Frontend Mock Interview - HTML, CSS, JavaScript

Section 1: HTML Questions (Semantic, Accessibility, Forms)

1. Difference between `<article>` and `<section>`

- `<article>`: Represents **independent, self-contained content** like blog posts, news articles, or forum comments. It makes sense even when removed from the page.
- `<section>`: Represents a **thematic grouping** of content with a heading. It's like organizing content into chapters or logical blocks within a page.
- 🔍 **Example**: Use `<article>` for each blog post, and `<section>` for "About", "Services", or "Contact" within a page.

2. Accessible Forms — `label` and `aria-label`

- Use `<label for="id">` so screen readers can associate the label with the input.
- `aria-label="..."` is used when there's **no visible label** (e.g., icon-only search bars).
- **Why important**: Helps screen readers and improves usability for disabled users.
- ✅ Tip: Always use a `<label>` unless you absolutely need to hide it visually.

3. How HTML is parsed — DOM Tree

- The browser reads HTML from top to bottom, turning it into tokens and building a **Document Object Model (DOM)** — a tree structure of nodes.
- This tree represents the **rendered webpage** and allows JavaScript to read and manipulate it.
- Think of the DOM as a **live map of your page**.

4. `disabled` vs `readonly` vs `required`

- `disabled`: The input **cannot be clicked, focused, or submitted**. Often grayed out.
- `readonly`: The input is **visible and can be selected** but not edited.
- `required`: The field must be filled before the form can be submitted.
- □ Use `disabled` to lock out a field entirely. Use `readonly` when you want users to copy or see data but not change it.

5. Custom Checkbox/Radio

- Hide the native input (`display: none`) and **style the label or span**.
- Use `:checked` in CSS to change appearance when selected.
- 🔧 Example:

```
<input type="checkbox" id="cb1" hidden>
<label for="cb1" class="custom-checkbox"></label>
```

- Then style `.custom-checkbox` and `.custom-checkbox:checked` with CSS.

6. When to use `<template>`

- `<template>` holds HTML that isn't rendered immediately. You can **clone and render it via JavaScript**.
- Useful when building components like modals, cards, dynamic lists.

Section 2: CSS Questions (Layout, Responsive Design, Styling)

1. Box Model & Margin Collapsing

- Box model: `content → padding → border → margin`
- Vertical margins of two adjacent elements may **collapse into one**, taking the larger value.
- 💡 To prevent collapsing: add padding, borders, or overflow on the parent.

2. z-index & stacking context

- `z-index` only works on **positioned elements** (`relative`, `absolute`, etc.)
- A **stacking context** is like a mini layer system — `z-index` only affects elements **within the same context**.
- Debug with DevTools: hover over elements and check stacking context.

3. Units: `rem`, `em`, `%`, `px`

- `px`: Absolute pixel value.
- `em`: Relative to parent font-size.
- `rem`: Relative to **root (`html`) font-size**.
- `%`: Relative to parent size.
- ✅ Prefer `rem` for font sizes (consistency), `%` for layouts, and avoid `px` in responsive UIs.

4. Responsive 3-column layout

- **Grid**:

```
display: grid;
grid-template-columns: repeat(3, 1fr);
```

- **Flexbox**:

```
display: flex;
flex-wrap: wrap;
.child { width: 33.33%; }
```

- Grid is more declarative and powerful for complex layouts.

5. Card with hover animation

- Reveal content using CSS transitions:

```
.card .extra { opacity: 0; transition: opacity 0.3s; }
.card:hover .extra { opacity: 1; }
```

- Can animate max-height, transform, or opacity.

6. Pseudo-elements vs Pseudo-classes

- `::before`, `::after`: insert content via CSS.
- `:hover`, `:focus`, `:nth-child(2)`: apply style based on interaction or position.

7. CSS Performance Tips

- Use class selectors instead of deep nesting.
- Minimize use of `*`, tag selectors, and complex rules.
- Avoid large repaints or layout thrashing.

8. Specificity

- `#id` = 100
- `.class` = 10
- `div` = 1
- Inline style = 1000
- Combine them: `#id .class div` = 111

9. CSS Variables (`--vars`)

- Declare once, reuse anywhere:

```
:root { --main-color: #333; }
button { color: var(--main-color); }
```

- Useful for themes, dark mode, spacing consistency.

10. Loading Skeleton

- Placeholder boxes with shimmer:

```
.skeleton {
  background: linear-gradient(...);
  animation: shimmer 2s infinite;
}
```

- Good for improving **perceived performance**.

Section 3: JavaScript (Core + DOM + Async)

1. `var` vs `let` vs `const`

- `var`: function-scoped, hoisted.
- `let`: block-scoped, reassignable.
- `const`: block-scoped, **not reassignable** (but objects are still mutable).

2. `this` in different contexts

- Global: `window` (in browsers)
- Object method: the object
- Arrow function: inherits `this` from surrounding scope

3. Event Loop

- JS runs synchronously, but async functions are pushed to a queue.
- **Microtasks** (e.g., `Promise.then`) run before **macrotasks** (`setTimeout`).
- Key concept: **JS is single-threaded but non-blocking.**

4. Promises vs Callbacks

- Promises: cleaner syntax, `then()/catch()`, avoid nesting.
- Example:

```
fetch("/api").then(...).catch(...);
```

5. Difference: `null`, `undefined`, `NaN`, `0`

- `undefined`: not assigned
- `null`: explicitly empty
- `NaN`: Not a Number
- `0`: a valid number (falsy)

6. Deep Clone

- Use recursion or `structuredClone(obj)`
- Avoid `JSON.parse(JSON.stringify(obj))` if you have functions or dates.

