

HTML & CSS Interview Questions and Answers

Intern to senior level

HTML and CSS interview questions and answers from beginner to senior: semantic HTML, the box model, Flexbox vs Grid, positioning, specificity, rem vs em, responsive design, the critical rendering path and Core Web Vitals.

12 questions Read online: <https://www.freedevtemplates.com/interview-questions/html-css>

1. What are semantic HTML elements and why do they matter?

Level: intern HTML & CSS

Semantic elements describe their meaning in the name, such as header, nav, main, article and footer, so browsers, assistive technology and search engines can understand the page structure.

A <div> or says nothing about the content. <header>, <nav>, <main>, <article>, <section>, <aside> and <footer> tell the browser what that region is for. Screen readers use them as landmarks, and search engines use them as a hint about what is primary content.

Use one <main> per page, put the site chrome in <header> and <footer>, and wrap a self-contained story or card in <article>. Keep using <div> when you only need a styling hook. Pair landmarks with a clear heading order (h1 then h2) rather than skipping levels.

Key points

Semantic tags describe meaning, not just layout

They help accessibility, SEO and maintainability

div and span are still correct when there is no meaning to express

2. What is the CSS box model?

Level: intern HTML & CSS

Every element is a box made of content, padding, border and margin. The default content-box size counts only the content; border-box includes padding and border in the width you set.

From the inside out: content (the text or image), padding (space inside the border), border, then margin (space outside the border). Adjacent vertical margins collapse into the larger of the two, which surprises people building stacked cards.

With box-sizing: content-box (the default), width: 200px is the content only, so padding and border make the element wider. box-sizing: border-box makes width include padding and border, which is why most teams set *, ::before, ::after { box-sizing: border-box; }.

```
*, ::before, ::after { box-sizing: border-box; }

.card {
width: 320px; /* includes padding and border */
padding: 16px;
border: 1px solid #ddd;
margin-bottom: 16px;
}
```

3. What is the difference between display none and visibility hidden in CSS?

Level: intern HTML & CSS

display: none removes the element from the layout so it takes no space; **visibility: hidden** hides it but leaves the empty space, and the element can still be read in the accessibility tree in some cases.

display: none is as if the element was not in the document for layout: later siblings move up. **visibility: hidden** keeps the box, so a 200px-tall banner still reserves 200px. **opacity: 0** is similar to hidden for layout, but the element can still receive pointer events unless you also set **pointer-events: none**.

For content that should not be announced to screen readers, prefer **display: none** or the HTML **hidden** attribute. To hide visually but keep it available to assistive technology (a skip link target, for example), use a visually-hidden class rather than **display: none**.

4. What is the difference between inline, block and inline-block in CSS?

Level: intern HTML & CSS

Block elements start on a new line and take the full available width; inline elements sit in the text flow and ignore width and height; inline-block sits in the flow but accepts width, height, padding and margin on all sides.

Headings, paragraphs and `div` are block by default. `span`, `a` and `strong` are inline: width, height and vertical margin do not apply the way you expect. **inline-block** is the middle ground used for buttons or icon-and-label pairs that should sit in a row but have a fixed size.

Today Flexbox or Grid usually replaces floats and long rows of **inline-block**, because **inline-block** still leaves gaps from the whitespace between tags. Use the **display** value that matches the layout job, not a habit.

5. What is the difference between relative, absolute, fixed and sticky positioning in CSS?

Level: junior HTML & CSS

Relative offsets an element from its normal place; absolute positions it against the nearest positioned ancestor; fixed is against the viewport; sticky sits in normal flow until a scroll threshold, then sticks.

position: relative with `top/left` moves the element but the original space is kept. **position: absolute** takes it out of flow; the containing block is the nearest ancestor with position other than static (or the initial containing block). That is why a modal overlay needs **position: relative** on its parent.

fixed stays put when the page scrolls, which is how sticky headers used to be built. **sticky** is a hybrid: the element scrolls with its container until `top: 0` (or similar) is reached, then it behaves like **fixed** within that container. **sticky** fails if any ancestor has **overflow: hidden**.

Key points

absolute is relative to the nearest positioned ancestor

fixed is relative to the viewport

sticky needs a threshold (`top/left`) and no **overflow-hidden** ancestors

6. What is the difference between Flexbox and CSS Grid?

Level: junior HTML & CSS

Flexbox is one-dimensional (a row or a column). CSS Grid is two-dimensional (rows and columns at once). Use Flexbox for alignment along a line; use Grid when the layout is a real 2D structure.

A nav bar, a cluster of buttons, or a card header with a title on the left and actions on the right is a Flexbox problem. A page with a sidebar, a main column and a footer, or a photo gallery with consistent tracks, is a Grid problem. They combine well: a Grid page whose cards use Flexbox internally.

