Paradigmas de Programación – Ejercicios: Objetos y Colecciones (map/some/every/filter/reduce)

Nivel: inicial–intermedio. Cada ejercicio incluye enunciado y una posible solución en JavaScript.

# 1) Saludos con métodos (map)

**Enunciado:** Definí un array de personas. Cada persona debe ser un objeto con nombre, edad y un método saludar() que devuelva un saludo con su nombre. Usá map para obtener un array con todos los saludos.

**Solución (JS):**

const personas = [  
 { nombre: 'Ana', edad: 20, saludar(){ return `Hola, soy ${this.nombre}`; } },  
 { nombre: 'Luis', edad: 22, saludar(){ return `Hola, soy ${this.nombre}`; } },  
 { nombre: 'Mia', edad: 19, saludar(){ return `Hola, soy ${this.nombre}`; } },  
];  
  
const saludos = personas.map(p => p.saludar());  
console.log(saludos); // ["Hola, soy Ana", "Hola, soy Luis", "Hola, soy Mia"]

# 2) Aprobados por promedio (filter + método)

**Enunciado:** Modelá estudiantes como objetos con nombre, notas: number[] y un método promedio() que devuelva su promedio. Usá filter para obtener los estudiantes con promedio >= 6.

**Solución (JS):**

const estudiantes = [  
 { nombre: 'Ana', notas: [8, 6, 7], promedio(){ return this.notas.reduce((a,b)=>a+b,0)/this.notas.length; } },  
 { nombre: 'Luis', notas: [5, 4, 6], promedio(){ return this.notas.reduce((a,b)=>a+b,0)/this.notas.length; } },  
 { nombre: 'Mia', notas: [9, 7, 8], promedio(){ return this.notas.reduce((a,b)=>a+b,0)/this.notas.length; } },  
];  
  
const aprobados = estudiantes.filter(e => e.promedio() >= 6);  
console.log(aprobados.map(e => ({ nombre: e.nombre, prom: e.promedio().toFixed(2) })));

# 3) ¿Hay tareas vencidas? (some + método)

**Enunciado:** Definí tareas como objetos con titulo y diasRestantes (número). Agregá un método estaVencida() que devuelva true si diasRestantes < 0. Usá some para saber si existe alguna tarea vencida.

**Solución (JS):**

const tareas = [  
 { titulo: 'Entregar TP1', diasRestantes: 2, estaVencida(){ return this.diasRestantes < 0; } },  
 { titulo: 'Leer cap. 3', diasRestantes: -1, estaVencida(){ return this.diasRestantes < 0; } },  
 { titulo: 'Repasar notas', diasRestantes: 0, estaVencida(){ return this.diasRestantes < 0; } },  
];  
  
const hayVencidas = tareas.some(t => t.estaVencida());  
console.log(hayVencidas); // true

# 4) Control de stock (every + método)

**Enunciado:** Modelá productos con nombre, stock y un método enStock(min) que devuelva true si stock >= min. Usá every para verificar que todos los productos tienen stock suficiente (min = 1).

**Solución (JS):**

const productos = [  
 { nombre: 'Cuaderno', stock: 10, enStock(min){ return this.stock >= min; } },  
 { nombre: 'Lápiz', stock: 3, enStock(min){ return this.stock >= min; } },  
 { nombre: 'Goma', stock: 0, enStock(min){ return this.stock >= min; } },  
];  
  
const todosConStock = productos.every(p => p.enStock(1));  
console.log(todosConStock); // false (Goma tiene 0)

# 5) Total del carrito (reduce + método)

**Enunciado:** Definí un carrito como array de ítems { nombre, precio, cantidad } y un método subtotal() en cada ítem que devuelva precio \* cantidad. Usá reduce para obtener el total a pagar.

**Solución (JS):**

const carrito = [  
 { nombre: 'Libro', precio: 2000, cantidad: 2, subtotal(){ return this.precio \* this.cantidad; } },  
 { nombre: 'Lápiz', precio: 200, cantidad: 3, subtotal(){ return this.precio \* this.cantidad; } },  
 { nombre: 'Cuaderno', precio: 800, cantidad: 1, subtotal(){ return this.precio \* this.cantidad; } },  
];  
  
const total = carrito.reduce((acc, item) => acc + item.subtotal(), 0);  
console.log(total); // 2000\*2 + 200\*3 + 800\*1 = 5200

# 6) Libros largos de un autor (filter + método + map)

**Enunciado:** Definí libros como objetos con titulo, autor, paginas y un método esLargo() que devuelve true si paginas >= 300. Filtrá los libros de un autor dado que sean largos y obtené sus títulos.

