# 🔥 Full-Stack Interview Questions with Detailed Answers

---

## 🦜 1. HTML & CSS

**Q1: What are semantic tags?**\ Semantic tags are HTML5 elements that carry meaning about the content within them. For example, `<article>`, `<header>`, `<footer>`, `<nav>`, and `<section>` clearly define the role of the content. These tags improve accessibility and SEO, and help developers understand the structure of the webpage more intuitively.

**Q2: Difference between `** and **`?**

- `id`: Uniquely identifies a single element. Cannot be reused on the same page.
- `class`: Used for styling multiple elements with the same style. Can be reused multiple times.

Use `id` when a style or script will apply to only one element. Use `class` when the same styling is applied to multiple elements.

**Q3: What is the CSS box model?**\ The box model consists of:

- **Content**: The actual content (text/images)
- **Padding**: Space around content
- **Border**: Surrounds the padding
- **Margin**: Space outside the border

Understanding the box model is key to proper layout and spacing.

**Q4: Difference between `**, **`, `**, and **`?**

- `relative`: Positioned relative to its normal position.
- `absolute`: Positioned relative to the nearest ancestor with `position` other than `static`.
- `fixed`: Stays fixed relative to the browser window (useful for navbars).
- `sticky`: Switches between relative and fixed depending on scroll.

**Q5: What are media queries?**\ Media queries allow you to apply styles based on device characteristics (like width, height, resolution).

```
@media (max-width: 768px) {
  body { font-size: 14px; }
}
```

**Q6: CSS specificity?**\ It determines which style rule gets applied. Specificity hierarchy:

- Inline styles (highest)
- ID selectors

- Class selectors / attributes / pseudo-classes
- Element selectors (lowest)

**Q7: Flexbox vs Grid?**

- **Flexbox**: One-dimensional layout. Aligns items in rows/columns.
- **Grid**: Two-dimensional layout. Aligns items in rows *and* columns.

Use Flexbox for simpler layouts and Grid for more complex designs.

**Q8: Difference between `** and **`?**

- `visibility: hidden`: Hides the element, but it still occupies space in layout.
- `display: none`: Completely removes the element from the layout.

**Use Case**: Use `visibility: hidden` for temporary toggling when you want the layout preserved.

**Q9: Difference between `**, **`, and ``?**

- `inline`: Cannot set width/height; sits beside other elements.
- `block`: Takes full width; starts on a new line.
- `inline-block`: Like inline, but allows setting width/height.

**Q10: How to make a layout responsive?**\ Use fluid layouts with percentages or `em/rem`, use media queries, flex/grid layouts, and avoid fixed pixel sizes. Use mobile-first design practices.

---

## 🦜 2. JavaScript

**Q1: What is a closure?**\ A closure is a function that remembers the variables from its outer lexical scope, even after the outer function has returned.

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    return count;
  };
}
const counter = outer();
console.log(counter()); // 1
```

**Q2: Difference between `** and **`?**

- `==`: Compares values with type coercion.
- `===`: Compares values *and* types.

```
"5" == 5  // true
"5" === 5 // false
```

**Q3: What is hoisting?**\ JavaScript moves function and variable declarations to the top of their scope before code execution.

```
console.log(a); // undefined
var a = 5;
```

**Q4: What is the event loop?**\ It's a mechanism that lets Node.js and browsers handle non-blocking async operations. It pushes callback functions to the task queue and executes them after the call stack is clear.

**Q5: What is lexical scope?**\ Lexical scope means that scope is determined by the position of functions in the source code.

**Q6: Difference between `**, **` , and ``?**

- `var` : Function-scoped, can be hoisted.
- `let` : Block-scoped, not hoisted like var.
- `const` : Block-scoped, cannot be reassigned.

**Q7: What is a promise?**\ A Promise is an object representing the eventual completion or failure of an asynchronous operation.

```
new Promise((resolve, reject) => {
  // async logic
})
```

**Q8: Difference between `**, **` , and ``?**

- `call` : Invokes function with context and arguments.
- `apply` : Same as call but arguments are passed as array.
- `bind` : Returns a new function with context bound.

**Q9: Debounce vs Throttle?**

- **Debounce**: Executes only after a pause in activity (e.g., search bar).
- **Throttle**: Executes at intervals, no matter how often the event is triggered (e.g., scroll).

**Q10: Explain `**, **` , and ``**

- `map()` : Transforms each array element.
- `filter()` : Filters elements based on condition.
- `reduce()` : Reduces array to single value.

## 🦜 3. TypeScript

**Q1: Type vs Interface?**

- Use `interface` for object shapes that you expect to extend.
- Use `type` for union types, primitives, tuples, etc.

**Q2: What is `** vs **`?**

- `any`: Turns off type checking.
- `unknown`: Requires type checking before usage.

**Q3: What is `` type?**\ Used for functions that never return (throw errors, infinite loops).

