🚀 Roadmap JavaScript Avanzado - Versión Híbrida Completa

Semana 3-4: Asincronía y Fundamentos Avanzados

1. Asincronía Avanzada

Duración: 2h

- **1.1 Throttle**
  - Implementación desde cero
  - Casos de uso: scroll events, API calls frecuentes
- **1.2 Debounce**
  - Diferencias con throttle
  - Aplicación en search inputs y resize events
- **1.3 Event Loop Profundo**
  - Call Stack, Microtasks, Macrotasks
  - Orden de ejecución: Promise vs setTimeout
- **1.4 Promesas Avanzadas**
  - `Promise.allSettled()`, `Promise.any()`
  - Cancelación con `AbortController`
- **1.5 Async Iterators**
  - `for await...of` loops
  - Crear async generators

Por qué: Dominar asincronía es fundamental en JS moderno. Estos patrones optimizan rendimiento y UX en aplicaciones reales.

2. Prototipos y Herencia (Fundamentos que SÍ importan)

Duración: 2h

- **2.1 Prototipos vs Clases**
  - Cómo funcionan las clases internamente
  - Cuándo usar cada uno
- **2.2 Herencia Prototipal**
  - `Object.create()`, `__proto__`

- Cadena de prototipos

- **2.3 Mixins**
  - Composición sobre herencia
  - Patrones de múltiple herencia

Por qué: Las clases son syntax sugar. Entender prototipos te ayuda a debuggear mejor y comprender librerías que los usan.

Semana 5-6: Programación Funcional y Memoria

3. Métodos Funcionales Avanzados

Duración: 2h

- **3.1 Inmutabilidad**
  - `Object.freeze()`, `immer.js` conceptos
  - Structural sharing básico
- **3.2 Transducers**
  - Transformaciones eficientes de datos
  - Composición de transformaciones
- **3.3 Currying y Partial Application**
  - Implementación manual
  - Casos de uso prácticos

Por qué: Te hace escribir código más predecible, testeable y performante. Especialmente útil en apps con mucho manejo de datos.

4. Gestión de Memoria

Duración: 1.5h

- **4.1 Garbage Collection**
  - Cómo V8 maneja la memoria
  - Generational GC concepts
- **4.2 Memory Leaks**
  - Closures problemáticos
  - Event listeners no removidos
  - Referencias circulares

Por qué: Memory leaks destruyen la performance de apps complejas. Saber identificarlos y prevenirlos es crucial para desarrollo profesional.

Semana 7-8: Metaprogramación y Rendimiento

5. Metaprogramación Básica

Duración: 2h

- **5.1 Proxy**
  - Trampas para objetos (get, set, has)
  - Crear APIs reactivas básicas
- **5.2 Reflect API**
  - Complemento perfecto para Proxy
  - Métodos de introspección
- **5.3 Symbols**
  - Private properties con symbols
  - Well-known symbols

Por qué: Proxy es la base de frameworks como Vue.js. Te permite crear abstracciones poderosas y APIs más elegantes.

6. Rendimiento Crítico

Duración: 2h

- **6.1 Optimización de Bucles**
  - Técnicas de micro-optimización
  - Profiling con DevTools
- **6.2 Web Workers**
  - Offloading CPU-intensive tasks
  - Comunicación con main thread
- **6.3 JIT Compilation**
  - Cómo V8 optimiza tu código
  - Escribir código "JIT-friendly"

Por qué: Las apps modernas manejan grandes cantidades de datos. Web Workers evitan bloquear la UI y entender JIT te ayuda a escribir código más rápido.

Semana 9-10: Módulos y Arquitectura

7. Módulos y Estándares

Duración: 2h

- **7.1 ESM vs CommonJS**
  - `import/export` vs `require`
  - Interoperabilidad entre sistemas
- **7.2 Dynamic Imports**
  - `import()` function
  - Code splitting manual
- **7.3 Module Resolution**
  - Cómo Node.js resuelve módulos
  - Package.json fields importantes

Por qué: Los módulos son la base de aplicaciones escalables. Dynamic imports permiten optimizar el bundle size cargando código solo cuando se necesita.

