Modern Frontend Web Development - Labs

Complete Lab Instructions for Modern Frontend Web Development Course

📚 **Course Repository:**

https://github.com/chrisminnick/modern-frontend-web-dev

Version: 3.0.0

Date: September 2025

Author: Chris Minnick

Website: https://www.watzthis.com

Lab 01: Working with the Command Line in VSCode

Objectives

- Open and use the integrated terminal in Visual Studio Code.
- Practice basic command line navigation and file management.
- Build confidence with commands used later for Git, npm, and project setup.

Instructions

1. Open the Integrated Terminal

- VSCode → **Terminal > New Terminal**. The terminal opens at your workspace root.

2. Check Your Current Directory

```
pwd
```

3. List Files and Folders

```
ls
# Windows alternative:
dir
```

4. Change Directories

```
cd labs/lab01
ls
```

5. Create, View, Copy, Move, and Delete Files

```
# create
touch notes.txt

# view
cat notes.txt

# copy
cp notes.txt copy.txt

# move/rename
mv copy.txt renamed.txt

# make a directory
mkdir practice

# move files into a folder
mv notes.txt renamed.txt practice/

# delete a file
rm practice/renamed.txt
```

6. Go "Up" a Directory

```
cd ..
```

7. Tab Completion and Help

- Type `cat no` then press **Tab** to auto-complete `notes.txt` (if present).

- Use `--help` for command help:

```
ls --help
```

8. Understanding Paths and Command History

- Learn about absolute vs relative paths:

```
pwd                    # shows absolute path
cd ../..               # relative path (go up two directories)
cd /Users/yourname     # absolute path
```

- View command history:

```
history                # see previous commands
# Use up/down arrows to navigate through history
```

Discussion and Sharing

- Which commands were most useful?
- How does tab completion speed up your workflow?
- What's the difference between absolute and relative paths?
- Why do developers still rely on the command line with powerful GUIs available?

Lab 02: Using Visual Studio Code Basics

Objectives

- Create, open, and edit files in VSCode.
- Explore syntax highlighting, Emmet, and extensions.
- Learn Markdown syntax for documentation and note-taking.
- Customize VSCode for front-end development.

Instructions

1. Open Your Project Folder

- **File > Open Folder...** → choose `modern-frontend-web-dev` (or your workspace).

2. Create an HTML File

- In `labs/lab02/`, create `hello.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Hello VSCode</title>
  </head>
  <body>
    <h1>Hello, Visual Studio Code!</h1>
    <p>This is my first page created in VSCode.</p>
    <!-- Try Emmet below: type 'ul>li*3' and press Tab -->
  </body>
</html>
```

3. Preview with Live Server

- Right-click `hello.html` → **Open with Live Server**.

4. Use Emmet

- Inside `<body>`, type:

```
ul>li*3
```

- Press **Tab** to expand:

```
<ul>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

5. **Install & Use Extensions**

- Extensions panel → install **Prettier** and **ESLint**.

- Format document: **Right-click > Format Document** (Prettier).

6. **Open the Integrated Terminal**

```
ls    # or: dir
```

7. **Master Essential VSCode Features**

- Open Command Palette: **Cmd+Shift+P** (Mac) or **Ctrl+Shift+P** (Windows/Linux)

- Try these useful shortcuts:

- **Cmd+D** / **Ctrl+D**: Select next occurrence of current word

- **Cmd+Shift+L** / **Ctrl+Shift+L**: Select all occurrences

- **Cmd+/** / **Ctrl+/**: Toggle line comment

- **Alt+Shift+Down** / **Shift+Alt+Down**: Duplicate line

8. **Learn Markdown for Documentation**

- Install the **Markdown Preview Enhanced** extension

- Create `notes.md` in your `labs/lab02/` folder:

````
# My VSCode Learning Notes

## What I Learned Today

- How to use **Emmet** for faster HTML writing
- VSCode shortcuts that save time
- Installing and using extensions

## Useful Extensions

1. **Live Server** - for previewing HTML files
2. **Prettier** - for code formatting
3. **Markdown Preview Enhanced** - for viewing Markdown files

## Code Example

Here's some HTML I created with Emmet:

```html
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

## Next Steps

- [ ] Try more Emmet shortcuts
- [ ] Explore more VSCode extensions
- [ ] Practice keyboard shortcuts

> **Tip**: Use Markdown for all your project documentation!
````

```
- Right-click `notes.md` → **Open Preview to the Side**
- See your formatted Markdown in real-time!
```

9. Markdown Syntax Reference

Practice these common Markdown elements:

```
# Heading 1

## Heading 2

### Heading 3

**Bold text** and _italic text_

- Unordered list item
- Another item
  - Nested item

1. Ordered list item
2. Another ordered item

[Link text](https://example.com)

`inline code` and:

```javascript
// Code block
console.log('Hello, Markdown!');
```
```

> Blockquote for important notes

Column 1	Column 2
Data 1	Data 2

```
```

Discussion and Sharing

- How do syntax highlighting and Emmet improve productivity?
- Which keyboard shortcuts do you find most useful?
- How does the Command Palette speed up your workflow?
- Which extensions do you find most helpful?
- Why is editor mastery important early on?
- **Markdown Questions:**

- How could Markdown improve your project documentation?

- What advantages does Markdown have over plain text or Word documents?

- When would you use Markdown in web development projects?

Lab 03: Controlling Your Versions with Git

Objectives

- Understand the Git workflow: init, add, commit, branch, merge.
- Use branches to isolate work.
- (Optional) Push to GitHub.

Instructions

1. Verify Git

