30 Days Roadmap - Can you master JavaScript?

📦 Week 1 — Foundations & Core Behavior

📆 Day 1 — Values, Types, and Coercion

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟢 Beginner
>
> Category: Internals / Types

📖 What to Study (Time Budget: ~2 Hours)

- **A. Main Concept**

  Understand JavaScript’s primitive types, objects, value vs. reference, and how coercion works.

- **B. Subtopics / Gotchas / Patterns**

- `undefined` vs. `null`
- Strict (`===`) vs. loose (`==`) equality
- Truthy and falsy values
- Typeof limitations
- Object wrappers (e.g., `new Number(5)`)

- **C. Optional Nuance or Edge Case**
  - Behavior of `NaN` (self-inequality), Symbol uniqueness, `Object.is` edge cases

❓ Sample Interview Questions

1. How does JavaScript handle equality between different types (e.g., `==` vs `===`)?
2. Why is `typeof null` equal to `"object"`?

🧪 Practical Task

- Create a table for all JS types with sample values and results of `typeof` checks.
- Write a function that reliably checks for “plain” objects.

📆 Day 2 — Variable Declarations, Hoisting, and TDZ

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟢 Beginner
>
> Category: Internals / Scope

📖 What to Study

- **A. Main Concept**
  - Differences between `var`, `let`, and `const`
  - Understanding hoisting and TDZ (Temporal Dead Zone)

- **B. Subtopics / Gotchas / Patterns**
  - Shadowing
  - Mutations of const objects/arrays
  - Re-declarations
- **C. Optional Nuance or Edge Case**
  - Block vs. function scope with `var`

❓ Sample Interview Questions

1. Describe variable hoisting in JavaScript.
2. What happens if you access a `let` variable before its declaration?

🧪 Practical Task

- Conceptual tracing: Draw a timeline diagram for hoisting/TDZ in a code sample.

📆 Day 3 — Functions: Declarations, Expressions, and Arrow Functions

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟢 Beginner
>
> Category: Functions / Scope

📖 What to Study

- **A. Main Concept**
  - Function declaration vs. expression vs. arrow function (syntax and behavior)
- **B. Subtopics / Gotchas / Patterns**
  - Hoisting with function declarations

- Arrow functions and lexical `this`
- No `arguments` object for arrow functions

- **C. Optional Nuance or Edge Case**
  - Arrow functions as methods – what breaks?
  - Returning object literals from arrow functions

❓ Sample Interview Questions

1. How does the `this` keyword behave differently in arrow functions compared to traditional functions?
2. What would happen here?

```
jslet obj = {
 num: 5,
 arrow: () ⇒ this.num,
 regular: function() { return this.num; }
}
```

🧪 Practical Task

- Write or trace code highlighting `this` in arrow vs. traditional functions.

📆 Day 4 — Scope, Closures, and the LEGB Model

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Scope / Closures / Internals

📖 What to Study

- **A. Main Concept**
  - Lexical scoping in JavaScript
  - Closure formation and its role in callbacks/loops
- **B. Subtopics / Gotchas / Patterns**

    - Block vs. function vs. global scope
    - Common closure bugs (e.g., capturing loop variables)
    - IIFEs (Immediately Invoked Function Expressions)
- **C. Optional Nuance or Edge Case**
    - Memory leaks with closures retaining large scopes

❓ Sample Interview Questions

1. Explain how closures work. Why are they important?
2. How do you ensure that callbacks inside loops capture the correct value of an iterator?

🧪 Practical Task

- Code exercise: Implement a counter using closures.

📅 Day 5 — The `this` Keyword Deep Dive

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Runtime / Context / Internals

📖 What to Study

- **A. Main Concept**
    - How `this` is determined (call-site based, not author-time)
    - The four binding rules (implicit, explicit, new, default)
- **B. Subtopics / Gotchas / Patterns**
    - Function vs. method context
    - Arrow functions and `this`
    - Loss of binding in callbacks
- **C. Optional Nuance or Edge Case**

- Binding priority when combined (`bind` + `new`)
- `this` in strict mode

❓ Sample Interview Questions

1. Explain the different rules for how `this` is bound in JavaScript.
2. What happens to `this` inside a regular function when called as a callback?

