50 Must-Solve JavaScript Interview Problems – Concept Notes

> **Purpose of this note**
> These are **thinking notes**, not answers. Use them before solving each problem to recall **what concept is being tested**, **what mental model to apply**, and **what mistakes to avoid**.

1–10: Scope, Hoisting & TDZ

1. `var` hoisting with access before declaration

- Tests: hoisting + initialization phase
- Key idea: declaration hoisted, assignment not
- Common mistake: assuming runtime error

2. `let` access before declaration

- Tests: Temporal Dead Zone (TDZ)
- Key idea: block-scoped + no early access

3. Block scope shadowing

- Tests: lexical scoping
- Key idea: inner scope does not affect outer

4. Function scope + `var` hoisting

- Tests: hoisting inside function
- Key idea: local `var` shadows outer variable

5. Default parameters + scope order

- Tests: parameter evaluation order
- Key idea: parameters have their own scope
- Watch for: TDZ inside parameters

6. `typeof` + TDZ

- Tests: misconception that `typeof` is always safe
- Key idea: TDZ still applies

7. Function declaration hoisting

- Tests: function hoisting vs execution order
- Key idea: function declarations fully hoisted

8. TDZ inside function

- Tests: let/const lifecycle
- Key idea: declaration exists but inaccessible

9. IIFE + `var` hoisting

- Tests: function scope isolation
- Key idea: `var` hoists within IIFE

10. Function expression hoisting

- Tests: variable hoisting vs function hoisting
- Key idea: expression is not hoisted

11–20: Closures

11. Basic counter closure

- Tests: closure over private state
- Key idea: state persists across calls

12. Closure inside loop with `var`

- Tests: single shared binding
- Key idea: closure captures variable, not value

13. Closure inside loop with `let`

- Tests: block-scoped bindings
- Key idea: new binding per iteration

14. Function array + closure

- Tests: closure + loop trap
- Key idea: all functions share same variable

15. Nested closures

- Tests: lexical environment chain
- Key idea: inner function remembers outer args

16. Closure reads latest value

- Tests: closures reference, not snapshot
- Key idea: value resolved at execution time

17. `once` function

- Tests: real-world closure usage
- Key idea: encapsulated mutable state

18. Closure with mutation

- Tests: post-increment behavior
- Key idea: mutation inside closure

19. Factory functions

- Tests: function factories
- Key idea: each call creates new lexical scope

20. Multiple closures from same factory

- Tests: independent closure states
- Key idea: closures don't share state unless same scope

21–30: `this`, call, apply, bind

21. Global `this`

- Tests: runtime environment (browser vs Node)
- Key idea: depends on strict mode & environment

22. Method extraction

- Tests: lost `this` binding
- Key idea: `this` determined at call site

23. Arrow function `this`

- Tests: lexical `this`
- Key idea: arrows don't have own `this`

24. Normal function `this`

- Tests: default binding
- Key idea: non-method call

25. Constructor function

- Tests: `new` keyword behavior
- Key idea: `this` points to new object

26. Inner function inside method

- Tests: function vs method context
- Key idea: inner function loses `this`

27. Arrow inside method

- Tests: lexical `this` capture
- Key idea: arrow inherits `this` from method

28. `call`

- Tests: explicit binding
- Key idea: immediate invocation with custom `this`

29. `bind`

- Tests: permanent binding
- Key idea: returns new function

30. Callback context loss

- Tests: async + `this`
- Key idea: reference passed, not method call

31–40: Event Loop, Promises & Async

31. Sync vs async order

- Tests: call stack vs task queue
- Key idea: timers are async

32. Microtask vs sync

- Tests: promise microtasks
- Key idea: microtasks run after sync

33. Microtask vs macrotask

- Tests: priority order
- Key idea: microtasks first

34. `async/await` execution flow

- Tests: await pauses function only
- Key idea: rest runs as microtask

35. Promise chaining

- Tests: return value vs promise
- Key idea: promise flattening

36. Error handling in promises

- Tests: rejection propagation
- Key idea: `catch` handles rejection

37. Async function errors

- Tests: async returns promise
- Key idea: throw = rejected promise

38. `Promise.all`

- Tests: fail-fast behavior
- Key idea: first rejection wins

39. Async function return value

- Tests: implicit promise wrapping
- Key idea: async always returns promise

40. Top-level `await`

- Tests: module vs script
- Key idea: only valid in ES modules

41–50: Equality, Objects, Arrays

41. Object reference comparison

- Tests: reference equality
- Key idea: different objects ≠ equal

42. Abstract equality coercion

- Tests: coercion rules
- Key idea: tricky conversions

43. `null` vs `undefined`

- Tests: loose equality exception
- Key idea: special rule in spec

44. `typeof null`

- Tests: historical JS bug
- Key idea: legacy behavior

45. Same reference comparison

- Tests: reference assignment
- Key idea: both point to same object

46. Array length mutation

- Tests: array internals
- Key idea: truncation behavior

47. `Object.freeze`

- Tests: immutability vs strict mode
- Key idea: silent failure vs error

48. Shared array reference

- Tests: mutation side effects
- Key idea: arrays are objects

49. `||` vs `??`

- Tests: falsy vs nullish
- Key idea: `0` and `""` difference

50. `NaN` comparison

- Tests: IEEE floating rules
- Key idea: NaN is never equal

How to Practice

For every problem: 1. Identify **concept first** 2. Write execution steps 3. Predict output 4. Explain *why*, not *what* 5. Rewrite code to avoid bug

---

> This note is designed to build **interview-grade JavaScript thinking**, not memorized answers.