```
git --version
```

2. Initialize a Repo

```
cd labs/lab03
git init
```

3. Create & Commit a README

```
echo "# Lab 03: Controlling Your Versions with Git" > README.md
git status
git add README.md
git commit -m "Initial commit with README"
```

4. View History and Check Status

```
git status    # check current state (use this frequently!)
git log       # view commit history
git log --oneline  # compact view
# press 'q' to quit log view
```

5. Create and Use a Branch

```
git checkout -b feature-hello

# add a file
echo 'console.log("Hello, Git!");' > hello.js

git add hello.js
git commit -m "Add hello.js with greeting"
```

6. Merge to Main

```
# if 'main' doesn't exist yet, create/switch to it
git checkout -B main

# merge feature branch
git merge feature-hello

# delete the branch
git branch -d feature-hello
```

7. View Differences Between Commits

```
# See what changed between commits
git diff HEAD~1 HEAD    # compare last two commits
git diff                # see unstaged changes
git diff --staged       # see staged changes
```

8. (Optional) Push to GitHub

```
# create an empty repo on GitHub (no README)
git remote add origin https://github.com/YOUR-USERNAME/lab03.git
git push -u origin main
```

Discussion and Sharing

- How do branches protect `main` from unstable code?
- When should you make a new branch?
- How does `git diff` help you understand what changed?
- Why is `git status` considered essential for daily Git usage?
- Share your GitHub link if you pushed your repo.

Lab 04: Initializing npm

Objectives

- Initialize npm and explore `package.json`.
- Add scripts.
- Install and use a simple dependency.

Instructions

1. Initialize npm

```
cd labs/lab04
npm init -y
```

2. Set ESM (so you can use `import` in Node)

- Open `package.json` and add `"type": "module"` (if not present):

```
{
  "name": "lab04",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "echo \"Running dev server...\""
  }
}
```

3. Run a Script

```
npm run dev
```

4. Install a Package

```
npm install dayjs
```

5. Use the Package

- Create `index.js`:

```
import dayjs from 'dayjs';

console.log('The current date and time is:', dayjs().format());
```

- Run:

```
node index.js
```

6. Understanding Dependencies and Versioning

```
# View dependency tree
npm ls

# Check package versions
cat package.json | grep dayjs
```

- Learn about semantic versioning (semver):

- `1.2.3` = MAJOR.MINOR.PATCH

- `^1.2.3` = compatible within major version

- `~1.2.3` = compatible within minor version

Discussion and Sharing

- Why is `package.json` important?
- How does npm simplify sharing a project?
- What's the difference between `^` and `~` in version ranges?
- How does semantic versioning help prevent breaking changes?
- What other scripts might you add later?

Lab 05: Using npm

Objectives

- Install, update, and remove packages.
- Distinguish local vs global installs.
- Use `npx` to run tools without global installs.

Instructions

1. Verify npm

```
npm -v
```

2. Local Install (lodash)

```
cd labs/lab05
npm init -y
# ensure ESM
npm pkg set type=module

npm install lodash
```

3. Use lodash

- Create `index.js`:

```
import _ from 'lodash';

const numbers = [10, 20, 30, 40, 50];
console.log('Shuffled numbers:', _.shuffle(numbers));
```

- Run:

```
node index.js
```

4. Dev Dependency (Prettier)

```
npm install --save-dev prettier
```

- Run with `npx`:

```
npx prettier --write index.js
```

5. Update & Remove Packages

```
# Check for outdated packages
npm outdated

# Update packages
npm update

# Remove packages
npm uninstall lodash
```

6. Global vs Local Installs

```
# global (system-wide) example
npm install -g http-server
http-server

# without global install
npx http-server
```

Discussion and Sharing

- Why prefer local installs for project dependencies?
- When would a global install be appropriate?
- How does `npm outdated` help maintain your project?
- What's the difference between `^1.2.3` and `~1.2.3` version ranges?
- Pros/cons of `npx` vs global tools?

Lab 06: Creating a New Project with Vite

Objectives

- Scaffold a modern project with Vite.
- Use Vite's dev server and HMR.
- Explore project structure.

Instructions

1. Create the Project

```
cd labs/lab06
npm create vite@latest my-vite-app
# Choose: Framework = Vanilla, Variant = JavaScript
cd my-vite-app
npm install
```

2. Explore Files

- `index.html`, `main.js`, `style.css`, `package.json`

3. Run the Dev Server

```
npm run dev
```

- Open the shown URL (e.g., `http://localhost:5173/`).

4. Edit for HMR

- Open `main.js` and replace content:

```
document.querySelector('#app').innerHTML = `
  <h1>Hello Vite!</h1>
  <p>This is my first project using Vite 🚀</p>
`;
```

5. Build for Production (Optional)

```
npm run build
npm run preview
```

Discussion and Sharing

- How does HMR speed up development?
- What's the benefit of separate dev and build scripts?

Lab 07: Using Chrome Developer Tools – Elements Panel

Objectives

- Inspect and modify HTML/CSS live with DevTools.
- Understand temporary edits vs saved files.

Instructions

1. Create Files in `labs/lab07/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>DevTools Lab</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Welcome to Modern Front-End Web Development</h1>
    <p>This is a practice page for exploring Chrome DevTools.</p>
  </body>
</html>
```