🧪 Practical Task

- Diagram: For several call-sites, annotate which `this` binding rule applies.

📆 Day 6 — Prototypes and Inheritance

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Internals / Objects / Patterns

📖 What to Study

- **A. Main Concept**
  - Prototypal inheritance, the prototype chain, `__proto__`, `Object.create`, `class` sugar
- **B. Subtopics / Gotchas / Patterns**
  - Shared mutable state via prototypes
  - Overriding vs. shadowing
  - The difference between prototype and instance properties
- **C. Optional Nuance or Edge Case**
  - Customizing inheritance via `Object.setPrototypeOf`, `super` keyword

❓ Sample Interview Questions

1. How does JavaScript's prototype chain resolve property access?
2. What's the difference between `prototype` and `__proto__`?

🧪 Practical Task

- Trace a property lookup across an inheritance chain with a drawn diagram.

📆 Day 7 — Objects: Creation, Property Descriptors, and Modern Patterns

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Objects / Patterns / Internals

📖 What to Study

- **A. Main Concept**
  - Object literal shorthand, computed properties, property descriptors
- **B. Subtopics / Gotchas / Patterns**
  - Enumerability, configurability, and writability
  - Using `Object.freeze`, `Object.seal`
  - Shallow vs. deep copy
- **C. Optional Nuance or Edge Case**
  - Using symbols as property keys

❓ Sample Interview Questions

1. How do you make an object property read-only?
2. Describe the difference between enumerable and non-enumerable properties in practice.

🧪 Practical Task

- Write or analyze code using `Object.defineProperty` and compare properties created via shorthand.

📦 Week 2 — Asynchrony & Data Structures

📆 Day 8 — Execution Context and Event Loop

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Internals / Async / Runtime

📖 What to Study

- **A. Main Concept**
  - How JavaScript executes code: call stack, event loop, microtasks vs. macrotasks
- **B. Subtopics / Gotchas / Patterns**
  - Starvation (microtask queue growth)
  - `setTimeout`, `Promise.resolve`, `queueMicrotask`
  - Blocking the event loop
- **C. Optional Nuance or Edge Case**
  - Prioritization differences between micro/macro tasks

❓ Sample Interview Questions

1. Explain the differences between microtasks and macrotasks.
2. What will the output be?

```
jsconsole.log(1);
setTimeout(() => console.log(2));
Promise.resolve().then(() => console.log(3));
console.log(4);
```

🧪 Practical Task

- Draw a call stack and queue trace for given async code.

📆 Day 9 — Promises and Async/Await

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Async / Patterns

📖 What to Study

- **A. Main Concept**
  - How Promises work, chaining, error handling, and the async/await abstraction
- **B. Subtopics / Gotchas / Patterns**
  - Promise chaining — return vs. side effect
  - Error propagation and missed `catch`
  - Sequential vs. parallel async flows
- **C. Optional Nuance or Edge Case**
  - Await-ing non-Promise values; resolving with another Promise

❓ Sample Interview Questions

1. How does async/await make working with Promises easier, and what pitfalls remain?
2. What happens if you forget to return a Promise in a `.then` chain?

🧪 Practical Task

- Refactor a callback-based async function to Promise and then to async/await.

📆 Day 10 — Arrays: Methods, Iterability, and Mutation

🧨 Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟢 Beginner
>
> Category: Data Structures / Patterns

📖 What to Study

- **A. Main Concept**
  - Arrays as objects, mutative vs. non-mutative array methods, iteration protocols
- **B. Subtopics / Gotchas / Patterns**
  - Array holes vs. undefined values
  - Spread and destructuring with arrays
  - `Array.from`, array-likes, and iterables
- **C. Optional Nuance or Edge Case**
  - Creating sparse arrays vs. new Array(size)

❓ Sample Interview Questions

1. What’s the difference between `.forEach`, `.map`, and `.reduce`?
2. How would you convert an array-like object to a true array?