```
function error(msg: string): never {
  throw new Error(msg);
}
```

**Q4: What are Generics?**\ Generics allow code to be reusable for any data type.

```
function identity<T>(arg: T): T {
  return arg;
}
```

**Q5: Utility Types?**

- `Partial<T>`
- `Required<T>`
- `Readonly<T>`
- `Pick<T, K>`
- `Omit<T, K>`

---

## 🦜 4. React

**Q1: Functional vs Class Component?**\ Functional components are simpler and use hooks. Class components use lifecycle methods.

**Q2: What are Hooks?**\ Functions like `useState`, `useEffect`, `useRef`, `useMemo` that allow you to use React features without classes.

**Q3: useEffect Usage?**\ Runs after render. Common for API calls.

```
useEffect(() => {
  fetchData();
}, []);
```

**Q4: Controlled vs Uncontrolled components?**

- Controlled: Value controlled via React state.
- Uncontrolled: Value managed by DOM.

**Q5: What are keys in React?**\ Used to identify list elements uniquely. Helps React optimize re-rendering.

**Q6: What is lifting state up?**\ Moving state to a common parent component to share between child components.

**Q7: What is Reconciliation?**\ The process by which React updates the DOM. It uses diffing algorithm on virtual DOM.

---

(### ✅5. Redux & RTK Query

**Q1: What is Redux?**\ Redux is a predictable state management library for JavaScript apps. It stores the state in a single global store and lets components access state via actions and reducers.

**Q2: Core concepts of Redux?**

- **Store**: Single source of truth
- **Actions**: Plain JS objects with a `type` property
- **Reducers**: Pure functions that update the state
- **Dispatch**: Sends an action to the store

**Q3: What is Redux Toolkit?** Redux Toolkit (RTK) is the official, opinionated way to write Redux logic. It simplifies store setup and eliminates boilerplate.

```
const slice = createSlice({
  name: 'counter',
  initialState: 0,
  reducers: {
    increment: state => state + 1,
  }
});
```

**Q4: What is RTK Query?**\ RTK Query is a powerful data-fetching and caching tool built into Redux Toolkit. It auto-generates endpoints and handles caching, updating, and invalidation.

```
const api = createApi({
  baseQuery: fetchBaseQuery({ baseUrl: '/api' }),
  endpoints: builder => ({
    getUsers: builder.query({ query: () => 'users' })
  })
});
```

**Q5: Difference between Redux and Context API?**

- **Context API** is good for simple state sharing.
- **Redux** is better for complex, large-scale applications with predictable state updates and middleware support.

---

## 🦜6. Node.js

**Q1: What is Node.js?**\ Node.js is a JavaScript runtime built on Chrome's V8 engine that allows JavaScript to be run server-side.

**Q2: Event Loop in Node.js?**\ It handles asynchronous operations using a single-threaded loop with queues (event queue, callback queue).

**Q3: What is middleware in Node.js/Express?**\ Middleware functions are functions that have access to the `req`, `res`, and `next` objects and are used for things like authentication, logging, and parsing.

```
app.use((req, res, next) => {
  console.log("Request Received");
  next();
});
```

**Q4: What are streams?**\ Streams are objects used to read or write data piece-by-piece. Types: Readable, Writable, Duplex, Transform.

**Q5: Common built-in modules in Node.js?**

- `fs`: File System
- `http`: HTTP server
- `path`: File paths
- `events`: EventEmitter

---

## 🦜 7. NestJS

**Q1: What is NestJS?**\ NestJS is a framework for building scalable and maintainable server-side applications using Node.js and TypeScript. It uses Express (or optionally Fastify) under the hood.

**Q2: What is a Module in NestJS?** Modules are decorators that group related controllers and providers together.

```
@Module({
  controllers: [UserController],
  providers: [UserService],
})
export class UserModule {}
```

**Q3: What are Controllers and Providers?**

- **Controllers** handle incoming requests and return responses.
- **Providers** (mostly services) contain the business logic and can be injected using Nest's Dependency Injection.

**Q4: DTO and Validation?** DTO (Data Transfer Object) is used to define the shape of data. We use `class-validator` with DTOs to enforce validation rules.

```
class CreateUserDto {
  @IsEmail()
  email: string;

  @IsString()
  name: string;
}
```

**Q5: What is Dependency Injection in NestJS?** It's a design pattern where classes receive their dependencies from an external source. Nest handles this automatically via metadata and decorators.

**Q6: Middleware and Guards?**

- **Middleware**: Runs before route handlers. Good for logging or modifying request objects.
- **Guards**: Used for authorization checks before route handlers.

**Q7: Interceptors and Pipes?**

- **Interceptors**: Modify input/output of a request.
- **Pipes**: Used for validation and transformation of incoming data.

---

✅This concludes a full-stack interview preparation guide across your tech stack. Let me know if you'd like coding exercises or a mock interview next!)