## 

## 3. interval – Emits an infinite sequence of ascending numbers in a given time interval

```
typescript
import { interval } from 'rxjs';

const observable$ = interval(1000); // emits every 1 second
observable$.subscribe(value => console.log(value));
```

```
python-repl

0

1

2

3

...
```

### 6. fromEvent - Converts DOM events into Observables

```
import { fromEvent } from 'rxjs';

const clicks$ = fromEvent(document, 'click');
clicks$.subscribe(event => console.log('Clicked!', event));
```

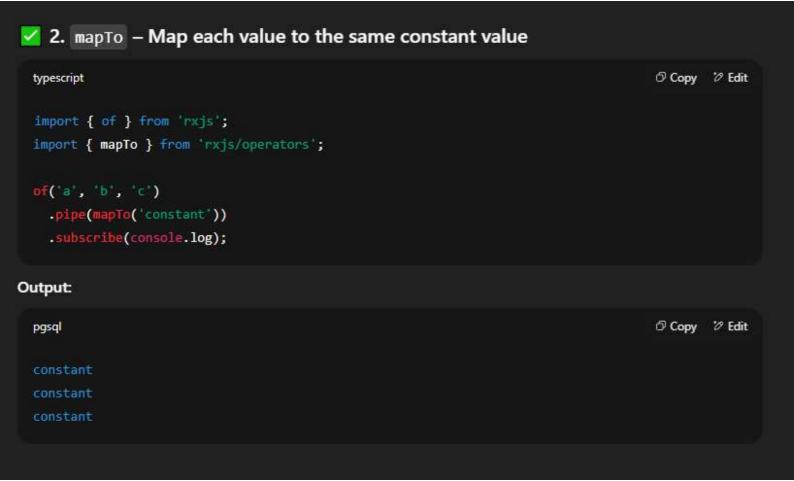
### 1. map – Apply a function to each value emitted

Transforms each value using a projection function.

```
typescript
import { of } from 'rxjs';
import { map } from 'rxjs/operators';

of(1, 2, 3)
   .pipe(map(x => x * 10))
   .subscribe(console.log);
```

```
10
20
30
```



```
3. pluck – Extract a property from each emitted object
  typescript

<sup>⑤</sup> Copy <sup>②</sup> Edit

  import { from } from 'rxjs ;
  import { pluck } from 'rxjs/operators';
  const source$ = from([
   { name: 'Alice', age: 30 },
    { name: 'Bob', age: 25 }
  ]);
  source$
    .pipe(pluck('name'))
    .subscribe(console.log);
Output:
  nginx

⑤ Copy 
② Edit

  Bob
```

### 5. switchMap – Map and switch to new inner observable (cancels previous)

```
import { fromEvent, interval } from 'rxjs';
import { switchMap } from 'rxjs/operators';

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

clicks$
   .pipe(switchMap(() => interval(1000)))
   .subscribe(console.log);
```

Explanation: Emits an increasing number every second, restarting on each click.

### 6. concatMap – Map and subscribe to inner observables one at a time, in order

```
import { of } from 'rxjs';
import { concatMap, delay } from 'rxjs/operators';

of(1, 2, 3)
   .pipe(
    concatMap(val => of(val * 10).pipe(delay(1000)))
   .subscribe(console.log);
```

Output: Emits 10, 20, 30, each 1 second apart, in order.

```
▼ 7. mergeMap – Map and flatten inner observables concurrently
```

```
typescript

import { of } from 'rxjs';
import { mergeMap, delay } from 'rxjs/operators';

of(1, 2, 3)
   .pipe(
    mergeMap(val => of(val * 10).pipe(delay(1000)))
   )
   .subscribe(console.log);
```

Output: Same values as concatMap but emitted concurrently (not guaranteed order).

```
8. exhaustMap – Ignores new emissions if one is already active
```

```
typescript

import { fromEvent, interval } from 'rxjs';
import { exhaustMap, take } from 'rxjs/operators';

fromEvent(document, 'click')
   .pipe(
    exhaustMap(() => interval(1000).pipe(take(3)))
   .subscribe(console.log);
```

Explanation: On click, emits 0,1,2. Ignores further clicks until that completes.

```
✓ 1. filter
```

Emits only those items that pass the condition.

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

of(1, 2, 3, 4, 5)
   .pipe(filter(x => x % 2 === 0))
   .subscribe(console.log);
```

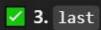
Output: 2, 4



Emits only the first value or the first that matches a condition.

