Modern Intro to JavaScript — Lab Pack

Chris Minnick

Version 1.0.0

These labs accompany the “Modern Intro to JavaScript” course. Each lab is designed to take ~30 minutes and pairs with the preceding lecture block.

Table of Contents

Lab 01: Command Line + VS Code Warm-Up

Objectives

- Use the integrated terminal in VS Code confidently.
- Navigate folders, create files, and take Markdown notes.

Instructions

1. Open VS Code → Terminal → New Terminal (Ctrl+`` ` `` / Cmd+`` ` ``).
2. Navigate with `cd`, list files with `ls` (Windows: `dir`).
3. Create `labs/day1/` and inside it create `index.html` and `notes.md`.
4. Put a minimal HTML skeleton in `index.html` and open in a browser.
5. In `notes.md`, record at least five commands you ran and what they do.
6. Practice split view in VS Code and repeat a command history item.

Deliverables

- `labs/day1/notes.md` with at least five commands + short environment verification note.

Tips

- Keep notes concise; include one thing you learned or got stuck on.

Lab 02: JavaScript Language Warm-Up

Objectives

- Practice variables, strings, arrays, and objects.
- Use the console for quick feedback.

Instructions

1. Create `labs/day1/js-basics/index.html` and `labs/day1/js-basics/main.js`.
2. Link `main.js` with `<script type="module" src="./main.js"></script>`.
3. Declare several variables using `let` and `const`; `console.log` them.
4. Create an array of favorite foods and use `.map()` to uppercase each.
5. Build an object for yourself (name, role, interests).
6. Write a template-literal greeting function and a small arrow function (e.g., `add(a,b)`).
7. Open DevTools Console to verify outputs.

Deliverables

- `index.html` and `main.js` that log the requested values clearly.

Tips

- Prefer descriptive variable names; keep functions pure and small.

Lab 03: Create a Vite App

Objectives

- Scaffold and run a Vite (vanilla) project.
- Observe HMR and produce a build.

Instructions

1. Run the scaffold: `npm create vite@latest my-vite-app -- --template vanilla`.
2. `cd my-vite-app && npm install`.
3. Start the dev server: `npm run dev` and open the shown URL.
4. Edit `main.js` to log a new message or change DOM text. Observe HMR.
5. Build for production: `npm run build`; explore `dist/`.
6. Preview the build: `npm run preview`.

Deliverables

- A working Vite app demonstrating HMR and a successful production build.

Tips

- If imports fail, ensure `<script type="module">` and correct `./` paths.

Lab 04: Language Drills

Objectives

- Apply modern syntax for arrays, objects, and logical operators.
- Practice chaining and destructuring.

Instructions

1. Create `labs/day1/syntax-drills/main.js`.
2. Define an array of user objects (name, age, city).
3. Use `.filter()` to find users over 25.
4. Apply `.map()` to format names with template literals.
5. Destructure one object into separate variables and log them.
6. Combine arrays with spread; clone an object and update one property.
7. Use `??`, `||`, and `&&` to handle missing values.
8. Log each step's result with a short explanatory comment.

Deliverables

- `main.js` that logs each transformation and explains the result.

Tips

- Avoid mutating arrays/objects; create copies when needed.

Lab 05: Code Quality Setup and Documentation Practice

Objectives

- Configure ESLint/Prettier and add JSDoc.
- Write a concise project README.

Instructions

1. Create `.eslintrc` with a modern JS configuration.
2. Add Prettier and enable format on save (VS Code settings).
3. Write a `README.md` describing project setup and conventions.
4. Add JSDoc to at least two functions in `main.js`.
5. Run `eslint --fix` and confirm uniform formatting.
6. Commit with a descriptive message.

Deliverables

- `.eslintrc`, `README.md`, and updated `main.js` with JSDoc.

Tips

- Keep rules pragmatic; document “why” in README when non-obvious.

Lab 06: Elements Panel Practice

Objectives

- Explore the DOM via DevTools and practice live style edits.

Instructions

1. Open `practice.html` in Chrome or Edge.
2. Inspect the DOM tree; hover to see layout highlighting.
3. Change a heading's color, font size, and margin in Styles.
4. Inspect the Box Model; modify padding/border.
5. Switch to responsive mode and adjust width.
6. Reload to confirm unsaved changes revert.

Deliverables

- `labs/day2/devtools-notes.md` with:
- CSS rules you changed
- Visual outcomes
- One accessibility or layout insight

Tips

- Note cascade and specificity as you inspect.

Lab 07: Debug a Countdown Bug

Objectives

- Find and fix a logic error using breakpoints and stepping.

