RxJS & observables

observables :-

provided by RxJS (not a concept specific to Angular)
- object that produces & controls a stream of data. (they emit values over time - we can set-up subscription to handle them).

subject (BehaviorSubjects) are special kinds of observables that act as event emitters

one another way of creating observables :-

1). We can create using interval fn. from rxjs.
It creates an observable that emits sequential numbers every specified interval of time.

Interval fn. observable won't do anything unless subscribed to.

RxJs library works on the principle if no listner is available, doesn't make sense to emit data

1000 ms = 1 sec

```
interval(1000).subscribe({
    next : (val) => console.log(val)
    complete : () => console.log("data emission completed.");
    error : () => console.log("error in emission")
})
```

next method ← for what to do with data emitted

RxJs operators can help manipulating data when dealing with observables.

- Call pipe() before subscribe to observables.

map operator

applies a given project function that in end converts a value emitted by observable.

Executed on every value that would be emitted. This subscribe method would get the transformed value to work with now, not the one emitted by observable originally.

signals (Angular-specific concept)

similarites & differences related to observable

- often signals & subjects might look like they're almost the same.

- Before signals were introduced Angular relied on observables for data-communication from services to components.

- signals have initial values, observables don't (subjects can have) (interval) have no initial value)

Observables = pipeline of values emitted over time

signals = value containers

to read values from observables subscription is a must, not the case with signal

() on signal gives out the value stored in the signal

Thumb Rule —

Observables are really good for events & streamed data (values arrive asynchronously)

Signals can be great for managing application state

In fact, Angular gives tools to convert observables ⟷ signals.

for observable properties ending with $ is a common pattern.

```
clickCount = signal(0);
clickCount$ = toObservable(this.clickCount);
```

import { toObservable } from '@angular/core/rxjs-interop';

↓

Don't execute () just pass the signal property.

interval$ = interval(1000); // interval$ is observable
intervalSignal = toSignal(this.interval$); // property

in template

{{ intervalSignal() }}

Because, by default observables have no initial value so when toSignal() is used it is given special "undefined" value by default by Angular, if we want a custom initial value, we can do that.

intervalSignal = toSignal(this.interval$, {initialValue: 0});

// This would override 'undefine' with 0.

toSignal() ⟸ for observable cleans up the subscription when the component is destroyed from DOM.

Building an observable from scratch —

customInterval$ = new Observable();