`style.css`:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  padding: 20px;
}
h1 {
  color: darkblue;
}
p {
  font-size: 18px;
  color: gray;
}
```

2. Open with Live Server

- Right-click `index.html` → **Open with Live Server**.

3. **Open DevTools**

- **F12** or **Cmd+Opt+I / Ctrl+Shift+I**, then select **Elements**.

4. **Edit HTML & CSS in DevTools**

- Right-click `<h1>` → **Edit as HTML**, change text.

- In **Styles** panel, change `color` to `crimson` and add:

```
text-transform: uppercase;
```

5. **Reload to Reset**

- Refresh to revert changes (they're temporary).

Discussion and Sharing

- How does live editing speed up troubleshooting?
- Why don't DevTools edits persist?

Lab 08: Using Chrome Developer Tools – Sources Panel (JavaScript Debugging)

Objectives

- Set breakpoints and step through code.
- Inspect variables and call stack.
- Use the Console alongside the debugger.

Instructions

1. Create Files in `labs/lab08/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Countdown Bug</title>
  </head>
  <body>
    <h1>JavaScript Rocket Launcher</h1>
    <button id="launch">Start the Countdown!</button>
    <p>Time remaining: <span id="timer">10</span></p>
    <script src="script.js"></script>
  </body>
</html>
```

`script.js` (with a bug):

```
const launchBtn = document.getElementById('launch');
const timerEl = document.getElementById('timer');

function startTimer(start = 10) {
  let timer = start;
  const id = setInterval(() => {
    // BUG: we decrement before checking, which skips 0 properly
    timer--; // <-- suspect line
    timerEl.textContent = timer; // shows updated time
    if (timer === 0) {
      // may never hit as expected
      clearInterval(id);
      console.log('🚀 Lift off!');
    }
  }, 1000);
}

function init() {
  launchBtn.addEventListener('click', startTimer);
}

init();
```

2. **Open DevTools → Sources**

- Set a line-of-code breakpoint on the `timer--` line.

- Click **Start the Countdown!**; execution pauses.

3. **Step Through**

- Use **Step Over / Step Into / Step Out** buttons.

- Observe **Scope** variables; run quick tests in **Console**:

```
timer; // what is its value now?
```

4. **Fix In DevTools**

- Edit code (temporarily) in Sources to check first, then decrement:

```
if (timer === 0) {
  clearInterval(id);
  console.log('🚀 Lift off!');
}
timer--;
```

- Resume and verify behavior, then apply the same fix in `script.js` using VSCode and save.

5. **Advanced Debugging Features**

- Try conditional breakpoints: right-click on line number, set condition like `timer < 5`

- Add watch expressions in the **Watch** panel to monitor variables

- Explore the **Call Stack** to understand function execution flow

Discussion and Sharing

- How do breakpoints compare to `console.log` debugging?
- When would you step **into** vs **over**?
- How do conditional breakpoints improve debugging efficiency?
- What's the benefit of watch expressions over console logging?

Lab 09: Creating an HTML Form

Objectives

- Create an accessible HTML5 form.
- Use semantic labels and input types.
- Leverage built-in validation attributes.

Instructions

1. Create Files in `labs/lab09/`

`index.html` :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Contact Us</title>
    <link rel="stylesheet" href="css/main.css" />
  </head>
  <body>
    <h1>Contact Us</h1>
    <p>Please fill out the following form and we'll get back to you.</p>

    <form action="#" method="post" novalidate>
      <div>
        <label for="contact-name">Your Name</label>
        <input
          id="contact-name"
          name="contact-name"
          type="text"
          placeholder="Enter your name"
          required
        />
      </div>

      <div>
        <label for="contact-phone">Phone Number</label>
        <input
          id="contact-phone"
          name="contact-phone"
          type="tel"
          placeholder="(555) 555-5555"
        />
      </div>

      <div>
        <label for="contact-email">Email</label>
        <input
          id="contact-email"
          name="contact-email"
          type="email"
          placeholder="you@example.com"
          required
        />
      </div>

      <div>
        <label for="contact-message">Message</label>
        <textarea
          id="contact-message"
          name="contact-message"
          rows="5"
          placeholder="How can we help?"
          required
        ></textarea>
      </div>

      <div>
        <label for="callback-time">Preferred Callback Time</label>
```

```
        <select id="callback-time" name="callback-time">
          <option value="">Select a time</option>
          <option value="morning">Morning</option>
          <option value="afternoon">Afternoon</option>
          <option value="evening">Evening</option>
        </select>
      </div>

      <div>
        <input type="submit" value="Submit" />
      </div>
    </form>
  </body>
</html>
```

`css/main.css` (optional starter):

