**Top 50 Web Development Interview Questions & Answers**

**1. What is the difference between id and class in HTML?**

* Both are attributes used for identifying and styling elements, but their usage is different.

| **Feature** | **id** | **class** |
| --- | --- | --- |
| Uniqueness | Must be unique on a page | Can be reused on multiple elements |
| Selector | #idname in CSS | .classname in CSS |
| Usage | Identify a single element | Group multiple elements for styling |
| Specificity | Higher specificity than class | Lower than id |

**2. How do HTML5 semantic elements improve SEO and accessibility?**

* **Semantic elements** (<header>, <nav>, <article>, <footer>) describe meaning, not just presentation.
* **SEO**: Search engines understand page structure → better indexing.
* **Accessibility**: Screen readers identify page sections → improves UX for visually impaired.

**3. What is the box model in CSS?**

* Every HTML element is a rectangular box with:

| **Part** | **Description** |
| --- | --- |
| **Content** | Actual text, image, or element content |
| **Padding** | Space between content & border |
| **Border** | Surrounds padding + content |
| **Margin** | Space between element and others |

**4. How do you center a div using Flexbox?**

.container {

display: flex;

justify-content: center; /\* horizontal \*/

align-items: center; /\* vertical \*/

}

This makes the child <div> perfectly centered.

**5. Difference between relative, absolute, and fixed positioning?**

| **Position** | **Relative To** | **Moves With Scroll?** | **Example Use** |
| --- | --- | --- | --- |
| **Relative** | Its normal position | Yes | Minor offset adjustments |
| **Absolute** | Nearest positioned ancestor | Yes | Tooltips, dropdowns |
| **Fixed** | Viewport (screen) | No | Sticky nav bars |

**6. What is specificity in CSS and how is it calculated?**

* Specificity determines which CSS rule applies when multiple rules target the same element.

**Calculation:**

* Inline styles = **1000**
* ID selector = **100**
* Class, attribute, pseudo-class = **10**
* Element, pseudo-element = **1**

Example:

div p span { } /\* 3 \*/

.content p { } /\* 11 \*/

#main { } /\* 100 \*/

**7. What are media queries? Give an example.**

* Allow different styles for different screen sizes.

@media (max-width: 768px) {

body { font-size: 14px; }

}

Used for **responsive design**.

**8. Difference between em, rem, px, and % units?**

| **Unit** | **Based On** | **Example** | **Use Case** |
| --- | --- | --- | --- |
| **px** | Fixed pixel value | 20px | Precise control |
| **em** | Parent’s font size | 2em → 2 × parent font-size | Nested scaling |
| **rem** | Root (html) font size | 2rem → 2 × root size | Consistent scaling |
| **%** | Parent element | 50% | Flexible layouts |

**9. How does z-index work in CSS?**

* Controls stacking order of elements.
* Higher value → appears on top.
* Works only on elements with position (not static).

**10. What are pseudo-elements and pseudo-classes in CSS?**

| **Type** | **Example** | **Purpose** |
| --- | --- | --- |
| **Pseudo-class** | :hover, :focus, :nth-child(2) | Apply styles in a state |
| **Pseudo-element** | ::before, ::after, ::first-letter | Style part of element |

**11. What is hoisting in JavaScript?**

* **Definition**: JavaScript moves variable and function declarations to the top of their scope before execution.
* But only **declarations** are hoisted, not **initializations**.

**Example:**

console.log(a); // undefined (hoisted but not initialized)

var a = 10;

foo(); // Works (function hoisted)

function foo() { console.log("Hello"); }

**12. Explain the difference between == and ===.**

| **Operator** | **Compares** | **Type Conversion?** | **Example** |
| --- | --- | --- | --- |
| **==** | Value | Yes (loose) | 5 == "5" → true |
| **===** | Value + Type | No (strict) | 5 === "5" → false |

Always prefer === for reliability.

**13. What is a closure? Provide an example.**

* A **closure** is when an inner function “remembers” variables from its outer function even after the outer function has finished.

**Example:**

function outer() {

let count = 0;

return function inner() {

count++;

return count;

};

}

const counter = outer();

console.log(counter()); // 1

console.log(counter()); // 2

Useful for **data privacy, currying, event handlers**.

