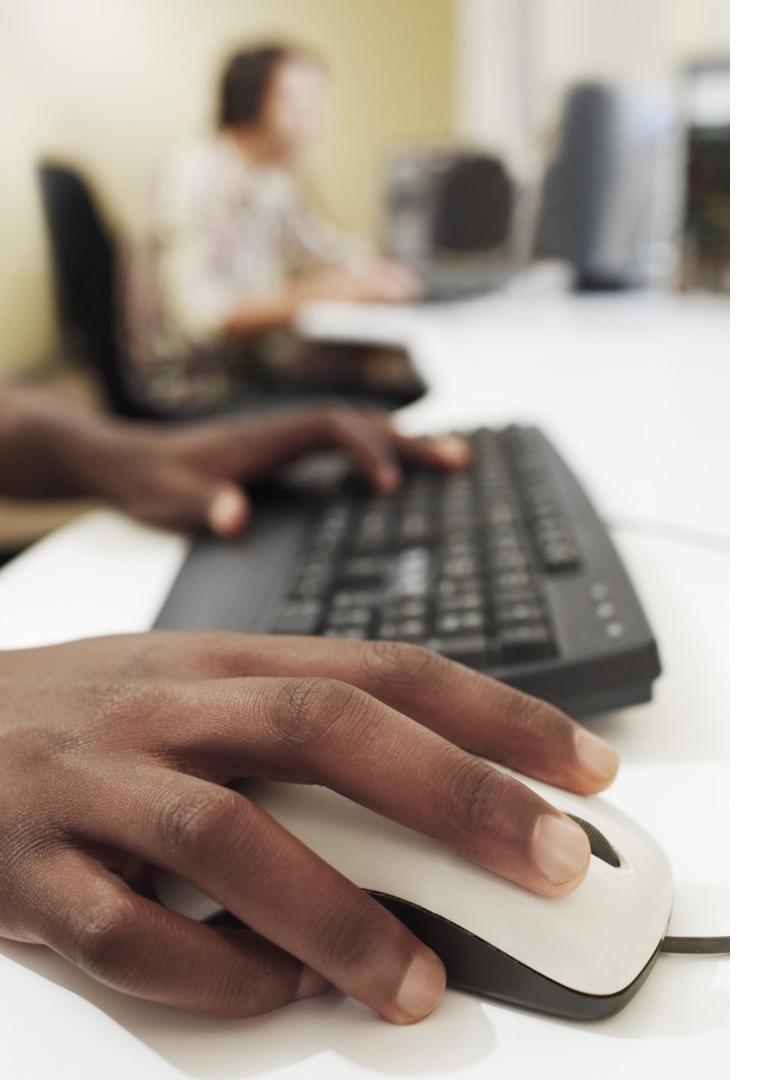
Reacting to Actions



Deborah Kurata Consultant | Speaker | Author | MVP | GDE

@deborahkurata





Reactive Development:

- A declarative programming paradigm
- Concerned with data streams
- And the propagation of change

Acme Product Management Home Product List Product List (Alternate UI)

- Display All -				Add Product
- Display All -				
Garden Toolbox	Code	Category	Price	In Stock
Gaming	GDN-0011	Garden	\$29.92	15
Garden Cart	GDN-0023	Garden	\$49.49	2
Hammer	TBX-0048	Toolbox	\$13.35	8
Saw	TBX-0022	Toolbox	\$17.33	6
Video Game Controller	GMG-0042	Gaming	\$53.93	12

