¡Perfecto! Continuemos con la serie de ejercicios detallados para los temas restantes. Aquí tienes 10 ejercicios para el tema de **Arrays**, seguidos de 10 ejercicios para **Objetos Literales** y finalmente 10 ejercicios para la **Combinación de Conceptos**.

Tema: 4. Arrays

Ejercicio 1: Crear y Acceder al Primer Elemento

Consigna:
Crea una función llamada `obtenerPrimerElemento` que reciba un argumento `miArray` (un array de cualquier tipo). La función debe retornar el primer elemento de ese array. Si el array está vacío, debe retornar `undefined`.

Ejemplo:

- `obtenerPrimerElemento([10, 20, 30])` debería retornar `10`.
- `obtenerPrimerElemento(["a", "b", "c"])` debería retornar `"a"`.
- `obtenerPrimerElemento([])` debería retornar `undefined`.

Ejercicio 2: Obtener el Último Elemento

Consigna:
Define una función llamada `obtenerUltimoElemento` que reciba un argumento `miArray` (un array de cualquier tipo). La función debe retornar el último elemento de ese array. Si el array está vacío, debe retornar `undefined`.

Ejemplo:

- `obtenerUltimoElemento([10, 20, 30])` debería retornar `30`.
- `obtenerUltimoElemento(["x", "y", "z"])` debería retornar `"z"`.
- `obtenerUltimoElemento([])` debería retornar `undefined`.

Ejercicio 3: Agregar Elemento al Final

Consigna:
Escribe una función llamada `agregarAlFinal` que reciba dos argumentos: `miArray` (un array) y `nuevoElemento` (cualquier tipo de dato). La función debe agregar `nuevoElemento` al final de `miArray` y luego retornar la **nueva longitud** del array.

Ejemplo:

- Si `miArray` es `[1, 2]` y `nuevoElemento` es `3`, la función debería modificar `miArray` a `[1, 2, 3]` y retornar `3`.

Ejercicio 4: Quitar Último Elemento

Consigna:
Implementa una función llamada `quitarUltimo` que reciba un argumento `miArray` (un array).

La función debe eliminar el último elemento de `miArray` y retornar el elemento que fue eliminado. Si el array está vacío, debe retornar `undefined`.

Ejemplo:

- Si `miArray` es `["apple", "banana", "cherry"]`, la función debería modificar `miArray` a `["apple", "banana"]` y retornar `"cherry"`.
- Si `miArray` es `[]`, debería retornar `undefined`.

Ejercicio 5: Verificar Existencia de Elemento

Consigna:
Crea una función llamada `contieneElemento` que reciba dos argumentos: `miArray` (un array) y `elementoBuscado` (cualquier tipo de dato). La función debe retornar `true` si `elementoBuscado` se encuentra en `miArray`, y `false` en caso contrario.

Ejemplo:

- `contieneElemento([10, 20, 30], 20)` debería retornar `true`.
- `contieneElemento(["perro", "gato"], "ratón")` debería retornar `false`.

Consideraciones:
Puedes usar el método `includes()` o `indexOf()`.

Ejercicio 6: Sumar Todos los Elementos Numéricos

Consigna:
Define una función llamada `sumarElementosArray` que reciba un argumento `numeros` (un array de números). La función debe retornar la suma total de todos los números en el array. Si el array está vacío, debe retornar `0`.

Ejemplo:

- `sumarElementosArray([1, 2, 3, 4])` debería retornar `10`.
- `sumarElementosArray([5, 10])` debería retornar `15`.
- `sumarElementosArray([])` debería retornar `0`.

Consideraciones:
Deberás iterar sobre el array usando un bucle `for` o `forEach`.

Ejercicio 7: Encontrar Índice de un Elemento

Consigna:
Escribe una función llamada `encontrarIndice` que reciba dos argumentos: `miArray` (un array) y `elemento` (el valor a buscar). La función debe retornar el índice de la **primera aparición** de `elemento` en `miArray`. Si el elemento no se encuentra, debe retornar `-1`.