**14. What are var, let, and const? Differences?**

| **Feature** | **var** | **let** | **const** |
| --- | --- | --- | --- |
| Scope | Function-scoped | Block-scoped | Block-scoped |
| Hoisting | Hoisted (initialized as undefined) | Hoisted (TDZ applies) | Hoisted (TDZ applies) |
| Re-declare | Allowed | Not allowed | Not allowed |
| Re-assign | Allowed | Allowed | Not allowed |

**15. What is the event loop in JavaScript?**

* JS is **single-threaded** but handles async tasks using the **event loop**.
* Flow:
  1. Call Stack executes code.
  2. Async tasks go to **Web APIs**.
  3. Results pushed to **Callback Queue / Microtask Queue**.
  4. Event loop pushes them back to Call Stack when empty.

Enables **non-blocking I/O** in Node.js and browsers.

**16. Explain how this keyword behaves in different contexts.**

| **Context** | **Value of this** |
| --- | --- |
| Global scope (browser) | window object |
| Inside function (strict) | undefined |
| Inside function (non-strict) | window |
| Object method | Refers to that object |
| Arrow function | Lexical scope (parent this) |
| In class constructor | Refers to the created instance |

**17. What is the difference between synchronous and asynchronous code?**

| **Type** | **Execution** | **Example** |
| --- | --- | --- |
| **Synchronous** | Code runs line by line, blocks execution | console.log("A"); console.log("B"); |
| **Asynchronous** | Non-blocking, handled via callbacks, promises, async/await | setTimeout(() => console.log("B"), 1000); console.log("A"); |

Async is crucial for **API calls, file handling, DB queries**.

**18. What are promises and async/await?**

* **Promise**: Represents a future value (pending → fulfilled/rejected).
* **async/await**: Syntactic sugar for working with promises in a synchronous-like style.

**Example:**

function getData() {

return new Promise((resolve) => {

setTimeout(() => resolve("Done!"), 1000);

});

}

async function showData() {

const result = await getData();

console.log(result);

}

showData(); // "Done!"

**19. What is debouncing and throttling?**

| **Concept** | **Definition** | **Use Case** |
| --- | --- | --- |
| **Debouncing** | Delays function execution until after a pause in events | Search box input |
| **Throttling** | Ensures function runs at most once in a given interval | Window resize, scroll events |

**20. What is the difference between null and undefined?**

| **Feature** | **null** | **undefined** |
| --- | --- | --- |
| Meaning | Explicitly "no value" | Variable declared but not assigned |
| Type | Object (legacy bug) | Undefined |
| Usage | Assigned by developer | Default by JS |
| Example | let x = null; | let y; console.log(y); // undefined |

**React (Q21–Q30)**

1. **What are React Hooks?**
   * Hooks are **functions** that let you “hook into” React state and lifecycle features in **functional components**.
   * Common hooks:
     + useState → manage state
     + useEffect → run side effects (fetching data, subscriptions)
     + useContext → access global state
2. **Controlled vs Uncontrolled Components**

| **Aspect** | **Controlled Component** | **Uncontrolled Component** |
| --- | --- | --- |
| State management | React state | DOM handles state |
| Accessing value | value prop + onChange handler | ref to read value |
| Example | <input value={state} onChange={...} /> | <input ref={inputRef} /> |
| Use case | When you need to validate inputs or react to changes | Quick form with minimal React logic |

1. **What is Virtual DOM? How React uses it?**
   * **Virtual DOM (VDOM)**: Lightweight **in-memory copy** of the real DOM.
   * React **diffs** old VDOM vs new VDOM → updates **only changed nodes** in real DOM.
   * Improves performance by minimizing expensive DOM operations.
2. **Props vs State**

| **Aspect** | **Props** | **State** |
| --- | --- | --- |
| Mutability | Immutable | Mutable |
| Source | Passed from parent | Managed within component |
| Purpose | Config data for child component | Component’s internal data |
| Example | <Button color="red" /> | const [count, setCount] = useState(0) |

1. **What is lifting state up?**
   * Moving state to the **nearest common ancestor** of components so multiple children can **share and sync state**.
   * Example: sharing input value between sibling components.
2. **What are keys and why important?**
   * Unique identifier for each list element.
   * Helps React **track elements** during re-render → **optimize diffing**.
   * Example: <li key={item.id}>{item.name}</li>
3. **Difference between useEffect and useLayoutEffect**

| **Hook** | **Timing of execution** | **Use case** |
| --- | --- | --- |
| useEffect | After DOM paint, async | Fetching API data, subscriptions |
| useLayoutEffect | Before DOM paint, sync | Measure DOM, apply layout changes before render |

8. **Reconciliation & Re-rendering**

* + React maintains **VDOM**, compares old vs new state → updates only changed nodes.
  + Keys in lists help React **avoid unnecessary re-renders**.

