Actividad Práctica 3 – Introducción a JS

Parte 1:

1. Crear un arreglo llamado **productos** que contenga al menos 5 objetos literales con las propiedades: id, nombre, precio y stock.
    - Agregar un nuevo producto usando push().
    - Eliminar el último producto con pop().
    - Mostrar el listado final en consola.

2. Usar **filter()** con una función **anónima** para obtener los productos cuyo stock sea mayor a 10. Luego, guardar el resultado en un nuevo arreglo productosEnStock y mostrarlo.

3. Usar **find()** con una función callback para encontrar un producto por su nombre. Si lo encuentra, mostrarlo en la consola. Si no, mostrar el mensaje "Producto no encontrado".

4. Usar **reduce()** con una **arrow function** para calcular el precio total de todos los productos. Luego, realizar una función anónima que calcule el promedio de precios.

5. Usar **some()** para verificar si existe al menos un producto con stock igual a 0. Luego, usar **every()** para comprobar si todos los productos cuestan más de 100. Mostrar ambos resultados.

6. Crear un arreglo clientes con al menos 3 objetos literales que tengan las propiedades: id, nombre, edad, compras (array de strings). Usar **forEach()** para mostrar en consola el nombre de cada cliente junto con la cantidad de compras que realizó.

Parte 2:

7. Crear una función *procesarClientes(clientes, callback)* que reciba el arreglo de clientes y una función de callback. Llamar a procesarClientes con distintos callbacks:
    - Un callback que muestre solo los nombres.
    - Otro callback que muestre solo la cantidad total de compras.

8. Crear un arreglo de números y ordenarlo en forma ascendente con **sort()** y una **arrow function.** Ordenar los mismos números en forma descendente.

9. Crear un objeto literal tienda que tenga:
    - Un arreglo productos.
    - Un método vender(idProducto, cantidad) que:
        - ❖ Busque el producto por id usando find().
        - ❖ Si hay stock suficiente, reste la cantidad al stock y muestre "Venta realizada".
        - ❖ Si no hay stock, muestre "Stock insuficiente".

Probar el método vendiendo algunos productos.

10. Crear un arreglo carrito vacío.
    - Usar **push()** para agregar objetos con las propiedades: producto, cantidad, precioUnitario.
    - Usar **reduce()** para calcular el total a pagar.
    - Usar **map()** para generar un arreglo con el detalle de cada ítem en formato: "Producto X - Cantidad Y - Subtotal Z".
    - Mostrar el detalle y el total en consola.

11. Crear un arreglo libros, cada elemento debe ser un objeto con: id, titulo, autor, genero, disponible (booleano). Luego:
    - Usar **filter()** para obtener todos los libros de un género específico.
    - Usar **map()** para generar un arreglo con solo los títulos en mayúscula.
    - Crear una función prestarLibro(id) que:
        - Busque el libro con **find()**.
        - Si está disponible, lo marque como no disponible.
        - Si no, devuelva "No disponible".

12. Crear un objeto literal agenda con un arreglo contactos que guarde objetos { id, nombre, telefono }. Luego, implementar los siguientes métodos en la agenda:
    - agregarContacto(contacto) usando **push()**.
    - eliminarContacto(id) usando **filter()**. (elimina todos los de mismo id por si hubiera repetidos)
    - buscarContacto(nombre) usando **find()**.
    - listarContactos() que muestre todos.

13. Crear un arreglo alumnos con objetos { id, nombre, notas: [números] }. Luego:

- Usar **map()** + **reduce()** para generar un nuevo array, que tenga guardado cada objeto con: el nombre del alumno, el id y el promedio de sus notas.
- Generar un nuevo arreglo con solo los aprobados (promedio >= 6).
- Mostrar en consola la lista de aprobados.

Parte 3:

14. Crear un arreglo productos con objetos { id, nombre, precio, stock }. Luego definir una función comprar(id, cantidad, callbackExito, callbackError) que:

- Si el producto no existe, ejecutar callbackError con un mensaje “Producto no encontrado”
- Si hay stock suficiente → que descuente y ejecute callbackExito mostrando el detalle de la compra.
- Si no hay stock → ejecute callbackError con mensaje “no hay stock suficiente.
- Probar la función con distintas compras.

15. Utilizar el arreglo de productos del ejercicio anterior, y crear una función aplicarDescuento(id, porcentaje, callbackExito, callbackError) que haga lo siguiente:

- Buscar el producto por su id.
- Si no existe, ejecutar callbackError con un mensaje “Producto no encontrado”.
- Si el porcentaje de descuento no es válido (≤0 o >100), ejecutar callbackError con un mensaje “Porcentaje inválido”.
- Si todo es correcto, aplicar el descuento sobre el precio del producto y ejecutar callbackExito mostrando el nombre del producto y su nuevo precio.

A continuación, invocar a aplicarDescuento() y dentro de su callback de éxito, llamar/invocar a comprar() con sus propios callbacks de éxito y error.

16. Crear una función filtrarPorStock(minStock, callbackExito, callbackError) que:

- Filtre todos los productos cuyo stock sea mayor o igual a minStock.
- Si existen productos que cumplen la condición, ejecutar callbackExito pasando el listado filtrado.
- Si no hay productos que cumplan la condición, ejecutar callbackError con un mensaje “No hay productos con ese stock”.
- Probar la función con distintos valores de minStock: Mostrar los productos disponibles usando un callback de éxito y Manejar los posibles errores usando un callback de error.