**Curso de programación reactiva RxJs.**

1. **Sección 2: Introducción a las extensiones reactivas y a la programación reactiva.**
2. **Sección 3: Observables.**
3. **Sección 4: Funciones para crear Observables.**
4. **Sección 5: Operadores básicos.**
5. **Sección 6: Operadores no tan comunes.**
6. **Sección 7: Operadores que trabajan con tiempo.**
7. **Sección 8: Ajax - Peticiones ajax usando RxJs/ajax.**
8. **Sección 9: Operadores de transformación.**
9. **Sección 10: Operadores y métodos de combinación de  
   observables.**
10. **Sección 11: Ejercicios de reforzamiento.**
11. **Sección 12: Fin del curso.**

**Sección 2: Introducción a las extensiones reactivas y a la programación reactiva.**

En esta sección aprenderemos sobre:

* Conceptos generales de RxJs y las extensiones reactivas
* ¿Qué es ReactiveX?
* ¿Cómo funciona?
* ¿Qué es el patrón observable?
* ¿Cómo leer los diagramas de canicas?
* Y otros temas

Es una sección corta llena de teoría, pero es indispensable que lo aprendamos para poder tener un mejor panorama de la programación reactiva.

**Video #07: conceptos generales.**

En esta clase empezamos con la pregunta “¿por qué usar extensiones reactivas?”, la respuesta que nos dan es porque desaseamos información en tiempo real, hoy en día no es viable para los usuarios hacer una recarga en la página para obtener información de manera de tiempo real sin que el usuario deba interactuar con un botón para obtener esa información.

¿En qué momento es correcto utilizar extensiones reactivas?

* Eventos de interfaz de usuario.
* Cuando es necesario notificar sobre cambios en un objeto.
* Comunicaciones por sockets.
* Cuando necesitamos trabajar con flujos de información.

Los pilares de la programación reactiva son los Observables Subscribers Operators.

**Observables:**

* Son la fuente de información.
* Pueden emitir múltiples valores, sólo uno o ninguno.
* Pueden emitir errores.
* Pueden ser infinitos o finitos. (completarse)
* Pueden ser síncronos o asíncronos.

**Subscribers**

* Se subscriben a un observable, es decir, estar pendiente de lo que realizar el observable.
* Consumen / observan la data del observable.
* Pueden recibir los errores y eventos del observable.
* Desconocen todo lo que se encuentra detrás del observable.

**Operators**

* Usados para transformar Observables (map, group, scan).
* Usados para filtrar observables (filter, distinct, skip, debounce).
* Usados para combinar observables.
* Usados para crear nuevos observables.

**Video #08: ¿Qué es ReactiveX ?,¿Como funciona?**

ReactiveX funciona con las tres siguientes bases, el patrón observador, el patrón iterador y la programación funcional.

**Observer Pattern**: es un patrón de diseño de software que define una dependencia del tipo de uno a muchos entre objetos, de manera que cuando uno de los objetos cambia de estado, notifica este el cambio a todos los dependientes.

**Iterador Pattern:** En POO, el patrón iterador define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección.

**Programación funcional:** Es básicamente, crear un conjunto de funciones que tengan un objetivo específico. Es decir, si tengo una función que reciba ‘A’ y retorna ‘A+1’, siempre que yo llame esa función retornara ‘A+1’.

Sin efectos secundarios, sin mutar la data.

**Video #09: ¿Cómo leer los diagramas de canicas?**

Para identificar un observable se usa el signo de dólar. Esto es un estándar para identificar observables en nuestro código.

Este observador trabajará con una línea de tiempo donde se representará en canicas los valores emitidos por este. diagrama canicas




**Nota:** Al momento de trabajar con extensiones reactivas es recomendable que la librería este en las dependencias de desarrollo.

**Sección 3: Observables.**

Estamos a las puertas de la sección que nos enseñará sobre los observables, el primer eslabón de la programación reactiva, veremos temas como:

1. Observers
2. Subscriber
3. Unsubscribe
4. Subjects
5. Hot y Cold Observables
6. Observables en cadena
7. Y más

Antes de utilizar funciones que crean observables, es importante comprender las bases para entrar a temas de la generación automática de observables.

