CSS / SCSS / TAILWIND - COMPLETE INTERVIEW GUIDE

Table of Contents

CSS FUNDAMENTALS ANSWERS

1. What is the CSS Box Model?

The **CSS Box Model** describes how elements are sized and spaced. From inside out: **Content** (actual content), **Padding** (inside spacing), **Border** (boundary), **Margin** (outside spacing). Each component can have different values. **Box-sizing** property controls what's included in width/height calculation. `border-box` includes padding and border in the width/height. Understanding box model is **fundamental** to CSS layout.

2. What are CSS selectors?

CSS selectors target HTML elements to apply styles. **Universal selector** (`*`): All elements. **Element selector**: Target specific tag. **Class selector** (`.classname`): Target by class. **ID selector** (`#id`): Target by ID (unique). **Attribute selector**: Target by attributes. **Descendant selector** (space): Select nested elements. **Child selector** (`>`): Direct children only.

3. What is CSS specificity?

Specificity determines which CSS rule applies when multiple rules target same element. Calculated as four-part value: `inline, id, class/attribute, element`. **Inline styles** have highest specificity (1000). **IDs** (100), **classes/attributes/pseudo-classes** (10), **elements/pseudo-elements** (1). **`!important`** overrides all (but discouraged). Higher specificity **wins** over lower specificity. Understanding specificity prevents styling issues.

4. What is CSS cascade?

CSS cascade is the algorithm that determines which style applies. Styles **flow downward** in stylesheets (later rules override earlier ones). **Specificity** determines importance of competing rules. **Specificity** wins over cascade order. **`!important`** can override cascade (but should be avoided). Selectors **closer to element** (more specific) usually win. Understanding cascade helps **manage styles effectively**.

5. What is the display property?

display property controls how element is rendered in layout. **`block`**: Takes full width, starts new line. **`inline`**: Flows within text, no line breaks. **`inline-block`**: Flows inline but behaves like block. **`flex`**: Flexible box layout. **`grid`**: CSS grid layout. **`none`**: Element removed from layout. Default depends on element type. Most important CSS property for layout.

6. What is Flexbox?

Flexbox (Flexible Box Layout) provides one-dimensional layout system. **Container** holds **flex items** in row or column. **`display: flex`** activates flexbox. **Main axis** and **cross axis** (perpendicular). Properties for container: `flex-direction`, `justify-content`, `align-items`, `flex-wrap`, `gap`. Properties for items: `flex`, `flex-grow`, `flex-shrink`, `flex-basis`, `align-self`. Modern standard for **responsive layouts**.

7. What is CSS Grid?

CSS Grid provides **two-dimensional** layout system. Uses **rows and columns** simultaneously. **`display: grid`** activates grid. Define template with `grid-template-rows`, `grid-template-columns`. **Gap** for spacing between items. **`grid-auto-flow`** controls item placement. **`grid-column`** and **`grid-row`** for item positioning. **Powerful for complex layouts** and **page structures**.

8. What is CSS positioning?

Positioning controls how element is placed in the layout. **`static`** (default): Normal document flow. **`relative`**: Positioned relative to normal position. **`absolute`**: Positioned relative to positioned parent. **`fixed`**: Positioned relative to viewport. **`sticky`**: Combination of relative and fixed. Use `top`, `right`, `bottom`, `left` properties. **`z-index`** controls stacking order for positioned elements.

9. What is CSS z-index?

z-index controls the **stacking order** of overlapping elements. Higher value appears **on top**. Only applies to **positioned elements** (`position` ≠ static). Creates **stacking context** (new reference frame). **Negative values** send element behind. **`auto`** is default (usually 0). Sibling elements compare z-index values. Understanding stacking context prevents **unexpected layering** issues.

10. What are CSS units?

CSS provides **absolute** and **relative** units. **Absolute**: `px` (pixels), `pt`, `in`, `cm`. **Relative**: `em` (relative to font-size), `rem` (relative to root), `%` (percentage), `vw/vh` (viewport), `ch` (character width). **`px`** commonly used for fixed sizes. **`em`/`rem`** better for **responsive** and **scalable** designs. **Viewport units** (`vw`, `vh`) for **full-screen** effects.