Ejemplo:

- `encontrarIndice(["rojo", "verde", "azul"], "verde")` debería retornar `1`.
- `encontrarIndice([1, 5, 2, 5], 5)` debería retornar `1`.
- `encontrarIndice(["a", "b"], "c")` debería retornar `-1`.

Ejercicio 8: Duplicar Elementos de un Array

Consigna:
Implementa una función llamada `duplicarArray` que reciba un argumento `originalArray` (un array). La función debe retornar un **nuevo array** que contenga todos los elementos del `originalArray` pero duplicados. Los elementos originales en `originalArray` no deben ser modificados.

Ejemplo:

- `duplicarArray([1, 2, 3])` debería retornar `[1, 2, 3, 1, 2, 3]`.
- `duplicarArray(["a", "b"])` debería retornar `["a", "b", "a", "b"]`.

Consideraciones:
Puedes usar el método `concat()` o el operador `spread (...)`.

Ejercicio 9: Filtrar Números Mayores a un Límite

Consigna:
Crea una función llamada `filtrarMayoresQue` que reciba dos argumentos: `numeros` (un array de números) y `limite` (un número). La función debe retornar un **nuevo array** que contenga solo los números de `numeros` que son estrictamente mayores que `limite`.

Ejemplo:

- `filtrarMayoresQue([10, 5, 20, 15], 12)` debería retornar `[20, 15]`.
- `filtrarMayoresQue([1, 2, 3], 5)` debería retornar `[]`.

Consideraciones:
Deberás iterar sobre el array y usar una condición para filtrar.

Ejercicio 10: Invertir el Orden de un Array

Consigna:
Define una función llamada `invertirOrdenArray` que reciba un argumento `miArray` (un array de cualquier tipo). La función debe retornar un **nuevo array** con los elementos de `miArray` en orden inverso. El array original no debe ser modificado.

Ejemplo:

- `invertirOrdenArray([1, 2, 3])` debería retornar `[3, 2, 1]`.
- `invertirOrdenArray(["a", "b", "c", "d"])` debería retornar `["d", "c", "b", "a"]`.

Consideraciones:
Investiga el método `reverse()` pero ten en cuenta que modifica el array original. Para no modificar el original, podrías crear una copia primero o construir un nuevo array manualmente.

Tema: 5. Objetos Literales

Ejercicio 1: Crear un Objeto Básico

Consigna:
Crea una función llamada `crearPersona` que no reciba ningún argumento. La función debe retornar un nuevo objeto literal con las siguientes propiedades y sus valores:

- `nombre` : un string con el valor `"Juan"`
- `edad` : un número con el valor `30`
- `esEstudiante` : un booleano con el valor `false`

Ejercicio 2: Acceder a Propiedades de un Objeto

Consigna:
Define una función llamada `obtenerNombreProducto` que reciba un argumento `producto` (un objeto). El objeto `producto` siempre tendrá una propiedad `nombre` (string). La función debe retornar el valor de la propiedad `nombre` de ese objeto.

Ejemplo:

- Si `producto` es `{ nombre: "Laptop", precio: 1200 }` , debería retornar `"Laptop"` .

Ejercicio 3: Modificar una Propiedad de Objeto

Consigna:
Escribe una función llamada `actualizarEdad` que reciba dos argumentos: `persona` (un objeto con una propiedad `edad` numérica) y `nuevaEdad` (un número). La función debe modificar la propiedad `edad` del objeto `persona` con el valor de `nuevaEdad` . La función no necesita retornar nada.

Ejemplo:

- Si `persona` es `{ nombre: "Ana", edad: 25 }` y `nuevaEdad` es `26` , `persona` debería modificarse a `{ nombre: "Ana", edad: 26 }` .

Ejercicio 4: Agregar una Nueva Propiedad

Consigna:
Implementa una función llamada `agregarEmail` que reciba dos argumentos: `usuario` (un objeto) y `email` (un string). La función debe agregar una nueva propiedad llamada `correo` al objeto `usuario` con el valor de `email` . La función no necesita retornar nada.

