JS Assignment Questions

Instructions

- Create a new GitHub repository named `js-coding-challenges`.
- Create a separate folder for each topic, using the topic name as the folder name (e.g., `functional-programming`).
- Solve each question in its own file, using filenames like `q1.js`, `q2.js,` etc.
- Use clear, descriptive function names. Export functions when necessary or when multiple files depend on each other.
- Commit after completing each question with a clear message explaining what was solved. This helps us track your progress and effort.
  Example: `git commit -m "Solved q1: Implemented array flatten function"`
- Submit your GitHub repository link once all questions are completed.
- Deadline: November 25, 2025

JavaScript Basics

Q1. Type Checker

Write a function `checkTypes(values)` that takes an array of mixed values and returns an array of their types using `typeof`.
Input: `[1, "a", true, null, undefined]`
Output: `["number", "string", "boolean", "object", "undefined"]`

Q2. Template Literal Formatter

Write a function `formatUser(name, age)` that returns: `"User <name> is <age> years old"`
Use template literals.

Q3. Truthy/Falsy Filter

Write a function `filterTruthy(arr)` that returns only truthy values. Do not use `Boolean()` —use an `if` condition.

Q4. Safe Division

Write a function `divide(a, b)` that returns `"Invalid"` if division is not possible (e.g., dividing by 0 or passing non-numbers). Use explicit type conversion for input validation.

Q5. Variable Scope Demo

Inside a function, declare variables using `var`, `let` and `const`. Log their accessibility inside and outside blocks.
Expected: Demonstrate block scope vs. function scope.

Control Flow & Functions

Q1. Grade Calculator

Write `getGrade(score)` using `if/else`:

90–100 → "A"

80–89 → "B"

...
Return `"Invalid"` for out-of-range values.

Q2. Day Name (Switch)

Write `getDayName(num)` using `switch-case`.
Input: `3` → Output: `"Wednesday"`
Include one intentional fall-through case.

Q3. Reverse Array (Using `for...of`)

Write `reverseArray(arr)` that builds a reversed array without using the built-in `reverse()` method. Use `for...of`.

Q4. Sum All (Rest Parameters)

Write `sumAll(...nums)` that returns the sum of unlimited input values.

Q5. Counter (Closure)

Write a function `createCounter()` that returns:

`increment()` → increments count

`get()` → returns current count

Must use closure, not global variables.

Arrays, Objects & JSON

Q1. Array Editor

Given an array, perform these operations in order:

`push(10)`

`unshift(5)`

`pop()`

Splice at index 1, remove 1 item
Return the final array.

Q2. Deep Access

Write `getCity(user)` that returns `user.address.current.city`. If any part is missing, return `"Unknown"`.

Q3. Dynamic Keys

Write `createUser(id, name, age)` that returns:

```
\{
 user_id: id,
 user_name: name,
 user_age: age
\}
```

Use computed property names.

Q4. JSON Converter

Write `safeParse(jsonStr)` that returns parsed JSON or `"Invalid JSON"` using `try/catch`.

Q5. Shallow vs. Deep Copy

Write code that:

Makes a shallow copy of an object using spread

Deep copies using JSON methods

Modifies both copies
Log which one affects the original.

Asynchronous JavaScript

Q1. Delayed Greeting

Write `delayedHello()` that logs `"Hello"` after 2 seconds using `setTimeout`.

Q2. Interval Counter

Write code that logs numbers 1–5 every second using `setInterval`. Stop automatically after reaching 5.

Q3. Promise Calculator

Write `addAsync(a, b)` that returns a Promise resolving with the sum in 1 second.

Q4. Parallel Fetch

Fetch these two URLs in parallel using `Promise.all()` and return a combined array:

https://jsonplaceholder.typicode.com/users

https://jsonplaceholder.typicode.com/posts

Q5. Async/Await Fetch

Write `getUser(id)` using `async/await` to fetch user data. Wrap in `try/catch`. If fetch fails, return `"Error"`.

Functional Programming

Q1. Square All

Write `squareAll(arr)` using `map()` only.
Input: `[1, 2, 3]` → Output: `[1, 4, 9]`

Q2. Filter Adults

Write `filterAdults(users)` using `filter()`, where age ≥ 18.
Input: `\[\{name:"A", age:17\}, \{name:"B", age:20\}\]`
Output: `\[\{name:"B", age:20\}\]`

Q3. Count Words

Given array: `\["apple", "banana", "apple"\]`
Write `countWords(arr)` using `reduce()` to return:
`\{ apple: 2, banana: 1 \}`

Q4. Find By ID

Write `findById(arr, id)` using `find()` only.

Q5. Chained Transformation

Given: `\[1, 2, 3, 4\]`
Use `map` → `filter` → `reduce` to compute the sum of squares of even numbers.
Should result in: `(2² + 4²) = 20`

ES6 Modules & npm

Q1. Named Export Calculator

Create a file `math.js` exporting `add()` and `sub()` (named exports). Import them in `main.js` and use them.

Q2. Default Export

Create `greet.js` exporting a default function that returns `"Hello <name>"`. Import it as `g` in `main.js`.

Q3. Namespace Import

Create `utils.js` with functions and import them using:
`import \* as utils from './utils.js'`

Q4. npm Script Runner

Inside `package.json`, create a script:

`"start": "node index.js"`

Write `index.js` that prints: `"App started"`.

Q5. Using the Installed Package

Install any npm package (like `lodash`). Write code using one method (e.g., `_.chunk()`).

Destructuring, Spread, Rest

Q1. Array Destructure

Write a function that takes an array of 3 numbers and returns:

First element

Third element

Default value = 0 if missing

Q2. Object Destructure & Rename

Input:

`\{ firstName: "John", address: \{ city: "Kathmandu" \} \}`

Extract `firstName` as `name` and nested `city`.

Q3. Rest Parameters Sum

Write `sum(...nums)` that returns the sum of unlimited numbers.

Q4. Merge Config

Given:

`defaultConfig = { dark: false, lang: "en" }`
`userConfig = { dark: true }`

Merge using spread, where user overrides default.

Q5. Safe Access

Given a nested object, safely access `user.profile.avatar.url` using optional chaining, and fall back to `"No Avatar"` using nullish coalescing.

Errors, Debugging & Storage

Q1. Throw Custom Error

Write `validateAge(age)` that throws `InvalidAgeError` if age < 0.

Q2. Async Error Handler

Write an async function that fetches from an invalid URL, catches the error, and returns `"Failed"`.

Q3. Debugger Demo

Write a function where inserting `debugger` helps inspect loop execution.

Q4. Storage Setter

Store the object `{theme:"dark", logged:true}` in:

`localStorage`

`sessionStorage`

Convert to/from JSON properly.

Q5. Cookie Creator

Write code to set a cookie `"token=abc123"` that expires in 24 hours.

REST API

Q1. Fetch Users

Write `getUsers()` that fetches users from JSONPlaceholder and returns an array of names.

Q2. Create Post

Write `createPost(title, body)` using POST with `fetch()`. Return the created post.

Q3. Auth Fetch

Write a fetch request that adds:

`Authorization: "Bearer 12345"`

Print the response JSON.

Q4. Parallel Fetch Merge

Fetch `/posts` and `/comments` in parallel using `Promise.all()` and return:

`{ posts: [...], comments: [...] }`

Q5. Paginated Loader

Fetch:

`/posts?_page=1` then `/posts?_page=2`

Combine results into one array.