Resurse suplimentare
pe care recomand să le parcurgi

Higher-Order Functions pe scurt

O higher-order function (HOF) este o funcție care primește ca parametru o altă funcție sau returnează o funcție nouă. Majoritatea metodelor array sunt HOF-uri.

```
const numbers = [1, 2, 3]

// map primește un callback
const doubled = numbers.map((n) => n * 2)
```

`map()`

Transformă fiecare element și returnează un **nou** array cu aceeași lungime.

```
const products = [
  { id: 1, name: 'Laptop', price: 3500 },
  { id: 2, name: 'Telefon', price: 2000 },
]

const priceList = products.map(({ name, price }) => `${name}: ${price} RON`)
// ['Laptop: 3500 RON', 'Telefon: 2000 RON']
```

În React

```
const ProductList = ({ products }) => (
  <ul>
    {products.map(({ id, name }) => (
      <li key={id}>{name}</li>
    ))}
  </ul>
)
```

`filter()`

Selectează elementele care îndeplinesc o condiție.

```
const affordable = products.filter(({ price }) => price < 3000)
```

`reduce()`

Calculează o valoare „rezumat” (sumă, obiect, array) parcurgând elementele.

```
const total = products.reduce((sum, p) => sum + p.price, 0) // 5500
```

Pattern comun: Group By

```
const orders = [
  { id: 1, status: 'pending' },
  { id: 2, status: 'shipped' },
  { id: 3, status: 'pending' },
]

const grouped = orders.reduce((acc, order) => {
  const { status } = order
  acc[status] = [...(acc[status] ?? []), order]
  return acc
}, {})
```

Alte metode utile

Metodă	Descriere
`forEach`	Iterează (NU returnează nimic – produce side-effects)
`find`	Returnează primul element care satisface condiția
`some`	`true` dacă CEL PUȚIN un element respectă condiția
`every`	`true` dacă TOATE elementele respectă condiția
`flat` / `flatMap`	Aplatizează arrays imbricate

```
const tags = ['js', 'react', 'css']
const hasReact = tags.some((t) => t === 'react') // true
```

Combinarea metodelor (method chaining)

```
const users = [
  { id: 1, name: 'Ana', age: 25 },
  { id: 2, name: 'Ion', age: 17 },
  { id: 3, name: 'Maria', age: 30 },
]

const namesOfAdults = users
  .filter(({ age }) => age >= 18)
  .map(({ name }) => name)
  .join(', ')
// 'Ana, Maria'
```

Principii de Programare Funcțională

1. Imutabilitate – Nu modifica obiectele/array-urile originale.
2. Funcții pure – Aceeași intrare ⇒ aceeași ieșire, fără efecte secundare.
3. Compoziție – Combină funcții mici pentru a construi logica.

   În React, aceste principii ajută la prevenirea bug-urilor legate de state și la re-randări eficiente.

Studiu de caz: gestionarea unei liste Todo

```
const [todos, setTodos] = useState([{ id: 1, text: 'Learn JS', done: false }])

const addTodo = (text) =>
  setTodos((t) => [...t, { id: Date.now(), text, done: false }])

const toggleTodo = (id) =>
  setTodos((t) =>
    t.map((todo) => (todo.id === id ? { ...todo, done: !todo.done } : todo))
  )

const clearCompleted = () => setTodos((t) => t.filter((todo) => !todo.done))
```