```
import { of } from 'rxjs';
import { first } from 'rxjs/operators';

of(1, 2, 3)
   .pipe(first())
   .subscribe(console.log);
```



Emits only the last value emitted by the source Observable.

```
import { of } from 'rxjs';
import { last } from 'rxjs/operators';

of(1, 2, 3)
   .pipe(last())
   .subscribe(console.log);
```

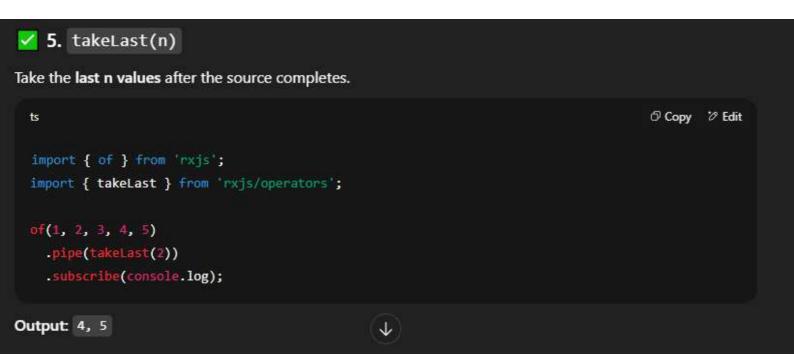
```
✓ 4. take(n)
```

Take the first n values emitted by the source.

```
import { of } from 'rxjs';
import { take } from 'rxjs/operators';

of(1, 2, 3, 4, 5)
   .pipe(take(3))
   .subscribe(console.log);
```

Output: 1, 2, 3



```
Skip the first n values.

ts

① Copy ② Edit

import { of } from 'rxjs';
import { skip } from 'rxjs/operators';

of(1, 2, 3, 4)
   .pipe(skip(2))
   .subscribe(console.log);
Output: 3, 4
```



Wait for the specified time before emitting the value (useful for type-ahead search).

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

fromEvent(document, 'keyup')
   .pipe(
    debounceTime(300),
    map((event: any) => event.target.value)
   )
   .subscribe(console.log);
```

```
✓ 9. distinct
```

Emits all items that are distinct from the previous.

```
import { of } from 'rxjs';
import { distinct } from 'rxjs/operators';

of(1, 2, 2, 3, 1)
   .pipe(distinct())
   .subscribe(console.log);
```

Output: 1, 2, 3

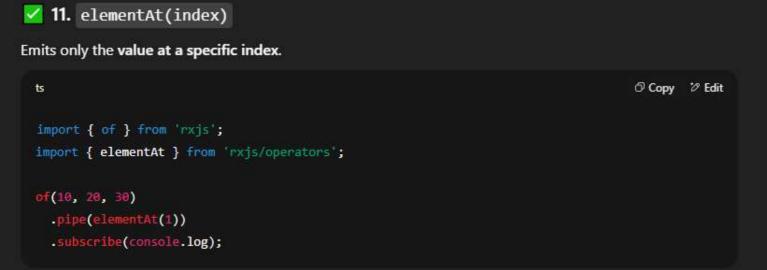


Emits only when the current value is different from the last.

```
import { of } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';

of(1, 1, 2, 2, 3, 3)
   .pipe(distinctUntilChanged())
   .subscribe(console.log);
```

Output: 1, 2, 3





Take values while the condition is true.

```
import { of } from 'rxjs';
import { takeWhile } from 'rxjs/operators';

of(1, 2, 3, 4, 1)
   .pipe(takeWhile(x => x < 4))
   .subscribe(console.log);</pre>
```

Output: 1, 2, 3

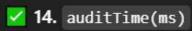


Skip values while the condition is true, then take the rest.

```
import { of } from 'rxjs';
import { skipWhile } from 'rxjs/operators';

of(1, 2, 3, 4)
   .pipe(skipWhile(x => x < 3))
   .subscribe(console.log);</pre>
```

Output: 3, 4



Ignore values for a set time, then emit the latest one.

