of / from / fromEvent

convert non-obervables to Observable.

f.E. of(null); from ([1,2,3]); fromEvent(s,'keyup');

toPromise()

convert observables to Promises.

pipe()

Pipes let you **combine multiple functions into a single function.**

map()

f.E. source.pipe(map(val => val +10));

subscribe() / unsubscribe()

const o = interval(1000);

const subscription = o.subscribe(x=>console.log(x));

switchMap()

The Angular SwitchMap **maps each value from the source observable into an inner observable,** subscribes to it, and then starts emitting the values from it. It creates a new inner observable for every value it receives from the Source.

f.E. const o1 = fromEvent(document, 'click');

const o2 = interval(1000);

const finalo = o1.pipe(switchMap(x=>o2));

mergeMap()

merge response from 2 observables. The final observable subscription will have to wait until the merged observable to emit some response.

f.E. const fname = of('Sebastian');

const lname = of('Kronschläger');

const fullname = fname.pipe(mergeMap(x => lname.pipe(map(y=> x + ' ' + y)));

forkJoin()

Group of Observables -> **only care about final emitted value**

f.E. Multiple Requests on event and only take action when every request has received a response

forkJoin({req1: …, req2: …}).
subscribe(x=>{allrequests=x};

tap()

perform transparent actions. Logging f.E.

debounceTime()

wait specific time before emit.

f.E. passwordCheck / type-ahead: wait 1 sec, do not emit every users input.

const keyup = fromEvent(searchbox, 'keyup');

keyup.pipe(map((i:any)=>i.currentTarget.value),

debounceTime(500))

.subscribe(….log);

distinctUntilChanged()

Only emit when the current value is different than the last.

catchError()

Catch Error operator is used to gracefully handle errors, returning observable with error messages as below.

f.E. sourceObs$.pipe(catchError(error => of(`Bad Promise: ${error}`)))

takeUntil()

emit values until provided observable emits.

f.E. const source = interval (1000);

const clicks = fromEvent(document, 'click');

const result = source.pipe(takeUntil(clicks));

combineLatest()

Whenever any input Observable emits a value, it computes a formula **using the latest values from all the inputs, then emits the output of that formula.**

f.e. var bmi = weight.combineLatest(height, (w,h) => w/(h*h));

throttleTime()

This operator is used to emit the latest value when a specified throttle time duration has passed.

const ob = interval(1000).pipe(throttleTime(5000));

Useful links: https://medium.com/@naveen_chandupatla/rxjs-most-commonly-used-operators-b987e526ceb6 https://rxjs.dev/operator-decision-tree https://angular.io/guide/observables

by Sebastian Kronschläger