Instructions

1. Open `countdown.html` and `countdown.js`.
2. Run the page; observe incorrect countdown behavior.
3. Set a breakpoint in the countdown loop (Sources panel).
4. Step through and inspect `timeRemaining`, `intervalId`.
5. Identify the off-by-one error; fix and test.
6. Log start and end times for validation.

Deliverables

- Fixed `countdown.js` and `labs/day2/countdown-fix.md` summarizing:
- Root cause
- Fix applied
- DevTools features used

Tips

- Use `debugger;` to pause precisely when needed.

Lab 08: Build a Contact Form

Objectives

- Use semantic markup and native validation.
- Capture input with FormData.

Instructions

1. Create `labs/day2/contact-form/index.html` and `main.js`.
2. Add fields: Name, Email, Message.
3. Add attributes: `required`, `type="email"`, `minlength` for message.
4. In `main.js`, add a `submit` listener; `preventDefault()`.
5. Use `FormData` to log field values.
6. Call `checkValidity()` and show custom messages if needed.

Deliverables

- A working form with validation and console logging.

Tips

- Use roles/status messages for accessible feedback.

Lab 09: Selector Scavenger Hunt

Objectives

- Align CSS selectors with DOM queries and dynamic class toggles.

Instructions

1. Create `labs/day2/selectors/index.html`, `style.css`, `main.js`.
2. Build a page with IDs, classes, and attribute-marked elements.
3. Apply distinctive styles for visibility.
4. In `main.js`, `querySelectorAll()` different selectors; log counts.
5. Toggle classes on click to change appearance.
6. Try `:has()` or `:nth-child()` if supported; note results.

Deliverables

- A small project showing selection, logging, and class toggles.

Tips

- Use DevTools to confirm style application.

Lab 10: Interactive Page (DOM & Events)

Objectives

- Build a small interactive list with event delegation.

Instructions

1. Create `labs/day2/dom-events/index.html` and `main.js`.
2. Add a heading, an “Add Item” button, and a `<ul>` container.
3. Implement an “add item” function on button click.
4. Add a single listener on the `<ul>` to handle remove actions.
5. Show a dynamic counter of total items.
6. Add keyboard accessibility (Enter triggers add).
7. Toggle a class for active items.

Deliverables

- A working interactive list app with delegation and accessibility.

Tips

- Use event.target and `closest()` to manage delegation.

Lab 11: Objects → Classes → Modules

Objectives

- Reinforce OOP and ESM in the browser.

Instructions

1. Create `labs/day2/modules-lab/main.js`, `formatter.js`, and `User.js`.
2. Move a utility into `formatter.js` and export functions.
3. Define a `User` class in `User.js` with `name`, `email`, and `greet()`; add a private `#id`.
4. In `main.js`, import both modules and log formatted greetings.
5. Use `export default` in one module, named exports in the other.
6. Run in the browser with `<script type="module">` and verify console output.

Deliverables

- A modular app demonstrating classes, encapsulation, and imports.

Tips

- Remember explicit file extensions in browser imports (e.g., `./User.js`).

Lab 12: Collections Workout

Objectives

- Normalize and clean a small array-of-objects dataset
- Dedupe values efficiently with `Set`
- Build a `Map` index for fast lookups
- Produce locale-aware sorted output with `Intl.Collator`

Instructions

1. Create `labs/day3/collections/collections.js`.
2. Start with an array of user-like objects (name, city, tags), including duplicates and inconsistent casing.
3. Normalize fields (trim, consistent casing) without mutating the original data.
4. Use `Set` to dedupe tag values; log unique tag count.
5. Build a `Map` keyed by `city` → array of users; log the index keys.
6. Sort user names with `Intl.Collator('en', { sensitivity: 'base' })` and print a final report.
7. Add 2–3 inline comments explaining key choices (immutability, copy-before-sort).

Deliverables

- `labs/day3/collections/collections.js` that logs intermediate and final results clearly.

Instructor Notes

- Remind students that `sort` mutates; prefer copying prior to sorting.
- Encourage descriptive logging to make transformations auditable.

Lab 13: Formatters & Validators

Objectives

- Practice `Intl.NumberFormat` and `Intl.DateTimeFormat`.
- Validate and sanitize two input fields.
- Show accessible feedback messages.

Instructions

1. Create `labs/day3/formatters/index.html` and `labs/day3/formatters/formatters.js`.
2. In JS, write helpers `formatCurrency(amount, locale)` and `formatDate(date, locale)`.
3. Validate two fields (e.g., price ≥ 0, date present) with small predicate functions.
4. Display success/error messages with role/status semantics in the HTML.
5. Log both raw values and formatted outputs to the console.

Deliverables

- A minimal demo page plus `formatters.js` showing locale-aware formatting and validation.

Instructor Notes

- Encourage clear separation of parsing/validation/formatting.
- Remind to use semantic roles for screen reader feedback.

