JavaScript Zero → Hero Challenge (Part 1: Bank App, Day 1–10)

General Instructions

- Pure JavaScript only – no HTML, no CSS, no frameworks.
- Use functions, arrays, and objects (no class).
- Each day's work should be in its own folder:
- Example: day01/, day02/, etc.
- Inside each folder, create your .js files (e.g., bank.js).
- Push all code to a single GitHub repo, one branch only.
- Name your functions clearly. Keep your code readable.
- Use console.log() to test and display results.
- By Day 10, you should have a mini banking system with transactions, transfers, reports, and analytics.

Day 1: JavaScript Basics

- Create a new file bank.js.
- Declare a variable for the bank name.
- Create an empty customers array to hold customer records.
- Create an empty transactions array to log all transactions.
- Write a function that takes a customer name and returns a greeting.
- Write a function that adds two numbers and returns the result.
- Write a function that subtracts one number from another.

Day 2: Arrays & Objects

- Add 3 sample customers as objects with id, name, and balance.
- Write a function to list all customer names.
- Write a function to find and return a customer by their ID.
- Create a transactions array where each transaction has {id, customerId, type, amount}.
- Write a function to add a new transaction to this array.
- Use a loop to print all transactions.
- Write a function to count how many transactions exist.

Day 3: Core Banking Functions

- Write a function to deposit money into a customer's account.

- Write a function to withdraw money from a customer's account (only if enough funds exist).
- Update the transactions list whenever a deposit or withdrawal occurs.
- Write a function to return a customer's current balance.
- Add a validation rule: deposits must be greater than 0.
- Add a validation rule: withdrawals must not exceed balance.
- Write a function to return all transactions for a given customer.

Day 4: Loops & Conditionals

- Use a loop to print the balance of all customers.
- Write a function to calculate the total deposits for a customer.
- Write a function to calculate the total withdrawals for a customer.
- Write a function to calculate the net balance using deposits – withdrawals.
- Add a bonus: give ■500 to customers with a balance above ■10,000.
- Write a function to return all customer balances in an array.
- Write a function to check if a customer is "Rich" (> ■10,000) or "Poor".

Day 5: Building Customer Helpers

- Write a function to create a new customer object with name and balance.
- Add this customer to the customers array.
- Write a helper function to generate unique IDs.
- Ensure transactions also use unique IDs.
- Write a function to delete a customer by ID.
- Write a function that gives a full summary of one customer (balance, total deposits, withdrawals).
- Write a function that calculates the bank's total balance (sum of all customers).

Day 6: Advanced Transactions

- Write a function to get the last transaction of a customer.
- Write a function to get the largest deposit a customer has made.
- Write a function to get the largest withdrawal a customer has made.
- Write a function to count how many transactions a customer has.
- Write a function to check if a customer is in overdraft (withdrew more than deposited).
- Write a function to return all unique transaction types for a customer.
- Write a function to calculate the average transaction amount for a customer.

Day 7: Error Handling & Validation

- Ensure customer IDs exist before running deposit/withdraw functions.
- Throw an error if a deposit or withdrawal amount is not a number.
- Throw an error if the amount is ≤ 0.
- Prevent duplicate customer IDs.
- Write a safe version of withdraw that returns “success” or “fail” instead of crashing.
- Test the deposit function with invalid values (like strings).
- Test the withdraw function with edge cases (too much money, zero, etc.).

Day 8: Refactoring & Helpers

- Write a helper function to find a customer’s index in the array.
- Replace repeated code with calls to helper functions.
- Extract amount validation into its own function.
- Extract transaction-logging into a reusable function.
- Update deposit/withdraw to reuse helpers.
- Rename any unclear function names.
- Add documentation comments above each function to explain its role.

Day 9: Algorithms in Banking (12 Tasks)

- Sort all customers by balance (ascending).
- Sort all customers by balance (descending).
- Write a function to search customers by name (ignore case).
- Find the richest customer.
- Find the poorest customer.
- Return the top N richest customers.
- Calculate the average balance across all customers.
- Find the customer with the most transactions.
- Find the customer with the largest single deposit.
- Find the customer with the largest single withdrawal.
- Group customers into “Rich” (> ■10k) and “Poor”.
- Write a function to produce a customer leaderboard ranked by balance.

Day 10: Finalizing the Bank System (12 Tasks, Hard)

- Write a transfer(fromId, toId, amount) function to move money.
- Validate transfers (both users must exist, sender must have enough funds).
- Log transfers as two transactions (transfer-out, transfer-in).
- Write a function applyInterest(rate) that increases all balances by a given percentage.

- Write a function getBankReport() summarizing total deposits, withdrawals, and balances.
- Write a function to get inactive customers (those with no transactions).
- Write a function to get the top N most active customers by transaction count.
- Write a function to get all debt customers (balance < 0).
- Write a function simulateMonthlyCharge(amount) that deducts a fixed fee from everyone.
- Write a function to reverse a transaction by its ID.
- Optimize: write one loop that calculates both deposits and withdrawals for all customers.
- Write a README explaining how the system works, with examples.