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

fromEvent(document, 'mousemove')
.pipe(
   auditTime(1000),
   map((event: MouseEvent) => 'X: ${event.clientX}, Y: ${event.clientY}')
)
.subscribe(console.log);
```

### ⊘ 1. concat – Sequentially emits all values from multiple Observables

```
import { of, concat } from 'rxjs';

const obs1$ = of(1, 2, 3);

const obs2$ = of(4, 5);

concat(obs1$, obs2$).subscribe(console.log);
```

Output: 1, 2, 3, 4, 5
(Waits for obs1\$ to complete before starting obs2\$)

# ② 2. merge – Emits values from multiple Observables as they arrive (interleaved)

```
import { of, merge } from 'rxjs';
import { delay } from 'rxjs/operators';

const obs1$ = of('A', 'B').pipe(delay(1000));
const obs2$ = of('1', '2');

merge(obs1$, obs2$).subscribe(console.log);
```

Output: Interleaved based on timing (e.g., 1, 2, A, B)

# ⊘ 3. combineLatest – Emits the latest values from each observable whenever any emits

```
import { combineLatest, interval } from 'rxjs';
import { map, take } from 'rxjs/operators';

const obs1$ = interval(1000).pipe(take(3)); // 0, 1, 2
const obs2$ = interval(1500).pipe(take(2)); // 0, 1

combineLatest([obs1$, obs2$])
   .pipe(map(([x, y]) => `obs1: ${x}, obs2: ${y}`))
   .subscribe(console.log);
```

```
yaml

obs1: 1, obs2: 0

obs1: 2, obs2: 0

obs1: 2, obs2: 1
```

# ⊘ 4. forkJoin – Waits for all Observables to complete, then emits the last values as an array

```
import { of, forkJoin } from 'rxjs';
import { delay } from 'rxjs/operators';

const obs1$ = of('A').pipe(delay(1000));
const obs2$ = of('B').pipe(delay(2000));

forkJoin([obs1$, obs2$]).subscribe(console.log);

Output: ['A', 'B']
(Only after both complete)
```

### Ø 5. zip − Pairs values by index and emits them together

```
import { of, zip } from 'rxjs';

const names$ = of('Alice', 'Bob');
const scores$ = of(90, 80);

zip(names$, scores$).subscribe(([name, score]) => {
   console.log(`${name}: ${score}`);
});
Output:
```

```
makefile

Alice: 90
Bob: 80
```

### ∅ 6. startWith - Prepends an initial value before the source emits

```
import { of } from 'rxjs';
import { startWith } from 'rxjs/operators';

of(2, 3, 4)
   .pipe(startWith(1))
   .subscribe(console.log);
```

Output: 1, 2, 3, 4

## ⊘ 7. withLatestFrom – Combine source value with latest value from another observable

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

const fast$ = interval(1000);
const slow$ = interval(3000);

fast$
   .pipe(
    withLatestFrom(slow$),
    map(([fastVal, slowVal]) => `Fast: ${fastVal}, Slow: ${slowVal}`)
   )
   .subscribe(console.log);
```

Output: Emits only when fast\$ emits, using the latest slow\$ value.

### Ø 8. race – Emits from the first observable to emit, and ignores the rest

```
import { of, race } from 'rxjs';
import { delay } from 'rxjs/operators';

const obs1$ = of('A').pipe(delay(1000));
const obs2$ = of('B').pipe(delay(500));

race(obs1$, obs2$).subscribe(console.log);
```

Output: B (obs2\$ wins the race)

# Ø 9. combineLatestWith − Same as combineLatest but used as a pipeable operator

```
import { interval } from 'rxjs';
import { combineLatestWith, map, take } from 'rxjs/operators';

interval(1000)
   .pipe(
    take(3),
    combineLatestWith(interval(1500).pipe(take(2))),
    map(([x, y]) => `X: ${x}, Y: ${y}`)
   .subscribe(console.log);
```



Performs a side effect (like logging) without modifying the stream.

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

of(1, 2, 3)
   .pipe(
    tap(value => console.log('Before map:', value)),
    map(value => value * 10),
    tap(value => console.log('After map:', value))
)
   .subscribe();
```

```
arduino

Before map: 1

After map: 10

Before map: 2

After map: 20

Before map: 3

After map: 30
```



Executes code when the observable completes or errors out.

```
import { of } from 'rxjs';
import { finalize } from 'rxjs/operators';

of('RxJS')
   .pipe(finalize(() => console.log('Observable completed')))
   .subscribe(console.log);
```

```
nginx

RXJS

Observable completed
```

```
3. delay
```

Delays each emission by the specified amount of time.

```
ts
import { of } from 'rxjs';
import { delay } from 'rxjs/operators';

of('Hello')
   .pipe(delay(1000))
   .subscribe(console.log); // emits after 1 second
```



Delays each value based on another observable.