```
body {
  font-family: system-ui, sans-serif;
  padding: 1.5rem;
}
form {
  display: grid;
  gap: 1rem;
  max-width: 480px;
}
label {
  display: block;
  font-weight: 600;
  margin-bottom: 0.25rem;
}
input,
textarea,
select {
  width: 100%;
  padding: 0.5rem;
}
input[type='submit'] {
  width: auto;
  cursor: pointer;
}
```

2. **Open in Browser and Test Validation**

- Try submitting with required fields empty.

Discussion and Sharing

- Which HTML attributes improved accessibility?
- What validation do you get "for free" from HTML5?

Lab 10: Using CSS Selectors

Objectives

- Practice type, class, ID, attribute, and pseudo selectors.
- Style a page without changing its HTML.

Instructions

1. Create Files in `labs/lab10/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Selectors Practice</title>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <header>
      <h1 id="title">A Modest Practice</h1>
      <h3 class="subhead">by Jonathan Swift (1729)</h3>
      <h3 id="pubdate">1729</h3>
      <hr />
    </header>

    <main>
      <p>First paragraph. This one will be larger.</p>
      <p>Subsequent paragraphs should be indented.</p>
      <p>
        Another paragraph with a
        <a href="https://example.com" target="_blank">link</a>.
      </p>
      <ul class="items">
        <li>Alpha</li>
        <li>Beta</li>
        <li>Gamma</li>
      </ul>
    </main>
  </body>
</html>
```

`css/style.css`:

```
/* element selector */
p {
  font-family: sans-serif;
}

/* class selector */
.subhead {
  font-style: italic;
}

/* multiple selectors */
h1,
h2,
h3 {
  text-align: center;
}

/* id selector */
#pubdate {
  background-color: gray;
  color: white;
  width: 300px;
  margin-left: auto;
  margin-right: auto;
}

/* pseudo-class selector */
p:first-of-type {
  font-size: 24px;
}

/* indent subsequent paragraphs */
p:not(:first-of-type) {
  text-indent: 2em;
}

/* attribute selector */
a[target='_blank'] {
  text-decoration: underline;
}

/* pseudo-element example: initial cap */
p:first-of-type::first-letter {
  font-size: 200%;
  font-weight: bold;
}

/* center the hr */
hr {
  width: 50%;
  margin: 1rem auto;
}

/* Advanced selectors - nth-child examples */
.items li:nth-child(odd) {
  background-color: #f0f0f0;
}
```

```
.items li:nth-child(2) {
  font-weight: bold;
  color: darkblue;
}

/* Sibling combinators */
h1 + h3 {
  margin-top: -10px; /* reduce space after h1 if followed by h3 */
}

h3 ~ hr {
  border-color: darkblue; /* style hr that comes after any h3 */
}

/* Modern :has() selector (where supported) */
header:has(#pubdate) {
  border: 2px solid #ccc;
  padding: 1rem;
  border-radius: 8px;
}
```

2. **Open with Live Server and Verify Styles**

3. **Update HTML to demonstrate advanced selectors**

- Update the list items to show nth-child effects:

```
<ul class="items">
  <li>Alpha (odd - should be gray)</li>
  <li>Beta (even, 2nd child - should be bold blue)</li>
  <li>Gamma (odd - should be gray)</li>
  <li>Delta (even)</li>
  <li>Epsilon (odd - should be gray)</li>
</ul>
```

Discussion and Sharing

- Which selectors reduced the most HTML changes?
- How do pseudo-classes and pseudo-elements differ?
- What's the advantage of `:nth-child()` over adding classes?
- How do sibling combinators (`+` and `~`) help with styling relationships?
- What browser support considerations exist for modern selectors like `:has()`?

Lab 11: Positioning with CSS (and Flexbox)

Objectives

- Understand normal flow, relative/absolute/fixed positioning.
- Build a simple layout with Flexbox.

Instructions

1. Create Files in `labs/lab11/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Positioning & Flexbox</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <nav class="nav">My Fixed Nav</nav>

    <section class="stage">
      <img
        src="https://via.placeholder.com/150"
        alt="astronaut"
        class="pic pic-1"
      />
      <img
        src="https://via.placeholder.com/120"
        alt="dinosaur"
        class="pic pic-2"
      />
      <p class="blurb">This is a paragraph of text.</p>
      <div class="box">This is a div</div>
    </section>

    <div class="sticky-nav">Sticky Navigation - Scroll to see it stick!</div>

    <section class="flex-demo">
      <div class="card">Card 1 (Flexbox)</div>
      <div class="card">Card 2 (Flexbox)</div>
      <div class="card">Card 3 (Flexbox)</div>
    </section>

    <section class="grid-demo">
      <div class="grid-item">Grid Item 1</div>
      <div class="grid-item">Grid Item 2</div>
      <div class="grid-item">Grid Item 3</div>
      <div class="grid-item">Grid Item 4</div>
    </section>

    <!-- Add some content to make scrolling meaningful -->
    <div
      style="height: 1000px; padding: 2rem; background: linear-gradient(to botto
    >
      <h2>Scroll down to see the sticky navigation in action!</h2>
      <p>
        This section has extra height to demonstrate the sticky positioning.
      </p>
    </div>
  </body>
</html>
```

`style.css`:

```
body {
  margin: 0;
  font-family: system-ui, sans-serif;
}

/* fixed positioning */
.nav {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  height: 56px;
  background: #222;
  color: #fff;
  display: flex;
  align-items: center;
  padding: 0 1rem;
  z-index: 10;
}

.stage {
  margin-top: 72px;
  position: relative;
  min-height: 400px;
}

/* relative and absolute positioning */
.pic {
  position: absolute;
  left: 50px;
}
.pic-1 {
  top: 50px;
}
.pic-2 {
  top: 140px;
  left: 220px;
}

.blurb {
  background: yellow;
  width: 220px;
  height: 100px;
  position: relative;
  top: 240px;
  left: 370px;
  padding: 0.5rem;
}

