RxJS Operators

Deborah Kurata

Developer

https://www.youtube.com/@deborah_kurata

RxJS Operators

Start

Notified as each apple is emitted

Item passes through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop

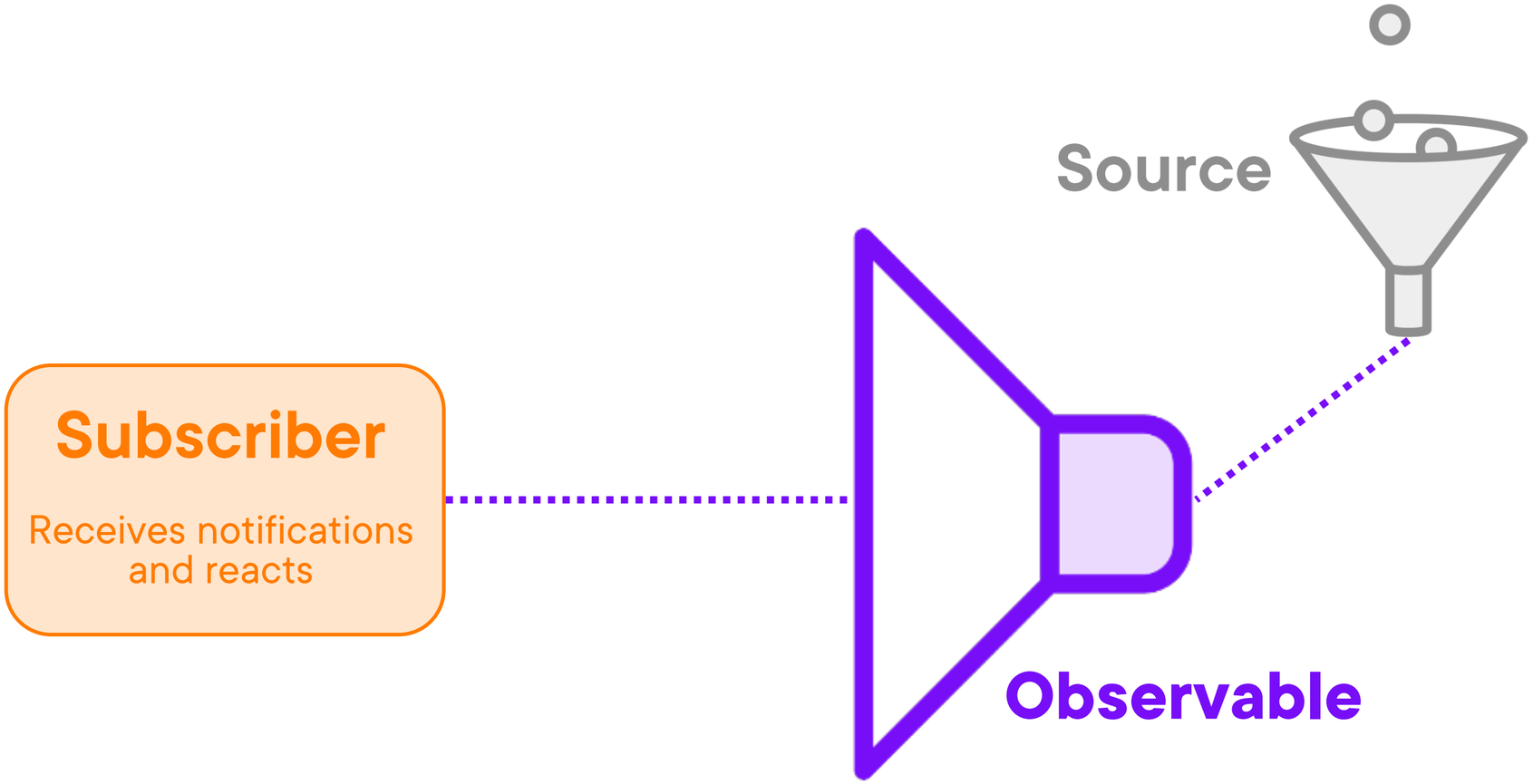
Each emitted item can be piped through a set of operators

- Transform, filter, process, ...
- Combine, aggregate, ...
- Handle errors
- Delay, timeout, ...

