1. Creation Operators

of

Creates an Observable that emits the arguments you provide.

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

const source$ = of(1, 2, 3, 'hello', true);
source$.subscribe(value => console.log(value));
//=>  Output: 1, 2, 3, 'hello', true
```

from

Creates an Observable from an array, promise, or iterable.

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

// From array
const arraySource$ = from([1, 2, 3]);
arraySource$.subscribe(value => console.log(value));
//=> Output: 1, 2, 3

// From promise
const promiseSource$ = from(fetch('https://api.example.com/data'));
```

interval

Creates an Observable that emits sequential numbers at specified intervals.

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

const source$ = interval(1000);
source$.subscribe(value => console.log(value));
//=> Output: 0, 1, 2, 3... (every second)
```

2. Transformation Operators

map

Transforms each value emitted by the source Observable.

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

const source$ = of(1, 2, 3, 4, 5);
const doubled$ = source$.pipe(
  map(x => x * 2)
);

doubled$.subscribe(value => console.log(value));
//=> Output: 2, 4, 6, 8, 10
```

2. Transformation Operators

filter

Filters items emitted by the source Observable.

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

const source$ = of(1, 2, 3, 4, 5, 6);
const evenNumbers$ = source$.pipe(
  filter(x => x % 2 === 0)
);

evenNumbers$.subscribe(value => console.log(value));
//=> Output: 2, 4, 6
```

2. Transformation Operators

switchMap

Maps to a new Observable and cancels previous inner subscriptions.

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

previous requests
const button = document.getElementById('search');
const clicks$ = fromEvent(button, 'click');

const search$ = clicks$.pipe(
  switchMap(() => fetch('/api/search'))
 //=> Cancels previous fetch if new click occurs
);
```

2. Transformation Operators

merge

Combines multiple Observables into one by merging their emissions.

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

const timer1$ = interval(1000).pipe(map(() => 'Timer 1'));
const timer2$ = interval(2000).pipe(map(() => 'Timer 2'));

const merged$ = merge(timer1$, timer2$);
merged$.subscribe(value => console.log(value));
//=> Output: Mixed emissions from both timers
```

concat

Concatenates multiple Observables sequentially.

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

const first$ = of(1, 2, 3);
const second$ = of(4, 5, 6);

const concatenated$ = concat(first$, second$);
concatenated$.subscribe(value => console.log(value));
//=> Output: 1, 2, 3, 4, 5, 6 (waits for first to complete)
```

4. Utility Operators

tap

Performs side effects without affecting the stream.

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

const source$ = of(1, 2, 3);
const processed$ = source$.pipe(
  tap(value => console.log('Before map:', value)),
  map(x => x * 2),
  tap(value => console.log('After map:', value))
);

processed$.subscribe();
// Output:  Before map: 1  After map: 2 Before map: 2 After map: 4
```

take

Takes the first N values from the source.

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

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

firstThree$.subscribe(value => console.log(value));
//=> Output: 0, 1, 2 (then completes)
```

4. Utility Operators

debounceTime

Ignores values for a specified time period.

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

const searchBox = document.getElementById('search');
const input$ = fromEvent(searchBox, 'input');

const debouncedInput$ = input$.pipe(
  map(event => event.target.value),
  debounceTime(300)
 //=> Wait 300ms after last input
);

debouncedInput$.subscribe(value => console.log('Search:', value));
```

5. Error Handling Operators

catchError

Handles errors from the source Observable.

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

const source$ = of(1, 2, 3, 'invalid', 4);
const processed$ = source$.pipe(
  map(value => {
    if (value === 'invalid') throw new Error('Invalid value');
    return value * 2;
  }),
  catchError(error => {
    console.log('Error caught:', error.message);
    return of('default value');
 //=> Continue with fallback value
  })
);

processed$.subscribe(value => console.log(value));
//=>  Output: 2, 4, 6, 'default value'
```

5. Error Handling Operators

retry

Retries the source Observable a specified number of times on error.

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

const unstableSource$ = of(1, 2, 3).pipe(
  mergeMap(value => {
    if (value === 2) throw Error('Temporary failure');
    return of(value);
  })
);

const retried$ = unstableSource$.pipe(retry(3));
retried$.subscribe({
  next: value => console.log(value),
  error: err => console.log('Failed after retries:', err)
});
```

Practical Example: Search Functionality

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

const searchBox = document.getElementById('search');

fromEvent(searchBox, 'input')
  .pipe(
    map(event => event.target.value.trim()),
    debounceTime(300), //=> Wait 300ms after user stops typing
    distinctUntilChanged(), //=> Only emit if value changed
    switchMap(searchTerm =>
      fetch(`/api/search?q=${searchTerm}`)
        .then(response => response.json())
    ),
    catchError(error => of([])) //=> Return empty array on error
  )
  .subscribe(results => {
    console.log('Search results:', results);
    //=> Update UI with results
  });
```