Lab 14: Hardening a Feature

Objectives

- Add guard clauses and meaningful errors to a brittle function.
- Demonstrate improved behavior with simple checks.

Instructions

1. Create `labs/day3/hardening/hardened.js` with a starting function (e.g., formatting or parsing) that fails on edge cases.
2. Add input validation and throw custom errors with helpful messages.
3. Use `??` and `?.` where appropriate to supply safe defaults.
4. Log before/after outputs for at least three edge cases.
5. Summarize the changes in a short Markdown note.

Deliverables

- `hardened.js` and `labs/day3/hardening/notes.md` describing the root causes and fixes.

Instructor Notes

- Encourage explicit error messages that aid debugging.
- Ask students to articulate their guard strategy.

Lab 15: Refactor to Modules

Objectives

- Introduce modular structure and clear public APIs.
- Separate data access, UI, and composition layers.

Instructions

1. Create `labs/day3/app/` with `api.js`, `ui.js`, `main.js`.
2. Move data-fetching or transformation logic into `api.js` (mock if needed).
3. Move DOM rendering helpers into `ui.js`.
4. In `main.js`, compose the two and initialize on DOMContentLoaded.
5. Export only the functions needed by other modules; keep the rest private.

Deliverables

- A working `app/` folder with a minimal page demonstrating modular wiring.

Instructor Notes

- Emphasize explicit imports/exports and avoiding implicit globals.

Lab 16: Promise Patterns

Objectives

- Practice converting callbacks to promises.
- Use combinators to coordinate multiple async operations.

Instructions

1. Create `labs/day3/promises/promises.js`.
2. Wrap `setTimeout`-style callbacks into a `delay(ms)` promise.
3. Chain 2–3 small async steps; log order of execution.
4. Use `Promise.allSettled` over 3 fetches (one should fail) and summarize results.
5. Add an example with `AbortController` to cancel a fetch.

Deliverables

- `promises.js` containing three self-contained demos with console output.

Instructor Notes

- Encourage explicit `.catch` and clear logging of success/failure paths.

Lab 17: async/await Drills

Objectives

- Refactor promise chains to `async/await`.
- Implement a timeout with `AbortController`.
- Manage loading/error/empty UI states.

Instructions

1. Create `labs/day3/async-demo/index.html` and `labs/day3/async-demo/async.js`.
2. Refactor an existing promise chain to `async/await` with try/catch.
3. Create a `withTimeout(promise, ms)` helper using `AbortController` for fetch.
4. Show a spinner while loading; handle error and empty states explicitly.
5. Log timings with `console.time` / `console.timeEnd`.

Deliverables

- Small demo app with a “Load” button, spinner, and robust error handling.

Instructor Notes

- Emphasize state transitions: idle → loading → success/error.

Lab 18: Data Fetcher

Objectives

- Fetch from a public API and render a small list.
- Handle loading, error, and empty states accessibly.

Instructions

1. Create `labs/day4/fetch-demo/index.html`, `main.js`, and `render.js`.
2. In `main.js`, wire a button to trigger a fetch.
3. In `render.js`, export `renderList(data)` and `renderState(state)` helpers.
4. Show loading → success/error/empty transitions in the DOM.
5. Log network timing and status; handle non-200 gracefully.

Deliverables

- `fetch-demo/` mini-app with resilient UI states.

Instructor Notes

- Encourage `try/catch` around `await fetch` and explicit status checks.
- Use ARIA live regions for announcing state changes.

Lab 19: Queryable List

Objectives

- Persist search/sort state in the URL.
- Keep UI and URL in sync on change/navigation.

Instructions

1. Create `labs/day4/query-list/index.html`, `filters.js`.
2. Build a basic list UI with search and sort inputs.
3. On input changes, update `URLSearchParams` and re-render.
4. On page load and `popstate`, read URL and restore UI state.
5. Ensure keyboard accessibility and announce results summary.

Deliverables

- `filters.js` and HTML with synchronized UI↔URL behavior.

Instructor Notes

- Emphasize handling back/forward reliably and accessibly.
- Keep query param names stable and documented.

Lab 20: SPA-Lite

Objectives

- Build a small SPA-lite with History API navigation.
- Persist user preferences in `localStorage`.

Instructions

1. Create `labs/day4/spa-lite/index.html`, `router.js`, `storage.js`, and `main.js`.
2. Implement a tiny router that switches views on path changes.
3. Store and load a preference (e.g., theme) from `localStorage`.
4. Manage focus on navigation; restore scroll when appropriate.
5. Verify back/forward behavior and deep links.

Deliverables

- `spa-lite/` with `router.js`, `storage.js`, and a working demo page.