.box {
  background: red;
  width: 200px;
  height: 200px;
  position: relative;
  top: 150px;
  left: 40px;
  color: #fff;
  display: grid;
```

```
  place-items: center;
}

/* Flexbox layout */
.flex-demo {
  display: flex;
  gap: 1rem;
  padding: 1rem;
  margin: 2rem 1rem;
  border-top: 1px solid #ccc;
  flex-wrap: wrap;
  justify-content: center;
}

.card {
  flex: 1 1 220px;
  min-height: 120px;
  background: #f5f5f5;
  border: 1px solid #ddd;
  border-radius: 0.5rem;
  display: grid;
  place-items: center;
}

/* CSS Grid example */
.grid-demo {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 1rem;
  padding: 1rem;
  margin: 2rem 1rem;
  border-top: 2px solid #333;
}

.grid-item {
  background: linear-gradient(45deg, #667eea, #764ba2);
  color: white;
  padding: 2rem;
  border-radius: 0.5rem;
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100px;
}

/* Sticky positioning example */
.sticky-nav {
  position: sticky;
  top: 56px; /* below the fixed nav */
  background: rgba(255, 255, 255, 0.9);
  backdrop-filter: blur(10px);
  padding: 0.5rem 1rem;
  border-bottom: 1px solid #ddd;
  z-index: 5;
}

@media (max-width: 600px) {
  .stage {
    min-height: 480px;
```

```
  }
}
```

2. **Experiment**

- Adjust `top/left` on absolutely positioned elements.

- Observe how fixed nav stays during scroll.

- Resize browser to see Flexbox responsiveness.

- Scroll down to see sticky navigation behavior.

- Compare CSS Grid vs Flexbox layouts.

Discussion and Sharing

- When is absolute positioning appropriate?
- Why is Flexbox often preferred for modern layout?
- How does CSS Grid differ from Flexbox, and when would you use each?
- What's the difference between `position: sticky` and `position: fixed`?
- How does `backdrop-filter` enhance modern UI design?

Lab 12: Variables, Arrays, and Constants in JavaScript

Objectives

- Declare with `let` and `const`.
- Create and manipulate arrays.
- Understand mutating vs reassigning.

Instructions

1. Create Files in `labs/lab12/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Variables, Arrays, and Constants</title>
    <script src="main.js" defer></script>
  </head>
  <body>
    <h1>JavaScript Variables, Arrays, and Constants</h1>
    <p>Open the console to see output.</p>
  </body>
</html>
```

`main.js`:

```
// Declaring and initializing a variable
let userName = 'Alice';

// Declaring and initializing an array
let primeNumbers = [2, 3, 5, 7, 11, 13];

// Declaring and initializing a constant
const GRAVITY = 9.81;

// Using variables
console.log('The user name is:', userName);

// Accessing array elements
console.log('The first prime number is:', primeNumbers[0]);
console.log('The third prime number is:', primeNumbers[2]);

// Using a constant
console.log('The gravity constant is:', GRAVITY);

// Mutating vs reassigning
const pets = ['cat', 'dog'];
pets.push('parrot'); // OK (mutates array)
console.log(pets);
// pets = []                // NOT OK (reassigns the const)

// Modern JavaScript features
console.log('\n--- Modern JavaScript Features ---');

// Array destructuring
const [first, second, ...rest] = primeNumbers;
console.log('First prime:', first);
console.log('Second prime:', second);
console.log('Rest of primes:', rest);

// Object destructuring
const person = { name: 'Bob', age: 30, city: 'New York' };
const { name, age, city = 'Unknown' } = person;
console.log(`${name} is ${age} years old and lives in ${city}`);

// Spread operator with arrays
const morePrimes = [17, 19, 23];
const allPrimes = [...primeNumbers, ...morePrimes];
console.log('All primes:', allPrimes);

// Spread operator with objects
const extendedPerson = { ...person, occupation: 'Developer', age: 31 };
console.log('Extended person:', extendedPerson);
```

Discussion and Sharing

- When to use `let` vs `const`?
- Why is reassigning a `const` not allowed, but mutating objects/arrays is?
- How does destructuring make code more readable?
- What are the practical uses of the spread operator?
- How do these modern features improve upon older JavaScript patterns?

Lab 13: Using Chrome DevTools – JavaScript Console

Objectives

- Run JavaScript directly in the console.
- Read error messages and fix issues.
- Use `console.*` for debugging.

Instructions

1. Create Files in `labs/lab13/`

`index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Console Practice</title>
    <script src="script.js" defer></script>
  </head>
  <body>
    <h1>Console Practice</h1>
    <p>Open DevTools → Console</p>
  </body>
</html>
```

`script.js` (with an error to fix):

```
// Intentional error:
console.log(somethingiswrong);

// Some data to explore:
const people = [
  { name: 'Ada', role: 'Engineer' },
  { name: 'Grace', role: 'Scientist' },
  { name: 'Linus', role: 'Hacker' },
];

// Try in console:
// console.table(people);
```

2. Open Console and Read the Error

- You'll see `ReferenceError: somethingiswrong is not defined`.

3. Fix the Error

- Option A: define the variable:

```
const somethingiswrong = 'Nope, all good now!';
console.log(somethingiswrong);
```

- Option B: treat as string:

```
console.log('somethingiswrong');
```

- Option C: remove the line.

4. Explore Advanced Console Methods

