

# Web Design and Client Side Scripting



brunkonjaa

Week 2

# IDs vs Classes + Specificity

## What they are (and why they exist)

- **Hooks for code:**
  - ✓ id and class give CSS/JS reliable handles when elements alone aren't enough.
- **Analogy:**
  - ✓ Class = barcode (shared by many).
  - ✓ ID = serial number (unique to one).

## Core rules

- **IDs are unique:**
  - ✓ One element may have at most one id.
  - ✓ A document may have only one element with a given id.
  - ✓ Duplicate IDs break validation and confuse JS.
- **Classes are reusable:**
  - ✓ Same class on many elements.
  - ✓ An element can have multiple classes (space-separated).
- **No built-in magic:**
  - ✓ id/class do nothing until CSS or JS targets them.

## Browser & JS behavior that actually matters

- **Deep links:**
  - ✓ page.html#comments scrolls to id="comments". This is why IDs must be unique.
- **CSS doesn't care, JS cares:**
  - ✓ CSS can style by either #id or .class.
  - ✓ JS often expects unique IDs (getElementById), while classes are ideal for toggling state (element.classList.add/remove).

### CSS specificity — who wins?

- **Ladder: Inline style**
  - ✓ `"..." > ID #x > Class/attr/pseudo .x > Tag p.`
- **Ties:**
  - ✓ If specificity ties, the later rule in the stylesheet wins.
- **Implications:**
  - ✓ Overusing #id in selectors makes later overrides painful; prefer classes to keep specificity low.

### Naming & semantics (future-proof hooks)

- Prefer purpose over position/appearance:
  - ✓ `id="sidebar"` beats `id="right-col"`.
- Avoid meaningless hooks like class
  - ✓ `"link"` on `<a>`—the tag already states it's a link.
  - ✓ Keep names consistent and human-readable.

### Microformats (quick primer)

- **Microformats**
  - ✓ standardized class names for real-world data (e.g., contact info). They add machine-readable meaning without changing appearance.

### Common gotchas (and fixes)

- **Duplicate IDs:**
  - ✓ Make them unique or convert repeated ones to classes.
- **Inline styles trump everything:**
  - ✓ Move styles into CSS files.
- **Order blindness:**
  - ✓ When selectors tie, later wins—check stylesheet order.
- **Selector typos:**
  - ✓ `.class` uses a dot; `#id` uses a hash.
- **User-agent defaults:**
  - ✓ Browser defaults (e.g., blue underlined links) can mislead—reset or override explicitly.

## Tiny examples

- **Reusable + modifier classes**
  - ✓ `<div class="widget"></div>`
  - ✓ `<div class="widget big"></div>`
  - ✓ `<div class="widget"></div>`
  - ✓ `/* .widget { ... }.big { ... } */`
- **ID for deep link + class for shared style**
  - ✓ `<li id="comment-27299" class="item">...</li>`
  - ✓ `/* #comment-27299 { ...anchor-specific... }.item { ...shared... } */`
- **Specificity tie broken by order**
  - ✓ `p { color: red; }`
  - ✓ `p { color: green; } /* green wins (later in source) */`
- **Quick checklist before you ship**
  - ✓ Use classes for anything that repeats;
  - ✓ layer variations as extra classes
  - ✓ Reserve IDs for unique anchors or one-off JS targets
  - ✓ Never duplicate an ID;
  - ✓ validate your HTML.
  - ✓ Keep selectors low-specificity;
  - ✓ avoid sprinkling #id everywhere.
  - ✓ Remove inline styles; centralize rules in CSS.
  - ✓ Name hooks semantically, not by location or color.

### Rule-of-thumb conclusion:

Classes are your reusable uniforms; IDs are one-off passports. Design with classes first, sprinkle IDs only where uniqueness or deep-linking truly matters—and let the specificity ladder guide your overrides.

Next step: apply this to your CV page—replace any duplicate IDs with classes, rename positional IDs to semantic names, and move any inline styles into your stylesheet.