Ejemplo:

- Si `usuario` es `{ id: 1, nombre: "Pedro" }` y `email` es `"pedro@example.com"` , `usuario` debería modificarse a `{ id: 1, nombre: "Pedro", correo: "pedro@example.com" }` .

Ejercicio 5: Eliminar una Propiedad

Consigna:
Crea una función llamada `eliminarPropiedad` que reciba dos argumentos: `objeto` (un objeto) y `clave` (un string que representa el nombre de una propiedad). La función debe eliminar la propiedad con el nombre `clave` del `objeto` . La función no necesita retornar nada.

Ejemplo:

- Si `objeto` es `{ a: 1, b: 2, c: 3 }` y `clave` es `"b"`, `objeto` debería modificarse a `{ a: 1, c: 3 }`.

Ejercicio 6: Iterar sobre Propiedades (claves)

Consigna:
Define una función llamada `listarClaves` que reciba un argumento `miObjeto` (un objeto). La función debe retornar un **array de strings** que contenga solo los nombres (claves) de las propiedades de `miObjeto`.

Ejemplo:

- Si `miObjeto` es `{ marca: "Ford", modelo: "Fiesta", año: 2020 }`, debería retornar `["marca", "modelo", "año"]`.

Consideraciones:
Investiga `Object.keys()`.

Ejercicio 7: Iterar sobre Propiedades (valores)

Consigna:
Escribe una función llamada `listarValores` que reciba un argumento `miObjeto` (un objeto). La función debe retornar un **array** que contenga solo los valores de las propiedades de `miObjeto`.

Ejemplo:

- Si `miObjeto` es `{ nombre: "Laura", edad: 28, ciudad: "Madrid" }`, debería retornar `["Laura", 28, "Madrid"]`.

Consideraciones:
Investiga `Object.values()`.

Ejercicio 8: Comprobar Existencia de Propiedad

Consigna:
Implementa una función llamada `existePropiedad` que reciba dos argumentos: `objeto` (un objeto) y `nombrePropiedad` (un string). La función debe retornar `true` si `objeto` tiene una propiedad con el nombre `nombrePropiedad`, y `false` en caso contrario.

Ejemplo:

- `existePropiedad({ a: 1, b: 2 }, "a")` debería retornar `true`.
- `existePropiedad({ x: 10 }, "y")` debería retornar `false`.

Consideraciones:
Puedes usar el operador `in` o el método `hasOwnProperty()`.

Ejercicio 9: Copiar un Objeto (Superficial)

Consigna:
Crea una función llamada `copiarObjeto` que reciba un argumento `objetoOriginal` (un objeto).

La función debe retornar una **nueva copia superficial** de `objetoOriginal`. Los cambios en el objeto original no deben afectar la copia y viceversa.

Ejemplo:

- Si `objetoOriginal` es `{ a: 1, b: 2 }`, la función debería retornar `{ a: 1, b: 2 }`, pero siendo un objeto distinto en memoria.

Consideraciones:
Puedes usar `Object.assign()` o el operador `spread (...)`.

Ejercicio 10: Combinar Dos Objetos

Consigna:
Define una función llamada `combinarObjetos` que reciba dos argumentos: `obj1` (un objeto) y `obj2` (otro objeto). La función debe retornar un **nuevo objeto** que contenga todas las propiedades de `obj1` y `obj2`. Si ambos objetos tienen propiedades con el mismo nombre, las propiedades de `obj2` deben sobrescribir las de `obj1`.

Ejemplo:

- `combinarObjetos({ a: 1, b: 2 }, { b: 3, c: 4 })` debería retornar `{ a: 1, b: 3, c: 4 }`.

Consideraciones:
Puedes usar `Object.assign()` o el operador `spread (...)`.

Tema: 6. Combinación de Conceptos

Ejercicio 1: Calcular Promedio de Edades

Consigna:
Crea una función llamada `calcularPromedioEdades` que reciba un argumento `personas` (un array de objetos). Cada objeto en el array representa una persona y tiene una propiedad `edad` (número). La función debe retornar el promedio de todas las edades. Si el array está vacío, debe retornar `0`.