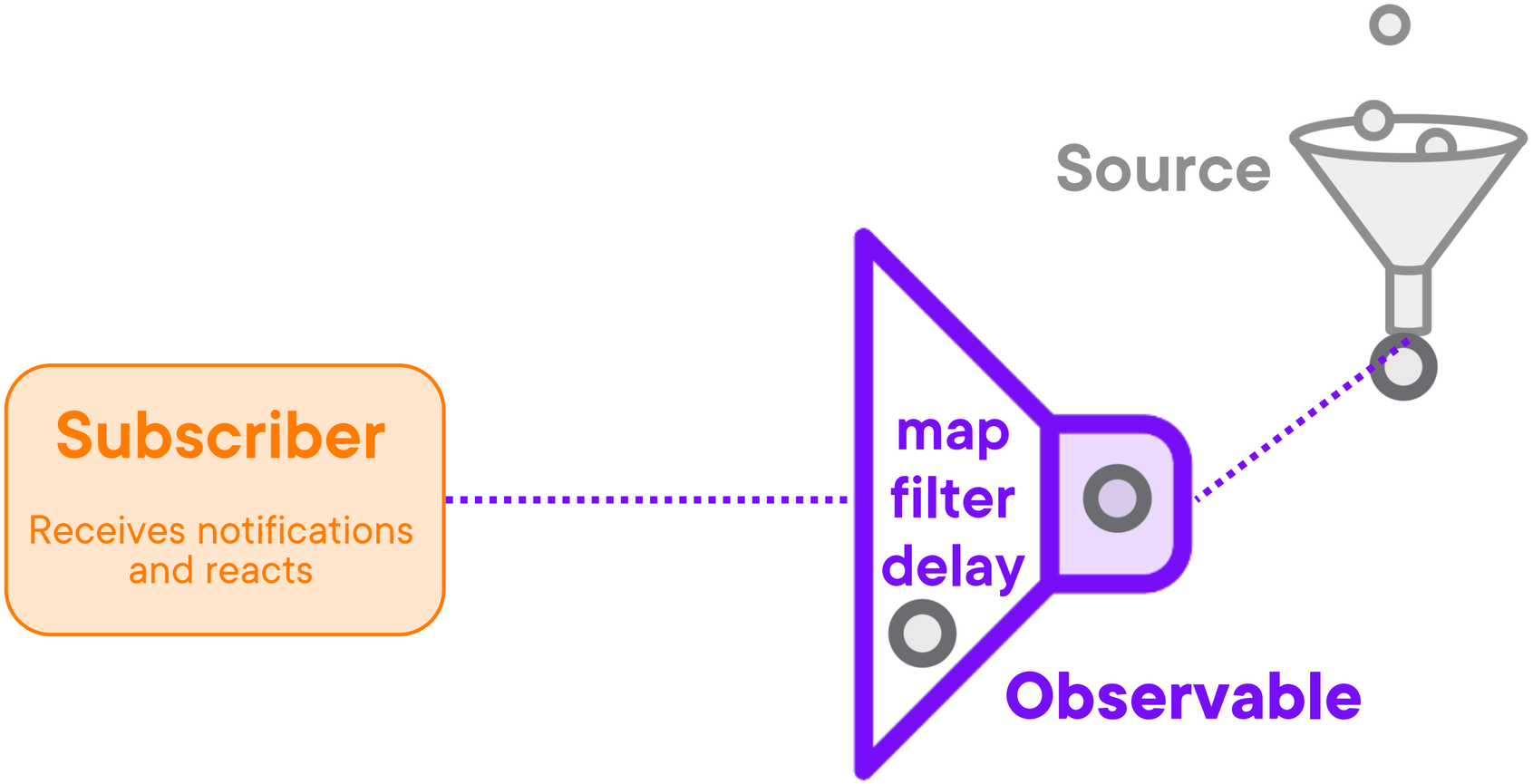
Fashioned after .NET LINQ operators

Similar to JavaScript array methods such as filter and map

Subscribing to an Observable



Subscribing to an Observable



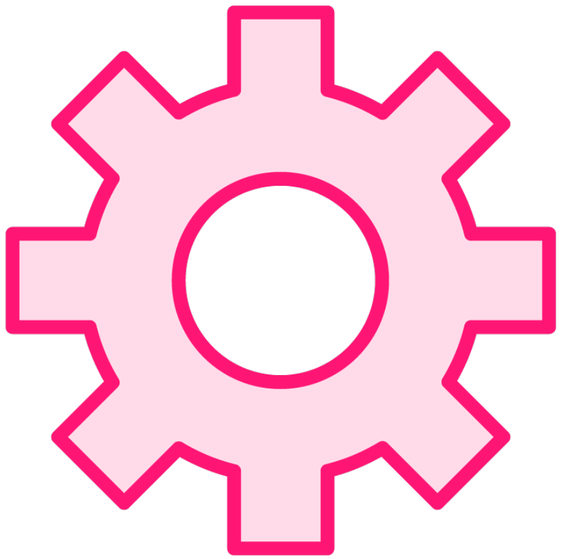
Overview

Introduce RxJS operators

Examine operators:

- map
- tap
- filter
- take

What Is an RxJS Operator?



An operator is a function

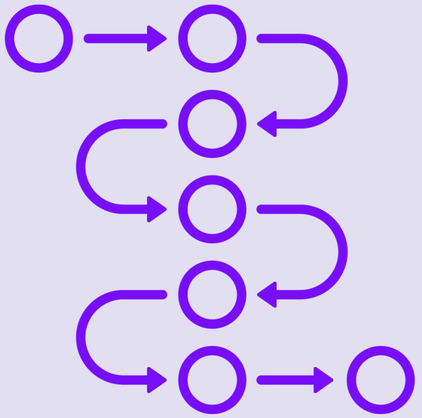
Used to transform, manipulate, or operate on items received from the source

- Before they are emitted to the subscribers

Apply operators in sequence using the observable's `pipe()` method

RxJS Operators

```
of(2, 4, 6)
  .pipe(
    map(item => item * 2),
    tap(item => console.log(item)),
    take(3)
  ).subscribe(item => console.log(item));