Module Overview



Filtering emitted items

Data stream vs. action stream

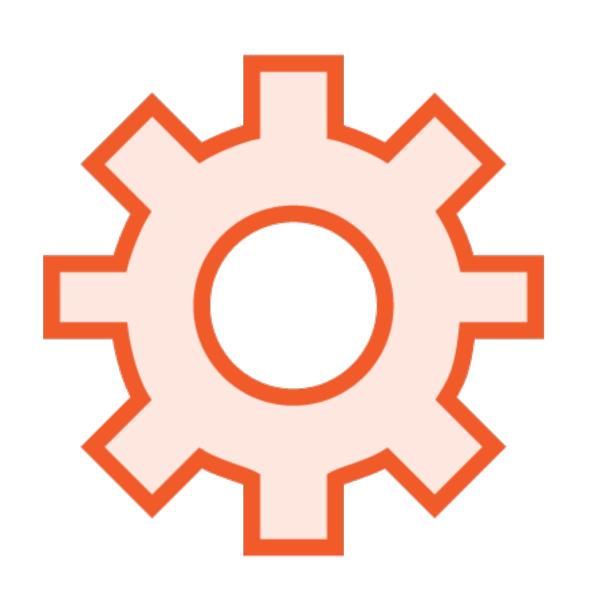
Subject and BehaviorSubject

Reacting to actions

Starting with an initial value



RxJS Features



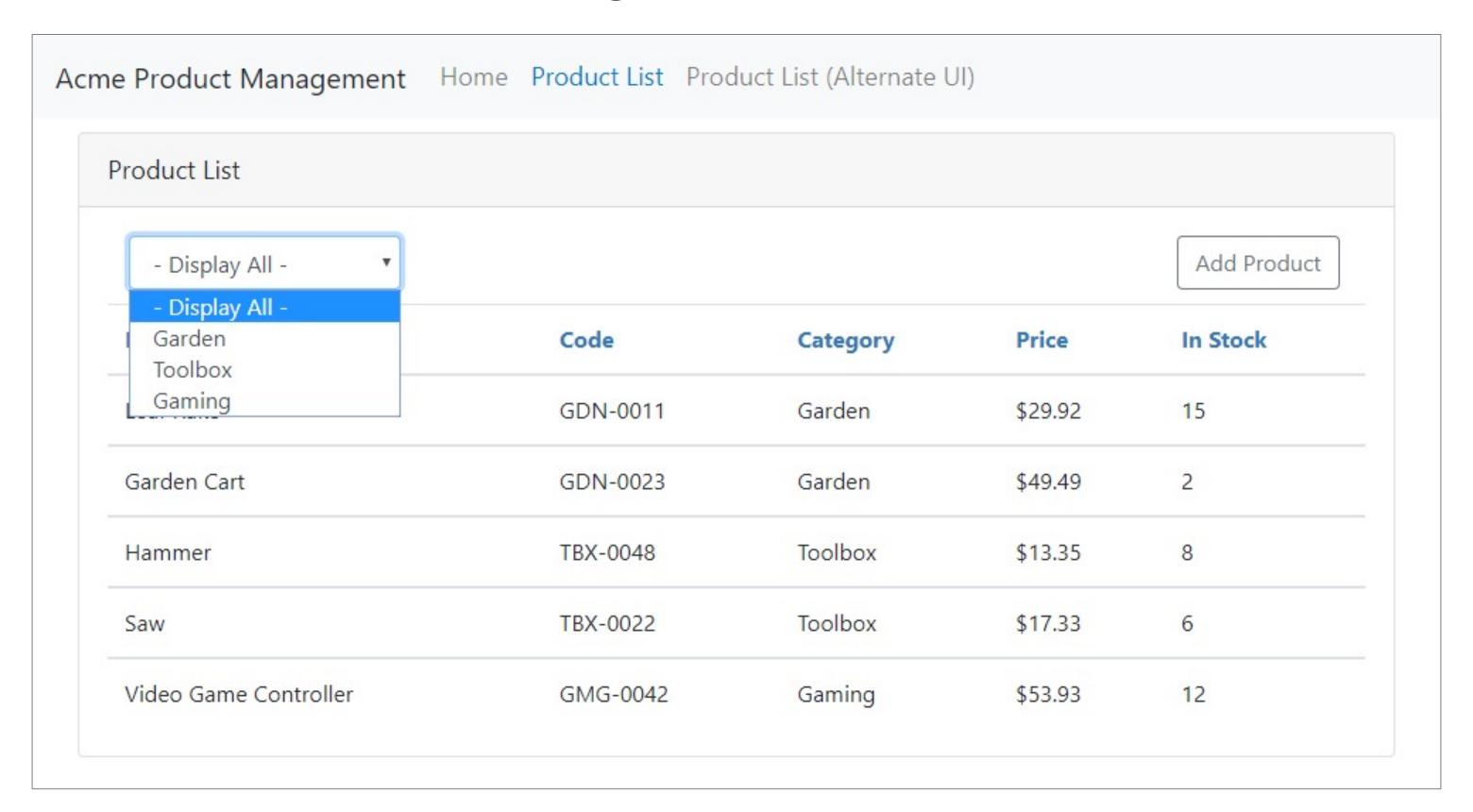
filter

startWith

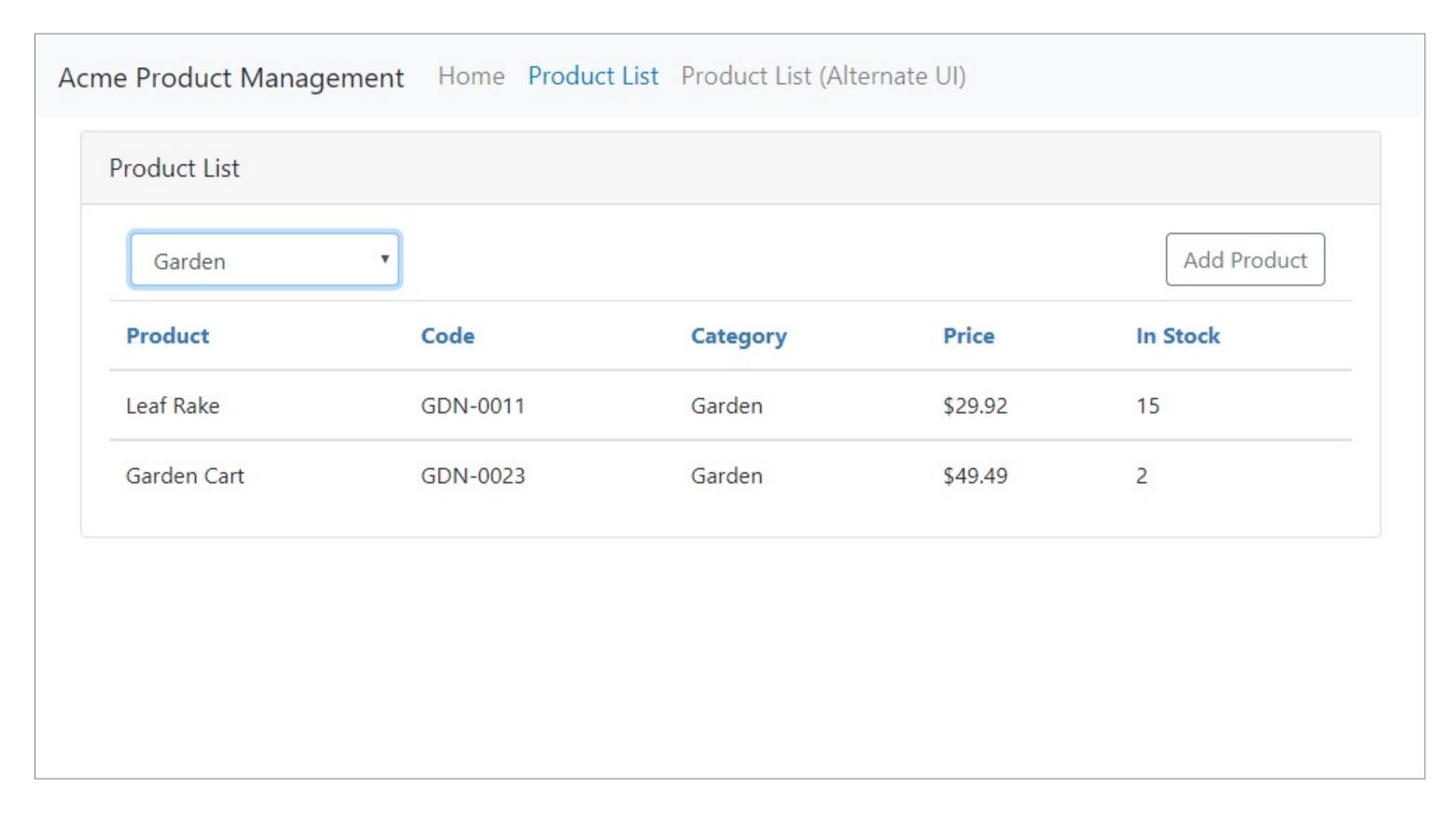
Subject

BehaviorSubject

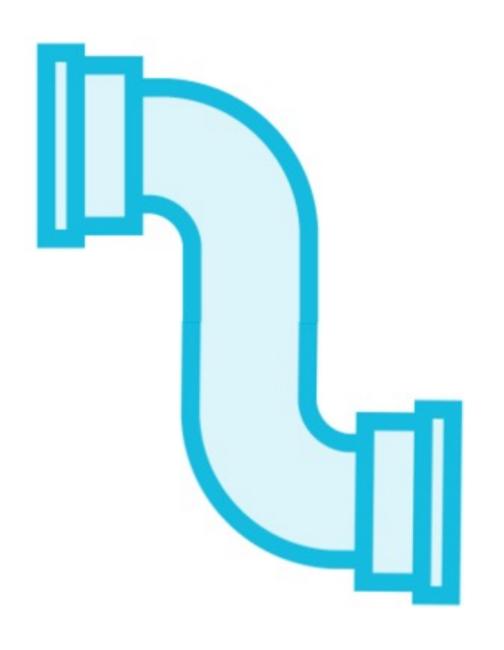
Filtering Emitted Items



