Paradigmas de Programación (JS en Node.js)

SOLUCIONES – Uso exclusivo del docente.

# Dataset base (igual que en la guía)

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// TRABAJAMOS CON ARRAYS - 2 clase  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
const lista = [4, 1, 2, 4, 5, 8, 7, 6, 9, 10];  
  
// Estructura de datos - Objeto persona  
const personas = [  
 { nombre: "Ana", edad: 18, pais: "Argentina", sexo: "F" },  
 { nombre: "Luis", edad: 30, pais: "Perú", sexo: "M" },  
 { nombre: "Fernanada", edad: 25, pais: "Argentina", sexo: "F" },  
 { nombre: "Laura", edad: 15, pais: "Chile", sexo: "F" },  
 { nombre: "Veronica", edad: 22, pais: "Chile", sexo: "F" },  
 { nombre: "Miguel", edad: 22, pais: "Chile", sexo: "M" },  
];

# Soluciones paso a paso con comentarios pedagógicos

## E1

// Filtrar números > 5 (filter recorre y conserva los que cumplan la condición)  
const res = lista.filter(v => v > 5);   
// Pedagogía: filter NO modifica el array original; crea uno nuevo con los que pasaron el "colador".

## E2

// Números pares (x % 2 === 0)  
const pares = lista.filter(x => x % 2 === 0);

## E3

// Extraer edades con map (transforma cada objeto persona en su edad)  
const edades = personas.map(p => p.edad);

## E4

// Mapear a frases "<nombre> vive en <pais>"  
const frases = personas.map(p => `${p.nombre} vive en ${p.pais}`);

## E5

// ¿Existe el 6? some devuelve true si al menos uno cumple  
const existeSeis = lista.some(x => x === 6);

## E6

// ¿Todos >= 1? every exige que todos cumplan  
const todosMayoresIgualUno = lista.every(x => x >= 1);

## E7

// Mayores estrictos a 18  
const mayores = personas.filter(p => p.edad > 18);  
// Pedagogía: preferir >= o > según el criterio de negocio.

## E8

// Entre 20 y 25 inclusive  
const entre20y25 = personas.filter(p => p.edad >= 20 && p.edad <= 25);

## E9

// Viven en Chile  
const enChile = personas.filter(p => p.pais === "Chile");

## E10

// Masculinos chilenos  
const mascChilenos = personas.filter(p => p.pais === "Chile" && p.sexo === "M");

## E11

// Primer masculino en Chile (find devuelve el primero o undefined)  
const primeroMascChile = personas.find(p => p.pais === "Chile" && p.sexo === "M");

## E12

// Nombres de Argentina ordenados alfabéticamente  
const nombresArgOrden = personas  
 .filter(p => p.pais === "Argentina")  
 .map(p => p.nombre)  
 .sort((a,b) => a.localeCompare(b));

## E13

// ¿Hay menores de edad?  
const hayMenores = personas.some(p => p.edad < 18);

## E14

// ¿Todas son F?  
const todasF = personas.every(p => p.sexo === "F");

## E15

// Nombres de Chile ordenados por edad ascendente  
const nombresChileOrdenEdad = personas  
 .filter(p => p.pais === "Chile")  
 .sort((a,b) => a.edad - b.edad)  
 .map(p => p.nombre);

## E16

// Suma total de edades  
const sumaEdades = personas.reduce((acum, p) => acum + p.edad, 0);

## E17

// Promedio de edades por país  
const porPais = personas.reduce((acc, p) => {  
 if (!acc[p.pais]) acc[p.pais] = { suma: 0, count: 0 };  
 acc[p.pais].suma += p.edad;  
 acc[p.pais].count += 1;  
 return acc;  
}, {});  
const promedioPorPais = Object.fromEntries(  
 Object.entries(porPais).map(([pais, {suma, count}]) => [pais, suma / count])  
);  
// Pedagogía: dividimos la agregación (reduce) del cálculo final (mapeo de entries).

## E18

// Estructura { pais: { F: n, M: n } }  
const conteoSexoPorPais = personas.reduce((acc, p) => {  
 if (!acc[p.pais]) acc[p.pais] = { F: 0, M: 0 };  
 acc[p.pais][p.sexo] = (acc[p.pais][p.sexo] || 0) + 1;  
 return acc;  
}, {});

## E19

// Partición adultos/menores en una sola pasada  
const esAdulto = p => p.edad >= 18;  
const { adultos, menores } = personas.reduce((acc, p) => {  
 (esAdulto(p) ? acc.adultos : acc.menores).push(p);  
 return acc;  
}, { adultos: [], menores: [] });

## E20

// Países únicos - con reduce  
const paisesUnicosReduce = personas.reduce((acc, p) => {  
 if (!acc.includes(p.pais)) acc.push(p.pais);  
 return acc;  
}, []);  
// Países únicos - con Set (más simple y eficiente)  
const paisesUnicosSet = [...new Set(personas.map(p => p.pais))];

## E21

// Eliminar repetidos con reduce  
const a = [3, 5, 6, 2, 4, 2, 2, 4, 5, 6];  
const sinRepeReduce = a.reduce((acc, v) => {  
 if (!acc.includes(v)) acc.push(v);  
 return acc;  
}, []);  
// Comparar con Set  
const sinRepeSet = [...new Set(a)];

## E22

// Elegir la función con mayor resultado por cada valor  
const fns = [  
 { name: 'Doble', fn: x => x \* 2 },  
 { name: 'Cuadrado', fn: x => x \* x },  
 { name: 'Suma5', fn: x => x + 5 },  
];  
const values = [3, 1, 2, 6];  
  
const mejoresPorValor = values.map(v => {  
 const mejor = fns.reduce((a, b) => (a.fn(v) > b.fn(v) ? a : b));  
 return { name: mejor.name, value: v, result: mejor.fn(v) };  
});  
// Pedagogía: reduce compara pares (a,b) y retiene el que "gana" según la métrica fn(v).

## E23

// Del arreglo anterior, elegir el de menor result  
const menorResultado = mejoresPorValor.reduce((a, b) => (a.result <= b.result ? a : b));

## E24

// uniqueBy: deduplicar por clave computada  
const uniqueBy = (arr, selector) => {  
 const vistos = new Set();  
 return arr.reduce((acc, item) => {  
 const key = selector(item);  
 if (!vistos.has(key)) {  
 vistos.add(key);  
 acc.push(item);  
 }  
 return acc;  
 }, []);  
};  
// Ejemplo:  
const unicosPorPais = uniqueBy(personas, p => p.pais);

## E25

// partition: separa en [trueItems, falseItems]  
const partition = (arr, predicate) => arr.reduce((acc, item) => {  
 (predicate(item) ? acc[0] : acc[1]).push(item);  
 return acc;  
}, [[], []]);  
// Ejemplo con adultos  
const [adultos2, menores2] = partition(personas, p => p.edad >= 18);

## E26

// Falla porque con llaves {} no hay return implícito  
// Incorrecto: lista.filter(v => { v > 5 }) -> siempre 'undefined'  
// Correcciones:  
const r1 = lista.filter(v => v > 5); // sin llaves, return implícito  
const r2 = lista.filter(v => { return v > 5; }); // con llaves, return explícito