1. Introducción breve

Ejercicio 1: Convertir 3 funciones de JavaScript a TypeScript añadiendo tipado.

Función 1: Suma de dos números

```
function sumar(a, b) {
  return a + b;
}
```

Función 2: Filtrar elementos mayores que un valor

```
function filtrarMayores(arr, limite) {
  return arr.filter(item => item > limite);
}
```

Función 3: Obtener propiedad de un objeto

```
function obtenerPropiedad(obj, clave) {
  return obj[clave];
}
```

Ejercicio 2: Detectar y corregir errores de tipado en ejemplos dados.

- let edad: number = "30";
- const activo: Boolean = new Boolean(false);
- const numeros: number[] = [1, 2, "3"]; // "3" es string
- let coordenada: [number, number] = [40.4168]; // Falta segundo elemento
  coordenada = [40.4168, "3.7038"]; // Orden y tipo incorrectos

Ejercicio 3: Explicar en 2 frases qué aporta TypeScript frente a JavaScript.

2. Tipos básicos y funciones

Ejercicio 1: Declarar variables con todos los tipos primitivos (string, number, boolean, null, undefined).

Ejercicio 2: Crear alias y literales (type Edad = number, type Rol = "admin" | "user").

Ejercicio 3: Escribir funciones con parámetros tipados y valores de retorno.

Ejercicio 4: Validar datos con funciones (ej. comprobar si una edad es válida).

Ejercicio 5: Practicar coerción y conversión explícita (parseInt, parseFloat).

Ejercicio 6: Crear una función que devuelva "Aprobado" o "Reprobado" según un booleano.

3. Estructuras de datos

Ejercicio 1: Crear arrays tipados y aplicar métodos (map, filter, reduce).

Ejercicio 2: Definir tuplas para coordenadas y respuestas de funciones.

Ejercicio 3: Crear enumeraciones (enum Rol, enum Estado) y usarlas en funciones.

Ejercicio 4: Implementar una función que muestre permisos según el rol.

Ejercicio 5: Simular un listado de usuarios con arrays y enums, y filtrar por rol.

4. Programación orientada a objetos

Ejercicio 1: Definir una clase Persona con atributos y métodos.

Ejercicio 2: Crear constructores con parámetros y valores por defecto.

Ejercicio 3: Usar modificadores de acceso (public, private, protected).

Ejercicio 4: Implementar herencia (Animal → Perro, Gato) y sobrescribir métodos.

Ejercicio 5: Crear una interfaz Mascota y aplicarla en varias clases.

Ejercicio 6: Definir métodos y propiedades estáticas (Usuario.totalUsuarios).

Ejercicio 7: Polimorfismo: usar una lista de Animal y recorrerla llamando a hablar().

5. Proyecto integrador guiado

Mini-aplicación de gestión de usuarios con Node.js y TypeScript

Paso 1: Crear proyecto con npm init -y y configurar tsconfig.json.

Paso 2: Definir clase Usuario con atributos nombre, edad, rol.

Paso 3: Crear enum Rol { Admin, Usuario, Invitado }.

Paso 4: Implementar un array de usuarios.

Paso 5: Funciones:

añadirUsuario(usuario: Usuario)

buscarUsuario(nombre: string)

mostrarUsuarios()

Paso 6: Probar en consola: añadir varios usuarios y mostrar resultados.