**Solución (JS):**

const libros = [  
 { titulo: 'El Aleph', autor: 'Borges', paginas: 146, esLargo(){ return this.paginas >= 300; } },  
 { titulo: 'Ficciones', autor: 'Borges', paginas: 224, esLargo(){ return this.paginas >= 300; } },  
 { titulo: 'It', autor: 'King', paginas: 1138, esLargo(){ return this.paginas >= 300; } },  
 { titulo: 'Fundación', autor: 'Asimov', paginas: 255, esLargo(){ return this.paginas >= 300; } },  
 { titulo: 'Dune', autor: 'Herbert',paginas: 688, esLargo(){ return this.paginas >= 300; } },  
];  
  
const autor = 'Borges';  
const largosDelAutor = libros  
 .filter(l => l.autor === autor && l.esLargo())  
 .map(l => l.titulo);  
  
console.log(largosDelAutor); // []

# 7) Autenticación simple (some + método)

**Enunciado:** Definí usuarios con usuario y pin y un método validaPin(p) que compare el pin. Implementá una función autenticar(usuario, pin) que devuelva true si existe un usuario válido.

**Solución (JS):**

const usuarios = [  
 { usuario: 'ana', pin: '1234', validaPin(p){ return this.pin === p; } },  
 { usuario: 'luis', pin: '0000', validaPin(p){ return this.pin === p; } },  
 { usuario: 'mia', pin: '9999', validaPin(p){ return this.pin === p; } },  
];  
  
function autenticar(u, p) {  
 return usuarios.some(x => x.usuario === u && x.validaPin(p));  
}  
  
console.log(autenticar('ana','1234')); // true  
console.log(autenticar('ana','1111')); // false

# 8) Ventas por categoría (reduce)

**Enunciado:** Dado un array de productos vendidos con { categoria, precio }, usá reduce para producir un objeto con el total de ventas por categoría.

**Solución (JS):**

const ventas = [  
 { categoria: 'libreria', precio: 2000 },  
 { categoria: 'libreria', precio: 200 },  
 { categoria: 'tecnologia', precio: 15000 },  
 { categoria: 'libreria', precio: 800 },  
 { categoria: 'tecnologia', precio: 5000 },  
];  
  
const totalesPorCat = ventas.reduce((acc, v) => {  
 acc[v.categoria] = (acc[v.categoria] || 0) + v.precio;  
 return acc;  
}, {});  
  
console.log(totalesPorCat); // { libreria: 3000, tecnologia: 20000 }

# 9) Array de comandos (map)

**Enunciado:** Modelá un array de comandos, cada uno con nombre y un método fn(x). Usá map para obtener el resultado de aplicar cada comando a un valor dado.

**Solución (JS):**

const comandos = [  
 { nombre: 'doble', fn(x){ return x \* 2; } },  
 { nombre: 'cuadrado', fn(x){ return x \* x; } },  
 { nombre: 'incremento', fn(x){ return x + 1; } },  
];  
  
const valor = 5;  
const resultados = comandos.map(c => ({ nombre: c.nombre, resultado: c.fn(valor) }));  
console.log(resultados);  
// [ {nombre:'doble', resultado:10}, {nombre:'cuadrado', resultado:25}, {nombre:'incremento', resultado:6} ]

# 10) Eliminar duplicados por id (reduce + some)

**Enunciado:** Dado un array de objetos con id y nombre, usá reduce para construir un nuevo array sin duplicados por id. Tip: en cada paso, verificá con some si el id ya está agregado.

**Solución (JS):**

const items = [  
 { id: 1, nombre: 'A' },  
 { id: 2, nombre: 'B' },  
 { id: 1, nombre: 'A bis' },  
 { id: 3, nombre: 'C' },  
 { id: 2, nombre: 'B bis' },  
];  
  
const unicos = items.reduce((acc, item) => {  
 return acc.some(x => x.id === item.id) ? acc : [...acc, item];  
}, []);  
  
console.log(unicos); // [{id:1,'A'}, {id:2,'B'}, {id:3,'C'}]