Ejemplo:

- Si `personas` es `[{ nombre: "Ana", edad: 20 }, { nombre: "Pedro", edad: 30 }]`, debería retornar `25`.

Ejercicio 2: Encontrar la Persona Mayor

Consigna:
Define una función llamada `encontrarPersonaMayor` que reciba un argumento `personas` (un array de objetos, donde cada objeto tiene una propiedad `edad`). La función debe retornar el **objeto persona** que tenga la edad más alta. Si el array está vacío, debe retornar `null`.

Ejemplo:

- Si `personas` es `[{ nombre: "Juan", edad: 25 }, { nombre: "Maria", edad: 35 }, { nombre: "Luis", edad: 30 }]`, debería retornar `{ nombre: "Maria", edad: 35 }`.

Ejercicio 3: Filtrar Productos por Precio

Consigna:
Escribe una función llamada `filtrarProductosPorPrecio` que reciba dos argumentos: `productos` (un array de objetos, donde cada objeto tiene propiedades `nombre` (string) y `precio` (número)) y `precioMaximo` (un número). La función debe retornar un **nuevo array** que contenga solo los objetos `producto` cuyo `precio` sea menor o igual a `precioMaximo`.

Ejemplo:

- Si `productos` es `[{ nombre: "Laptop", precio: 1200 }, { nombre: "Teclado", precio: 75 }, { nombre: "Mouse", precio: 30 }]` y `precioMaximo` es `100`, debería retornar `[{ nombre: "Teclado", precio: 75 }, { nombre: "Mouse", precio: 30 }]`.

Ejercicio 4: Contar Productos por Categoría

Consigna:
Implementa una función llamada `contarProductosPorCategoria` que reciba un argumento `productos` (un array de objetos). Cada objeto `producto` tiene propiedades `nombre` (string) y `categoria` (string). La función debe retornar un **objeto** donde las claves sean los nombres de las categorías y los valores sean la cantidad de productos en cada categoría.

Ejemplo:

- Si `productos` es `[{ nombre: "Leche", categoria: "Lácteos" }, { nombre: "Pan", categoria: "Panadería" }, { nombre: "Queso", categoria: "Lácteos" }]`, debería retornar `{ "Lácteos": 2, "Panadería": 1 }`.

Ejercicio 5: Actualizar Stock de Productos

Consigna:
Crea una función llamada `actualizarStock` que reciba dos argumentos: `inventario` (un array de objetos, donde cada objeto tiene `id` (número), `nombre` (string) y `stock` (número)) y `ventas` (un array de objetos, donde cada objeto tiene `productoId` (número) y `cantidadVendida` (número)).
La función debe modificar el `inventario` reduciendo el `stock` de cada producto según las `ventas` realizadas. La función no necesita retornar nada.

Ejemplo:

- Si `inventario` es `[{ id: 1, nombre: "A", stock: 10 }, { id: 2, nombre: "B", stock: 5 }]` y `ventas` es `[{ productoId: 1, cantidadVendida: 2 }, { productoId: 2, cantidadVendida: 1 }]`, entonces `inventario` debería modificarse a `[{ id: 1, nombre: "A", stock: 8 }, { id: 2, nombre: "B", stock: 4 }]`.

Consideraciones:
Deberás iterar sobre `ventas` y, para cada venta, encontrar el producto correspondiente en `inventario` para actualizar su `stock`.

Ejercicio 6: Formatear Lista de Tareas

Consigna:
Define una función llamada `formatearTareas` que reciba un argumento `tareas` (un array de objetos). Cada objeto `tarea` tiene propiedades `descripcion` (string) y `completada` (booleano). La función debe retornar un **array de strings** donde cada string represente una tarea. Si la tarea está completada, debe ir prefijada con `"[X]"` . Si no, con `"[ ]"` .