Instructor Notes

- Keep router minimal; highlight ARIA/focus considerations.
- Avoid storing sensitive data in `localStorage`.

Lab 21: Lazy Images + Subtle Effects

Objectives

- Use Intersection Observer to lazy-load images.
- Add a subtle fade-in animation via requestAnimationFrame.
- Respect reduced motion preferences.

Instructions

1. Create `labs/day4/lazy/index.html`, `observer.js`, and add a few sample images (use placeholders if needed).
2. Mark images with `data-src` and a low-res placeholder or blank `src`.
3. In `observer.js`, observe images; when an image enters the viewport, swap `data-src` → `src`.
4. In the next animation frame, add a `loaded` class to trigger a CSS fade-in.
5. Detect `prefers-reduced-motion` and skip animations when enabled.
6. Verify performance impact using DevTools (Network throttling + Performance).

Deliverables

- `lazy/` mini-demo with lazy loading and accessible motion behavior.

Instructor Notes

- Encourage before/after measurements (first paint differences).
- Keep effects tasteful and short; prefer opacity over transforms for simplicity.

Lab 22: Prep the Final Project Skeleton

Objectives

- Scaffold the final project with clear module boundaries.
- Establish a minimal “home” view and wiring.

Instructions

1. Create `final/` with `api/`, `components/`, `styles/`, `utils/`, and `main.js`.
2. Stub `api/` with a sample function and mock data.
3. Create a simple `components/Card.js` and `components/List.js`.
4. Render a placeholder list in `main.js`; wire click navigation stubs.
5. Add a basic stylesheet and normalize default styles.

Deliverables

- Bootable skeleton with a placeholder home screen.

Instructor Notes

- Prioritize clear file names and small exported surfaces.
- Keep the DOM update functions pure where possible.

Lab 23: Debugging a Small App

Objectives

- Practice network throttling and offline modes.
- Capture logs and fix at least one bug.

Instructions

1. In the skeleton app, enable network throttling and simulate failures.
2. Capture logs with `console.group` and `console.table` to trace data flow.
3. Pause on exceptions and inspect call stacks.
4. Fix at least one bug; verify with a minimal repro.
5. Write a short bug report describing the root cause and fix.

Deliverables

- A commit with a fix and `final/BUGFIX.md` summarizing the issue.

Instructor Notes

- Encourage one change at a time and tight feedback loops.
- Save console output snippets in the bug report if helpful.

Lab 24: Capstone Part 1 — Fetch & Render (Data + UI Skeleton)

Objectives

- Build the fetch layer with retry/timeout wrappers.
- Establish the base list UI and state handling.

Instructions

1. In `final/`, add `api/fetcher.js` with `fetchJSON(url, { signal })` and retry-with-timeout.
2. Create `components/Skeleton.js` and `components/ErrorState.js` for UI states.
3. In `main.js`, load initial data, show skeletons, then render or error.
4. Log timings and failures; ensure the app remains interactive.

Deliverables

- Working list view that fetches and renders with robust state handling.

Instructor Notes

- Keep the API wrapper small and well-documented.
- Use `AbortController` to prevent stale updates on navigation.

Lab 25: Capstone Part 1 — Lists + Pagination

Objectives

- Add pagination or “Load more” controls.
- Maintain scroll/focus and shareable URLs.

Instructions

1. Add `components/Paginator.js` or a “Load more” control.
2. Implement page state in the URL (`page` param) and restore on load.
3. Append vs replace items; manage scroll and focus.
4. Handle empty last pages and disable controls appropriately.

Deliverables

- Paginated list with accessible controls and URL state.

Instructor Notes

- Announce page changes for screen readers.
- Ensure keyboard and touch targets are usable.

Lab 26: Capstone Part 2 — Routing & Storage

Objectives

- Implement detail routing and basic persistence.

Instructions

1. Add `api/detail.js` and a detail view component.
2. Push/replace state on navigation; handle `popstate`.
3. Persist theme and last filter to `localStorage` and restore on load.
4. Ensure focus and scroll behavior are accessible.

Deliverables

- Working detail view with URL routing and persisted preferences.

Instructor Notes

- Keep router and storage helpers small and reusable.
- Avoid storing sensitive data in `localStorage`.

Lab 27: Capstone Part 2 — Polish & Ship It

Objectives

- Run an accessibility and performance pass; prep for delivery.

Instructions

1. Fix at least one accessibility issue and one rendering or network performance issue.
2. Add a small theming token file (colors/spacing) and ensure consistent usage.
3. Create a production build and verify the preview server.
4. Write a README with features, decisions, and how to run.

Deliverables

- Final app with README describing features, decisions, and how to run.

Instructor Notes

- Keep the README concise and practical; include tradeoffs and future work.
- Capture before/after data for perf fixes when possible.