JavaScript Training Protocol: Prerequisites for React

This protocol simulates 30 escalating operational challenges for Section 9, targeting the core JavaScript features essential for modern React component development.

🟢 Tier 1: Fundamentals and Initial Deployment (Easy - 10 Exercises)

Focus: let/const, Arrow Functions, Basic Syntax, Conditionals.

#	Concept	Section 9 Scenario
1.	**let vs. const**	Define **Motoko's cyberbrain firmware version** using a variable that cannot be reassigned. Define her current **target coordinates** using a variable that might change during the mission.
2.	**Template Literals**	Construct a status report string for Aramaki: "Agent [Agent Name] is deployed at Sector [Sector Number]. Current threat level: [Level]." Use template literals for interpolation.
3.	**Arrow Function (Simple)**	Create a concise arrow function calculateEnergy that accepts powerConsumption and returns powerConsumption * 0.95 (accounting for a 5% system loss).
4.	**Ternary Operator**	Batou is checking equipment. Use a ternary operator to set status to "Active" if batteryLevel > 20, otherwise set it to "Recharge".
5.	**Block Scoping**	Inside an if block, declare a new variable tempCode using let. Demonstrate that this variable is not accessible outside the block.
6.	**Object Access**	Given a Tachikoma object with nested properties for pilot and weapons, log the pilot's name (Motoko) and the first weapon's name.
7.	**Array Access**	The logFiles array contains three strings. Access and log the string at the center index (index 1).
8.	**Function Declaration**	Write a named function authenticateAgent(id, password) that simply returns true if id is "Major" and password is "Ghost", otherwise false.
9.	**Arrow Function (Implicit Return)**	Write a single-line arrow function isOperational(status) that implicitly returns status === 'Ready'.
10.	**Default Parameters**	Create a function deployAgent(name, sector = 'Unknown') that uses a default parameter for the sector. Call it once with, and once without, the sector argument.

🟡 Tier 2: ES6 Mastery and Functional Patterns (Intermediate - 10 Exercises)

Focus: Destructuring, Spread, Rest, Immutability, map, filter, reduce. **These are crucial for component state and rendering.**

#	Concept	Section 9 Scenario
11.	**Object Destructuring**	Extract the name and rank from the Aramaki object into separate variables with a single line of code.
12.	**Array Destructuring**	The missionTimeline array has 5 steps. Use array destructuring to assign the first step to initialScan and the third step to infiltrationPoint.
13.	**Spread Operator (Array Copy)**	Create a new array newTargets by performing a **shallow copy** of the existing targetList array and adding one new target without mutating the original targetList.
14.	**Spread Operator (Object Merge)**	Merge a TachikomaBase object and a TachikomaUpgrades object into a single fullTachikoma object.
15.	**Array map()**	Use map() on the sensorReadings array of numbers to generate a new array where each reading is multiplied by 1.5 (simulating signal boost).
16.	**Object map() in JSX-Context**	You have an array of agent objects [{id: 1, name: 'Motoko'}, {id: 2, name: 'Batou'}]. Use map() to transform this into an array of strings like [' - Motoko (ID: 1) ', ' - Batou (ID: 2) '].
17.	**Array filter()**	Filter the agentRoster array of objects to create a new array containing only agents whose securityLevel is greater than or equal to 7.
18.	**Filtering (Immutability)**	Create a function removeAgent(roster, agentId) that uses filter() to remove an agent without modifying the original roster array.
19.	**Array reduce() (Summing)**	Calculate the total **damage potential** by summing the attackValue property across all objects in the weaponInventory array.
20.	**Array reduce() (Tallying/Grouping)**	Use reduce() on the logEntries array to count how many entries have the status: 'Error'.

🔴 Tier 3: Asynchronous Operations and Advanced Structure (Hard - 10 Exercises)

Focus: Closures, Promises, Async/Await, Immutability/Deep Copy, this context. **These mimic complex state and data flow in React.**

#	Concept	Section 9 Scenario
21.	**Rest Parameters**	Create a function logMissionDebrief(agent, ...notes) that collects an arbitrary number of debriefing notes into a single array parameter.
22.	**Closure (Counter)**	Create a factory function createTachikomaCounter() that returns an inner function. The inner function, when called, should increment and return a private Tachikoma deployment count, demonstrating a **closure**.
23.	**Shallow vs. Deep Copy**	Given a nested missionData object, use the spread operator (...) to create a shallow copy. Then, demonstrate a change to a nested property in the copy *still* affects the original. Use JSON.parse(JSON.stringify(obj)) to create a deep copy and show that changes are isolated.
24.	**Promise (Basic)**	Create a new Promise called hologramCheck that resolves with the string "Hologram stable" if a random number is greater than 0.5, otherwise it rejects with "Hologram glitch".
25.	**Promise Chaining**	Chain the hologramCheck promise with a .then() to log the success message, and a .catch() to log the error message.
26.	**Async/Await**	Write an async function runTacticalDeployment() that uses await to wait for the resolution of a simulated data fetch function (fetchTargetInfo()) before proceeding to the next step (logging a "Data processed" message).
27.	**Error Handling (Async)**	Refactor the runTacticalDeployment function to use a try...catch block to gracefully handle rejection from fetchTargetInfo().
28.	**Module Exports (Conceptual)**	Define a function getPublicSecurityFeed() in a conceptual file and use the export default syntax. Show how to import it into a main file using a named import.
29.	**this Context (Function vs. Arrow)**	Define a simple class CyberneticAgent with a property name and a regular method logName(). Then, define a property that is an arrow function logNameArrow = () => {...}. Call both to show how this behaves differently.
30.	**Class Definition (Context)**	Define a class Tachikoma with a constructor that accepts serialNumber and a method speak(). Create an instance and call the method.

Note: For exercises 23, 28, and 29, the implementation should be clearly demonstrated as if separate files were being used or within conceptual code blocks to illustrate the concept. The deeper understanding of **Immutability** (23) and **Async/Await** (26/27) are the final steps before true mastery of React's data flow.