```
// Basic console methods
console.log('Hello from console.log');
console.error('This is an error message');
console.warn('This is a warning');
console.info('This is an info message');

// Data display methods
console.table(people);
console.dir(people); // shows object structure

// Grouping methods
console.group('User Information');
console.log('Name: Alice');
console.log('Age: 25');
console.log('Role: Developer');
console.groupEnd();

// Timing methods
console.time('Performance Test');
// Simulate some work
for (let i = 0; i < 100000; i++) {
  Math.random();
}
console.timeEnd('Performance Test');

// Trace method
function functionA() {
  functionB();
}
function functionB() {
  console.trace('Call stack trace:');
}
functionA();
```

Discussion and Sharing

- How did the console help identify the exact problem?
- When is `console.table` handy?
- How do `console.group()` and `console.time()` improve debugging?
- What information does `console.trace()` provide?
- Which console methods are most useful for different debugging scenarios?

Lab 14: Using JavaScript Methods

Objectives

- Practice common string, array, and object methods.
- Solve a simple data transformation task.

Instructions

1. Create Files in `labs/lab14/`

`index.html` (optional if running in browser):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>JavaScript Methods</title>
    <script type="module" src="methods.js" defer></script>
  </head>
  <body>
    <h1>JavaScript Methods</h1>
    <p>Open the console for output.</p>
  </body>
</html>
```

`methods.js`:

```
// String methods
const title = 'modern front-end web development';
const proper = title
  .split(' ')
  .map((w) => w[0].toUpperCase() + w.slice(1))
  .join(' ');
console.log(proper); // "Modern Front-End Web Development"

// Array methods
const nums = [5, 12, 8, 130, 44];
const doubled = nums.map((n) => n * 2);
const large = nums.filter((n) => n > 10);
const sum = nums.reduce((acc, n) => acc + n, 0);

console.log({ doubled, large, sum });

// Object methods
const person = { name: 'Chris', role: 'Instructor', active: true };
console.log(Object.keys(person)); // ['name','role','active']
console.log(Object.values(person)); // ['Chris','Instructor',true]

// Small exercise: filter numbers > 10
function filterGreaterThanTen(arr) {
  return arr.filter((n) => n > 10);
}
console.log(filterGreaterThanTen(nums)); // [12,130,44]

// Additional useful array methods
console.log('\n--- More Array Methods ---');

// find() - returns first matching element
const firstLarge = nums.find((n) => n > 10);
console.log('First number > 10:', firstLarge); // 12

// some() - tests if any element passes the test
const hasLargeNumbers = nums.some((n) => n > 100);
console.log('Has numbers > 100:', hasLargeNumbers); // true

// every() - tests if all elements pass the test
const allPositive = nums.every((n) => n > 0);
console.log('All numbers positive:', allPositive); // true

// includes() - checks if array contains a value
const hasEight = nums.includes(8);
console.log('Contains 8:', hasEight); // true

// Method chaining example
const processedNums = nums
  .filter((n) => n > 10) // [12, 130, 44]
  .map((n) => n * 2) // [24, 260, 88]
  .sort((a, b) => a - b); // [24, 88, 260]
console.log('Processed numbers:', processedNums);

// Advanced string methods
console.log('\n--- Advanced String Methods ---');
const sentence = 'The quick brown fox jumps over the lazy dog';
console.log('Original:', sentence);
console.log('Starts with "The":', sentence.startsWith('The'));
```

```
console.log('Ends with "dog":', sentence.endsWith('dog'));
console.log('Includes "fox":', sentence.includes('fox'));
console.log('Repeated:', '🚀'.repeat(5));
```

Discussion and Sharing

- Which array method (map/filter/reduce) felt most useful?
- How do built-in methods reduce boilerplate loops?
- What's the difference between `find()` and `filter()`?
- When would you use `some()` vs `every()`?
- How does method chaining improve code readability?

Lab 15: Using JavaScript Objects

Objectives

- Learn how to create and use JavaScript objects.
- Understand key–value pairs and property access.
- Add methods to objects and use `this`.

Instructions

1. Create a file `objects.js`.

2. Define and log an object:

```
const person = {
  name: 'Alice',
  age: 25,
  greet: function () {
    console.log(`Hi, I'm ${this.name} and I'm ${this.age} years old.`);
  },
};

console.log(person.name);
console.log(person['age']);
person.greet();
```

3. Add a new property:

```
person.job = 'Web Developer';
console.log(person);
```

4. Loop through object properties:

```
for (let key in person) {
  console.log(key, person[key]);
}
```

5. **Modern Object Features** (explore these additional features):

Object destructuring:

```
const { name, age } = person;
console.log(`Destructured: ${name} is ${age} years old`);
```

Shorthand property syntax:

```
const city = 'New York';
const country = 'USA';
const location = { city, country }; // same as { city: city, country: country }
```

Computed property names:

```
const propName = 'dynamicProperty';
const dynamicObj = {
  [propName]: 'This property name was computed!',
};
```

Compare arrow functions vs regular functions as methods:

```
const modernPerson = {
  name: 'Bob',
  greet: function () {
    console.log(`Hello, I'm ${this.name}`); // 'this' works
  },
  arrowGreet: () => {
    console.log(`Arrow: ${this?.name || 'undefined'}`); // 'this' is different!
  },
};
```

6. **ES6 Classes** (modern alternative to object literals):