Filtering Emitted Items



RxJS Operator: filter



Filters to the items that match criteria specified in a provided function

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

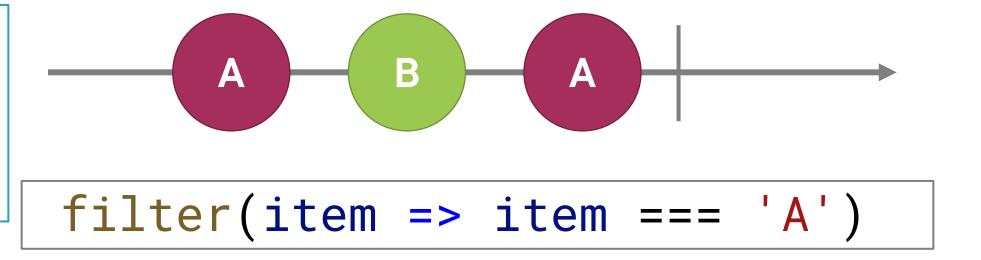
Similar to the array filter method

Used for

- Emitting items that match specific criteria

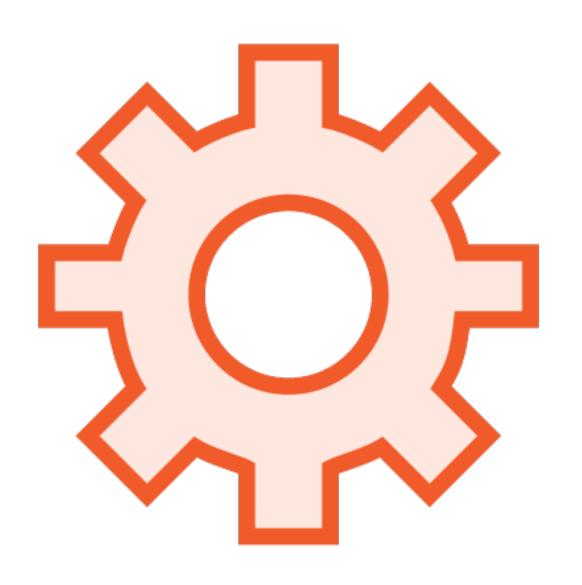
Marble Diagram: filter

```
of('A', 'B', 'A')
  .pipe(
    filter(item => item === 'A')
  );
```





RxJS Operator: filter



filter is a transformation operator

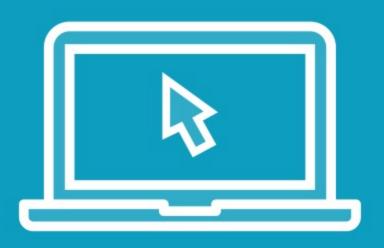
- Subscribes to its input Observable
- Creates an output Observable

When an item is emitted

- Item is evaluated as specified by the provided function
- If the evaluation returns true, item is emitted to the output Observable



