Homework - CSS Teaching Methodology

Below are **clean, exam-ready + interview-ready revision notes** for **CSS Selectors**, followed by a **clear, step-by-step specificity calculation section**.

This is written so you can:

- Revise in **10–15 minutes**
- Answer **theory + practical questions**
- Debug CSS confidently

No fluff. Only what matters.

✅ CSS SELECTORS — REVISION NOTES (CONSOLIDATED)

1️⃣ What a Selector Is (1-Line Definition)

A **CSS selector is a DOM query** that decides **which elements are allowed to receive styles**.

> CSS does not style elements directly — it styles **selector matches**.

2️⃣ Selector Categories (Mental Map)

Think of selectors in **5 logical groups**:

```pgsql
WHAT   → element, class, id
WHERE  → descendant, child, sibling
FILTER → attribute selectors
WHEN   → pseudo-classes (state/position)
PART   → pseudo-elements
```

3️⃣ Core Selector Types (Quick Notes)

🔹 Element Selector

```css
p { }
```

- Selects all `<p>`
- Global
- Low priority
- Used for base styles

🔹 Class Selector (MOST IMPORTANT)

```css
.card { }
```

- Reusable
- Scalable
- Component-based
- Preferred in production

🔹 ID Selector

```css
#header { }
```

- Unique
- Very high priority
- Avoid for styling
- Hard to override

🔹 Universal Selector

```css
* { }
```

- Selects everything
- Used in resets

- Dangerous if overused

4️⃣ Relationship Selectors (WHERE)

🔹 Descendant (space)

```
css

.card p { }
```

- Any `<p>` inside `.card`
- Fragile if DOM changes

🔹 Child (`>`)

```
css

.card > p { }
```

- Only direct children
- Safer for layouts

🔹 Adjacent Sibling (`+`)

```
css

label + input { }
```

- Only next sibling

🔹 General Sibling (`~`)

```
css

h2 ~ p { }
```

- All following siblings

5️⃣ Attribute Selectors (FILTERING)

Pattern	Meaning	Example
`[attr]`	Exists	`input[type]`
`=`	Exact	`type="email"`
`^=`	Starts with	`href^="https"`
`$=`	Ends with	`src$=".png"`
`*=`	Contains	`class*="card"`
`~=`	Whole word	`data-role~="admin"`
`` ` ``	`=` `	Hyphen prefix

Used heavily in:

- Forms
- Variants
- Clean HTML without extra classes

6️⃣ Pseudo-Classes (WHEN)

```css
:hover
:focus
:active
:first-child
:last-child
:nth-child(n)
```

- State-based or position-based
- No DOM change
- Critical for UX & accessibility

7️⃣ Pseudo-Elements (PART)

```css
```

```
::before
::after
::first-letter
```

- Virtual elements
- Not in DOM
- Requires `content`
- Used for decoration or structure helpers

8️⃣ Professional Selector Rules (VERY IMPORTANT)

- Prefer **class selectors**
- Avoid deep nesting
- Avoid IDs for styling
- Style **roles**, not HTML tags
- CSS should survive DOM refactor
- If selector breaks after HTML change → selector was wrong

✅ CSS SPECIFICITY — REVISION NOTES

9️⃣ What Is Specificity?

Specificity is the priority score used by the browser when **multiple selectors target the same element**.

> Higher specificity wins — order matters only if specificity is equal.

🔟 Specificity Calculation Model (MEMORIZE THIS)

Specificity is calculated as:

```css
(A, B, C, D)
```

Where:

Column	Counts
A	Inline styles
B	ID selectors
C	Class, attribute, pseudo-class
D	Element, pseudo-element

Comparison rule:

- Compare **A**
- If tie → compare **B**
- Then **C**
- Then **D**

1️⃣1️⃣ Specificity Calculation Examples (VERY IMPORTANT)

Example 1

```css
p { }
```

Specificity:

```scss
(0, 0, 0, 1)
```

Example 2

```css
.card { }
```

Specificity:

```scss
```

```
(0, 0, 1, 0)
```

Example 3

```css
#header { }
```

Specificity:

```scss
(0, 1, 0, 0)
```

Example 4

```css
.card p { }
```

Counts:

- Class → 1
- Element → 1

Specificity:

```scss
(0, 0, 1, 1)
```

Example 5

```css
#app .card p:hover
```

Counts:

- ID → 1 (`#app`)
- Class → 1 (`.card`)
- Pseudo-class → 1 (`:hover`)
- Element → 1 (`p`)

Specificity:

```scss
(0, 1, 2, 1)
```

Example 6 (Inline Style)

```html
<p style="color:red">
```

Specificity:

```scss
(1, 0, 0, 0)
```

👉 Beats everything except `!important`.

1️⃣2️⃣ Specificity Comparison Example (IMPORTANT)

```css
p { color: blue; }          /* (0,0,0,1) */
.card p { color: red; }     /* (0,0,1,1) */
#box p { color: green; }    /* (0,1,0,1) */
```

Result:

- `#box p` wins → **green**

Because:

```scss
```

```
(0,1,0,1) > (0,0,1,1) > (0,0,0,1)
```

1️⃣3️⃣ `!important` (INTERVIEW NOTE)

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

Rules:

- Overrides specificity
- Breaks cascade
- Hard to debug
- Avoid in production

Allowed only for:

- Utility overrides
- Rare edge cases

1️⃣4️⃣ Common Specificity Mistakes

❌ Using IDs to fix CSS
❌ Adding more selectors blindly
❌ Overusing `!important`
❌ Writing deeply nested selectors

🧠 FINAL MEMORY BLOCK (WRITE THIS ON PAPER)

```yaml
Selector Types:
element < class < id < inline

Specificity:
(A, B, C, D)
```

Golden Rule:

Lower specificity = easier maintenance

🔜 NEXT (STRICT ORDER)

We now move to:

👉 **Colors & Units → Color Formats (hex, rgb, hsl, etc.)**

Reply **“Next”** when ready.