
🟢 **Beginner Level (Core JS Fundamentals)**

- primitive vs non-primitive datatypes
- immutable datatype
- falsy values
- null vs undefined
- null pointer exception
- nullish operator (??)
- nullish assignment (??=)
- coercion
- pass by value vs pass by reference
- default parameters
- double exclamation (!!)
- console.log(1 + +'2')
- console.log('A' - 1)
- console.log(null == undefined)
- 1 | 2 vs 1 || 2
- typeof, instanceof
- “use strict”
- eval()
- why eval() is considered dangerous
- mark and sweep
- boxing / String prototype
- print current time without printing date
- current date in D/M/Y format
- homogeneous
- is JIT used all the time?
- features of JS runtime

🟡 **Intermediate Level (Functions, Scope, and Objects)**

- var / let / const

- variable shadowing
- illegal shadowing
- block scope
- function scope
- this keyword
- this in arrow function
- this substitution
- arrow function vs regular function
- pure function
- advantages of closures
- closure benefits
- limitations of closure
- IIFE (Immediately Invoked Function Expression)
- creating private scope
- private constructor function
- private properties
- class: private property
- getters and setters
- rest operator
- rest operator (use it outside a function)
- spread operator
- function composition (what is it & where used)
- method chaining
- call / apply / bind
- bind concept and code
- shallow copy object but exclude a key
- deep cloning of an object
- swap keys and values of object ignoring null values
- object methods
- object workouts
- polyfills

- proxy object implementation
- Proxy object for object validation
- implementation of proxy object

🟠 **Intermediate–Advanced (Asynchronous & Advanced Functionality)**

- asynchronous programming (async / await)
- timer functions
- reduce
- every / some
- generator function
- generator function implementation
- generator function that yields random unique elements
- callback hell → promise → async/await
- callbacks
- promise example
- promise methods
- promise.allSettled
- promise with fs
- promise with includes
- Promise vs Observables
- creating child_process
- phases in event loop
- process.nextTick
- setImmediate vs process.nextTick
- microtask vs macrotask queue
- reactor pattern
- event module, event.on, event.emit
- create a custom event emitter
- custom event emitter implementation
- execute a callback function every n seconds up to 10 times
- stopPropagation

- event delegation
- event propagation vs delegation
- native events vs synthetic events
- event pooling
- communication between components using useRef (JS foundation)
- asynchronous vs synchronous code execution

🔵 **Advanced (Performance, Patterns, Internals)**

- mark and sweep (garbage collection)
- JIT compilation
- pure vs impure functions
- immutable data structures
- polyfills (custom implementation)
- function composition in production
- proxy for validation & interception
- WeakSet vs WeakMap
- WeakMap and WeakSet implementation
- hash map vs hash table (conceptual, JS Map)
- custom map / filter function implementation
- creating private variables (closures, symbols, WeakMaps)
- Symbol()
- flatMap()
- unary function
- Object.entries(), Object.keys(), Object.values()
- encode a URL
- lambda expressions
- destructuring with defaults (const {name = "abcd"} = res)
- shallow vs deep copy (with structuredClone or recursion)
- creating custom event emitter
- custom hook logic understanding (from JS concept)
- Reflect API

- Proxy traps (get, set, deleteProperty)
- event loop phases in detail
- JS concurrency vs parallelism
- JS memory management
- call stack & heap visualization
- Weak references & GC behavior
- building polyfills for map, filter, reduce
- currying
- memoization
- debounce & throttle (core JS implementation)

⚫ **Expert (Low-level, Runtime, and Engine Concepts)**

- JS runtime architecture (call stack, heap, Web APIs, queue)
- V8 Engine internals
- Just-In-Time (JIT) compilation flow
- garbage collection algorithms (Mark & Sweep, Generational GC)
- blocking vs non-blocking calls
- starvation
- i/o bound vs cpu bound (in JS event loop context)
- concurrency vs parallelism in JS
- worker thread concept (JS-level understanding)
- thread pool (libuv)
- reactor pattern (in JS)
- Event loop phases (Timers, I/O, Poll, Check, Close)
- async task scheduling (Promise microtasks, nextTick queue)
- WeakRefs and FinalizationRegistry
- performance optimization in JS runtime
- security concepts (XSS, CSRF from JS POV)