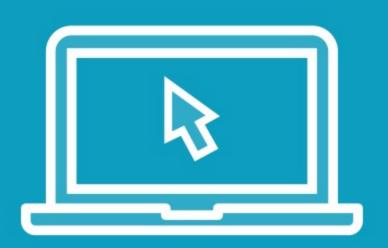
Demo



Filtering emitted items: Demo I

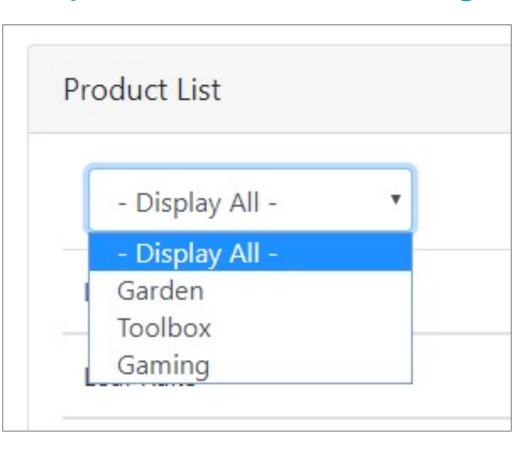
Hard-coded category

Demo



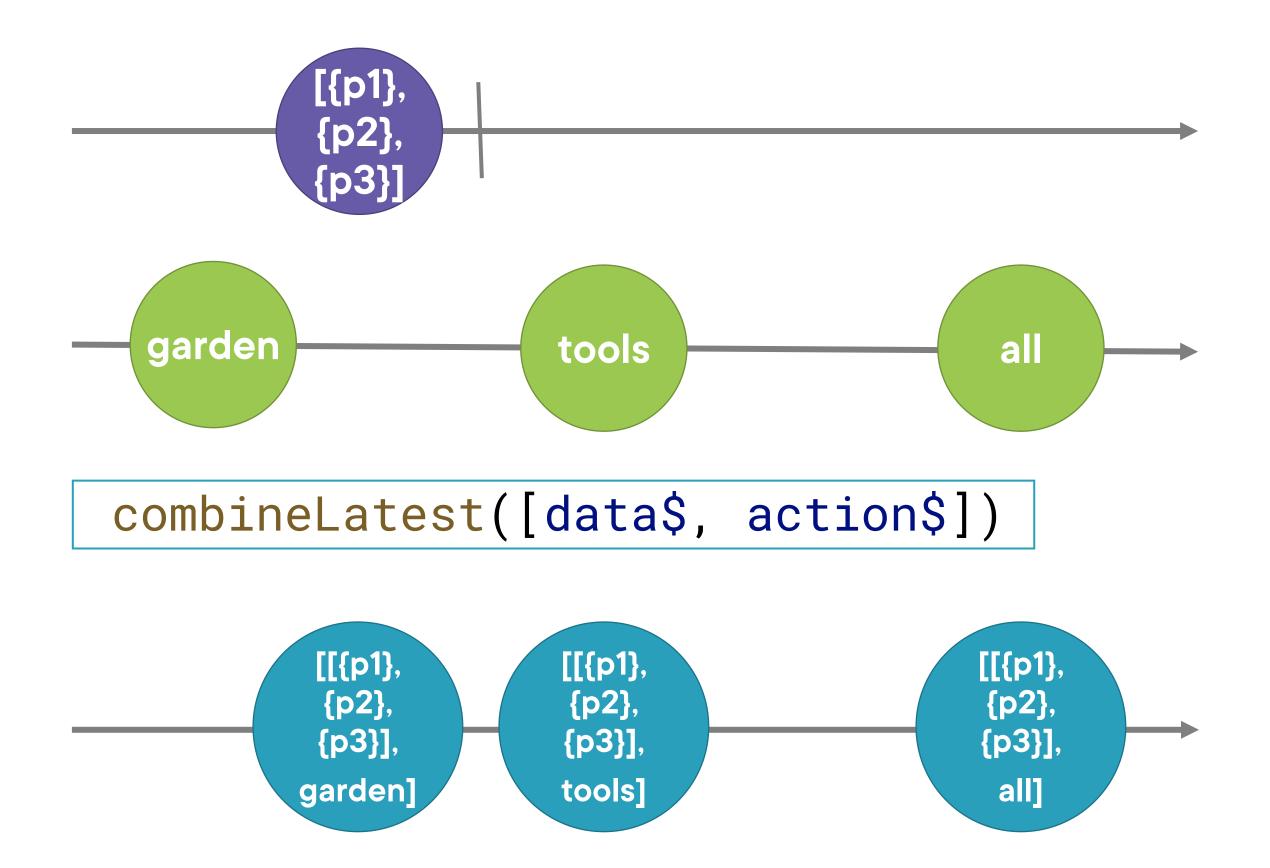
Filtering emitted items: Demo II

- Dropdown list of categories



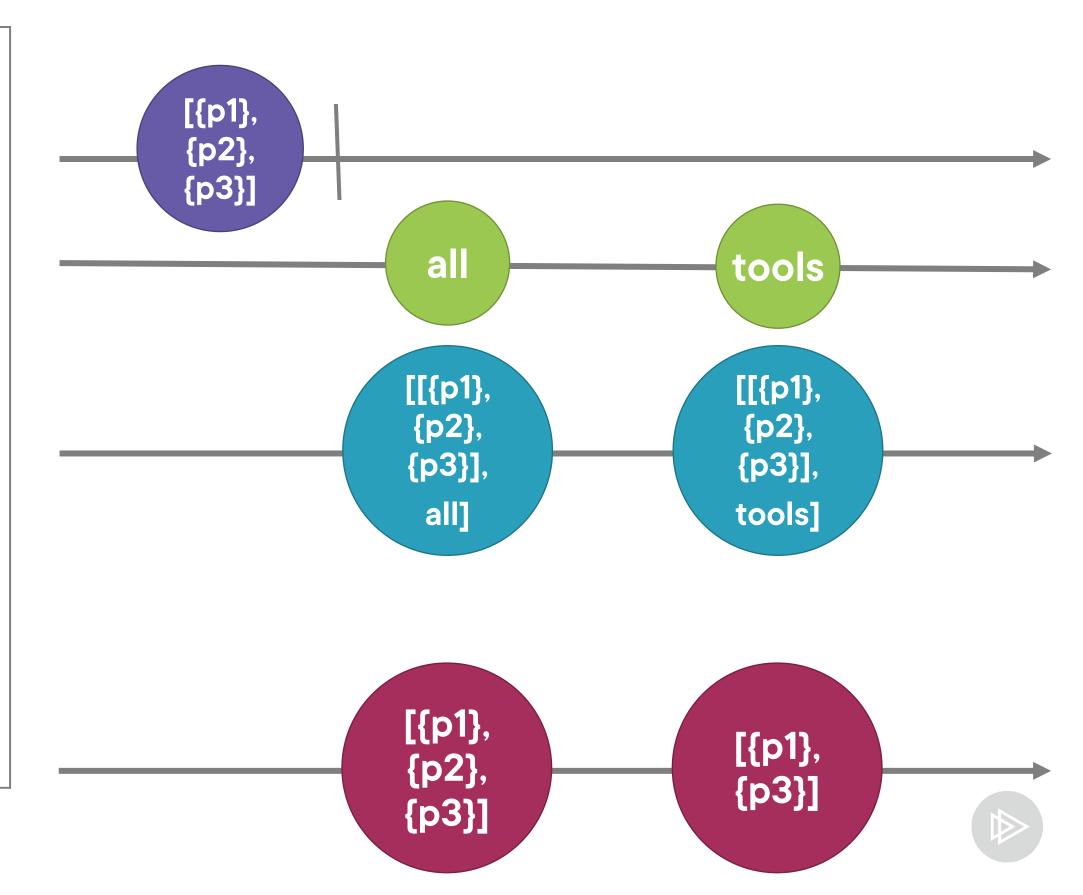


Data Stream vs. Action Stream

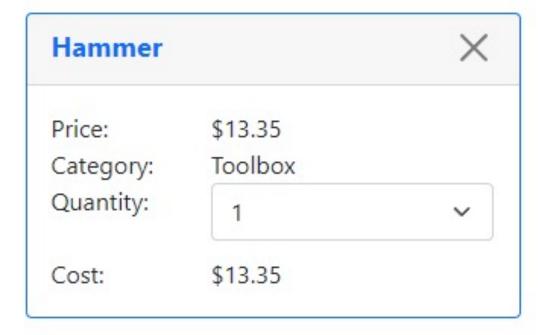


Combining a Data Stream and Action Stream

```
products$ = combineLatest([
  this.productService.products$,
  this.action$
.pipe(
 map(([products, category]) =>
  products.filter(product =>
     product.category === category)
```

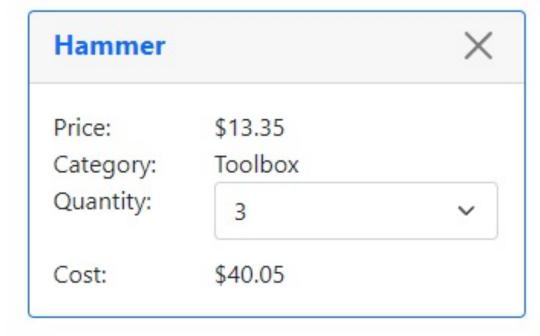








React when a user changes the quantity







Observable

Pseudo Code

Declare an Observable

Emit when an action occurs

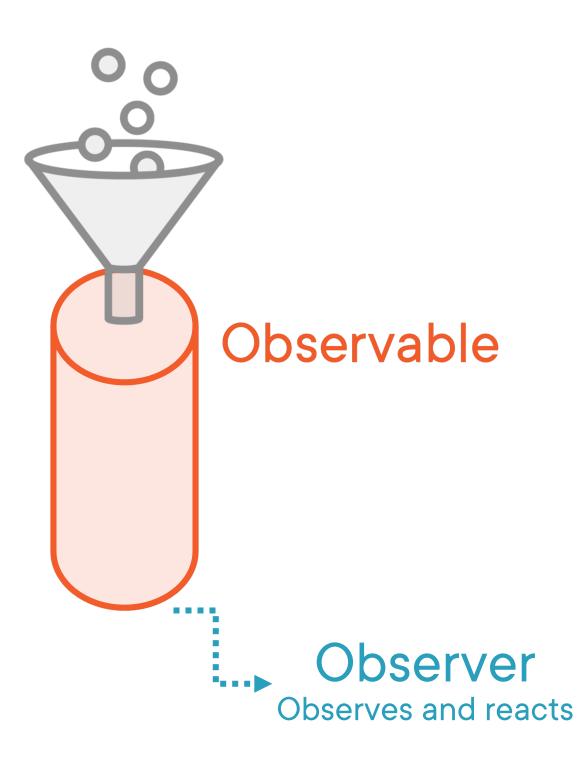
React to emissions

```
item = "Hammer";
price = 13.35;
quantity = 1;
qty$ = new Observable();
```

```
onQuantityChanged(newQty) {
   qty$.emit(newQty);
}
```

```
exPrice$ = qty$.pipe(
   map(q => q * price)
);
```

Observable



From the point of view of the subscriber

- An Observable is read only
- Subscribe to react to its notifications
- Can't emit anything into it

Only the creator of the Observable can emit items into it

Only the Observable Creator Can Emit Items

```
const apples$ = of('Apple1', 'Apple2');

const apples$ = from(['Apple1', 'Apple2']);

products$ = this.http.get<Product[]>(this.productsUrl)
   .pipe(
    catchError(this.handleError)
);
```

```
const apples$ = new Observable(appleSubscriber => {
   appleSubscriber.next('Apple 1');
   appleSubscriber.next('Apple 2');
   appleSubscriber.complete();
});
```