```



RxJS Operators

```
of(2, 4, 6)
  .pipe(
```

Observable

subscribe

```
      map(item => item * 2),
```

create

Observable

subscribe

```
      tap(item => console.log(item)),
```

create

Observable

subscribe

```
      take(3)
```

create

Observable

```
  ).subscribe(item => console.log(item));
```

Minimize the number of operators

Key Point

```
of(2, 4, 6)
  .pipe(
     map(item => item * 2),
     map(item => item + ' items')
  ).subscribe(item => console.log(item));
```

```
of(2, 4, 6)
  .pipe(
     map(item => {
        item = item * 2;
        return item + ' items';
     })
  ).subscribe(item => console.log(item));
```

Why?

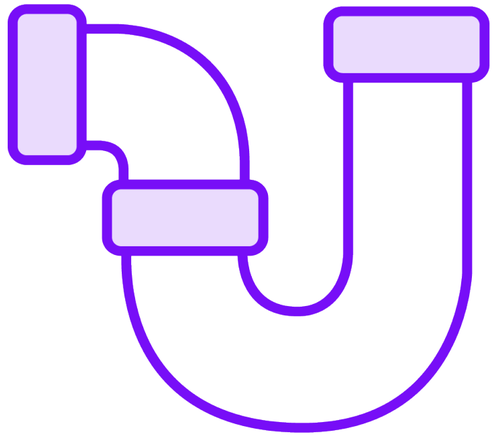
To minimize operator setup and tear down, improving performance

