# Trabajo Práctico – Programación con Arrays y Objetos en JavaScript

Objetivo: Practicar el uso de funciones de orden superior y métodos de arrays (map, reduce, filter, some, every, push, includes) aplicados a objetos y colecciones.

Instrucciones: Lea cada enunciado, analice el código y ejecute las soluciones propuestas. Cuando corresponda, modifique los datos para observar comportamientos diferentes.

## Sección A · Ejercicios provistos

### Ejercicio 1

Enunciado: Dado un conjunto de funciones y una lista de valores, generar una estructura que muestre, para cada valor, el resultado de aplicar cada función.

const fns = [{  
 name: "Doble",  
 fn: (x) => x \* 2,  
 },  
 {  
 name: "Cuadrado",  
 fn: (x) => x \* x,  
 },  
 {  
 name: "Suma5",  
 fn: (x) => x + 5,  
 },  
];  
  
/\* solo para entender al nombre  
 const result = fns.map(({ name, fn }) => {  
 return { name: name }}  
 );  
\*/  
  
const values = [3, 4, 5, 6];  
  
const r = values.map((v) => {  
 return fns.map(({ name, fn }) => {  
 return { name, valor: v, result: fn(v) };  
 });  
});  
  
console.log(r);

### Ejercicio 2

Enunciado: Para cada valor, determinar cuál función de fns produce el mayor resultado y, entre todos, quedarse con el menor de esos máximos.

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 r = values  
 .map((v) => {  
 const f = fns.reduce((a, b) => (a.fn(v) > b.fn(v) ? a : b));  
 return { name: f.name, value: v, result: f.fn(v) };  
 })  
 .reduce((a, b) => (a.result <= b.result ? a : b));  
  
console.log(r);

### Ejercicio 3

Enunciado: Validar contraseñas con una lista de funciones de validación. Mostrar cuáles reglas no se cumplen para el valor dado.

const fns = [{  
 name: "Long Minima de 8",  
 fn: (pwd) => pwd.length >= 8,  
 },  
 {  
 name: "Sin Blancos",  
 fn: (pwd) => !pwd.includes(" "),  
 },  
 {  
 name: "Con Caractes especiales",  
 fn: (pwd) => ["#", "&", "$", "!"].some((c) => pwd.includes(c)),  
 },  
];  
  
fns.push({  
 name: "Con Digitos",  
 fn: (pwd) => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].some((d) => pwd.includes(d)),  
});  
  
const values = "gn3 sfd";  
  
console.log(  
 fns.map((f) => {  
 return { name: f.name, value: f.fn(values) };  
 })  
 .filter((f) => !f.value)  
);

## Sección B · Ejercicios adicionales (con soluciones)

### Ejercicio 4 – Catálogo de productos

Enunciado: Dado un arreglo de productos, realizar: (a) filtrar productos con el tag "oferta" (some), (b) calcular el total de unidades en stock (reduce), (c) obtener el producto más barato (reduce).

// Datos  
const productos = [  
 { id: 1, nombre: "Mouse", precio: 12000, stock: 15, tags: ["periferico","oferta"] },  
 { id: 2, nombre: "Teclado", precio: 22000, stock: 0, tags: ["periferico"] },  
 { id: 3, nombre: "Monitor", precio: 150000, stock: 7, tags: ["oferta","fragil"] },  
 { id: 4, nombre: "Webcam", precio: 35000, stock: 12, tags: ["periferico"] },  
];  
  
// (a) Filtrar por "oferta"  
const enOferta = productos.filter(p => p.tags.some(t => t === "oferta"));  
console.log(enOferta.map(p => p.nombre)); // ["Mouse","Monitor"]  
  
// (b) Total de stock  
const totalStock = productos.reduce((acc, p) => acc + p.stock, 0);  
console.log(totalStock); // 34  
  
// (c) Producto más barato  
const masBarato = productos.reduce((min, p) => p.precio < min.precio ? p : min);  
console.log(masBarato.nombre); // "Mouse"

### Ejercicio 5 – Usuarios y permisos

Enunciado: Con el siguiente arreglo de usuarios, (a) verificar si todos tienen el permiso "leer" (every), (b) listar los que no tienen "escribir" (filter + some), (c) agregar el permiso "invitar" a quienes tengan el rol "admin" (push si no existe).

const usuarios = [  
 { usuario: "Ana", roles: ["admin"], permisos: ["leer","escribir","borrar"] },  
 { usuario: "Luis", roles: ["editor"], permisos: ["leer","escribir"] },  
 { usuario: "Mia", roles: ["lector"], permisos: ["leer"] },  
];  
  
// (a) ¿Todos tienen "leer"?  
const todosLeen = usuarios.every(u => u.permisos.includes("leer"));  
console.log(todosLeen); // true  
  
// (b) ¿Quiénes NO tienen "escribir"?  
const noEscriben = usuarios  
 .filter(u => !u.permisos.includes("escribir"))  
 .map(u => u.usuario);  
console.log(noEscriben); // ["Mia"]  
  
// (c) Agregar "invitar" a quienes tengan rol "admin"  
usuarios.forEach(u => {  
 if (u.roles.includes("admin") && !u.permisos.includes("invitar")) {  
 u.permisos.push("invitar"); // uso de push  
 }  
});  
console.log(usuarios.find(u => u.usuario === "Ana").permisos);  
// ["leer","escribir","borrar","invitar"]

