JavaScript Interview Q&As (Part 2)

🔹 Section 1: ES6+ Features

Q21. What are Template Literals in JavaScript?
Answer: Template literals use backticks (``) instead of quotes and allow string interpolation using ${}. They also support multi-line strings.
Example:

```
const name = "Yash";
console.log(`Hello, ${name}!`);
```

💡 Tip: Mention — “They make dynamic strings easier and cleaner.”

Q22. What are Modules (import/export) in JS?
Answer: Modules split code into multiple files for better organization and reusability. export shares code, import brings it into another file.
Example: export const x=10; → import {x} from './file.js'.
💡 Tip: Say — “Modules prevent polluting the global scope.”

Q23. Difference between Default and Named Exports?
Answer:

- Default export → only one per file, imported without {}.
- Named export → multiple exports, imported with {}.
  Example: export default func vs export {a, b}.
  💡 Tip: Default = main thing, Named = multiple helpers.

Q24. What is Destructuring in JavaScript?
Answer: Destructuring extracts values from arrays or objects into separate variables.
Example:

```
const {name, age} = user;
const [a,b] = [1,2];
```

💡 Tip: Say — "Destructuring makes code shorter and more readable."

Q25. What are Rest & Spread Operators?
Answer:

- Rest (...) collects arguments into an array.
- Spread (...) expands array/object elements.
  Example: **function sum(...nums){**

  return nums.reduce((a,b)=>a+b) }

💡 Tip: Mention — "Same syntax, different use: collect vs expand."

🔹 Section 2: Error Handling & Debugging

Q26. Difference between try...catch and promise .catch()?
Answer: try...catch handles synchronous errors, while .catch() is for promise rejections in async code.
Example:

```
try { riskyFn() } catch(e){ console.log(e) }

fetch('/api').catch(err => console.log(err))
```

💡 Tip: Say — "Use try...catch inside async/await functions."

Q27. What is the finally block used for?
Answer: finally runs after try/catch, regardless of success or failure. Often used for cleanup like closing DB connections.
Example:

```
try { ... } catch(e){ ... } finally { console.log("Done") }
```

💡 Tip: Say — "finally always runs → cleanup guarantee."

Q28. How do you create custom errors in JS?
Answer: Extend the Error class to make meaningful custom errors.
Example:

```
class AuthError extends Error {
  constructor(msg){ super(msg); this.name="AuthError"; }
}
```

💡 Tip: Say — “Custom errors improve debugging & clarity.”

Q29. What are common debugging techniques in JS?
Answer:

- Use console.log to inspect values.
- Use Chrome DevTools breakpoints.
- Debugging with debugger keyword.
- Linting tools (ESLint).
  💡 Tip: Always mention DevTools + breakpoints in interviews.

Q30. Difference between throw and return in error handling?
Answer:

- throw immediately stops execution and passes error to nearest catch.
- return just exits function with a value.
  Example: `throw new Error("Invalid") vs return false.`
  💡 Tip: Say — “throw = error, return = exit gracefully.”

🔹 Section 3: Advanced Concepts

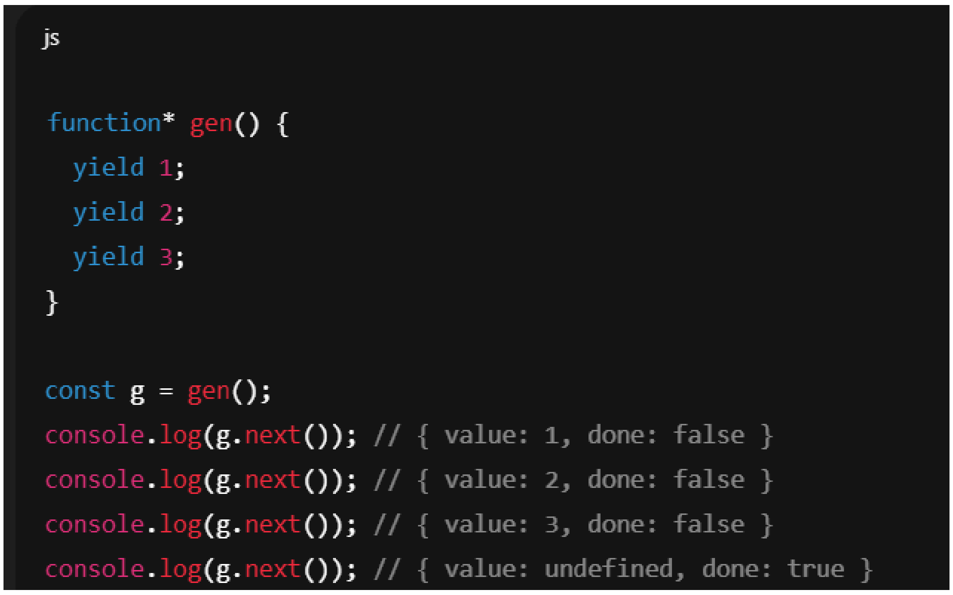
Q31. What is Event Bubbling vs Capturing?
Answer:

- Bubbling → event goes from child → parent (default).
- Capturing → parent → child (set with {capture:true}).
  💡 Tip: Use bubbling for most cases, capturing when order matters.