**Video #13: Nuestro primer observable.**

Podemos crear un observable de la siguiente forma:



este es el ejemplo de un observable, podemos ver que para identificar que es un observable utilizamos el signo de dólar, dentro de los observables existe un objeto llamado Subscriber, este permite crear suscripciones que nos permitiría estar al pendiente de las emisiones de nuestro observable.

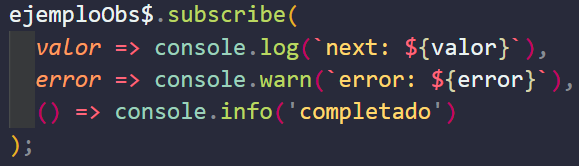
Para ejecutar un observable debemos suscribirnos a él para que nos notifique los cambios de nuestro observable.



Con el Subscriber.next() podemos emitir información a nuestras suscripciones, y con Subscriber.Complete() detenemos las emisiones de nuestro observable.

**Nota:**  es recomendado que nuetros obserbable debe tener la expecificacion del tipo de datos que fluyen en el.

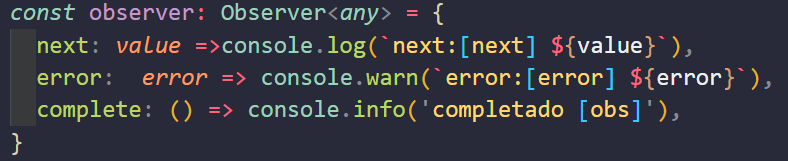
**Video 14#: Observer y Subscriber.**



Existe tres posibles funciones que puede recibir nuestro Subscriber, la primera, es la respuesta correcta de nuestro observable es esta el que procesa el next del observable, la segunda, es el posible error de nuestro observable a la hora de ejecutarse, la tercera una función que no recibe ningún parámetro y notifica que el observable ya termino su ejecución.

Estas funciones son los argumentos de nuestro Subscriber.

Existe otra forma de recibir los datos de nuestro observable y es crear un objeto que obtenga nuestros argumentos. Ejemplo:



Me parece que esta forma es un poco más ordenada y ocupa mucho menos código en nuestro Subscriber ya que solo debemos enviar el objeto como parámetro.

**Video #16: Subscription y Unsubscribe.**

Para realizar una suscripción a un observable debemos llamar a nuestro observable y agregarle el método subscribe. Ejemplo

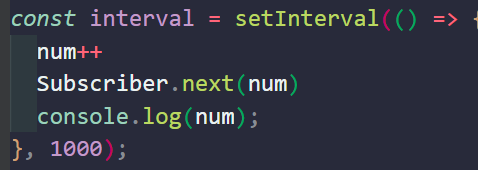


Para cancelar nuestra suscripción de nuestro observable es un tanto diferente, pues debemos crear una variable que alojara nuestra suscripción, esta variable debe ser de tipo Subscription, una vez realizada este proceso a nuestra variable la invocamos junto al método Unsubscribe. Ejemplo:





Primero guardamos el dato y después lo llamamos y le pasmos el método

**Nota:** conozco una nueva función de tiempo llamada ‘**setInterval**’ la cual se ejecuta su contenido cada espacio de tiempo que nosotros especifiquemos. Ejemplo: 

Y podemos terminar su ejecución de la siguiente forma



**Video #18 y #19: Subject part-1 y Subject part-2.**

Esta clase la iniciamos con una pregunta a resolver la cual es, que pasa cuando tenemos más de una suscripción a un observable y deseamos que los datos que arroje a cada suscripción sean los mismos.

Es aquí cuando llega el Observer **Subject** para solucionar nuestros problemas, ya que normalmente los Observer simples generar una ejecución independiente para cada suscripción, el Subject nos permite generar multidifusión lo que facilita para tener los mismos datos en cada una de nuestras suscriciones, en resumen, el Subject permite que cada una de nuestras suscripciones emita los mismos datos.