🧪 Practical Task

- Given an “array-like” object, implement a generic function to make it iterable.

📆 Day 11 — Maps, Sets, WeakMaps, WeakSets

🧨 Tags:

> Difficulty: 🟡 Moderate
>
> Category: Data Structures / Patterns

📖 What to Study

- **A. Main Concept**
  - Key differences between traditional objects and Maps/Sets, use cases for weak collections
- **B. Subtopics / Gotchas / Patterns**
  - Key types allowed in Map vs. Object
  - Garbage collection with WeakMap/WeakSet
  - Uniqueness guarantee and performance considerations
- **C. Optional Nuance or Edge Case**
  - Weak collections and non-enumerability

❓ Sample Interview Questions

1. When should you use a Map instead of an Object in JavaScript?
2. What are the memory management implications of WeakMap?

🧪 Practical Task

- Table: Compare use-cases and methods between Map, Set, WeakMap, and WeakSet.

📆 Day 12 — String & Number Methods, Template Literals

🏷️ Tags:

> Difficulty: 🟢 Beginner
>
> Category: Core / Patterns

📖 What to Study

- **A. Main Concept**
  - Modern string and number methods, interpolation, and immutability
- **B. Subtopics / Gotchas / Patterns**
  - Method chaining on strings
  - Numeric separators (`1_000_000`)
  - Tagged templates
- **C. Optional Nuance or Edge Case**
  - Unicode and surrogate pairs in strings

❓ Sample Interview Questions

1. How are template literals different from traditional string concatenation?
2. Give an example where method-chaining on string values would fail.

🧪 Practical Task

- For a sample tagged template, explain how interpolation and tag functions interact.

📅 Day 13 — Destructuring and Spread/Rest

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Patterns / Modern JavaScript

📖 What to Study

- **A. Main Concept**
  - Array and object destructuring; using spread and rest (...syntax) for copies/parameters
- **B. Subtopics / Gotchas / Patterns**

    - Default values in destructuring
    - Nested destructuring with renaming
    - Shallow vs. deep copy issues
- **C. Optional Nuance or Edge Case**
    - Performance/footguns with large spreads

❓ Sample Interview Questions

1. Explain how object destructuring assignment works in function parameters.
2. What's the difference between shallow and deep copy when using spread syntax?

🧪 Practical Task

- Practice: Write a function with destructuring and rest for arguments, then explain the destructured values.

📆 Day 14 — Modern ES Features (from ES2021 to ES2025)

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Modern JavaScript

📖 What to Study

- **A. Main Concept**
    - Key recent features: optional chaining, nullish coalescing, logical assignment, top-level await, records & tuples (if available)
- **B. Subtopics / Gotchas / Patterns**
    - Safe property access with `?.`
    - When `??` differs from `||`
    - Potential runtime surprises with new features

- **C. Optional Nuance or Edge Case**
  - How would legacy code react to modern patterns? Polyfill issues?

❓ Sample Interview Questions

1. Describe a scenario where optional chaining solves a real bug.
2. What's the difference between `??` and `||`?

🧪 Practical Task

- Refactor a "deep property access" code block using optional chaining and nullish coalescing.

📦 Week 3 — Internals, Performance, and Advanced Patterns

📆 Day 15 — Memory Management & Garbage Collection

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Internals / Runtime

📖 What to Study

- **A. Main Concept**
  - How garbage collection works, memory leaks in JS, reference counting vs. mark-and-sweep
- **B. Subtopics / Gotchas / Patterns**
  - Retaining references via closures or DOM
  - Leaks in event listeners, timers, and caches
- **C. Optional Nuance or Edge Case**
  - Weak references, `FinalizationRegistry`

❓ Sample Interview Questions

1. Explain a real-world cause and fix for a memory leak in JS.
2. What is the difference between strong and weak reference?

🧪 Practical Task

- Diagram: Trace object reachability and GC eligibility in a complex sample.