Grid lets you name areas (`grid-template-areas`) and place items on explicit lines. Flexbox shines at distributing leftover space (`justify-content`, `flex-grow`) and wrapping. Neither replaces the other; interviews often ask you to pick one and say why.

```
.page {
display: grid;
grid-template-columns: 240px 1fr;
grid-template-rows: auto 1fr auto;
min-height: 100vh;
}

.toolbar {
display: flex;
justify-content: space-between;
align-items: center;
gap: 1rem;
}
```

7. How does CSS specificity work?

Level: junior HTML & CSS

Specificity is the score the browser uses to decide which rule wins: inline styles beat IDs, IDs beat classes/attributes/pseudo-classes, and those beat elements. When scores tie, the last rule in the source wins.

Count a tuple (inline, IDs, classes, elements). #nav .item is (0, 1, 1, 0). div.card p is (0, 0, 1, 2). !important jumps the queue and should be a last resort; too many of them mean the specificity war is already lost.

A more maintainable approach is low-specificity utilities or a method such as BEM, plus CSS layers (@layer) so you can control order without stacking IDs. Attribute selectors and pseudo-classes (.btn:hover) count as classes. The universal selector * and the :where() pseudo-class add zero specificity.

8. What is the difference between em, rem, %, vh and vw in CSS?

Level: junior HTML & CSS

em is relative to the parent font size, rem is relative to the root font size, % is relative to the parent's relevant size, and vh/vw are 1% of the viewport height or width.

rem is the default choice for type and spacing because it stays predictable when components nest. em compounds: a 1.2em padding inside a 1.2em heading is larger than you think. % on width is the parent's width; % on height only works if the parent has an explicit height.

100vh is the full viewport height, which on mobile can include the area hidden by the browser chrome and cause jumpiness. Newer units such as svh, lvh and dvh address that. For fluid type, clamp(1rem, 2vw, 2rem) is a common pattern.

9. What is the difference between cookies, localStorage and sessionStorage?

Level: junior HTML & CSS

Cookies are sent to the server with every matching request and can expire; localStorage persists until cleared and stays in the browser; sessionStorage lasts for the current tab only.

Use cookies for authentication when the server must see the value (prefer HttpOnly, Secure and SameSite). Use localStorage for non-sensitive UI preferences that should survive a restart. Use sessionStorage for a wizard or draft that should die when the tab closes.

Both storage APIs are readable by JavaScript, so they are the wrong place for access tokens if you have XSS. Cookies set without HttpOnly have the same problem. All three are origin-scoped. localStorage has a typical quota of about 5 MB; cookies should stay tiny because they travel on every request.

10. What is the difference between async and defer on script tags?

Level: mid HTML & CSS

Both download the script without blocking HTML parsing. defer waits until parsing finishes and runs scripts in document order; async runs a script as soon as it is downloaded, which can be before or after parsing and in any order.

A classic `<script src>` in the head blocks parsing until the file is downloaded and executed. defer is the right default for module-style application code that depends on the DOM and on other scripts. async is for independent tags such as analytics, where order does not matter.

`type="module"` scripts are deferred by default. async and defer have no effect on inline scripts. Put critical, tiny inline scripts only when you must paint something before the main bundle arrives.

11. How does the browser render a web page?

Level: mid HTML & CSS

The browser parses HTML into a DOM and CSS into a CSSOM, combines them into a render tree, computes layout (reflow), paints pixels and composites layers onto the screen.

This path is called the critical rendering path. Scripts without async/defer pause HTML parsing. CSS is render-blocking: the browser waits for stylesheets before first paint so it does not flash unstyled content. After the first paint, changing geometry (width, height, top) causes reflow; changing only color or transform can often skip to paint or composite.

Performance work targets each stage: reduce render-blocking CSS, defer non-critical JS, avoid layout thrashing (reading layout then writing styles in a loop), and keep animations on transform and opacity so they run on the compositor thread. Senior interviews often follow this with Core Web Vitals: LCP, INP and CLS.

Key points

DOM + CSSOM render tree layout paint composite

CSS blocks first paint; JS without async/defer blocks parsing

Animate transform and opacity to avoid layout

12. What are Core Web Vitals and how do you improve them?

Level: senior HTML & CSS

Core Web Vitals are Google's field metrics for real-user experience: LCP (loading), INP (interactivity) and CLS (visual stability). They influence search ranking and are measured on real devices, not only Lighthouse.

LCP (Largest Contentful Paint) should happen within about 2.5 seconds: compress the hero image, preload it, and cut render-blocking CSS/JS. INP (Interaction to Next Paint) replaced FID: keep event handlers short, break up long tasks, and show immediate feedback. CLS (Cumulative Layout Shift) should stay below 0.1: always set width and height (or aspect-ratio) on images and embeds, and avoid inserting banners above existing content.

Lab tools (Lighthouse, DevTools Performance) are for debugging. Rankings use field data from the Chrome User Experience Report. Fix the slowest template first, measure with web-vitals or Search Console, and do not chase a 100 Lighthouse score at the cost of features users need.

These answers are written for real interview conversations. Prefer explaining a trade-off out loud over reciting a definition. Full explanations and related questions: <https://www.fredevtemplates.com/interview-questions/html-css>