1. El Desafío de la Auditoría Empresarial (Objetos, Herencia, Prototipos y Arrays)

Este ejercicio combina la creación de objetos complejos, la implementación de herencia y el uso de funciones de arrays para análisis de datos.

Parte A: Definiendo la Jerarquía (Objetos y Herencia)

1. Define la siguiente jerarquía utilizando **tres métodos diferentes** de creación de objetos para cada nivel (literal, constructor y clase, aunque solo necesitas implementar una por nivel para la funcionalidad, debes describir cómo lo harías con las otras dos):

   ◦ **Nivel Base:** `Persona` (usando la sintaxis de **Clases** de ES6). Debe tener propiedades `nombre`, `edad` y `género`.

   ◦ Añade a la clase `Persona` un método llamado `obtenerDetalles()` que muestre por consola un mensaje del tipo “XXX tiene XX años”.

   ◦ **Nivel 2.1:** `Estudiante` (que hereda de `Persona`). Debe añadir `curso` y `grupo`.

   ◦ **Nivel 2.2:** `Profesor` (que hereda de `Persona`). Debe añadir `asignatura` y `nivel`.

2. **Extensión de Prototipos:** Al **prototipo** de `Estudiante` (o su equivalente en la clase), añade la función `obtenerTipo()` que muestre un mensaje del tipo “XXX tiene XX años y es estudiante”. Esta función debe **reutilizar** la función `obtenerDetalles()` de su prototipo padre para la primera parte del mensaje.

3. **Cálculo Derivado:** Dentro de la clase `Persona`, implementa una función que devuelva los años que le quedan para jubilarse, asumiendo que la edad de jubilación es 65 años. (Similar al Ejercicio 5 de tus fuentes).

Parte B: Análisis de la Plantilla (Arrays y Funciones de Iteración)

Crea un *array* llamado `plantilla` que contenga una mezcla de 10 objetos `Estudiante` y `Profesor`.

1. **Iteración y Mapeo:** Utiliza la función `map` para generar un nuevo *array* que contenga, para cada persona, su `nombre` y la cantidad de años que le quedan para jubilarse (usando el método de la Parte A).

2. **Filtrado:** Utiliza la función `filter` para crear un nuevo *array* que solo contenga a los `Profesores` de "Nivel Alto".

3. **Agrupación:** Utiliza `Object.groupBy` para agrupar toda la `plantilla` por su `género`.

4. **Comprobación Universal:** Utiliza la función `every` para verificar si **todos** los miembros de la `plantilla` tienen más de 18 años.

---

2. El Gestor de Seguridad y Fechas (Prototipos, Contraseñas y Fechas)

Este ejercicio se centra en extender los objetos nativos de JavaScript (`Array` y `Date`) mediante la manipulación de prototipos, y la creación de un objeto complejo para generar y validar contraseñas.

Parte A: Extensión de Objetos Nativos (Prototypes)

1. **Extensión de `Array`:** Añade al prototipo del objeto `Array` una función llamada `cuantasVecesEsta(numero)`. Esta función debe recibir un número y devolver cuántas veces se encuentra dicho número en el *array*. Pruébala con un *array* de prueba.
2. **Extensión de `Date`:** Añade al prototipo del objeto `Date` una función llamada `añoNuevo()`. Esta función debe devolver el número de días que quedan hasta el 1 de Enero del siguiente año, desde la fecha que se le pase como parámetro. (Recuerda trabajar con milisegundos).

Parte B: El Objeto Contraseña Segura (Lógica y Estructura)

Crea un objeto constructor (prototipo o clase) llamado `Contraseña`.

1. **Constructor:** El objeto debe recibir su `longitud` en el momento de su creación.
2. **Generación:** Implementa una función interna que genere aleatoriamente la contraseña, respetando la longitud indicada. La contraseña debe estar formada por letras (mayúsculas y minúsculas) y por números.
3. **Validación de Fortaleza:** Implementa una función llamada `esFuerte()` que devuelva `true` o `false`. Una contraseña es fuerte si cumple con las siguientes condiciones estrictas:
   - Tiene al menos 6 letras mayúsculas.
   - Tiene al menos 3 letras minúsculas.
   - Tiene al menos 2 números.
4. **Programa de Prueba:** Diseña un programa que, haciendo uso de este objeto, **genere 10 objetos `Contraseña` de tamaño aleatorio (entre 12 y 20 caracteres), genere la contraseña para cada uno de ellos, y por último nos diga si cada una de esas 10 contraseñas es fuerte o no lo es**.

---

3. Manejo Crítico de Arrays y Contexto ()

Este ejercicio se centra en la programación estructurada frente al uso de funciones propias de *Array*, manejo de secuencias y el entendimiento del contexto `this` en diferentes tipos de funciones.

Parte A: Arrays sin Ayuda vs. con Ayuda (Programación Estructurada)

Dado un *array* `tVector` de números enteros (ej.):

1. **Búsqueda de Mínimo:** Crea una función que devuelva el valor mínimo de `tVector` **sin utilizar funciones propias de array**. Luego, crea una segunda versión de la función **haciendo uso de alguna de las funciones de array**.
2. **Búsqueda de Índice:** Crea una función que reciba el *array* `tVector` y un valor `x`. Debe devolver la posición (índice) donde lo encuentre por primera vez, o `-1` si no lo encuentra. **No utilices funciones propias de array**. Luego, crea una segunda versión de la función **haciendo uso de alguna de las funciones de array** (por ejemplo, `includes` o `find`).
3. **Análisis de Secuencia:** Desarrolla una función que calcule la **longitud de la subsecuencia común más larga** (LCSL), es decir, la longitud de la mayor secuencia de elementos repetidos dentro del vector. Por ejemplo, en `(2, 5, 4, 2, 2, 5, 6, 6, 6, 3)` sería 3.

Parte B: El Contexto Crítico de (Funciones)

Define un objeto llamado `Calculadora` que tenga un *array* de números llamado `valores` y dos métodos:

1. Un método `mostrarValoresNormal()` que utilice una **función normal (declaración de función o expresión de función)** dentro de un `forEach` para intentar acceder a una propiedad `titulo` definida en el objeto `Calculadora` (similar al ejemplo de `group` en tus fuentes). **Explica** por qué fallaría al intentar leer `this.titulo` dentro del `forEach`.
2. Un método `mostrarValoresFlecha()` que haga lo mismo, pero utilizando una **función flecha** dentro del `forEach`. **Explica** por qué este método funciona correctamente para acceder a `this.titulo`, basándote en cómo manejan el contexto `this` las funciones flecha.

Parte C: Seguridad en el Acceso (Operadores)

Imagina que tienes un objeto `Configuracion` que puede o no tener una propiedad anidada `seguridad.nivel`.

1. Muestra cómo accederías a `seguridad.nivel` sin que la aplicación se rompa si la propiedad `seguridad` no existe, utilizando el **operador de encadenamiento opcional (`?.`)**. Indica qué valor devolvería en ese caso.
2. Muestra cómo comprobarías si la propiedad `seguridad` existe o no en el objeto `Configuracion` usando el operador `in`.