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hi from class! I'm ${this.name}`);
  }

  static species() {
    return 'Homo sapiens';
  }
}

const classPerson = new Person('Charlie', 35);
classPerson.greet();
```

Discussion and Sharing

- What's the difference between dot and bracket notation?
- Why is `this` useful inside methods?
- How do objects help organize data?
- **Enhanced Questions:**

- How does object destructuring make code more readable?

- When would you use computed property names?

- What's the difference between arrow functions and regular functions as object methods?

- How do ES6 classes compare to object literals for creating multiple similar objects?

Lab 16: Performing DOM Manipulation

Objectives

- Learn how to select and update DOM elements with JavaScript.
- Respond to user events like clicks.
- Compare DOM manipulation with jQuery and React.

Instructions

1. Create `index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>DOM Lab</title>
  </head>
  <body>
    <h1 id="title">Original Title</h1>
    <button id="btn">Click Me</button>
    <p id="message"></p>
    <script src="main.js"></script>
  </body>
</html>
```

2. Create `main.js`:

```
// Modern DOM selection methods (preferred over getElementById)
const title = document.querySelector('#title');
const btn = document.querySelector('#btn');
const msg = document.querySelector('#message');

btn.addEventListener('click', () => {
  title.textContent = 'You clicked the button!';
  msg.textContent = 'DOM updated successfully 🎉';
});
```

3. **Advanced DOM Features** (explore these enhancements):

Dynamic element creation:

```
function addNewSection() {
  const newSection = document.createElement('section');
  newSection.innerHTML = `
    <h2>Dynamically Created Section</h2>
    <p>This section was created with JavaScript!</p>
    <button class="remove-btn">Remove This Section</button>
  `;

  // Add styling
  newSection.style.cssText = `
    background: linear-gradient(45deg, #ff6b6b, #4ecdc4);
    color: white;
    padding: 1rem;
    margin: 1rem 0;
    border-radius: 8px;
  `;

  document.body.appendChild(newSection);

  // Add event listener to remove button
  const removeBtn = newSection.querySelector('.remove-btn');
  removeBtn.addEventListener('click', () => {
    newSection.remove(); // Modern way to remove elements
  });
}

// Create button to add new sections
const createBtn = document.createElement('button');
createBtn.textContent = 'Create New Section';
createBtn.addEventListener('click', addNewSection);
document.body.appendChild(createBtn);
```

Event delegation:

```
document.body.addEventListener('click', (event) => {
  if (event.target.matches('.remove-btn')) {
    console.log('Remove button clicked via delegation!');
  }
});
```

4. **Comparison with jQuery and React**:

- **jQuery**:

```
  $('#btn').click(() => {
    $('#title').text('Updated with jQuery');
  });
```

- **React (conceptual)**: React updates the DOM indirectly using components.

Discussion and Sharing

- How does DOM manipulation make a page interactive?
- Why is direct DOM manipulation less common in large apps?
- How do libraries like React change the process?
- **Enhanced Questions:**

- What's the difference between `querySelector` and `getElementById`?

- When would you use event delegation instead of individual event listeners?

- How does dynamic element creation compare to template-based approaches?

- What are the performance implications of frequent DOM manipulation?

Lab 17: Building a Movie Review Webpage with jQuery

Objectives

- Learn the basics of jQuery for DOM manipulation.
- Build a simple interactive page with dynamic content.
- Compare jQuery with vanilla JavaScript solutions.

Instructions

1. Create `index.html`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Movie Reviews</title>
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
  </head>
  <body>
    <h1>Movie Reviews</h1>
    <input id="movie" placeholder="Enter movie name" />
    <input id="review" placeholder="Enter your review" />
    <button id="add">Add Review</button>
    <ul id="reviews"></ul>
    <script src="app.js"></script>
  </body>
</html>
```

2. Create `app.js`:

```
let reviewCounter = 0;

$('#add').click(() => {
  const movie = $('#movie').val().trim();
  const review = $('#review').val().trim();

  if (movie && review) {
    reviewCounter++;
    const reviewItem = $(`
      <li data-id="${reviewCounter}" style="margin-bottom: 10px; padding: 10px;
        <strong>${movie}:</strong> ${review}
        <button class="delete-btn" style="float: right; background: #dc3545; col
          Delete
        </button>
      </li>
    `);

    $('#reviews').append(reviewItem);
    $('#movie').val('');
    $('#review').val('');
    $('#movie').focus();

    // Show success message
    $('body').append(
      '<div id="success-msg" style="position: fixed; top: 20px; right: 20px; bac
    );
    setTimeout(
      () => $('#success-msg').fadeOut(() => $('#success-msg').remove()),
      2000
    );
  } else {
    alert('Please enter both movie name and review!');
  }
});

// Delete functionality using event delegation
$('#reviews').on('click', '.delete-btn', function () {
  if (confirm('Are you sure you want to delete this review?')) {
    $(this)
      .parent()
      .fadeOut(() => $(this).parent().remove());
  }
});

// Enter key support
$('#movie, #review').keypress((e) => {
  if (e.which === 13) {
    // Enter key
    $('#add').click();
  }
});
```

3. **Vanilla JavaScript Comparison** (explore the modern alternative):