7. `map`, `filter`, `reduce`

- `map`: transform array
- `filter`: select elements
- `reduce`: combine to a single value

8. Synchronous vs Asynchronous

- Sync: blocks execution
- Async: uses events/callbacks/promises to defer execution

9. Debounce

- Waits for the user to stop typing before running the function.
- Avoids excessive calls (e.g., on search input).

```
function debounce(fn, delay) { ... }
```

10. Storage Types

- `localStorage`: persists until cleared
- `sessionStorage`: cleared on tab close
- `cookies`: small, sent to server

Section 4: JavaScript DOM Manipulation

1. Change Button Text on Click

```
document.getElementById("btn").addEventListener("click", function() {
  this.innerText = "Clicked!";
});
```

- `this` refers to the button clicked.
- `innerText` changes the visible label.
- Good for toggles, feedback UI, etc.

2. Show/Hide Element (Toggle Visibility)

```
const el = document.getElementById("box");
el.style.display = el.style.display === "none" ? "block" : "none";
```

- Use `display` or `visibility`.
- For smoother UX, prefer adding/removing classes with transitions.

3. Create Element Dynamically

```
const p = document.createElement("p");
p.textContent = "Hello!";
document.body.appendChild(p);
```

- Useful for alerts, chat messages, lists, etc.
- Always set attributes or text **before** appending.

4. Event Delegation

```
document.getElementById("list").addEventListener("click", function(e) {
  if (e.target.tagName === "LI") {
    console.log("Clicked:", e.target.textContent);
  }
});
```

- Avoids attaching listeners to each item.
- Great for dynamic lists (e.g., todo apps, search suggestions).

5. Form Validation Example

```
document.querySelector("form").addEventListener("submit", function(e) {
  const name = document.getElementById("name").value;
  if (name.trim() === "") {
    alert("Name is required!");
    e.preventDefault(); // Stops form submission
  }
});
```

- Frontend validation = UX improvement
- Always combine with **server-side validation** for security.

6. ClassList Usage

```
element.classList.add("active");
element.classList.remove("hidden");
element.classList.toggle("dark-mode");
```

- `classList` makes it easy to manage styling programmatically.
- Widely used for toggling themes, menus, tabs, etc.

7. Debounced Search

```
let timeout;
input.addEventListener("input", function() {
  clearTimeout(timeout);
  timeout = setTimeout(() => {
    fetchResults(this.value);
  }, 300);
});
```

- Avoids API calls on every keystroke.
- **Good UX** and **reduces load on backend**.

Section 5: Real-World Frontend Scenarios

1. Build a Modal

- HTML: hidden modal with close button.
- JS: toggle class on button click.
- CSS: fade in/out or slide animations.

Key Parts:

```
btn.onclick = () => modal.classList.add("open");
close.onclick = () => modal.classList.remove("open");
```

2. Dark Mode Toggle

```
toggleBtn.onclick = () => {
  document.body.classList.toggle("dark");
  localStorage.setItem("theme", document.body.classList.contains("dark") ? "dark" :
"light");
};
```

- Save preference using `localStorage`
- Add `dark` class to `<body>` or root element.

3. Responsive Navbar Toggle

```
burger.addEventListener("click", () => {
  nav.classList.toggle("visible");
});
```

- Use CSS `@media` queries to handle mobile/desktop layout.
- Combine with transitions for smoother UX.

4. Real-time Character Counter

```
input.addEventListener("input", () => {
  count.innerText = `${input.value.length}/100`;
});
```

- Shows live feedback for textareas or inputs (bio, feedback, etc.)
- Add limit warning: `input.maxLength = 100;`

5. Infinite Scroll Implementation (basic)

```
window.addEventListener("scroll", () => {
  if (window.innerHeight + window.scrollY >= document.body.offsetHeight) {
    loadMoreItems();
  }
});
```

- Used in social feeds, e-commerce catalogs.
- Combine with loading spinners and throttling for performance.

6. Drag and Drop

- Use `dragstart`, `dragover`, and `drop` events.
- Add `dataTransfer` for passing data.

Basic Idea:

```
item.addEventListener("dragstart", e => {
  e.dataTransfer.setData("text", e.target.id);
});
dropZone.addEventListener("drop", e => {
  const id = e.dataTransfer.getData("text");
  dropZone.appendChild(document.getElementById(id));
});
```

7. Tabs UI Component

```
tabBtns.forEach(btn => {
  btn.addEventListener("click", () => {
    hideAllTabs();
    showTab(btn.dataset.target);
  });
});
```

- Each button toggles visibility of its related tab.
- Add `aria-selected`, `aria-hidden` for accessibility.

8. Client-Side Form Validation with Regex

```
emailInput.addEventListener("blur", () => {
  const valid = /^[^@\s]+@[^@\s]+\.[^@\s]+$/.test(emailInput.value);
  emailInput.classList.toggle("invalid", !valid);
});
```

- Validate emails, phone numbers, passwords without server request.

9. Page Load Spinner

- CSS loader (spinner or pulse)
- JS removes it once `window.onload` fires:

```
window.onload = () => document.querySelector(".loader").style.display = "none";
```

10. Lazy Load Images

```
<img data-src="image.jpg" class="lazy" />
const imgs = document.querySelectorAll("img.lazy");
const observer = new IntersectionObserver(...);
```

- Loads images only when they enter the viewport.
- Boosts performance for image-heavy sites.

.