Prueba teorica para GPTX

1.- Está recibiendo dos flujos de datos de dos sensores de temperatura (2 observables que devuelven números enteros), ¿qué operador de RxJS utilizaría para recibir estos dos datos en la misma suscripción?

Respuesta:

Para recibir ambos flujos de datos de dos sensores, se puede utilizar el operador `combineLatest`, para mantener ambas suscripciones en una.

El código quedaría algo parecido a esto:

```
import { combineLatest, Observable } from 'rxjs';

const sensor1: Observable = dataSensor1();
const sensor2: Observable = dataSensor2();

combineLatest([sensor1, sensor2]).subscribe(([data1, data2]) => {
    //Hacer algo con data 1 y data 2
})
```

2.- Si tiene dos llamadas al servidor y la segunda llamada depende de la primera, ¿cómo manejaría con RxJS esta secuencia de llamadas?

Respuesta: Metiendo una llamada dentro de la otra y realizando la segunda en la suscripción de la primera, de esta forma: (suponiendo que esto fuera código de Angular)

```
constructor(private userService: UserService) {}

ngOnInit(){
    this.userService.getUser().subscribe({
        next: (data) => {
            //Suponiendo que dependa de los datos de getUser
```

```
            this.userService.segundaLlamada(data).subscribe({
                next: (data2) => {
                    //algo con esta data
                }
            })
        },
    })
}
```

3.- Tengo en un servidor un archivo de texto que está en minúscula y ocupa 2GB en el disco duro, pero le solicitan que todo el archivo debe ser pasado a mayúsculas, ¿cómo lo haría?

Dependiendo de la situacion: Si tengo acceso a una maquina con GitBash o con WSL, simplemente usando el comando `tr '[:lower:]' '[:upper:]'`

4.- Tiene un arreglo de strings los cuales deben ser filtrados por su longitud mayor a dos y a la vez convertidos a un array de enteros con la longitud de cada string, ¿cómo lo haría? Ejemplo de entrada y salida: [“hola”, “mundo”, “es”, “una”, “prueba”] => [4, 5, 3, 6]

En caso de contar con Javascript, solo es necesario usar el metodo filter para filtrar las palabras que tienen mas de 2 letras, y el metodo map para transformarlas, de esta forma:

```
const palabras = ["hola", "mundo", "es", "una", "prueba"];

const longitudes = palabras.filter(palabra => palabra.length > 2).map(palabra => palabra.length);

console.log("longitudes", longitudes);
```

para no utilizar ningun metodo como filter o map, sería cuestion de declarar un arreglo vacio. Recorrer el arreglo original, validar si cada elemento del arreglo tiene mas de dos caracteres, y de ser el caso, ingresar la longitud del elemento en el declarado anteriormente, de esta forma

```
const palabras = ["hola", "mundo", "es", "una", "prueba"];

function getLongitudes() {
    let longitudes = [];
    for (let i = 0; i < palabras.length; i++) {
        if (palabras[i].length > 2) {
            longitudes.push(palabras[i].length);
        }
    }
    return longitudes;
}

console.log("longitudes", getLongitudes());
```

5.- Tiene un arreglo de números, los cuales pueden ser o no repetidos, ¿cómo eliminaría los repetidos? ¿Cómo los ordenaría en forma ascendente? Ejemplo de entrada y salida: [1, 2, 5, 10, 8, 8, 1, 3, 4, 5] => [1, 2, 3, 4, 5, 8, 10]

Al igual que el ejemplo anterior, solo es necesario usar tanto el metodo filter, para filtrar aquellos numeros que ya se encuentran en el arreglo, mediante el metodo array.indexOf.

Se puede saber que un numero se encuentra ahí, porque indexOf retorna -1 cuando no se encuentra un ítem en el arreglo.

De esta forma:

```
const numeros = [1, 2, 5, 10, 8, 8, 1, 3, 4, 5];

const numerosUnicos = numeros.filter((valor, indice, array) => {
  return array.indexOf(valor) === indice;
})

const numerosOrdenados = numerosUnicos.sort((a, b) => a - b);

console.log(numerosOrdenados);
```

En caso de que no se quiera utilizar ninguno de estos metodos, se necesita una solucion parecida a la anterior. Primero se tiene que declarar un arreglo vacio, recorrer el arreglo ya existente, y mediante el metodo indexOf, checar si existe cada ítem. Si existe, se ingresa al arreglo declarado.

Después, solo es necesario ordenarlos mediante un bubble sort. De esta forma:

```
const numeros = [1, 2, 5, 10, 8, 8, 1, 3, 4, 5];

function filtrarNumerosRepetidos(numeros) {
    const numerosUnicos = [];
    for (let i = 0; i < numeros.length; i++) {
        if (numerosUnicos.indexOf(numeros[i]) === -1) {
            numerosUnicos.push(numeros[i]);
        }
    }
    return numerosUnicos;
}

function ordenarNumeros(numeros) {
   for (let i = 0; i < numeros.length; i++) {
       for (let j = 0; j < numeros.length - 1; j++) {
           if (numeros[j] > numeros[j + 1]) {
               const acumulador = numeros[j];
               numeros[j] = numeros[j + 1];
               numeros[j + 1] = acumulador;
           }
       }
   }
    //Bubble sort :D
    return numeros;
}

const numerosUnicos = filtrarNumerosRepetidos(numeros);

const numerosOrdenados = ordenarNumeros(numerosUnicos);
```