```
// Modern vanilla JS equivalent:
const addBtn = document.querySelector('#add');
const movieInput = document.querySelector('#movie');
const reviewInput = document.querySelector('#review');
const reviewsList = document.querySelector('#reviews');
let reviewCounter = 0;

addBtn.addEventListener('click', () => {
  const movie = movieInput.value.trim();
  const review = reviewInput.value.trim();

  if (movie && review) {
    reviewCounter++;
    const li = document.createElement('li');
    li.innerHTML = `<strong>${movie}:</strong> ${review}
      <button class="delete-btn">Delete</button>`;

    reviewsList.appendChild(li);
    movieInput.value = '';
    reviewInput.value = '';
    movieInput.focus();
  }
});

// Event delegation for delete buttons
reviewsList.addEventListener('click', (e) => {
  if (e.target.classList.contains('delete-btn')) {
    e.target.parentElement.remove();
  }
});
```

Discussion and Sharing

- Why was jQuery so popular historically?
- How would you do this today with vanilla JS or React?
- What benefits remain for using jQuery in 2025?
- **Enhanced Questions:**

- What are the advantages of event delegation over individual event listeners?

- How does jQuery's chaining syntax compare to vanilla JavaScript?

- When might you still choose jQuery over vanilla JavaScript in modern development?

- How do the file sizes and performance compare between jQuery and vanilla JS solutions?

Lab 18 (Final Project): Interactive Web App with Fetch

Objectives

- Apply all skills from the course: HTML5, CSS3, JavaScript, DOM, and Fetch API.
- Build a comprehensive interactive app that pulls data from a public API.
- Demonstrate mastery of responsive design, error handling, and modern JavaScript.

Instructions

Complete project files are available in `labs/final-project/`

1. **Create the HTML Structure** (`index.html`):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Random User Generator</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="container">
      <header>
        <h1>🎲 Random User Generator</h1>
        <p>Generate random user profiles from around the world!</p>
      </header>
      <main>
        <div class="controls">
          <button id="load" class="load-btn">
            <span class="btn-text">Load Random User</span>
            <span class="loading" style="display: none;">Loading...</span>
          </button>
          <div class="options">
            <label>
              <input type="checkbox" id="multiple" />
              Load 5 users at once
            </label>
            <select id="nationality">
              <option value="">Any nationality</option>
              <option value="us">United States</option>
              <option value="gb">United Kingdom</option>
              <option value="ca">Canada</option>
              <option value="au">Australia</option>
            </select>
          </div>
        </div>
        <div id="output" class="output">
          <div class="placeholder">
            <p>
              👆 Click the button above to generate your first random user!
            </p>
          </div>
        </div>
      </main>
    </div>
    <script src="main.js"></script>
  </body>
</html>
```

2. **Add Modern CSS Styling** (`style.css`):

- Responsive grid layout using CSS Grid and Flexbox

- Gradient backgrounds and smooth animations

- Mobile-first responsive design

- Loading states and hover effects

3. **Implement JavaScript Functionality** (`main.js`):

```
const loadBtn = document.getElementById('load');
const output = document.getElementById('output');
const multipleCheckbox = document.getElementById('multiple');
const nationalitySelect = document.getElementById('nationality');

async function loadUsers() {
  try {
    setLoadingState(true);

    const isMultiple = multipleCheckbox.checked;
    const results = isMultiple ? 5 : 1;
    const nationality = nationalitySelect.value;

    let apiUrl = `https://randomuser.me/api/?results=${results}`;
    if (nationality) apiUrl += `&nat=${nationality}`;

    const response = await fetch(apiUrl);
    if (!response.ok)
      throw new Error(`HTTP error! status: ${response.status}`);

    const data = await response.json();
    displayUsers(data.results);
  } catch (error) {
    showError('Failed to load users. Please try again.');
  } finally {
    setLoadingState(false);
  }
}

function displayUsers(users) {
  // Dynamic DOM manipulation to create user cards
  // (Complete implementation in final-project/main.js)
}

loadBtn.addEventListener('click', loadUsers);
```

4. **Test and Run**:

- Use Live Server or run `python -m http.server` in the project directory

- Test responsive design in different screen sizes

- Verify error handling by disconnecting internet

5. **Enhanced Features Included**:

- Loading animations and user feedback

- Error handling for network failures

- Responsive design for mobile and desktop
- Keyboard navigation support
- Multiple user generation option
- Nationality filtering
- Modern CSS animations and effects

Skills Demonstrated

This project showcases mastery of all course concepts:

- **Labs 1-3**: Project structure, version control ready
- **Labs 4-6**: Modern development workflow concepts
- **Labs 7-8**: Debug-ready code with proper error handling
- **Lab 9**: Form controls and user input handling
- **Lab 10**: Advanced CSS selectors and styling
- **Lab 11**: Responsive layout with Grid and Flexbox
- **Lab 12**: Modern JavaScript variable handling
- **Lab 13**: Console logging and debugging
- **Lab 14**: Array methods and data transformation
- **Lab 15**: Object manipulation and destructuring
- **Lab 16**: Dynamic DOM manipulation and events
- **Lab 17**: Modern vanilla JavaScript (no jQuery needed)

Discussion and Sharing

- Which part of the course helped you the most in completing the project?
- How did using Fetch and async/await simplify working with data?
- What challenges did you face with responsive design?
- How does error handling improve user experience?
- What modern JavaScript features made development easier?
- How might you expand this project further?

Extension Ideas

- Add user favorites with localStorage
- Implement data visualization
- Add more filtering options
- Include user search functionality
- Add print/export features
- Implement accessibility improvements