Código Facilito
Bootcamp Desarrollo Frontend Moderno

Resumen de la Clase: Ejercicios prácticos de JavaScript y Typescript

🎯 Introducción

La clase se centró en el uso de herramientas esenciales para el desarrollo frontend moderno, con énfasis en JavaScript, TypeScript y el manejo de dependencias con NPM. Se abordaron conceptos clave como scopes, tipos de datos, interfaces y enums, además de prácticas recomendadas para la gestión de proyectos.

🧰 Herramientas necesarias

- **Editor de código**: Visual Studio Code (u otro equivalente)
- **Terminal**: PowerShell, Terminal de Windows o integrada en VS Code
- **Node.js**: Verificar instalación con `node -v`
- **GitHub**: Crear cuenta para futuras clases
- **TypeScript**: Instalación global con `npm install -g typescript`

🧠 Temas tratados

1. Template Literals

- Permiten interpolar variables dentro de strings usando backticks `` ` `` y `${variable}`.
- Más legibles que la concatenación con `+`.
- Útiles para saltos de línea y textos dinámicos.

2. Scopes en JavaScript

- **Var**: Tiene scope de función. Permite redeclaración, lo que puede causar errores.
- **Let**: Tiene scope de bloque. No permite redeclaración en el mismo bloque.
- **Const**: Scope de bloque. No permite reasignación ni redeclaración.

3. TypeScript

- Superset de JavaScript creado por Microsoft.
- Permite tipado estático y mejora la calidad del código.
- Se compila a JavaScript usando `tsc archivo.ts`.

4. Tipos de datos en TypeScript

- `string`, `number`, `boolean`, `array`, `any`, `union types`
- Tipado explícito vs. implícito

- Validación en tiempo de compilación

5. Funciones tipadas

- Parámetros obligatorios y opcionales (`?`)
- Valores por defecto
- Tipado de retorno

6. Enums

- Definen conjuntos de valores específicos (ej. `Dispositivo.Web`, `Dispositivo.Mobile`)
- Útiles para representar estados o categorías fijas

7. Interfaces

- Describen la forma de un objeto
- Permiten reutilización y validación estructural
- Ejemplo: `interface Gato { nombre: string; edad: number; colores: string[] }`

📦 NPM y manejo de dependencias

Inicialización de proyecto

- `npm init` → crea `package.json`
- Scripts personalizados con `npm run nombre_script`

Instalación de dependencias

- `npm install typescript` → agrega a `node_modules`
- `package-lock.json` → registra dependencias exactas
- `node_modules` → carpeta que no se sube al repositorio

Versionado semántico

- Formato: `major.minor.patch`
    - `major`: cambios incompatibles
    - `minor`: nuevas funcionalidades
    - `patch`: correcciones menores

📚 Conclusiones y próximos pasos

- Consolidar el uso de TypeScript para mejorar la calidad del código
- Practicar la creación de scripts en `package.json`
- Familiarizarse con el uso de `npm`, `node_modules` y `package-lock.json`
- Prepararse para integrar Git en el flujo de trabajo en la próxima clase

📖 Glosario técnico (opcional)

- **Scope**: Ámbito donde una variable es accesible
- **Template Literal**: String que permite interpolación de variables
- **Enum**: Tipo de dato con valores predefinidos
- **Interface**: Estructura que define la forma de un objeto
- **NPM**: Gestor de paquetes para Node.js
- **TypeScript**: Lenguaje que extiende JavaScript con tipado estático