8. Design Patterns Esenciales

Duración: 1.5h

- **8.1 Module Pattern**
  - IIFE y namespacing
  - Revealing module pattern
- **8.2 Observer Pattern**
  - Custom events system
  - Publisher/Subscriber básico
- **8.3 Factory Pattern**
  - Crear objetos sin new
  - Configuración flexible

Por qué: Los patrones son soluciones probadas. Te ayudan a estructurar código de manera profesional y facilitan comunicación con el equipo.

Semana 11-12: Testing y Herramientas Modernas

9. Testing Fundamentals

Duración: 2h

- **9.1 Unit Testing con Jest**
  - Test suites y assertions
  - Setup y teardown
- **9.2 Mocking y Spies**
  - Mock functions y modules
  - Testing dependencies
- **9.3 Testing Async Code**
  - Promises en tests
  - Async/await patterns

Por qué: Testing no es opcional en desarrollo profesional. Te da confianza para refactorizar y asegura que el código funcione como esperas.

10. Build Tools y Workflow

Duración: 1.5h

- **10.1 Bundlers Básicos**
  - Webpack vs Vite conceptos
  - Entry points y outputs
- **10.2 Linting y Formatting**
  - ESLint configuration
  - Prettier integration
- **10.3 Git Hooks**
  - Pre-commit hooks
  - Automated testing

Por qué: Las herramientas modernas aumentan productividad enormemente. Un buen setup te permite enfocarte en resolver problemas en lugar de configurar.

11. TypeScript Fundamentals

Duración: 2h

- **11.1 Tipos Básicos**
  - Primitives, arrays, objects

    - Type annotations vs inference
- **11.2 Interfaces y Types**
    - Cuándo usar cada uno
    - Extending y composition
- **11.3 Migración Gradual**
    - De .js a .ts paso a paso
    - Configuración básica

Por qué: TypeScript es estándar en la industria. Te ayuda a detectar errores en desarrollo y hace el código más autodocumentado.

Semana 13-14: Proyecto Final Integrador

12. Proyecto: Task Manager Avanzado

Duración: 8h total

Características a implementar:

- **Backend simulation con Web Workers**
- **Debounced search con async data**
- **Memory-efficient data handling**
- **Proxy-based reactive system**
- **Modular architecture**
- **Comprehensive testing**
- **TypeScript migration**

Estructura del proyecto:

```
src/
├── core/           # Proxy-based reactivity
├── workers/        # Background processing
├── utils/          # Debounce, throttle, etc.
├── patterns/       # Observer, Factory implementations
├── tests/          # Jest test suites
└── types/          # TypeScript definitions
```

Por qué: Un proyecto integrador consolida todo el conocimiento. Te da algo concreto para mostrar en entrevistas y portafolio.

📊 Distribución de Tiempo Total

- **Fundamentos Avanzados:** 4h (Async + Prototipos)
- **Programación Funcional:** 3.5h (Funcional + Memoria)
- **Metaprogramación:** 4h (Proxy + Performance)
- **Arquitectura:** 3.5h (Módulos + Patterns)
- **Herramientas:** 5.5h (Testing + Build + TS)
- **Proyecto Final:** 8h
- **Total:** ~28 horas (14 semanas × 2h promedio)

🎯 Hitos de Evaluación

Semana 4: Implementar debounce/throttle desde cero **Semana 6:** Crear un sistema de herencia con mixins **Semana 8:** Construir un mini-framework reactivo con Proxy **Semana 10:** Configurar pipeline completo de testing **Semana 12:** Migrar un proyecto existente a TypeScript **Semana 14:** Presentar proyecto final completo

📚 Recursos Recomendados

- **Libros:** "You Don't Know JS" (Kyle Simpson)
- **Práctica:** Leetcode para algoritmos funcionales
- **Comunidad:** JavaScript subreddit, Discord servers
- **Tools:** Chrome DevTools, VS Code extensions
- **Proyectos:** Contribuir a librerías open source pequeñas

🚀 Siguiente Nivel (Post-Roadmap)

Una vez completado, estarás listo para:

- **Frameworks avanzados** (React, Vue, Angular arquitectura)
- **Node.js backend** development
- **Performance optimization** especializada
- **Micro-frontends** architecture
- **Contribution** a proyectos open source grandes