Ejemplo:

- Si `tareas` es `[{ descripcion: "Comprar pan", completada: false }, { descripcion: "Pagar facturas", completada: true }]` , debería retornar `["[ ] Comprar pan", "[X] Pagar facturas"]` .

Ejercicio 7: Encontrar Elemento con Múltiples Criterios

Consigna:
Escribe una función llamada `buscarUsuario` que reciba un argumento `usuarios` (un array de objetos) y `criterios` (un objeto). Cada objeto `usuario` en el array tiene propiedades como `nombre` (string), `edad` (número) y `ciudad` (string). La función debe retornar el **primer objeto** `usuario` que coincida con **todos** los `criterios` proporcionados en el objeto `criterios` . Si no se encuentra ninguna coincidencia, debe retornar `null` .

Ejemplo:

- Si `usuarios` es `[{ nombre: "Ana", edad: 25, ciudad: "Buenos Aires" }, { nombre: "Pedro", edad: 30, ciudad: "Córdoba" }]` y `criterios` es `{ edad: 25, ciudad: "Buenos Aires" }` , debería retornar `{ nombre: "Ana", edad: 25, ciudad: "Buenos Aires" }` .
- Si `criterios` es `{ ciudad: "Rosario" }` , debería retornar `null` .

Consideraciones:
Necesitarás iterar sobre el array de usuarios y, para cada usuario, verificar si todas las propiedades en el objeto `criterios` coinciden con las del usuario.

Ejercicio 8: Calcular Total de Carrito de Compras

Consigna:
Implementa una función llamada `calcularTotalCarrito` que reciba un argumento `carrito` (un array de objetos). Cada objeto en el `carrito` tiene propiedades `nombre` (string), `precioUnitario` (número) y `cantidad` (número). La función debe retornar el costo total de todos los artículos en el carrito.

Ejemplo:

- Si `carrito` es `[{ nombre: "Camisa", precioUnitario: 20, cantidad: 2 }, { nombre: "Pantalón", precioUnitario: 50, cantidad: 1 }]` , debería retornar `90` (*20*2 + *50*1).

Ejercicio 9: Ordenar Estudiantes por Calificación Promedio

Consigna:
Crea una función llamada `ordenarEstudiantesPorPromedio` que reciba un argumento `estudiantes` (un array de objetos). Cada objeto `estudiante` tiene propiedades `nombre` (string) y `calificaciones` (un array de números). La función debe retornar un **nuevo array de objetos estudiante**, ordenados de forma descendente por su promedio de calificaciones.

Ejemplo:

- Si `estudiantes` es `[{ nombre: "A", calificaciones: [8, 9] }, { nombre: "B", calificaciones: [7, 6] }]` , debería retornar `[{ nombre: "A", calificaciones: [8, 9] }, { nombre: "B", calificaciones: [7, 6] }]` (o una copia ordenada, como `{ nombre: "A", promedio: 8.5 }, { nombre: "B", promedio: 6.5 }`).

Consideraciones:
Primero, para cada estudiante, necesitarás calcular su promedio. Luego, puedes usar el método `sort()` de los arrays para ordenar los estudiantes basándote en esos promedios.

Ejercicio 10: Generar Reporte de Ventas por Vendedor

Consigna:
Define una función llamada `generarReporteVentas` que reciba un argumento `ventas` (un array de objetos). Cada objeto `venta` tiene propiedades `vendedor` (string), `producto` (string) y `monto` (número). La función debe retornar un **objeto de reporte** donde las claves sean los nombres de los vendedores y los valores sean el monto total de ventas que cada vendedor ha realizado.

Ejemplo:

- Si `ventas` es `[{ vendedor: "Juan", monto: 100 }, { vendedor: "Maria", monto: 150 }, { vendedor: "Juan", monto: 50 }]` , debería retornar `{ "Juan": 150, "Maria": 150 }` .

Consideraciones:
Deberás iterar sobre el array de ventas y, para cada venta, acumular el monto en un objeto de reporte, creando nuevas propiedades para los vendedores si aún no existen.