



## Objetivo

Dominar programación funcional: funciones puras, inmutabilidad, composición y técnicas avanzadas.

## Conceptos clave

### Funciones puras

Sin efectos secundarios, mismo input → mismo output.

```
// ✅ Pura
const sumar = (a: number, b: number) => a + b;

// ❌ Impura (efecto secundario)
let contador = 0;
const incrementar = () => ++contador;
```

### Inmutabilidad

Datos no cambian, se crean nuevas versiones.

```
// ❌ Mutación
const nums = [1, 2, 3];
nums.push(4);

// ✅ Inmutable
const nums2 = [...nums, 4];
const actualizado = nums.map(n => n * 2);
```

## Funciones de orden superior

Reciben/devuelven otras funciones.

```
const aplicar = (arr: number[], fn: (n: number) => number) => arr.map(fn);
const duplicar = (n: number) => n * 2;
const resultado = aplicar([1, 2, 3], duplicar); // [2, 4, 6]

// Currying
const multiplicar = (a: number) => (b: number) => a * b;
const por3 = multiplicar(3);
console.log(por3(5)); // 15
```

## Composición

Combinar funciones simples.

```
const agregarIVA = (precio: number) => precio * 1.21;
const descuento = (precio: number) => precio * 0.9;
const redondear = (precio: number) => Math.round(precio * 100) / 100;

// Composición
const pipe = <T>(...fns: Array<(arg: T) => T>) =>
  (value: T) => fns.reduce((acc, fn) => fn(acc), value);

const calcularPrecio = pipe(descuento, agregarIVA, redondear);
console.log(calcularPrecio(100)); // 108.9
```

## Métodos funcionales en arrays

map, filter, reduce

```
const productos = [
  { nombre: "Laptop", precio: 1000, categoria: "tech" },
  { nombre: "Mesa", precio: 200, categoria: "muebles" },
  { nombre: "Mouse", precio: 25, categoria: "tech" }
];

// Transformar (map)
const nombres = productos.map(p => p.nombre);

// Filtrar
const tech = productos.filter(p => p.categoria === "tech");

// Reducir
const total = productos.reduce((sum, p) => sum + p.precio, 0);

// Combinar operaciones
const preciosTechConIVA = productos
```

```
.filter(p => p.categoría === "tech")
.map(p => p.precio * 1.21)
.reduce((sum, precio) => sum + precio, 0);
```

flatMap, some, every

```
const pedidos = [
  { id: 1, items: ["laptop", "mouse"] },
  { id: 2, items: ["mesa", "silla"] }
];

// flatMap: aplanar y transformar
const todosLosItems = pedidos.flatMap(p => p.items);

// some: al menos uno cumple
const hayTech = productos.some(p => p.categoría === "tech");

// every: todos cumplen
const todosCostosos = productos.every(p => p.precio > 20);
```

## Recursión vs iteración

```
// Iterativo
function factorialIter(n: number): number {
  let result = 1;
  for (let i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}

// Recursivo
function factorialRec(n: number): number {
  return n <= 1 ? 1 : n * factorialRec(n - 1);
}
```