■ Interview Questions & Answers

■ HTML Structure & Semantics

Q: Why start with ``?
A: It tells the browser to render in standards-compliant mode (HTML5), avoiding quirks mode and ensuring consistent interpretation across browsers.

Q: Purpose of ``
A: Declares the document language for accessibility tools, search engines, and proper hyphenation or pronunciation by screen readers.

Q: Roles of `` and ``
A: `` holds metadata, links, and scripts; `` contains content your users actually see and interact with.

Q: What does `` do?
A: Sets the document character encoding to UTF-8, supporting virtually all characters (emojis, international text) correctly.

Q: Why ``?
A: Instructs mobile browsers to control layout width and scaling, making designs responsive out of the box.

Q: Semantic alternatives to ``
A: Use ``, ``, ``, ``, or `` for meaningful regions—boosts accessibility and SEO.

Q: Adding ARIA attributes
A: If these boxes represent UI regions or widgets, add `role="region"` and `aria-label="Description"` to improve screen-reader context.

■ CSS Fundamentals

Q: Selector specificity
A: Inline styles > IDs > classes/attributes > elements. Combining selectors (e.g., `.parentEnclosureDiv .outerBox`) increases specificity.

Q: Units (`px`, `%`, `vh`, `dvh`)
A: `px` is absolute; `%` is relative to parent; `vh` is 1% of viewport height (including browser chrome); `dvh` is 1% of the visible viewport, adjusting for dynamic browser UI.

Q: Percentage sizing calculation
A: A child’s `%` width/height is based on its containing block’s content box. Using `dvh` avoids jumps when mobile address bars show/hide.

Q: Box Model & `border-box`
A: By default, borders/padding add outside the content size. Using `box-sizing: border-box;` includes them inside the width/height, simplifying layout math.

■ Layout with Flexbox

Q: `display: flex` basics
A: Turns a container into a flexbox, enabling straightforward alignment and distribution without hacks like absolute positioning.

Q: `align-items` vs. `justify-content`
A: `align-items: center` centers items along the cross-axis (vertical); `justify-content: center` centers along the main-axis (horizontal).

Q: Default flex properties
A: `flex-grow: 0` (no expansion), `flex-shrink: 1` (shrinks if needed), `flex-basis: auto` (initial size). Prevent shrinking with `flex-shrink: 0`.

Q: Old centering methods vs. Flexbox
A: Older tricks (absolute + transform, table/table-cell) were clunky; Flexbox is more semantic, predictable, and robust.

■ Viewport Units & Responsiveness

Q: `100vh` vs `100dvh`
A: `100vh` includes browser chrome's show/hide, causing layout jumps; `100dvh` matches the visible area, smoothing mobile viewport changes.

Q: Rotation & responsiveness
A: Percentages and viewport units recalculate on orientation change, keeping nested boxes proportional in portrait and landscape.

Q: Browser support & fallbacks
A: For browsers without `dvh`, fallback to `100vh` or update a CSS variable via JS on `resize` events.

■■ Best Practices & Optimization

Q: DRY up repeated styles
A: Create a utility class: `.centered-flex { display:flex; align-items:center; justify-content:center; }` or use a Sass/LESS mixin.

Q: Naming conventions (BEM)
A: Example: `.parent-enclosure__outer-box { }` and `.parent-enclosure__inner-box--third { }` clarify hierarchy and purpose.

Q: Nesting & performance
A: Deep nesting can impact paint times; flatten structure or simplify containers when possible.

Q: Preprocessors & utility libraries
A: Sass mixins or Tailwind utility classes keep code DRY, maintainable, and consistent.

■ Cross-Browser & Future-Proofing

Q: Testing legacy browsers
A: Use BrowserStack or VMs to verify flex and viewport-unit support; polyfill or provide fallbacks as needed.

Q: Progressive enhancement
A: Let modern browsers use `dvh`, and serve fallback layouts (e.g., `vh` or JS-driven CSS) for older ones without breaking.

■ Additional Angles

Q: Flexbox vs CSS Grid
A: Use Grid for two-dimensional layouts (rows + columns), and Flexbox for one-dimensional alignment tasks.

Q: JavaScript dynamic styling
A: Listen to `window.resize`, measure `window.innerHeight`, and assign a CSS variable (e.g., `--vh`) to adapt styles.

Q: Visual regression testing
A: Integrate tools like Percy or BackstopJS to snapshot and compare rendered components across changes.

Q: Accessibility deep dive
A: Use `tabindex` thoughtfully and maintain DOM order matching visual flow to keep keyboard navigation intuitive.