📆 Day 16 — Error Handling, Try/Catch, and Defensive Code

📌 Tags:

> Difficulty: 🟡 Moderate
>
> Category: Internals / Patterns

📖 What to Study

- **A. Main Concept**
  - Error propagation with try/catch/finally, `throw`, custom Error classes, Promise rejection
- **B. Subtopics / Gotchas / Patterns**
  - Errors inside async functions
  - Synchronous vs. asynchronous throw/catch
  - Defensive coding patterns (guard clauses, assertions)
- **C. Optional Nuance or Edge Case**
  - Re-throwing errors, error boundary patterns

❓ Sample Interview Questions

1. How does error handling differ between synchronous and asynchronous JavaScript code?
2. What's the risk of swallowing errors in a promise chain?

🧪 Practical Task

- Write a resilient function that always logs but never completely swallows unexpected errors.

📆 Day 17 — Immutability, Copy Patterns, and Functional Practices

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Patterns / Modern JavaScript

📖 What to Study

- **A. Main Concept**
  - Value vs. reference, copying arrays/objects, functional JS basics
- **B. Subtopics / Gotchas / Patterns**
  - Shallow copy pitfalls
  - `Object.assign` vs. spread vs. structuredClone
  - Immutability enforcement patterns
- **C. Optional Nuance or Edge Case**
  - Risks around deeply nested objects

❓ Sample Interview Questions

1. How would you perform a deep clone in JS, and when is it needed?
2. When does modifying one object unintentionally affect another?

🧪 Practical Task

- Code: Demonstrate structured cloning, explain what fails with non-cloneable objects.

📆 Day 18 — Iterators, Generators, and Custom Iteration

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Internals / Async / Patterns

📖 What to Study

- **A. Main Concept**
  - Protocols for iterable/iterator/generator. Use-cases for custom iteration patterns.
- **B. Subtopics / Gotchas / Patterns**
  - Implementing Symbol.iterator
  - Lazy evaluation with generators
  - Async iteration basics
- **C. Optional Nuance or Edge Case**
  - When would you use a generator vs. a traditional function?

❓ Sample Interview Questions

1. How do you make an object iterable in JS?
2. Write a simple generator for a custom data source.

🧪 Practical Task

- Implement and trace a custom iterator or generator function.

📆 Day 19 — Modules, Imports/Exports, and Scope Privacy

🏷️ Tags:

> 🔥 DON’T DARE TO SKIP
>
> Difficulty: 🟡 Moderate
>
> Category: Modern JavaScript / Patterns

📖 What to Study

- **A. Main Concept**
  - ES modules, named vs default exports, import patterns, top-level await
- **B. Subtopics / Gotchas / Patterns**
  - Module scope vs. global scope
  - Live bindings and circular dependencies
  - Static analysis and tree-shaking
- **C. Optional Nuance or Edge Case**
  - How module caching works

❓ Sample Interview Questions

1. What is the difference between CommonJS and ES modules?
2. How does top-level await affect module loading?

🧪 Practical Task

- Diagram: Visualize module load order and update on exports.

📆 Day 20 — Symbols, Hidden Properties, and Meta-programming

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Internals / Patterns / Modern JavaScript

📖 What to Study

- **A. Main Concept**
  - Using symbols for property keys, hiding properties, customizing behavior with meta-programming
- **B. Subtopics / Gotchas / Patterns**
  - Global vs. private symbols
  - Well-known symbols (`Symbol.iterator`, `Symbol.toStringTag`)
  - Uses for symbol properties in libraries
- **C. Optional Nuance or Edge Case**
  - Enumeration and property/key visibility

❓ Sample Interview Questions

1. How would you make a property non-enumerable and non-guessable in JS?
2. What are “well-known” symbols and why are they important?

🧪 Practical Task

- Trace: Create an object with mixed string and symbol keys, show the results with `for...in` and `Object.getOwnPropertySymbols`.