CSS ADVANCED ANSWERS

11. What are media queries?

Media queries apply styles based on **device characteristics**. Syntax: `@media (condition) { styles }`. Check **screen size** with `(max-width)`, `(min-width)`. **Breakpoints** define responsive tiers (mobile, tablet, desktop). **Mobile-first** approach: styles for small screens first, then add desktop. **Desktop-first** approach: opposite. Essential for **responsive design**. Media features: width, height, orientation, color.

12. What are CSS variables?

CSS custom properties (CSS variables) store reusable values. Defined with `--` prefix (e.g., `--primary-color`). Accessed with `var(--variable-name)`. Scoped to selector and children. **Fallback values**: `var(--var, fallback)`. Can be **updated dynamically** with JavaScript. Improves **maintainability** and **consistency**. Supported in **modern browsers**, requires fallback for old browsers.

13. What are CSS transitions?

Transitions smoothly animate property changes over time. Syntax: `transition: property duration easing-function`. Triggers on **property change** (hover, class change, etc.). Properties: `transition-property`, `transition-duration`, `transition-timing-function`, `transition-delay`. **`ease`** smoothest, **`linear`** constant speed, **`ease-in/out/in-out`** various curves. Cannot animate all properties. Useful for **interactive effects**.

14. What are CSS animations?

Animations create movement using **`@keyframes`**. Define animation with **start (0%) to end (100%)**. Apply with `animation` property: name, duration, timing, etc. **`animation-iteration-count`**: How many times to play. **`animation-direction`**: forward, reverse, alternate. **Can loop infinitely** with `infinite`. More control than transitions. Useful for **continuous effects** and **sequences**.

15. What are CSS transforms?

Transform property applies **2D or 3D transformations**. **`translate(x, y)`**: Move element. **`scale(x, y)`**: Resize element. **`rotate(angle)`**: Rotate element. **`skew(x, y)`**: Distort element. **`matrix()`**: Combined transformation. 3D transforms: `translateZ()`, `rotateX()`, `rotateY()`, `perspective`. Transforms don't affect document flow (unlike margin changes). Animatable with transitions/animations.

16. What are CSS filters?

Filters apply visual effects to elements. **`blur()`**: Blur effect. **`brightness()`**: Increase/decrease brightness. **`contrast()`**: Change contrast. **`grayscale()`**: Convert to grayscale. **`hue-rotate()`**: Rotate hue. **`invert()`**: Invert colors. **`opacity()`**: Change transparency. **`saturate()`**: Change color intensity. **`sepia()`**: Sepia tone effect. Can combine multiple filters. Often used with images.

17. What are CSS gradients?

Gradients create smooth color transitions. **Linear gradient**: Colors transition in a line. **Radial gradient**: Colors radiate from center. **Conic gradient**: Colors rotate around center. Syntax: `linear-gradient(direction, color1, color2)`. **Color stops**: Specify exact positions. **Angle** for linear gradient direction. Can use **multiple colors** and **complex stops**. Very useful for **modern UI designs**.

18. What are CSS shadows?

Box shadows add depth and dimension. Syntax: `box-shadow: offset-x offset-y blur radius color`. **Text shadows**: `text-shadow` with similar syntax. **Offset**: How far from element. **Blur**: How soft the shadow. **Spread radius**: Size of shadow. **Color**: Shadow color. **Multiple shadows**: Comma-separated values. Useful for **depth perception** and **emphasis**. Inset shadows: `inset` keyword.

19. What is CSS overflow?

Overflow handles content that exceeds container. **`visible`** (default): Content overflows. **`hidden`**: Hides excess content. **`scroll`**: Always shows scrollbars. **`auto`**: Scrollbars only if needed. **`overflow-x`** and **`overflow-y`**: Control horizontal/vertical separately. Used with **fixed height/width** containers. Essential for **scrollable areas** and **scroll handling**.

