SOFTWARE DEVELOPMENT AND ENGINEERING PRACTICES

Modern JavaScript (ES6+)

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

JavaScript changed more from ES6 onward than in the fifteen years before it, and the ecosystem changed with it: every modern framework, tutorial, and codebase now assumes the new language. Developers who learned older JavaScript, or who picked the language up in fragments on the job, end up reading code full of arrow functions, destructuring, modules, and async/await that they can use but not fully reason about. That gap surfaces at the worst time: while debugging.

This is a hands-on, practitioner course. It rebuilds your JavaScript on a modern foundation in dependency order: first the core language improvements around scope, functions, and data, then how modern code is organized into modules, and finally asynchronous programming, the area where fuzzy understanding costs the most. Rather than list every feature added since 2015, it goes deep on the ones you will read and write every day. Every module ends with a lab and builds on the one before.

Who Should Attend

- Developers who know older JavaScript and need to modernize
- Developers coming from another language (C#, Java, Python) who need real JavaScript fluency
- Front-end or back-end developers preparing for framework work in React, Node.js, or similar

Learners who have never programmed should take *Introduction to Programming* first.

Prerequisites

- Working proficiency in some programming language, ideally including some JavaScript
- Basic HTML and how a browser loads a script
- Comfortable with a code editor and the browser developer console

What You Will Learn

- Apply modern scoping and declarations: let, const, and why var went away
- Use arrow functions, default parameters, and template literals fluently, including how this behaves
- Work with objects and arrays the modern way: destructuring, spread, and the key array methods
- Organize code with ES modules: import, export, and structuring a real project
- Explain the event loop and write asynchronous code with promises and async/await
- Fetch data from APIs with proper error handling, and navigate the modern tooling ecosystem

Course Outline

Day one: the modern language core

- Modern Foundations
  - let, const, and block scope: what var got wrong
  - Template literals and default parameters
  - Arrow functions, and the honest story about this

  - Lab: modernize a legacy script and fix the scoping bug hiding in it
- Working with Data: Objects and Arrays
  - Destructuring objects and arrays, in assignments and parameters
  - Spread and rest: copying, merging, and gathering
  - The array methods that replaced loops: map, filter, reduce, and find
  - Lab: transform a realistic dataset using destructuring and array methods, no for loops allowed
- Functions, Objects, and Classes
  - Closures: the concept behind half of modern JavaScript patterns
  - Enhanced object literals and optional chaining
  - Classes: what they are, and what they are underneath
  - Lab: build a small library of composable functions and a class that uses them

Day two: modules, async, and the real world

- ES Modules
  - import and export: named, default, and how to choose
  - Structuring a multi-file project with modules
  - Modules in the browser and in Node.js, and where bundlers fit
  - Lab: refactor day one's code into a clean multi-module project
- Asynchronous JavaScript
  - The event loop: why JavaScript never waits, and what that implies
  - Promises: creating, chaining, and handling failure
  - async/await: asynchronous code you can actually read
  - Lab: convert callback-based code to promises and then to async/await, handling every error path
- Talking to APIs and Modern Practice
  - fetch: requests, responses, JSON, and status handling
  - Coordinating multiple async operations: Promise.all and friends
  - The modern toolchain at a glance: npm, bundlers, and where TypeScript fits
  - Lab: build a small app that fetches live API data and renders it, with loading and error states

Extended Version

The three-day version keeps the same gradient and adds depth on the harder corners:

- Iterators, generators, and the machinery behind the language's syntax
- Deeper asynchronous patterns: cancellation, retries, and taming concurrent requests
- Tooling in practice: npm scripts, a bundler, linting, and formatting on a real project
- A capstone: build a complete API-driven mini-application from empty folder to working app (a natural bridge to *TypeScript for JavaScript Developers*)