INTERSECTION-LIST.

Ejercicio de nodos y listas.
Tenemos dos listas (pongo un ejemplo) con los siguientes datos.
Lista1:

- manzana
- pera
    - conferencia
        - conferenciaverde
        - conferenciaamarilla
        - conferenciaroja
    - peradeagua
    - peritadulce
        - Dimas
        - Gestas
        - Malakatos
        - Jose
- limon
- limon
- LIMON

Lista2:

- manzana
- manzana
- manzana
- manzana
- pera
- limon

Realiza un programa que muestre por consola la INTERSECCIÓN de las dos listas SIN DUPLICADOS de forma ordenada:

- LIMON
- MANZANA
- PERA

Lista1 y Lista2 son los identificadores en el DOM de ambas listas.
No puedes recuperar objetos del DOM por tag.
No puedes utilizar el método filter() ni la palabra clave IN.
Prueba tu programa con más listas para comprobar si funciona correctamente.

Corrección:

- Tu programa funciona: (5-10 puntos)
- Tu programa no funciona: (0-4,99 puntos)

Se valorará de tu programa:

- Si hay errores de consola. Un error de consola evidencia un error estructural que no has sabido corregir por lo tanto se penalizará de forma grave. Entrega tu programa sin errores de consola.
- La eficiencia.
- La modularidad (alta cohesión y bajo acoplamiento).
- Que el código sea limpio, flexible, reutilizable y mantenible.
- Sigue los principios SOLID.
- Otros criterios.

SOLUCIÓN PASO A PASO

PASO 1. GENERAR HTML

Lo primero será generar el documento HTML con las dos listas. El resultado del mismo será el siguiente:

```
<!DOCTYPE html>
<html>
<head>
 <title>INTERSECTION LIST</title>
</head>
<body>
PRIMERA LISTA:
<ul id="lista1">
 <li>manzana</li>
 <li>pera</li>
 <ul>
   <li>conferencia</li>
   <ul>
     <li>conferenciaverde</li>
     <li>conferenciaamarilla</li>
     <li>conferenciaroja</li>
   </ul>
   <li>peradeagua</li>
   <li>peritadulce</li>
     <ul>
     <li>Dimas</li>
     <li>Gestas</li>
```

```
      <li>Malakatos</li>
      <li>Jose</li>
    </ul>
  </ul>
  <li>limon</li>
  <li>limon</li>
  <li>LIMON</li>
</ul>
Segunda lista:
<ul id="lista2">
  <li>manzana</li>
  <li>manzana</li>
  <li>manzana</li>
  <li>manzana</li>
  <li>pera</li>
  <li>limon</li>
</ul>


  <script>

//el contenido del script lo colocaré aquí

  </script>

</body>
</html>
```

PASO 2. ¿QUÉ VA A HACER EL SCRIPT?

A continuación se muestra los pasos que tiene que realizar nuestro script:

1. Declarar las listas
2. Insertar las dos listas del DOM en 2 arrays diferentes.
3. Ordenar las dos listas y eliminar duplicados. Si ordeno las listas puedo quitar los duplicados de forma fácil porque estarán juntos.
4. Crear una nueva lista concatenando el resultado de las dos anteriores.
5. Ordenar la nueva lista
6. Los duplicados en esa nueva lista serán la intersección que voy buscando.
7. Mostrar el resultado.

¿Se podría programar de otra forma aunando funciones?
Sin duda sí, pero al hacer las operaciones más atómicas es más fácil de programar y depurar. Además esta solución se sustenta en tres operaciones muy sencillas de crear y son muy mantenibles.

PASO 3. PROGRAMACIÓN ESTRUCTURA (BACKBONE) DEL SCRIPT.

Para los pasos citados anteriormente se muestran codificados en JavaScript:

```
//Declarar las listas
 var lista1=[];
 var lista2=[];
 var newlista=[];

//Insertar las dos listas del DOM en 2 arrays diferentes.
 inserta(document.getElementById("lista1"),lista1);
 inserta(document.getElementById("lista2"),lista2);

//Ordenar las dos listas y eliminar duplicados. Si ordeno las listas
puedo quitar los duplicados de forma fácil porque estarán juntos.
 lista1.sort();
 eliminaDuplicados(lista1);
 lista2.sort();
 eliminaDuplicados(lista1);

//Crear una nueva lista concatenando el resultado de las dos
anteriores.
 lista1=lista1.concat(lista2);

//Ordenar la nueva lista
 lista1.sort();

//Los duplicados en esa nueva lista serán la intersección que voy
buscando.
 buscaDuplicados(lista1,newlista);
 eliminaDuplicados(newlista);

//mostrar el resultado
 console.log(newlista);
```

PASO 4. CREACIÓN DE LAS FUNCIONES DEL BACKBONE

Ahora solamente queda programar las funciones del backbone. Solamente son tres (`inserta, eliminaDuplicados y buscaDuplicados`).