Subscriber / Observer

```
const apples$ = new Observable(appleSubscriber => {
   appleSubscriber.next('Apple 1');
   appleSubscriber.next('Apple 2');
   appleSubscriber.complete();
});
```

Subscriber

An Observer with additional features to unsubscribe

Observer

- Observes and responds to notifications from an Observable
- An interface with next, error, and complete methods

A Subject is a special type of Observable that implements the Observer interface



A Subject is a special type of Observable that implements the Observer interface



A Subject is a special type of Observable that implements the Observer interface



Subject

A special type of Observable that is

- An Observable
- An Observer

```
actionSubject = new Subject<string>();
```

Call next() to emit items

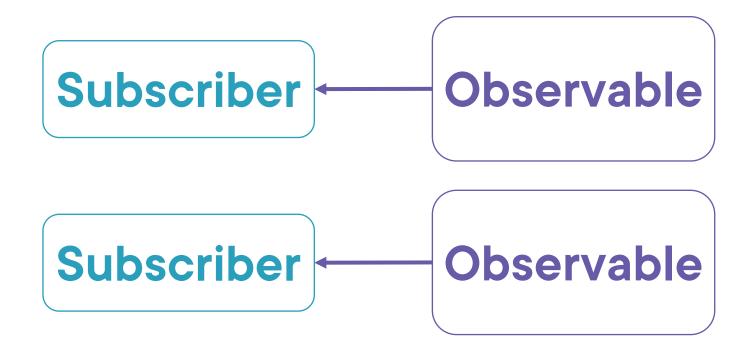
```
this.actionSubject.next('tools');
```

Call subscribe() to receive notifications

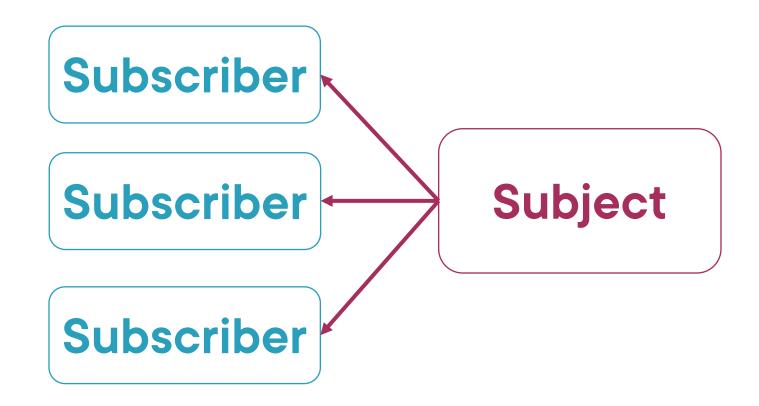
```
this.actionSubject.subscribe(
   item => console.log(item)
);
```



Unicast vs. Multicast



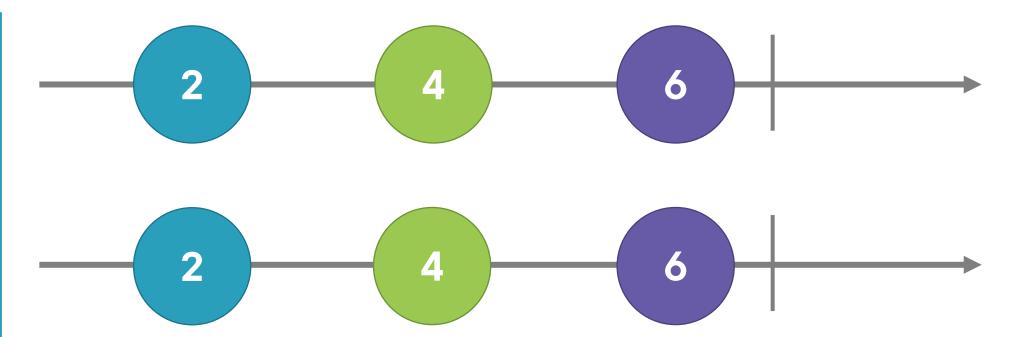
Observable is generally unicast



Subject is multicast

Observable: Unicast

```
numbers$ = of(2, 4, 6);
numbers$.subscribe(
    x => console.log('A', x)
);
numbers$.subscribe(
    x => console.log('B', x)
);
```



