



# RxJS

## **Ejecución asíncrona de código en Angular: RxJS**

**Librería de uso común para trabajar con flujos asíncronos de datos y eventos.**

**Disponible en muchos lenguajes: RxJAVA, RxCpp, RxNET, RxPy**

## RxJS - Elementos:

**Observable:** El observable representa un flujo de datos, una colección de eventos que se pueden emitir en algún momento futuro.

**Observer:** Los observers son objetos que están escuchando el flujo de datos y actúan sobre los valores que éste emite.

**Subscription:** vinculan observables y observers. NOTA: un observable solo se ejecuta si existe algún observer!

**Operador:** Los operadores son funciones puras que te permiten trabajar con el flujo de eventos mediante programación funcional.

## RxJS - creación, subscripción y operadores:

```
import { from } from 'rxjs';
import { filter } from 'rxjs/operators';

//create the "Hello world" observable
const myObs = from("Hello world");

//apply the filter operator
const filteredObs = myObs.pipe(
  filter(char => char !== ' ')
);

//subscribe to the filtered observable
const subscription = filteredObs.subscribe(char => console.log(char));
```

## RxJS - Pipe, encadendo operadores:

```
import { from } from 'rxjs';
import { filter, map } from 'rxjs/operators';

//create the "Hello world" observable
const myObs = from("Hello world");

const filteredObs = myObs.pipe(
  filter(char => char !== ' '),
  map(char => char.toUpperCase())
);

//subscribe to the filtered observable
const subscription = filteredObs.subscribe(char => console.log(char));
//output: H E L L O W O R L D
```

## RxJS - cancelar subscripciones:

**Los observables lanzan eventos asíncronamente, hasta que se completa la fuente de datos o hasta que cancelamos la subscripción.**

```
const subscription = filteredObs.subscribe(char => {  
  console.log(char);  
  if(char == 'e')  
    subscription.unsubscribe();  
});
```

# LABORATORIO – FIREBASE

## RxJS - Elementos:



`const o=interval(1000).pipe(take(4)); //UTIL!`

## RxJS - Promesas:

**Se ejecutan asíncronamente, UNA sola vez!**

```
let p:Promise<any>=myObs3.toPromise();  
console.log("-inicio-");  
  
p.then(x=>  
{  
    console.log(x);  
});  
  
console.log("-fin-");
```



## RxJS - Promesas:

**Sincronizando la ejecución de un promesa:**

```
function async myFunc()
{
    let p:Promise<any>=myObs3.toPromise();
    console.log("-inicio-");

    await p.then(x=>
    {
        console.log(x);
    });

    console.log("-fin-");
}
```