■ Positioning Interview Q&A;

■ Positioning Fundamentals

Q: Why set `position: relative` on the parent container?
A: It establishes the parent as the containing block for absolutely positioned child elements, allowing `top/left` offsets to be calculated relative to it.

Q: What does `position: absolute` do?
A: It removes the element from the normal flow and positions it according to its nearest positioned ancestor based on `top`, `left`, `right`, and `bottom` offsets.

Q: How does `transform: translate(-50%, -50%)` help centering?
A: It shifts the element by half of its own width and height along the X and Y axes, perfectly centering it when paired with `top: 50%; left: 50%;`.

Q: Difference between `static`, `relative`, `absolute`, and `fixed`?
A: `static` is default flow. `relative` offsets relative to itself without removing from flow. `absolute` is removed from flow and positioned relative to ancestor. `fixed` is positioned relative to the viewport.

■ Width, Height & Units

Q: How do percentage-based widths/heights calculate?
A: Percentages on absolutely positioned elements refer to the dimensions of the containing block (the nearest positioned ancestor).

Q: Why use `100vh` on the parent?
A: It makes the parent container's height match the viewport height, ensuring child centering spans full screen.

■ Nesting & Containment

Q: How do nested absolute elements position themselves?
A: Each child is positioned relative to its nearest ancestor with a `position` other than `static`, creating a chain of reference points.

Q: What happens if parent lacks `position: relative`?
A: The child's absolute positioning falls back to the next ancestor or ultimately the viewport, potentially misplacing the element.

■ Practical & Best Practices

Q: When to choose absolute positioning over Flex/Grid centering?
A: Use it for precise overlayed or layered layouts; prefer Flex/Grid for responsive, one-axis or two-axis centering with less manual math.

Q: Performance considerations?
A: Heavy use of absolute can complicate reflows; deep nesting increases layout calculation cost—keep structure as flat as possible.

■ Bonus Topics

Q: Stacking context & `z-index`?
   A: Absolutely positioned elements can create new stacking contexts; `z-index` controls overlap order within that context.

Q: Responsive adjustments without JS?
   A: Combine CSS media queries with percentage units or CSS custom properties to adapt offsets and sizes at breakpoints.