Ejercicio 1 — Arrays y caracteres

Enunciado
Dado un array de palabras, escribe una función `capitalizarYContarVocales(words)` que:

1. Devuelva un nuevo array con cada palabra capitalizada (primera letra en mayúscula y resto en minúscula).

2. Devuelva también un objeto que indique el número total de vocales (a, e, i, o, u — sin distinguir mayúsculas/minúsculas) encontradas en todas las palabras.

La función debe devolver un objeto con la forma `{ palabras: [...], totalVocales: N }`.

Ejemplo

- Entrada: `["hola", "mUndo", "árbol", "PIrrAmiDe"]`

- Salida esperada:
  `{ palabras: ["Hola", "Mundo", "Árbol", "Pirrámide"], totalVocales: 9 }`

Ejercicio 2 — Recursividad

Enunciado
Escribe una función recursiva `sumarHasta(n)` que calcule la suma de todos los enteros positivos desde 1 hasta `n`.

- Si `n` es menor o igual que 0, la función debe devolver 0.

- No usar bucles: resolver únicamente con recursión.

Ejemplo

- Entrada: `5` → Salida: `15` (1+2+3+4+5)
- Entrada: `0` → Salida: `0`
-

Ejercicio 3 — Funciones anónimas (expresiones de función / arrow functions)

Enunciado
Crea una función `filtrarYTransformar(arr, filtro, transformacion)` donde:

- `arr` es un array de números.
- `filtro` es una **función anónima** que recibe un número y devuelve `true` si debe incluirse.
- `transformacion` es una **función anónima** que recibe un número y devuelve su transformación.

La función debe devolver un nuevo array que primero filtra `arr` usando `filtro` y luego aplica `transformacion` a los elementos resultantes. Proporciona un ejemplo usando funciones anónimas (arrow functions).

Ejemplo

- Entrada:
  `arr = [1,2,3,4,5,6]`
  `filtro = n => n % 2 === 0` (pares)
  `transformacion = n => n * n` (elevar al cuadrado)
- Salida: `[4, 16, 36]`
-

Ejercicio 4 — Callbacks y funciones en línea (asincronía simulada con `setTimeout`)

Enunciado
Implementa una función `procesarTareas(tareas, callbackFinal)` donde:

- `tareas` es un array de objetos `{ id, duracion }` (duración en ms).
- Para cada tarea debes simular su ejecución con `setTimeout` usando su `duracion`.
- Cuando una tarea termina, debes llamar a una función callback en línea que imprima `Tarea <id> completada`.
- Cuando todas las tareas hayan terminado, llamar a `callbackFinal(resultado)` donde `resultado` es un array de ids de tareas completadas en orden de finalización.

Objetivo: practicar callbacks y pasar funciones en línea (inline) para manejar eventos completados.

Ejemplo

- Entrada: `[{id: 'A', duracion: 300}, {id: 'B', duracion: 100}, {id: 'C', duracion: 200}]`
- Salida por consola (orden según duraciones):
  ```
  Tarea B completada
  Tarea C completada
  Tarea A completada
  ```
- `callbackFinal` recibe `['B','C','A']`.