Para más información puede consultar la documentación <https://rxjs-dev.firebaseapp.com/guide/subject#subject>.

características del Subject

1- casteo múltiple: se refiere que muchas

suscripciones van estar sujetas al observable

y servirán para distribuir la misma data a

cada una de ellas

2- También es un Observer

3- maneja Next, Error, Complete

**Cold Observable:** cunado la data es producida por el mismo observable.

**Hot Observable:** Cunado la data es producida fuera del observable.

**Sección #4: Funciones para crear Observables.**

¡Es momento de utilizar funciones para crear observables!, esto nos ayudará a reducir mucho el tiempo a la hora de crear nuevos observables.

Los temas principales de la sección son:

* of
* fromEvent
* interval
* timer
* asyncScheduler

Mas adelante veremos otras funciones que crean observables, pero por ahora enfoquemos nuestra atención en las básicas y más comunes.

**Video # 23:** **OF**.

La función of devuelve un observable en base a un listado de elemento.

Este operador va emitir los valores uno por uno de manera asíncrona, cuando termina el último valor se finaliza la ejecución del observable.

**Nota**: cada argumento debe ir separado por una coma, en el caso que nuestro argumento sea un arreglo podremos usar el operador spread lo que nos va a desintegrar dicho argumento.



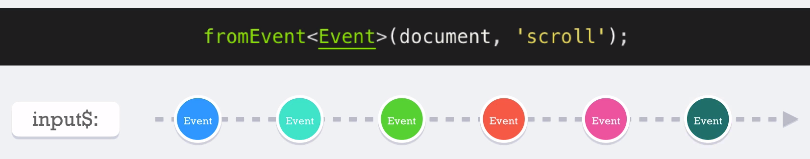
En resumen, el OF emite un observable por cada argumento que se le ingrese.

pero recomendable tener un tipado estricto de datos.

Para mayor información es recomendable que ingrese a la documentación.

[**https://rxjs-dev.firebaseapp.com/api/index/function/of**](https://rxjs-dev.firebaseapp.com/api/index/function/of)

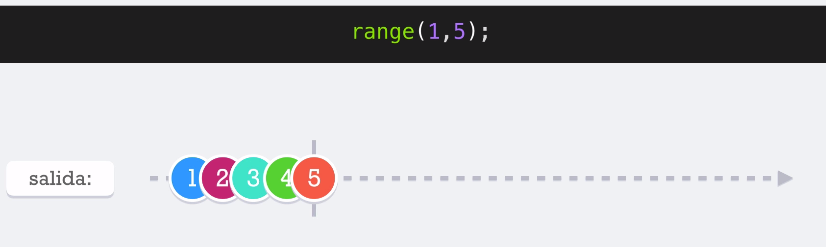
**video #24: fromEvent.**



En FromEvent emite eventos en forma observables lo que permite suscribirnos a sus cambios y controlar la data que recibe las suscripciones. Para mayor información entra a la documentación oficial.

<https://rxjs-dev.firebaseapp.com/api/index/function/fromEvent>.

**Video #25: Range.**

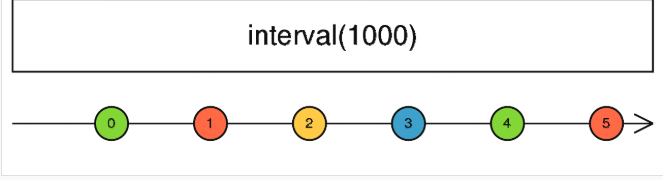


Range nos crea un observable que emite secuencia de números en base a un rango. Que por defecto son síncronos, pero estos también pueden ser asíncronos, mediante otro proceso llamado scheduler.

Este observable recibe dos argumentos obligatorios, el primero, la primera posición del rango que por defecto esta es 0, el segundo, luego cuantos elementos se desea mostrar. El ultimo es un argumento opcional el cual es el scheduler.