📆 Day 21 — Proxy, Reflect, and Runtime Traps

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Runtime / Patterns / Modern JavaScript

📖 What to Study

- **A. Main Concept**
  - Proxy handler traps, Reflect API, use-cases (logging, validation, reactive patterns)
- **B. Subtopics / Gotchas / Patterns**

    - The “get” and “set” traps
    - Proxy over arrays, classes, and functions
    - Performance and debugging challenges
- **C. Optional Nuance or Edge Case**
    - Revocable proxies, proxy invariants

❓ Sample Interview Questions

1. Give a real-world example of using a Proxy in JavaScript.
2. What happens if you break an invariant (e.g., set a non-writable property via Proxy)?

🧪 Practical Task

- Implement a logging proxy that tracks property reads and writes on an object.

📦 Week 4 — Mastery, Patterns, and Interview-Ready Edge-cases

📆 Day 22 — Event Delegation, Propagation, and Custom Events

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Patterns / Runtime / Async

📖 What to Study

- **A. Main Concept**
    - Event bubbling, capturing, delegation, and emitting custom events
- **B. Subtopics / Gotchas / Patterns**
    - `stopPropagation`, `preventDefault`

    - Pattern: using delegation for performance
    - Creating and dispatching CustomEvent
- **C. Optional Nuance or Edge Case**
    - Reentrancy and event queue timing with custom events

❓ Sample Interview Questions

1. How does event delegation work, and why is it important?
2. What would happen if you call stopPropagation() in a capturing handler?

🧪 Practical Task

- Visualization: Map event propagation path for a nested DOM structure.

📆 Day 23 — Async Patterns: Debouncing, Throttling, and Queues

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Patterns / Async

📖 What to Study

- **A. Main Concept**
    - Debounce and throttle patterns for handling async flows and performance
- **B. Subtopics / Gotchas / Patterns**
    - Difference between debounce and throttle
    - Queueing promises for concurrency control
- **C. Optional Nuance or Edge Case**
    - Edge-case: leading/trailing calls and cancellation

❓ Sample Interview Questions

1. Describe the difference between throttling and debouncing with examples.
2. How would you implement a concurrency-limited promise queue?

🧪 Practical Task

- Implement: Write a debounce function and show when it fires.

📆 Day 24 — Defensive Coding, Guard Patterns, and Type Resilience

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Patterns / Modern JavaScript

📖 What to Study

- **A. Main Concept**
  - Building fail-safe code (type checks, guards, input validation)
- **B. Subtopics / Gotchas / Patterns**
  - Pattern: Defensive “bail-out” (early return)
  - Making code robust to “invalid” inputs
  - Avoiding Type Coercion Surprises
- **C. Optional Nuance or Edge Case**
  - Pitfalls of overly-defensive code

❓ Sample Interview Questions

1. How do you make a function robust against bad input?
2. What’s the trade-off between fail-fast vs. fail-silent in a reusable JS utility?

📆 Day 25 — Composition, Higher-Order Functions, and Reusability

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Patterns / Functional JS

📖 What to Study

- **A. Main Concept**
  - Function composition, map/filter/reduce, higher-order function patterns
- **B. Subtopics / Gotchas / Patterns**
  - Currying and partial application
  - Passing functions and closures safely
  - Chaining and compositionality
- **C. Optional Nuance or Edge Case**
  - Law of Demeter in JS context

❓ Sample Interview Questions

1. What is a higher-order function? Give examples from built-in JS methods.
2. How can composition make code more testable and modular?

🧪 Practical Task

- Code: Compose several simple utilities for data transformation.

📆 Day 26 — Defensive Async: Race Conditions & Deadlocks

🏷️ Tags:

> Difficulty: 🔴 Advanced
>
> Category: Async / Patterns

📖 What to Study

- **A. Main Concept**
  - Understanding race conditions, avoiding double-resolutions, atomic operations in async code
- **B. Subtopics / Gotchas / Patterns**
  - Locking patterns (JS-only, with references)
  - Idempotency — making promises safe for repeats
  - Deadlocks with chained or cyclic async code
- **C. Optional Nuance or Edge Case**
  - Using `AbortController` for cancellation and cleanup

❓ Sample Interview Questions

1. How do you prevent race conditions in async JavaScript code?
2. What tools exist (in JS) to safely cancel or abort in-progress async work?