RxJS Operators

- audit
- auditTime
- BasicGroupByOptions
- buffer
- bufferCount
- bufferTime
- bufferToggle
- bufferWhen
- catchError
- ~~combineAll (deprecated)~~
- ~~combineLatest (deprecated~~
- combineLatestAll
- combineLatestWith
- ~~concat (deprecated)~~
- concatAll
- concatMap
- ~~concatMapTo (deprecated)~~
- concatWith
- connect
- ConnectConfig
- count
- debounce
- debounceTime
- defaultIfEmpty
- delay
- delayWhen
- dematerialize
- distinct
- distinctUntilChanged
- distinctUntilKeyChanged
- elementAt
- endWith
- every
- ~~exhaust (deprecated)~~
- exhaustAll
- exhaustMap
- expand
- filter
- finalize
- find
- findIndex
- first
- ~~flatMap (deprecated)~~
- groupBy
- GroupByOptionsWithElemer
- ignoreElements
- isEmpty
- last
- map
- ~~mapTo (deprecated)~~
- materialize
- max
- merge
- mergeAll
- mergeMap
- ~~mergeMapTo (deprecat~~
- mergeWith
- min
- observeOn
- ~~onErrorResumeNext (deprec~~
- pairwise
- ~~partition (deprecated)~~
- ~~pluck (deprecated)~~
- ~~publish (deprecated)~~
- ~~publishBehavior (deprecate~~
- ~~publishLast (deprecated)~~
- ~~publishReplay (deprecated)~~
- ~~race (deprecated)~~
- raceWith
- reduce
- ~~refCount (deprecated)~~
- repeat
- RepeatConfig
- ~~repeatWhen (deprecated)~~
- retry
- RetryConfig
- ~~retryWhen (deprecated)~~
- sample
- sampleTime
- scan
- sequenceEqual
- share
- ShareConfig
- shareReplay
- ShareReplayConfig
- single
- skip
- skipLast
- skipUntil
- skipWhile
- startWith
- subscribeOn
- switchAll
- switchMap
- ~~switchMapTo (deprecated)~~
- switchScan
- take
- takeLast
- takeUntil
- takeWhile
- tap
- TapObserver
- throttle
- ThrottleConfig
- throttleTime
- throwIfEmpty
- timeInterval
- timeout
- TimeoutConfig
- TimeoutInfo
- ~~timeoutWith (deprecated)~~
- timestamp
- toArray
- window
- windowCount
- windowToggle
- windowWhen
- ~~zip (deprecated)~~
- zipAll

https://rxjs.dev

RxJS Operator: `map`



Transforms each emitted item

```
map(item => item * 2)
```

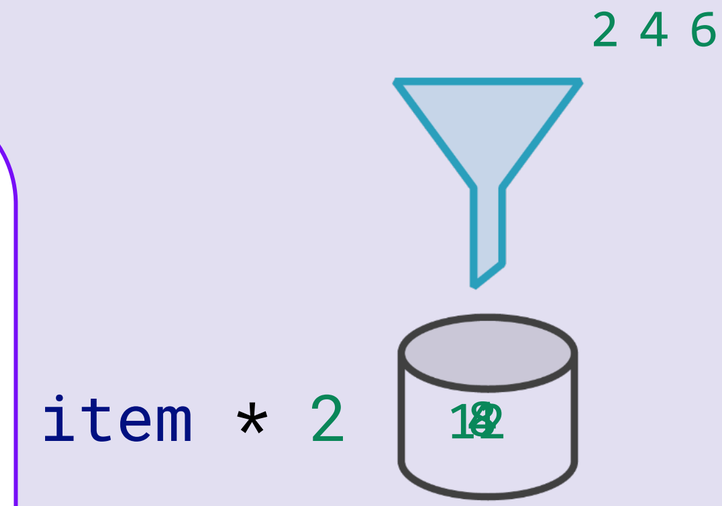
For each item emitted in, one mapped item is emitted out