**Video # 26: Interval y timer.**

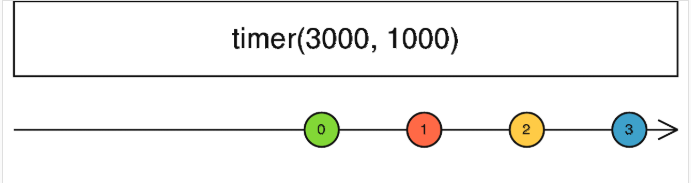
Crea un observable cada intervalo de tiempo especificado, en resumen, pro ejemplo enviamos por parámetro un segundo, apenas se complete ese segundo comenzara a emitir un secuencia de números accedente cada intervalo de tiempo asignado.



Para más información consulte la documentación:  
 <https://rxjs-dev.firebaseapp.com/api/index/function/interval>.

**Timer:**

Esta función, aunque muy parecida a al interval su funcionamiento es diferente, porque, aunque emita valor en un tiempo determinado a esta función le diremos en cuanto tiempo comience emitir y cada cuando queremos que emita nuestro valor.



Para más información ingrese a la documentación oficial

[**https://rxjs-dev.firebaseapp.com/api/index/function/timer**](https://rxjs-dev.firebaseapp.com/api/index/function/timer)

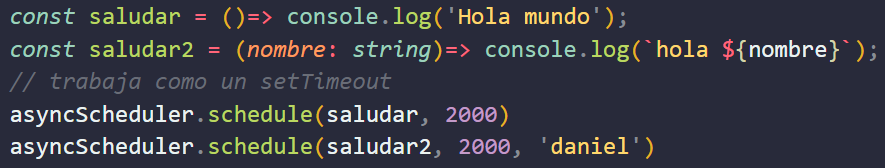
**Nota:** Recordemos que estas funciones se activan o ejecutan tan pronto el stock de col backs de JavaScript lo permita.

**Video #27: timer – configuraciones especiales.**

En esta clase nos mostro algunas posibles utilizaciones del timer, para más información diríjase a observables -> 07- interval&Timer, en la cual está el ejercicio realizado.

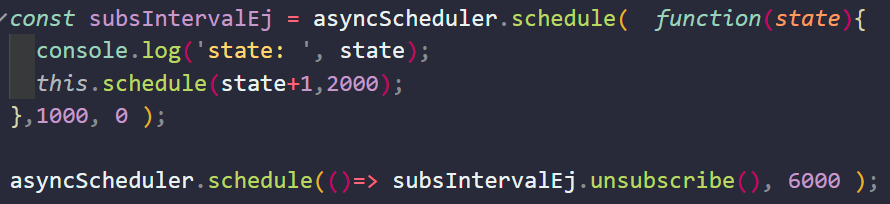
**Video #28: asyncScheduler.**

el asyncScheduler realiza el comportamiento del setTImeout y setInterval, pero la ventaja que tiene este método sobre los otros dos es que puede generar un Subscriber.



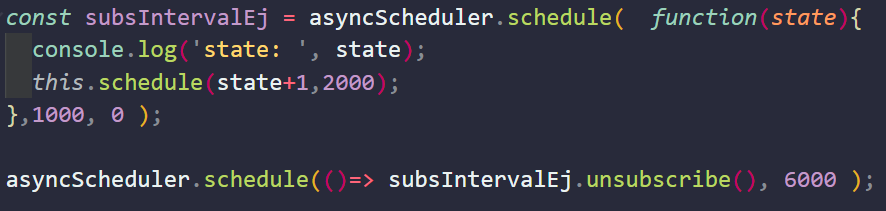
El código de arriba es un ejemplo de como trabaja el asyncScheduler como el setTimeout.

El siguiente ejemplo te mostrare como trabajar con el asyncScheduler como un setInterval



para trabajar con Scheduler como setInterval el parámetro que recibe no puede ser una función de flecha debe ser una función normal, esta función como parámetro recibe el state.

Para que cumpla con la función del setInterval debemos llamar nuestro Schedule dentro nuestra función y determinar cuál va ser su nuevo state y el intervalo de tiempo en el cual será ejecutado de nuevo.



**Nota**: la principal ventaja de estas formas de realizar la función del setTimeout y setInterval, es que genera suscripciones de sus procesos los que permite conocer su estado.

**Video #29: Mas ejemplos con Form y of.**