```
import { of, timer } from 'rxjs';
import { delayWhen } from 'rxjs/operators';

of('A', 'B', 'C')
   .pipe(delayWhen(() => timer(1000)))
   .subscribe(console.log); // All values delayed by 1 second
```



Throws an error if a value isn't emitted in the specified time.

```
import { of } from 'rxjs';
import { delay, timeout } from 'rxjs/operators';

of('Delayed value')
  .pipe(
    delay(2000),
    timeout(1000) // will throw an error because of 2s delay
)
  .subscribe({
    next: val => console.log(val),
    error: err => console.error('Timeout error:', err.message)
});
```



Retries the observable n times on error.

```
import { throwError } from 'rxjs';
import { retry } from 'rxjs/operators';

throwError(() => new Error('Failure'))
   .pipe(retry(2))
   .subscribe({
    error: err => console.log('Failed after retries:', err.message)
});
```



Changes the scheduler for downstream operators.

```
import { of, asyncScheduler } from 'rxjs';
import { observeOn } from 'rxjs/operators';

of('Hello')
   .pipe(observeOn(asyncScheduler))
   .subscribe(console.log); // executed asynchronously
```

1. share() – Share a single subscription among multiple subscribers

```
ts
import { interval } from 'rxjs';
import { take, share } from 'rxjs/operators';

const source$ = interval(1000).pipe(
   take(3),
   share()
);

source$.subscribe(val => console.log('Subscriber A:', val));

'setTimeout(() => {
   source$.subscribe(val => console.log('Subscriber B:', val));
}, 1500);
```

- Subscriber A: 0, 1, 2
- · Subscriber B: Might get 1 and 2 depending on timing

## 2. shareReplay() – Shares the source and replays specified number of last values

```
import { of, delay } from 'rxjs';
import { shareReplay } from 'rxjs/operators';

const source$ = of(1, 2, 3).pipe(shareReplay(2));

source$.subscribe(val => console.log('First:', val));

setTimeout(() => {
    source$.subscribe(val => console.log('Second (replay):', val));
}, 1000);
```

```
4. multicast() – Share a subject among subscribers
 ts

<sup>⑤</sup> Copy  <sup>②</sup> Edit

 import { interval, Subject } from 'rxjs';
 import { multicast, take } from 'rxjs/operators';
 const source$ = interval(1000).pipe(
   take(3),
   multicast(() => new Subject())
 ) as any;
 source$.subscribe(val => console.log('Subscriber A:', val));
 source$.subscribe(val => console.log('Subscriber 8:', val));
 source$.connect();
```

Deprecated in RxJS 7+. Use connectable() instead.

```
✓ 5. connectable() (RxJS 7+) – New alternative to publish() and multicast()
```

```
import { connectable, interval } from 'rxjs';
import { take } from 'rxjs/operators';

const source$ = connectable(interval(1000).pipe(take(3)));

source$.subscribe(val => console.log('A:', val));

source$.subscribe(val => console.log('B:', val));

source$.connect();
```

## 1. catchError – Catch errors and return a fallback Observable

```
import { of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

throwError(() => new Error('Oops!'))
   .pipe(
    catchError(err => {
        console.error('Caught error:', err.message);
        return of('Fallback value');
    })
   .subscribe(console.log);
```

```
nginx

Caught error: Oops!
Fallback value
```

## 3. retryWhen – Retry based on a custom condition or delay strategy

```
import { throwError, timer } from 'rxjs';
import { retryWhen, delayWhen } from 'rxjs/operators';

throwError(() => new Error('Oops'))
   .pipe(
    retryWhen(errors =>
        errors.pipe(delayWhen(() => timer(2000))) // retry after 2 seconds
    )
   .subscribe({
    error: err => console.log('Still failed:', err.message)
});
```

Behavior: Keeps retrying every 2 seconds until unsubscribed or an error is thrown that stops the stream.

✓ 4. onErrorResumeNext – Continue with another Observable on error (silently skips)

```
import { of, throwError, onErrorResumeNext } from 'rxjs';

onErrorResumeNext(
   throwError(() => new Error('fail')),
   of('Recovered', 'Continued')
).subscribe(console.log);
```

## Output:

nginx

Recovered
Continued

## 

## 7. timeout – Throws an error if no value is emitted in a given time

```
import { of } from 'rxjs';
import { delay, timeout } from 'rxjs/operators';

of('Too late')
   .pipe(
    delay(2000),
    timeout(1000) // throws timeout error
)
   .subscribe({
    next: val => console.log(val),
    error: err => console.log('Timeout:', err.message)
});
```