🧪 Practical Task

- Code: Implement a simple mutex/lock using promises.

📅 Day 27 — Defensive Patterns for Security and Robustness

📌 Tags:

> Difficulty: 🔴 Advanced
>
> Category: Patterns / Defensive JS

📖 What to Study

- **A. Main Concept**
  - Secure JS coding: injection risks, prototype pollution, and safe property access
- **B. Subtopics / Gotchas / Patterns**
  - Reading/writing only “own” properties

- Defensive input handling (escaping, filtering)
- Immutability for security

- **C. Optional Nuance or Edge Case**
  - Attacks via prototype chain manipulation

❓ Sample Interview Questions

1. What is prototype pollution, and how can you defend against it?
2. How do you ensure you only interact with your own object's properties in JS?

📆 Day 28 — Edge-Case Marathon: "What Actually Happens?"

🏷️ Tags:

> 🔥 DON'T DARE TO SKIP
>
> Difficulty: 🔴 Advanced
>
> Category: Edge Cases / Interview Practice

📖 What to Study

- **A. Main Concept**
  - Rapid-fire review of "gotchas", runtime quirks, and rarely tested behaviors
- **B. Subtopics / Gotchas / Patterns**
  - Edge-case coercions (`[] == ![]`, etc)
  - Scoping surprises (`for (var i...)`, closure bugs)
  - Async scheduler edge cases (chained promises inside setTimeouts)
- **C. Optional Nuance or Edge Case**
  - Odd interactions between new JS features and legacy patterns

❓ Sample Interview Questions

1. Predict the output:

```
jslet a = [];
if(a == !a) { console.log("true"); } else { console.log("false"); }
```

2. What happens if you call `resolve` and `reject` in the same Promise executor?

🧪 Practical Task

- Collect and solve at least 5 “weird output” JS snippets. Annotate your reasoning.

📆 Day 29 — (Soft Skill) Product/Systems Thinking

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Product Thinking / Soft Skill

📖 What to Study

- **A. Main Concept**
  - Thinking beyond code: how does your JS logic serve actual product/business goals?
- **B. Subtopics / Gotchas / Patterns**
  - User impact through code decisions
  - Trade-offs in code choices (performance, accessibility, team velocity)
- **C. Optional Nuance or Edge Case**
  - Bias toward action vs. avoidance of risk

❓ Sample Behavioral Interview Question

- Tell me about a time when your code decision improved a product outcome.

Mental Model

- Use the “Five Whys” technique to trace proposed features/bugs from design to code. Exercise: Trace a recent bug root-cause down to your code design.

📆 Day 30 — (Soft Skill) Technical Ownership + Communication

🏷️ Tags:

> Difficulty: 🟡 Moderate
>
> Category: Ownership / Soft Skill

📖 What to Study

- **A. Main Concept**
  - Communicate decisions to stakeholders, defend technical trade-offs, take ownership of bugs
- **B. Subtopics / Gotchas / Patterns**
  - Explain code clearly (“why not just ...?”)
  - Techniques for raising issues early
  - Demonstrating accountability and continued learning
- **C. Optional Nuance or Edge Case**
  - Navigating when you are “not the expert”

❓ Sample Behavioral Interview Question

- Tell me about a time you missed a bug in JS—how did you take ownership? How did you ensure it didn’t happen again?

Mental Model

- Use the “Ask for Feedback Early” exercise: Draft an email or post for code review, asking for feedback on a tricky design, showing transparency and openness.

Congratulations!

This roadmap focuses only on high-impact, interview-relevant skills and patterns. Spend time on the daily “what to study” and practical/diagramming tasks. Rigorously rehearse both your explanations and code reasoning to stand out in elite interview pipelines.