Q32. What are Generators in JavaScript?
Answer: Generators (function*) can pause execution using yield and

resume later. Useful for async workflows.
Example:

function* gen(){ yield 1; yield 2; }



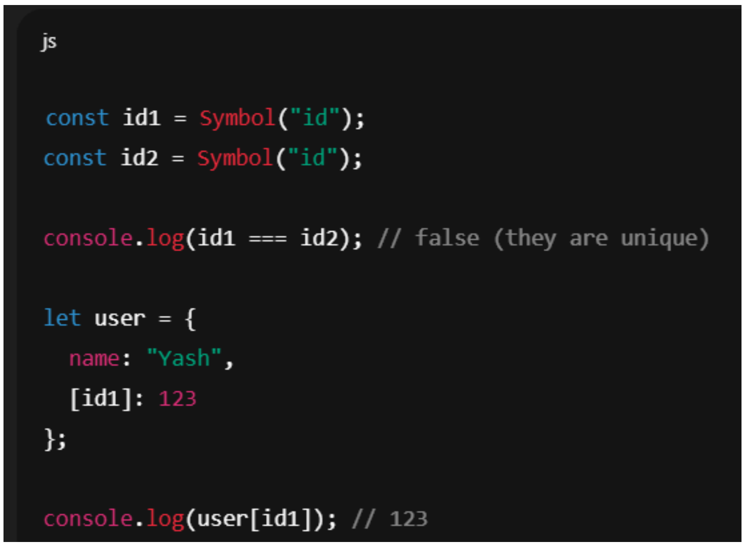
```js
function* gen() {
  yield 1;
  yield 2;
  yield 3;
}

const g = gen();
console.log(g.next()); // { value: 1, done: false }
console.log(g.next()); // { value: 2, done: false }
console.log(g.next()); // { value: 3, done: false }
console.log(g.next()); // { value: undefined, done: true }
```

💡 Tip: Mention — "Generators produce values lazily."

Q33. What is a Symbol in JavaScript?
Answer: Symbol is a unique, immutable primitive used as object keys. Even with same description, symbols are different.
Example:

const id = Symbol("id");



```js
const id1 = Symbol("id");
const id2 = Symbol("id");

console.log(id1 === id2); // false (they are unique)

let user = {
  name: "Yash",
  [id1]: 123
};

console.log(user[id1]); // 123
```

💡 Tip: Say — "Symbols prevent property name clashes."

Q34. What are WeakMap and WeakSet?
Answer: WeakMap = key-value pairs with object keys.

WeakSet = stores only objects. They don’t prevent garbage collection.
💡 Tip: Use when you need temporary associations with objects.

Q35. Explain Garbage Collection in JavaScript.
Answer: JS automatically clears memory by removing unused objects (no references left). Done via **mark-and-sweep** algorithm.
💡 Tip: Say — “GC is automatic, but memory leaks still happen.”

🔹 Section 4: Performance & Optimization

Q36. What causes memory leaks in JavaScript?
Answer: Memory leaks happen when references prevent garbage collection. Common causes:

- Global variables
- Forgotten timers (setInterval)
- Detached DOM nodes
  💡 Tip: Always say — “Leaks = unused memory never freed.”

Q37. How do you optimize large arrays or loops?
Answer: Use efficient methods like map, reduce instead of nested loops. Use pagination or generators for large datasets.
💡 Tip: Mention — “Break big tasks into smaller chunks.”

Q38. What is Lazy Loading in JavaScript?
Answer: Lazy loading means loading content/resources only when needed (like images below the fold).
Example: loading="lazy" for images.
💡 Tip: Say — “Improves performance by reducing initial load time.”

Q39. What is Tree Shaking in modern JS bundlers?
Answer: Tree shaking removes unused code during bundling (works with ES6 modules).
Example: import { usedFn } from 'lib' → unused functions are dropped.
💡 Tip: Mention Webpack/Rollup → common interview plus point.

Q40. How do you improve performance in a JS-heavy web app?
Answer:

- Use caching, lazy loading, and code splitting.
- Minimize DOM manipulation.
- Optimize loops and data structures.
- Use CDN for assets.
  💡 Tip: Always say — “Profile first, then optimize.”