Used for

- Making changes to each item

RxJS Operator: map

```
of(2, 4, 6)
  .pipe(
    map(item => item * 2)
  )
  .subscribe(x => console.log(x));
```



RxJS Operator: map


```
{
 "customers": [
   {
    "id": 1,
    "name": "microsoft",
    "address": "…"
   },
   {
    "id": 2,
    "name": "google",
    "address": "…"
   },
   {
    "id": 1,
    "name": "amazon",
    "address": "…"
   }
 ],
 "count": 3,
 "success": true
}
```



```
[
  {
    "id": 1,
    "name": "microsoft",
    "address": "…"
  },
  {
    "id": 2,
    "name": "google",
    "address": "…"
  },
  {
    "id": 1,
    "name": "amazon",
    "address": "…"
  }
]
```


```
this.response$
  .pipe(
    map(x => x.customers)
  ).subscribe(x => console.log(x));
```

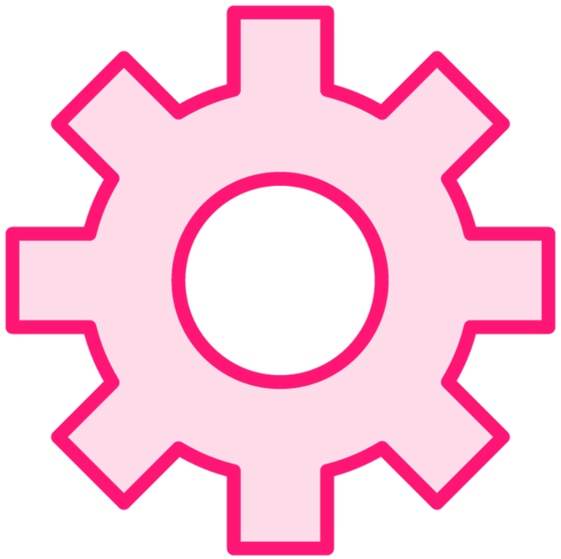

```
[{"id":1,"name":"microsoft","address":"…"},
{"id":2,"name":"google","address":"…"},
{"id":1,"name":"amazon","address":"…"}]
```

Demo

RxJS operator

- `map`

RxJS Operator: `map`

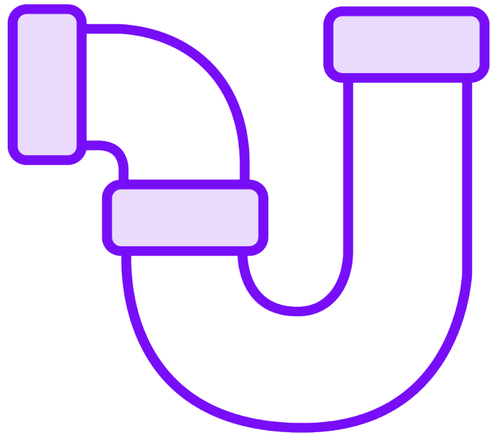


`map` **is a transformation operator**

When an item is emitted

- Item is transformed as specified by the provided function
- Transformed item is emitted