1. **Context API & usage**
   * Provides **global state** without prop drilling.
   * Example: theme, auth, language settings.
2. **Higher-Order Components (HOCs)**
   * A **function** that takes a component → returns a **new enhanced component**.
   * Example: withAuth(Component) adds authentication logic.

**Node.js & Express (Q31–Q40)**

1. **Node.js vs Browser JS**

| **Feature** | **Node.js** | **Browser JS** |
| --- | --- | --- |
| Runtime | Server-side | Client-side |
| APIs | File system, networking, OS | DOM, BOM |
| Module system | CommonJS, ES Modules | ES Modules |
| Use case | Backend servers, APIs | Interactivity on web pages |
|  |  |  |

1. **Middleware functions in Express.js**
   * Functions executed **in the request-response cycle**.
   * Example: logging, authentication, error handling:
   * app.use((req, res, next) => { console.log(req.url); next(); });
2. **Event-driven model in Node.js**
   * Uses **event loop** and **non-blocking I/O**.
   * Events trigger **callbacks** asynchronously → server can handle multiple requests concurrently.
3. **Benefits of Express.js**
   * Minimal and fast
   * Built-in **routing**
   * Supports **middleware**
   * Easily builds **REST APIs**
4. **Error handling in Node.js**
   * Synchronous → try/catch
   * Async → .catch() on promises
   * Express → error-handling middleware
   * app.use((err, req, res, next) => { res.status(500).send(err.message); });
5. **Role of package.json**
   * Stores metadata: project name, version, scripts, dependencies.
   * Essential for **npm install**, project management, and scripts.
6. **HTTP methods & use cases**

| **Method** | **Purpose** | **Example** |
| --- | --- | --- |
| GET | Fetch data | /users |
| POST | Create resource | /users |
| PUT | Replace resource | /users/1 |
| PATCH | Update part of resource | /users/1 |
| DELETE | Remove resource | /users/1 |

1. **Streams in Node.js**
   * Handle **data in chunks**, not all at once → efficient for large files.
   * Types: Readable, Writable, Duplex, Transform.
2. **require vs import**

| **Feature** | **require** | **import** |
| --- | --- | --- |
| Module | CommonJS | ES6 Modules |
| Execution | Synchronous | Asynchronous |
| Tree-shaking | No | Yes |

1. **Managing environment variables**
   * Use .env file + dotenv package

require('dotenv').config();

console.log(process.env.PORT);

**Databases & APIs (Q41–Q50)**

1. **SQL vs NoSQL**

| **Feature** | **SQL (Relational)** | **NoSQL (Non-relational)** |
| --- | --- | --- |
| Schema | Fixed/Structured | Flexible/Schema-less |
| Storage | Tables (rows & columns) | Documents, key-value, graph, column |
| Transactions | ACID compliant | BASE compliant |
| Examples | MySQL, PostgreSQL | MongoDB, Redis, Cassandra |

1. **CRUD operations**
   * **Create** → insert
   * **Read** → select
   * **Update** → modify
   * **Delete** → remove
2. **REST & principles**
   * **Representational State Transfer**
   * Principles: stateless, client-server, cacheable, uniform interface, layered system.
3. **PUT vs PATCH**

| **Method** | **Effect** |
| --- | --- |
| PUT | Replaces entire resource |
| PATCH | Updates partial resource |
|  |  |

1. **HTTP status codes**
   * 200 → OK
   * 404 → Not Found
   * 500 → Internal Server Error
2. **Securing REST APIs**
   * Use HTTPS
   * Authentication: JWT, OAuth
   * Input validation
   * Rate limiting
3. **CORS (Cross-Origin Resource Sharing)**
   * Restricts resources to be accessed from different origins
   * Handle via response headers:
   * res.setHeader('Access-Control-Allow-Origin', 'https://example.com');
4. **Indexing in databases**
   * Improves query performance by creating **B-tree or hash structure**
   * Example: CREATE INDEX idx\_name ON users(name);
5. **JWT (JSON Web Token)**
   * Authentication token:
     1. Server signs token → sends to client
     2. Client stores token → sends on each request
     3. Server verifies token for protected routes
6. **Structuring a RESTful API**
   * Use **HTTP methods** correctly
   * Meaningful **endpoints** (/users/:id)
   * **Stateless** design
   * Proper **error handling & versioning**