### Ejercicio 6 – Promedios y aprobaciones

Enunciado: Dado un conjunto de alumnos con sus notas: (a) obtener el promedio por alumno (map + reduce), (b) ordenar por promedio y tomar los 2 mejores, (c) verificar si todos aprobaron (every nota ≥ 6) y listar los que desaprobaron (some nota < 6).

const alumnos = [  
 { nombre: "A", notas: [10, 8, 9] },  
 { nombre: "B", notas: [6, 7, 5] },  
 { nombre: "C", notas: [9, 9, 10] },  
];  
  
// (a) Promedios  
const conPromedio = alumnos.map(a => {  
 const sum = a.notas.reduce((acc, n) => acc + n, 0);  
 return { nombre: a.nombre, promedio: sum / a.notas.length };  
});  
console.log(conPromedio);  
// [ {nombre:"A", promedio:9}, {nombre:"B", promedio:6}, {nombre:"C", promedio:9.333...} ]  
  
// (b) Top 2  
const top2 = [...conPromedio].sort((x,y) => y.promedio - x.promedio).slice(0,2);  
console.log(top2.map(a => a.nombre)); // ["C","A"]  
  
// (c) Aprobaciones  
const todosAprobaron = alumnos.every(a => a.notas.every(n => n >= 6));  
console.log(todosAprobaron); // false  
  
const desaprobaron = alumnos  
 .filter(a => a.notas.some(n => n < 6))  
 .map(a => a.nombre);  
console.log(desaprobaron); // ["B"]

### Ejercicio 7 – Frecuencia de palabras

Enunciado: Dado un texto, construir (con reduce) un objeto de frecuencias por palabra, obtener la palabra más frecuente y listar solo las palabras únicas (frecuencia = 1).

const texto = "la tecnología cambia la escuela y la comunidad con datos y práctica";  
  
// Normalizar y tokenizar  
const palabras = texto  
 .toLowerCase()  
 .replace(/[.,;:¡!¿?()"-]/g, '')  
 .split(/\s+/);  
  
// Frecuencia con reduce  
const freq = palabras.reduce((acc, w) => {  
 acc[w] = (acc[w] || 0) + 1;  
 return acc;  
}, {});  
console.log(freq);  
  
// Más frecuente  
const masFrecuente = Object.entries(freq).reduce((a, b) => (b[1] > a[1] ? b : a));  
console.log(masFrecuente); // e.g., ["la", 2] o ["y", 2] según texto  
  
// Únicas  
const unicas = Object.entries(freq)  
 .filter(([,count]) => count === 1)  
 .map(([w]) => w);  
console.log(unicas);

### Ejercicio 8 – Carrito de compras

Enunciado: Con un carrito de ítems: (a) calcular total sin IVA (reduce), (b) calcular total con IVA 21%, (c) aplicar cupón "DESC10" (10% menos) si existe, (d) verificar si hay algún ítem con cantidad inválida (some).

const carrito = [  
 { producto: "Libro", precio: 18000, cantidad: 2 },  
 { producto: "Lapicera", precio: 700, cantidad: 5 },  
 { producto: "Cuaderno", precio: 3500, cantidad: 1 },  
];  
  
// (a) Total sin IVA  
const totalSinIVA = carrito.reduce((acc, it) => acc + it.precio \* it.cantidad, 0);  
console.log(totalSinIVA); // 43000  
  
// (b) Con IVA 21%  
const totalConIVA = Math.round(totalSinIVA \* 1.21);  
console.log(totalConIVA); // 52030  
  
// (c) Cupón  
const cupones = ["DESC10"];  
const aplicaDESC10 = cupones.includes("DESC10");  
const totalConCupon = aplicaDESC10 ? Math.round(totalConIVA \* 0.90) : totalConIVA;  
console.log(totalConCupon); // 46827  
  
// (d) Validación  
const hayCantidadInvalida = carrito.some(it => it.cantidad <= 0);  
console.log(hayCantidadInvalida); // false

### Ejercicio 9 – Pipeline con reduce

Enunciado: Dado un objeto persona y un pipeline (arreglo) de transformaciones, aplicar todas las funciones en orden usando reduce.

const persona = { nombre: " Ada ", edad: 29 };  
  
const pipeline = [  
 p => ({ ...p, nombre: p.nombre.trim() }),  
 p => ({ ...p, tags: [ ...(p.tags || []), "nuevo" ] }),  
 p => ({ ...p, activo: true }),  
];  
  
const resultado = pipeline.reduce((acc, fn) => fn(acc), persona);  
console.log(resultado);  
// { nombre: "Ada", edad: 29, tags: ["nuevo"], activo: true }

### Ejercicio 10 – Agrupar por categoría con reduce

Enunciado: A partir de una lista de cursos, agruparlos por categoría (clave del objeto) usando reduce.

const cursos = [  
 { id: 1, titulo: "JS Básico", categoria: "programación" },  
 { id: 2, titulo: "HTML/CSS", categoria: "diseño" },  
 { id: 3, titulo: "JS Avanzado", categoria: "programación" },  
 { id: 4, titulo: "UI/UX", categoria: "diseño" },  
];  
  
const porCategoria = cursos.reduce((acc, c) => {  
 (acc[c.categoria] ||= []).push(c);  
 return acc;  
}, {});  
  
console.log(porCategoria);  
// {  
// "programación": [ {id:1,...}, {id:3,...} ],  
// "diseño": [ {id:2,...}, {id:4,...} ]  
// }

Sugerencia didáctica: invite a los/las estudiantes a reescribir soluciones de forma inmutable (copiando arrays/objetos con spread) y luego comparar con versiones mutables (p. ej., push vs. [...arr, x]).