RxJS Operator: `tap`



Taps into the emissions without affecting the item

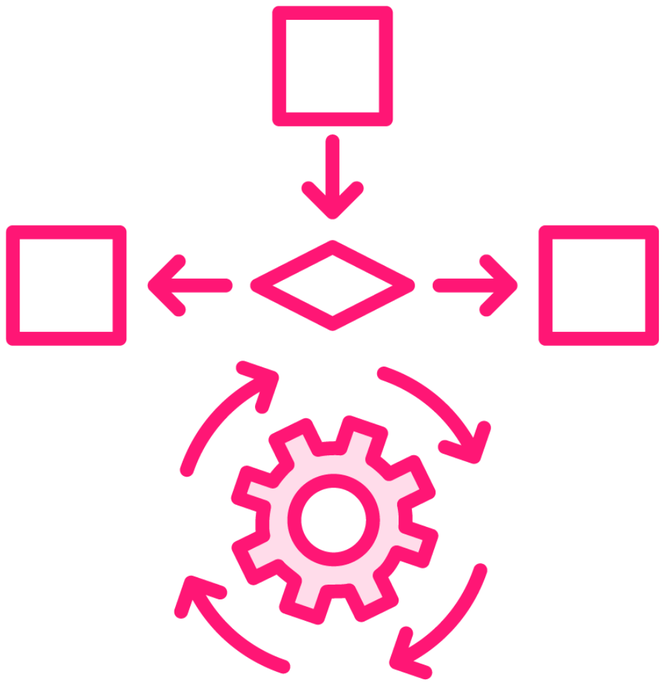
```
tap(item => console.log(item))
```

For each item emitted in, the same item is emitted out

Used for

- Debugging
- Performing actions outside of the flow of data (side effects)

`tap` for Side Effects



A function has a side effect if

- It has an effect other than its primary effect of returning a value
- We do anything other than modify the item emitted into the pipeline

Example

- Increment a counter
- Set a flag to turn on/off a loading indicator
- Log information to the console

Use the tap operator for debugging

Key Point

```
of(2, 4, 6)
    .pipe(
      tap(item => console.log(item)),
      map(item => item * 2),
      tap(item => console.log(item)),
      map(item => item - 3),
      tap(item => console.log(item))
    ).subscribe();
```

Why?

Log the value to the console at key points
Hover over to see the type emitted from the prior operator

Retaining Emitted Items

```
const keys: string[] = [];
this.sub = keyPresses$.subscribe(
  event => keys.push(event.key)
);
```

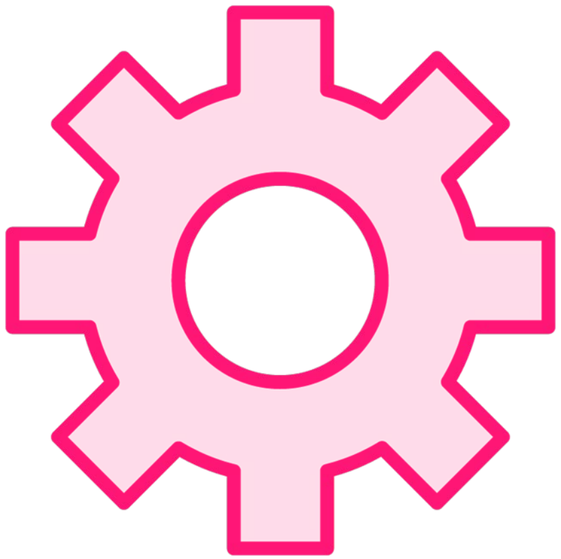
```
const keys: string[] = [];
this.sub = keyPresses$
  .pipe(
    tap(event => keys.push(event.key)),
    tap(() => console.log(keys))
  ).subscribe();
```

Demo

RxJS operator

- `tap`

RxJS Operator: `tap`

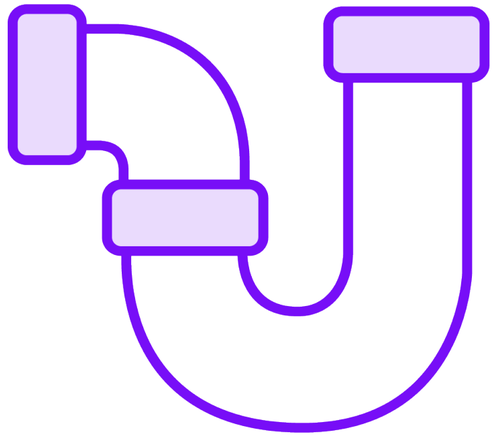


`tap` **is a utility operator**

When an item is emitted

- Performs a side effect as specified by a provided function
- Original item is emitted

