

Q1: What is the difference between var, let, and const?

A: `var` is function-scoped, hoisted, allows redeclaration. `let` and `const` are block-scoped, in Temporal Dead Zone until initialized. `const` cannot be reassigned.

Q2: What will be the output of console.log(a); var a = 5;

A: Output: undefined. Because `var` is hoisted but initialized as undefined.

Q3: What will be the output of console.log(b); let b = 3;

A: Output: ReferenceError. Because `let` is hoisted but not initialized (TDZ).

Q4: What is the difference between == and ===?

A: `==` does type coercion, `===` checks strict equality. Example: '5' == 5 true, '5' === 5 false.

Q5: What is the difference between null and undefined?

A: `null` is intentional absence of value, `undefined` means declared but not assigned. `typeof null` returns 'object' (legacy bug).

Q6: How does this differ in arrow functions and regular functions?

A: Arrow functions inherit `this` from lexical scope. Regular functions bind `this` at runtime.

Q7: What are closures?

A: Closure = function + lexical environment. Can cause memory leaks if holding unnecessary references.

Q8: In the event loop, what runs first setTimeout(fn,0) or Promise.resolve().then(fn)?

A: Promise first, because microtasks run before macrotasks.

Q9: What is the prototype chain?

A: Prototype chain is the lookup mechanism for inherited properties. `prototype` belongs to constructors, `__proto__` links object to prototype.

Q10: Explain map, filter, reduce.

A: `map` transforms, `filter` selects, `reduce` accumulates. Example: [1,2,3].reduce((a,b)=>a+b,0).

Q11: Why is NaN === NaN false?

A: Because NaN is not equal to itself. Use Number.isNaN().

Q12: What's the difference between parseInt('08') and parseInt('08',10)?

A: Old engines parsed '08' as octal. Always specify radix for safety.

Q13: Difference between for..in and for..of?

A: `for..in` loops keys (including inherited). `for..of` loops values (iterables).

Q14: What issue arises with default parameters as objects?

A: Object default parameters create shared references mutation across calls.

Q15: Difference between shallow copy and deep copy?

A: Shallow copies only top-level. Deep copy copies nested. JSON methods fail for dates, functions.

Q16: How do closures implement encapsulation? Show a loop issue.

A: Closures hide variables inside a function scope. Loop issue: closures in loops capture same variable reference.

Q17: Explain the 4 binding rules of this.

A: Default: global/undefined. Implicit: object before dot. Explicit: call/apply/bind. New: bound to instance.

Q18: Difference between call, apply, bind?

A: `call` passes args individually, `apply` uses array, `bind` returns new function. Bind is permanent.

Q19: How are properties looked up in prototype inheritance?

A: JS checks own object, then prototype chain until Object.prototype. Property shadowing hides parent property.

Q20: Explain event loop phases: microtasks vs macrotasks.

A: Microtasks (Promises) always run before next macrotask (Timers/I/O).

Q21: How does error propagation work in promises?

A: Errors bubble down the chain until caught. Catch placement changes behavior.

Q22: How do async/await work with promises?

A: They pause execution until resolved. Internally built on promises. Errors handled with try/catch.

Q23: Difference between generators and async functions?

A: Generators yield lazily, async functions return promises. Yield gives control, await waits.

Q24: How do memory leaks occur in JS?

A: From closures, never-cleared timers, detached DOM, global variables.

Q25: Difference between debounce and throttle?

A: Debounce delays until inactivity, throttle limits execution rate.

Q26: What is immutability issue with nested objects?

A: Shallow immutability fails because nested properties still mutate.

Q27: What is event delegation?

A: Listener on parent handles child events using bubbling. Efficient for dynamic content.

Q28: What are Object.defineProperty descriptors?

A: writable: changeable, enumerable: shows in loop, configurable: can be redefined.

Q29: What is prototype pollution?

A: Injection of properties into Object.prototype. Prevent by freezing/validating inputs.

Q30: Difference between CommonJS and ES Modules?

A: CommonJS loads synchronously, runtime. ES Modules static, hoisted, async loading.

Q31: What are hidden classes and inline caching?

A: Engine optimizations. Hidden classes speed property lookup, inline caching optimizes repeated access.

Q32: Explain JIT compilation and deoptimization.

A: Parse optimize deopt. Deopt triggers: polymorphism, eval, with, changing shapes.

Q33: Why does property order affect performance?

A: Different order = different object shape breaks optimization.

Q34: Strict mode vs sloppy mode?

A: Strict: no silent errors, safer, allows optimization. Example: disallows implicit globals.

Q35: Difference between SameValue, SameValueZero, == ?

A: SameValue = Object.is, SameValueZero = array equality (+0/-0 same), == does coercion.

Q36: What is Tail Call Optimization?

A: TCO = no extra stack in tail recursion. Rarely supported except Safari.

Q37: What are WeakMap, WeakSet, WeakRef, FinalizationRegistry?

A: Weak collections don't prevent GC. WeakRef + FinalizationRegistry give manual hooks, unreliable timing.

Q38: What invariants must proxies respect?

A: Spec rules: e.g. non-configurable properties consistent. Violating throws TypeError.

Q39: How do cyclic ES modules work?

A: They resolve partially with live bindings. Can cause undefined before init.

Q40: What does structured clone algorithm support?

A: Clones objects, arrays, maps, sets. Cannot clone functions, DOM nodes.

Q41: What are SharedArrayBuffer and Atomics?

A: Shared memory for threads. Atomics ensure ordering and atomic operations.

Q42: Web Workers vs threads?

A: Workers = isolated threads with message passing. Data cloned or transferred.

Q43: How do async/await affect stack traces?

A: They lose sync context. Debug tools add async stack traces.

Q44: Advanced uses of Symbol?

A: Symbol.iterator for iteration, Symbol.toStringTag for tag, Symbol.species controls subclass returns.

Q45: How does promise job queue work?

A: Promise callbacks are microtasks, run after current code, before rendering/macrotasks.

Q46: What causes engine deoptimization?

A: Eval, with, polymorphic calls, dynamic property addition.

Q47: What bundler pitfalls affect runtime?

A: Tree-shaking drops side-effect imports, circular deps cause undefined.

Q48: What JS security issues exist?

A: Prototype poisoning, eval escapes. Prevent with CSP, validation.

Q49: How does garbage collection work?

A: Mark-sweep, generational GC. Leaks if closures or long-lived objects hold refs.

Q50: What are realms in JS?

A: Separate global envs (iframes). instanceof fails across realms due to different constructors.