cascade — Whose Rule Wins?

When multiple rules apply to the same element, the browser picks one using:

1. **Origin & importance** — author rules beat browser defaults; `!important` beats normal
2. **Specificity** — more specific selector wins (next slide)
3. **Source order** — if both above are equal, the *later* rule wins

Mnemonic

Origin > Specificity > Source order.

Specificity — Scoring

Specificity is a 4-tuple: `(inline, id, class, element)`

Selector	Specificity
`p`	$(0, 0, 0, 1)$
`p.note`	$(0, 0, 1, 1)$
`.note`	$(0, 0, 1, 0)$
`ul li.note`	$(0, 0, 1, 2)$
`#main`	$(0, 1, 0, 0)$
`#main p.note`	$(0, 1, 1, 1)$
`style="..."` (inline)	$(1, 0, 0, 0)$

Compare left-to-right: the bigger tuple wins.

Specificity in Action

```
p               { color: blue; }      /* (0,0,0,1) */
p.note          { color: green; }     /* (0,0,1,1) -- wins over plain p */
#main p.note    { color: red; }       /* (0,1,1,1) -- wins over above */
```

Rule of thumb: prefer *simple* selectors and classes. Avoid `!important` except as a last resort — it breaks the cascade and is hard to override later.

!important — Use Sparingly

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

- Beats *all* other declarations of the same property
- Two `!important` rules battle by specificity, then source order
- Acceptable for: utility classes (`.hidden { display:none !important; }`)
- Bad for: regular component styles

Which Properties Inherit?

Some properties pass from parent to child automatically; others don't.

Inherit by default	**Do NOT inherit**
`color`	`margin`, `padding`
`font-family`, `font-size`	`border`, `background`
`line-height`	`width`, `height`
`text-align`	`display`, `position`
`visibility`	`box-shadow`

Force it: `property: inherit;`

Reset: `property: initial;` Use parent or initial: `property: unset;`

Good Practices

- Prefer **classes** over IDs for styling (lower specificity, reusable)
- Use **semantic** class names (`.alert-danger`, not `.red-box`)
- Keep selectors **short** (max 2–3 levels deep)
- Use **CSS variables** for colours, spacing, fonts
- Mobile-first; layer larger-screen styles via media queries
- Group related rules together; comment "why" choices
- Use **relative units** (`rem`, `em`, `%`) for fonts and spacing

Bad Practices — Avoid

- !important everywhere → specificity wars
- Inline `style="..."` for ordinary styling
- Deep selectors: `.page #main ul.list li.item a span`
- Magic numbers: `margin-top: 37px` (where did 37 come from?)
- ID selectors for shared styles
- Pixel-only sizing for text
- Duplicating colours/values everywhere instead of variables

Forms — HTML

```
<form action="/login" method="POST">
  <label for="email">Email</label>
  <input id="email" name="email" type="email" required>

  <label for="pw">Password</label>
  <input id="pw" name="password" type="password" required minlength="8">

  <button type="submit">Log in</button>
</form>
```

Input types: `text`, `email`, `password`, `number`, `date`, `file`, `checkbox`, `radio`, `range`, `color`, `tel`, `url`.

Why Accessibility Matters

- $\sim$15% of people live with some disability
- Many countries (incl. Tanzania moving) have legal a11y requirements
- Accessible design is **better for everyone**: keyboard users, slow networks, mobile, future-you
- Standards: **WCAG 2.1** levels A, AA (target), AAA

Quick Wins

- Always provide `alt` on images:

```
<img src="logo.png" alt="SUZA logo">
<img src="decorative.svg" alt="">        <!-- empty alt = decorative -->
```

- Use `<label for="...">` on every form field
- Use **semantic** elements: `<button>`, not `<div onclick>`
- Visible focus styles: don't `outline: none` without a replacement
- Sufficient **contrast** (WCAG AA: 4.5:1 for body text)
- Descriptive link text: "Read the report", not "click here"

ARIA — When Semantics Aren't Enough

```
<button aria-label="Close dialog" aria-pressed="false">X</button>

<nav aria-label="Main navigation"> ... </nav>

<div role="alert">Form saved successfully.</div>

<input aria-describedby="email-help">
<small id="email-help">We'll never share your email.</small>
```

First rule of ARIA: *don't use ARIA if a plain HTML element does the job.*

Test Your Page

- Navigate the whole page with the **Tab key** only
- Use a screen reader: NVDA (Win), VoiceOver (Mac, Cmd+F5), TalkBack
- Run a tool: **axe DevTools**, Lighthouse, WAVE
- Check colour contrast with browser dev tools
- Disable CSS — does the page still make sense?

Summary

- HTML provides **structure**; CSS provides **style**
- Selectors: type, class, id, attribute, pseudo-classes, combinators
- Cascade: **origin** > **specificity** > **source order**
- Specificity tuple `(inline, id, class, element)`; bigger wins
- Inheritance: text-related props inherit; layout props don't
- Accessibility = legally required, helps everyone, easy wins available

Next: CSS Layout — the box model, Flexbox, and Grid.

Questions?