RxJS Operator: `filter`



Emits items that match criteria specified in a provided function

```
filter(item => item === 'Apple')
```

Similar to the array filter method

Used for

- Emitting items that match specific criteria
- Filtering out null values

RxJS Operator: `filter`

```
of(2, 3, 4, 5, 6).pipe(
  filter(x => x % 2 === 0),
  tap(x => console.log(x))
).subscribe();
```

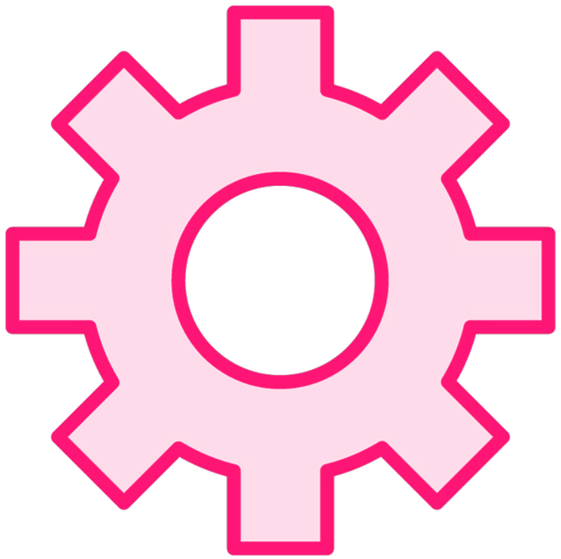
```
2
4
6
```

Demo

RxJS operator

- `filter`

RxJS Operator: `filter`

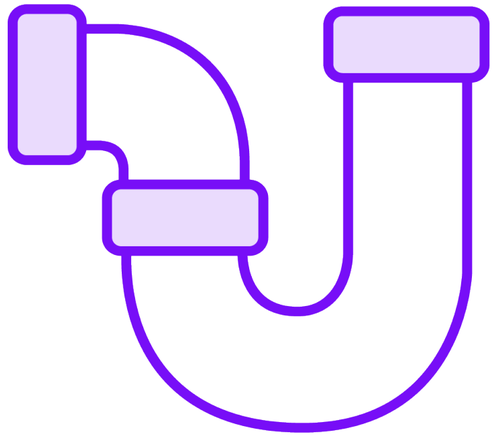


`filter` **is a filtering operator**

When an item is emitted

- Item is evaluated as specified by the provided function
- If the evaluation returns true, item is emitted down the pipeline
- If the evaluation returns false, item is not emitted

RxJS Operator: `take`



Emits a specified number of items

```
take(2)
```

Automatically completes after taking the specified number of items

Used for

- Taking a specified number of items
- Limiting unlimited observables

RxJS Operator: take

```
of(2, 4, 6)
    .pipe(
      take(2)
    ).subscribe(console.log);  // 2 4
```

```
of(2, 4, 6)
    .pipe(
      tap(item => console.log(item)),
      map(item => item * 2),
      take(2),
      map(item => item - 3),
      tap(item => console.log(item))
    ).subscribe();
```

2
1
4
5

RxJS Operator: take

```
of(2, 3, 4, 5, 6).pipe(
  filter(x => x % 2 === 0),
  take(2),
  tap(x => console.log(x))
).subscribe();
```

```
2
4
```

```
of(2, 3, 4, 5, 6).pipe(
  take(2),
  filter(x => x % 2 === 0),
  tap(x => console.log(x))
).subscribe();
```

```
2
```

Key Point

If you want to take the specified number of items, insert `take` last

```
of(2, 3, 4, 5, 6).pipe(
  filter(x => x % 2 === 0),
  take(2)
).subscribe();
```

Why?