20. Flexbox vs Grid comparison

Feature	Flexbox	Grid
Dimensionality	1D (row or column)	2D (rows and columns)
Use Case	Navigation, items in line	Page layouts, complex designs
Content-based	Yes (content size matters)	Layout-based (structure first)
Browser Support	Excellent	Excellent (modern)
Learning Curve	Easier	Slightly harder
Alignment	Main and cross axis	Row and column axes

Feature	Flexbox	Grid
Nesting	Works well	Also works well

SCSS / SASS ANSWERS

21. What is SCSS/Sass?

Sass (Syntactically Awesome Stylesheets) is a **CSS preprocessor**. Makes CSS more powerful with **variables**, **nesting**, **mixins**, **functions**. **SCSS** (Sassy CSS) is the modern syntax (CSS-compatible). **Sass** is the older indented syntax. Preprocessor **compiles to CSS** before browser receives it. **Production uses CSS** (browser can't execute Sass directly). Improves **maintainability** and **DRY** principles.

22. What are SCSS variables?

Variables store reusable values. Syntax: `$variable-name: value`. Used like `$primary-color: #3498db`. Scoped to **nesting level** and below. Can store **colors**, **sizes**, **fonts**, **anything**. Use throughout stylesheet. **DRY principle**: Change value once, updates everywhere. Variables improve **consistency** and **maintainability**. More powerful than CSS variables (compile-time vs runtime).

23. What is SCSS nesting?

Nesting allows writing selectors **hierarchically**. Selector inside another selector inherits parent. Use `&` to reference parent selector. Reduces repetition of parent selector. Makes code **more readable** and **organized**. Limited nesting (2-3 levels) improves readability. Parent reference `&` creates **pseudo-class selectors** (`:hover`, `:active`). Prevents selector duplication.

24. What are SCSS mixins?

Mixins are **reusable blocks** of styles. Defined with `@mixin name { styles }`. Used with `@include name`. Can accept **parameters** for customization. **No output** until included. More **flexible** than copy-paste. Common uses: **vendor prefixes**, **media queries**, **animations**. DRY alternative to utility classes. Can be **nested**, **parameterized**, **returned** from functions.

25. What are SCSS functions?

Functions return values used in styles. Syntax: `@function name($args) { @return value }`. Return **colors**, **calculations**, **strings**. Often use **math operations** and **built-in functions**. Called with `name($args)`. **Built-in functions**: `lighten()`, `darken()`, `saturate()`, `adjust-hue()`, etc. Enable **complex style calculations** and **dynamic values**.

26. What are control directives in SCSS?

Control directives add **logic** to SCSS. **`@if`**: Conditional styles. **`@else if`**: Additional conditions. **`@else`**: Default case. **`@for`**: Loop through values. **`@each`**: Iterate over lists/maps. **`@while`**: While loop. Enable **programmatic** style generation. Often used with **mixins** for **flexible** outputs. Compile to static CSS (no runtime logic).

27. What is `@extend` in SCSS?

Extend shares styles between selectors. Syntax: `@extend %placeholder` or `.class`. Second selector **inherits** all styles from extended selector. Creates **selector chains**. More efficient than mixins for **style inheritance**. Use **placeholder selectors** (`%`) for extend-only rules. Reduces code duplication. Can create **complex selector chains** (be careful with overuse).

28. What are SCSS imports?

Imports include other SCSS files. Syntax: `@import 'path/file'`. Imports processed by preprocessor (not browser). Can import **multiple files** in single stylesheet. **Partials** (files starting with _) are meant for importing. Organizes code into **modules**. Can import **variables**, **mixins**, **styles** from other files. Improves **code organization** and **reusability**.

29. What are SCSS partials?

Partials are SCSS files meant for importing. Named with **_** prefix (e.g., `_variables.scss`). Not compiled to standalone CSS. Imported into main stylesheets. Organize code by **concern**: variables, mixins, layouts, components. Better **code organization**. Prevents creation of **unnecessary CSS files**. Naming convention helps identify importable files.

30. SCSS best practices

- Use **meaningful variable names** for clarity
- Keep **nesting level low** (max 3) for readability
- Use **mixins** for repeated style blocks
- Use **functions** for dynamic calculations
- Organize files into **logical partials**
- Comment **complex logic** and calculations
- Use **extend** for style inheritance, **mixins** for variation
- Keep **specificity low** and manageable
- Use **vendor prefixes** mixins for browser support
- Follow **consistent naming conventions**

TAILWIND CSS ANSWERS

31. What is Tailwind CSS?

Tailwind CSS is a **utility-first CSS framework**. Provides **pre-built utility classes** for rapid development. **No pre-styled components** (like Bootstrap). Flexible and highly **customizable**. Uses **PurgeCSS** to remove unused styles. Written in **PostCSS** and **JavaScript**. Results in **smaller CSS** compared to component frameworks. Modern approach to CSS workflow. Improves **development speed** and **design consistency**.