Se muestra el código de las mismas a continuación:

```
//funciones del backbone
 function buscaDuplicados(lista, hasta){
     for (let i=0;i<lista.length-1;i++){
       if (lista[i]==lista[i+1]){
         hasta.push(lista[i]);
       }
     }
 }

 function eliminaDuplicados(lista){
   let dup=1;
   while (lista.length>1 && dup==1){
     dup = 0;
     for (let i=0;i<lista.length-1;i++){
       if (lista[i]==lista[i+1]){
         lista.splice(i,1);
         dup=1;
       }
     }
   }
 }

 function inserta(desde,hasta){
   if (desde.nodeName ==
"LI"){hasta.push(desde.innerHTML.toUpperCase());}
   for (let i=0;i<desde.children.length;i++){
     inserta(desde.children[i],hasta);
   }
 }
```

La función inserta como se puede observar es una función recursiva que va insertando todos los nodos LI des de una lista del DOM hasta un array en nuestro script.
Las funciones se han programado con parámetros por si existiese la necesidad de reutilizarlas en algún otro script.

PASO 5. REALIZACIÓN DE PRUEBAS.

No hay que olvidar realizar todo tipo de pruebas posibles con distintos elementos diferentes e iguales en ambas listas para comprobar que el script está funcionando correctamente.

SOLUCIÓN COMPLETA

```
<!DOCTYPE html>
<html>
<head>
 <title>INTERSECTION LIST</title>
</head>
<body>
PRIMERA LISTA:
<ul id="lista1">
 <li>manzana</li>
 <li>pera</li>
 <ul>
   <li>conferencia</li>
   <ul>
     <li>conferenciaverde</li>
     <li>conferenciaamarilla</li>
     <li>conferenciaroja</li>
   </ul>
   <li>peradeagua</li>
   <li>peritadulce</li>
     <ul>
     <li>Dimas</li>
     <li>Gestas</li>
     <li>Malakatos</li>
     <li>Jose</li>
     <li>Jose</li>
   </ul>
 </ul>
 <li>limon</li>
 <li>limon</li>
 <li>LIMON</li>
</ul>
```

```
Segunda lista:
<ul id="lista2">
 <li>manzana</li>
 <li>manzana</li>
 <li>manzana</li>
 <li>manzana</li>
 <li>pera</li>
 <li>limon</li>
 <li>jose</li>
 <li>jose</li>
</ul>


 <script>
//Declarar las listas
 var lista1=[];
 var lista2=[];
 var newlista=[];

//Insertar las dos listas del DOM en 2 arrays diferentes.
 inserta(document.getElementById("lista1"),lista1);
 inserta(document.getElementById("lista2"),lista2);

//Ordenar las dos listas y eliminar duplicados. Si ordeno las listas
puedo quitar los duplicados de forma fácil porque estarán juntos.
 lista1.sort();
 eliminaDuplicados(lista1);
 lista2.sort();
 eliminaDuplicados(lista1);

//Crear una nueva lista concatenando el resultado de las dos
anteriores.
 lista1=lista1.concat(lista2);

//Ordenar la nueva lista
 lista1.sort();

//Los duplicados en esa nueva lista serán la intersección que voy
buscando.
 buscaDuplicados(lista1,newlista);
 eliminaDuplicados(newlista);

//mostrar el resultado
```

```
 console.log(newlista);

//funciones del backbone
 function buscaDuplicados(lista, hasta){
     for (let i=0;i<lista.length-1;i++){
       if (lista[i]==lista[i+1]){
         hasta.push(lista[i]);
       }
     }
 }

 function eliminaDuplicados(lista){
   let dup=1;
   while (lista.length>1 && dup==1){
     dup = 0;
     for (let i=0;i<lista.length-1;i++){
       if (lista[i]==lista[i+1]){
         lista.splice(i,1);
         dup=1;
       }
     }
   }
 }

 function inserta(desde,hasta){
   if (desde.nodeName ==
"LI"){hasta.push(desde.innerHTML.toUpperCase());}
   for (let i=0;i<desde.children.length;i++){
     inserta(desde.children[i],hasta);
   }
 }

 </script>

</body>
</html>
```