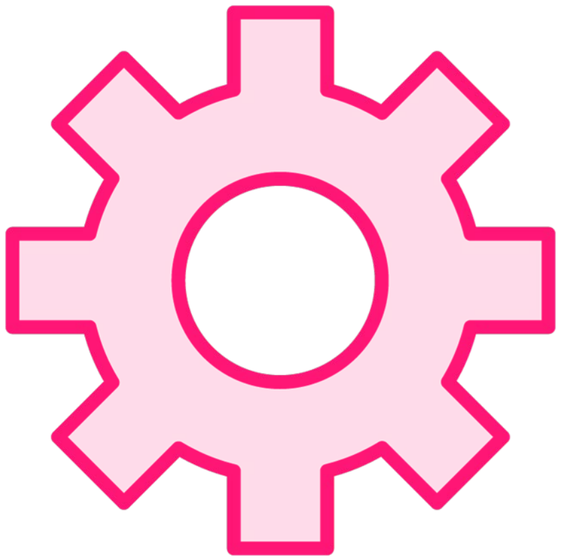
Otherwise, you may get more or fewer items than anticipated

Demo

RxJS operator

- `take`

RxJS Operator: `take`



`take` **is a filtering operator**

When an item is emitted

- Counts the item
  - If <= specified number, item is emitted
  - When count equals the specified number, the observable completes

Only emits the defined number of items

RxJS
Operators

Use the observable pipe method to pipe emitted items through a sequence of operators

```
from([30, 25, 20, 15, 10, 5, 0])
 .pipe(
   map(item => item + 2),
   tap(item => console.log(item)),
   filter(x => x % 2 === 0),
   take(3)
 ).subscribe();
```

Each operator's output observable
is the input observable to the following operator

RxJS Operators

map: transforms an emitted item

```
map(item => item + 2)
```

tap: taps into emissions without affecting the items

```
tap(item => console.log(item))
```

filter: filters the result to only items that match specific criteria

```
filter(x => x % 2 === 0)
```

take: takes a defined number of items and completes

```
take(3)
```

observer vs. tap

```
const keys: string[] = [];
this.sub = keyPresses$.subscribe(
  event => keys.push(event.key)
);
```

```
const keys: string[] = [];
this.sub = keyPresses$
  .pipe(
    tap(event => keys.push(event.key))
  ).subscribe();
```

Stop Receiving Notifications from an Observable

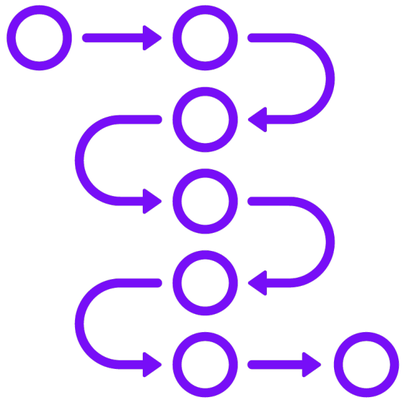
```
this.sub = apples$.subscribe(
  apple => console.log(`Emitted: ${apple}`)
);
```

```
this.sub.unsubscribe();
```

```
of(2, 4, 6)
  .pipe(
     map(item => item * 2)
  ).subscribe(item => console.log(item));
```

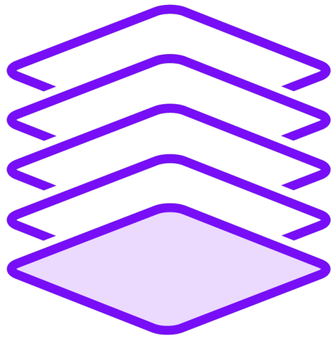
```
timer(0,1000)
    .pipe(
      take(5)
    ).subscribe(item => console.log(item));
```

Best Practices



Minimize the number of operators

Use tap to examine emissions as an aid in debugging



Insert the take operator last in the pipeline

For More Information

RxJS documentation

- https://rxjs.dev/

"RxJS in Angular: Terms, Tips and Patterns"

- https://youtu.be/vtCDRiG_D4

Demo code

- https://stackblitz.com/edit/rxjs-signals-m4-deborahk

Up Next:

Retrieving Data with HTTP and Observables