32. What are utility classes?

Utility classes are **small, single-purpose** CSS classes. **`text-center`**: Centers text. **`bg-blue-500`**: Blue background. **`m-4`**: Margin spacing. **`flex`**: Display flex. Each class does **one thing well**. Compose utilities to build **complex layouts**. Opposite of semantic classes. Reduces custom CSS writing. **Highly reusable** across projects.

33. What are responsive breakpoints in Tailwind?

Responsive prefixes apply styles at specific breakpoints. **`sm:`** (640px), **`md:`** (768px), **`lg:`** (1024px), **`xl:`** (1280px), **`2xl:`** (1536px). Mobile-first approach: base styles apply to all sizes. Use prefix for larger screens. Example: `md:text-lg` (large text on medium+). Easy **responsive design** without media queries. Can customize breakpoints in config.

34. What is responsive design in Tailwind?

Tailwind enables **responsive design** using **responsive prefixes**. Build **mobile-first**: Start with mobile, enhance for larger screens. Use **`sm:`, `md:`, `lg:`, etc.** prefixes. **No custom media queries** needed. Easily adjust **layouts**, **typography**, **spacing** per breakpoint. Simplifies **responsive workflows**. Same utility available at different breakpoints.

35. What are custom components in Tailwind?

Custom components group utilities into reusable classes. Defined in CSS with `@apply` directive. Example: `.btn { @apply px-4 py-2 bg-blue-500 text-white rounded; }`. Reduces **HTML duplication** for repeated patterns. Useful for **buttons**, **cards**, **forms**, etc. Define once, use multiple times. Balances **utility-first** and **component-based** approaches.

36. What are Tailwind plugins?

Plugins extend Tailwind's functionality. Add **custom utilities**, **components**, **variants**. Register in `tailwind.config.js`. Popular plugins: **Forms**, **Typography**, **Container Queries**, **DaisyUI**. Enable **advanced features** and **pre-built components**. Extend Tailwind without modifying core. Community plugins expand capabilities. Allows **consistent patterns** across projects.

37. Customizing colors in Tailwind

Tailwind provides **default color palette**. Customize in `tailwind.config.js` under `theme.colors`. Override or extend colors. Define **custom colors**: `'primary': '#3498db'`. Use in classes: `bg-primary`, `text-primary`. Supports **color opacity**: `bg-blue-500/50`. Complete control over **brand colors**. Generate consistent color schemes.

38. What is dark mode in Tailwind?

Dark mode automatically creates dark theme. Use `dark:` prefix for dark mode styles. Example: `bg-white dark:bg-gray-900`. Enable in config: `darkMode: 'class'` or `'media'`. **`class`** mode: Manual toggle with class. **`media`** mode: Follows system preference. **CSS custom properties** for efficiency. Provides **professional dark experience**. Modern UI standard.

39. Tailwind vs traditional CSS

Feature	Tailwind	Traditional CSS
Approach	Utility-first	Semantic
File Size	Small (PurgeCSS)	Larger
Learning Curve	Moderate	Easier
Development Speed	Very fast	Slower
Customization	Highly customizable	Limited
Components	Compose utilities	Pre-built
Consistency	Built-in	Manual
IDE Support	Excellent	Standard

40. Tailwind best practices

- Keep **naming conventions** consistent across team
- Use **config file** for customization (colors, spacing, fonts)
- Extract **reusable components** with `@apply`
- Keep **custom CSS minimal** (embrace utilities)
- Use **responsive prefixes** properly (mobile-first)

- Leverage **PurgeCSS** to optimize bundle size
- Organize **Tailwind files** logically
- Document **custom utilities** and components
- Use **browser extensions** for class reference
- Avoid **over-customization** that defeats purpose

END OF CSS / SCSS / TAILWIND INTERVIEW GUIDE