Console

A2

A4

A 6

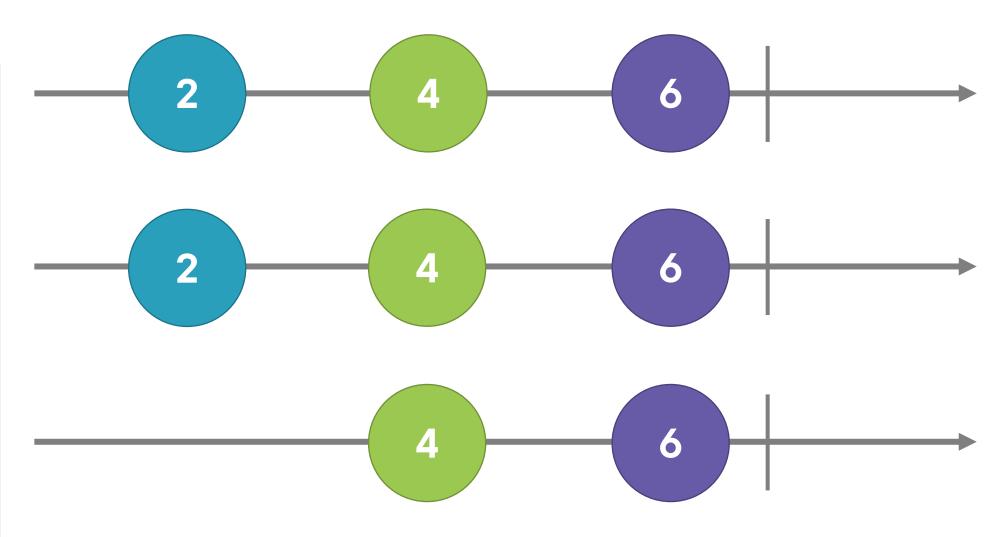
B 2

B4

B 6

Subject: Multicast

```
numbers = new Subject<number>();
numbers.subscribe(
  x => console.log('A', x)
numbers.next(2);
numbers.subscribe(
  x => console.log('B', x)
                      Console
numbers.next(4);
numbers.next(6);
                        A2
numbers.complete();
                        A4
                        B4
                        A 6
                        B 6
```



Hammer		×
Price:	\$13.35	
Category:	Toolbox	
Quantity:	3	~
Cost:	\$40.05	

Cart Total	
Subtotal:	\$13.35
Delivery:	\$5.99
Estimated	\$1.44
Tax:	
Total:	\$20.78

Behavior Subject



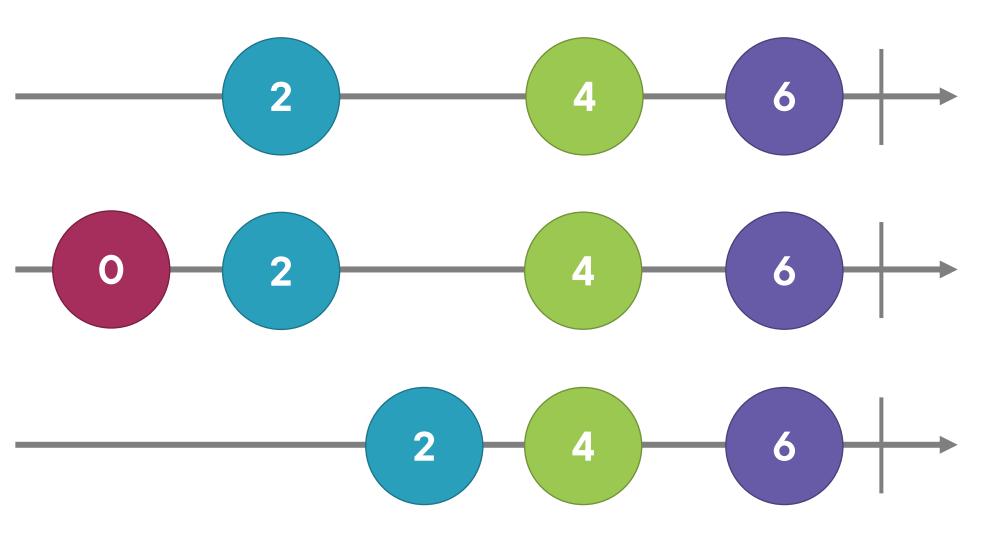
A special type of Subject that

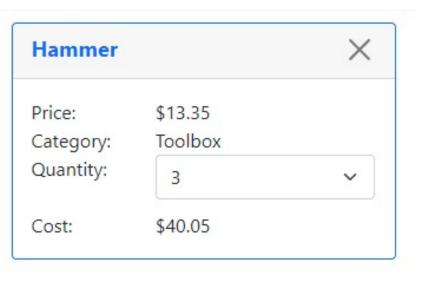
- Buffers its last emitted value
- Emits that value to any late subscribers
- Requires a default value
- Emits that default value if it hasn't yet emitted any items

aSub = new BehaviorSubject<number>(0);

BehaviorSubject

```
n = new BehaviorSubject<number>(0);
n.subscribe(
  x => console.log('A', x)
n.next(2);
n.subscribe(
  x => console.log('B', x)
                       Console
n.next(4);
                         AO
n.next(6);
                        A2
n.complete();
                         B 2
                         A4
                         B4
                         A6
                         B 6
```





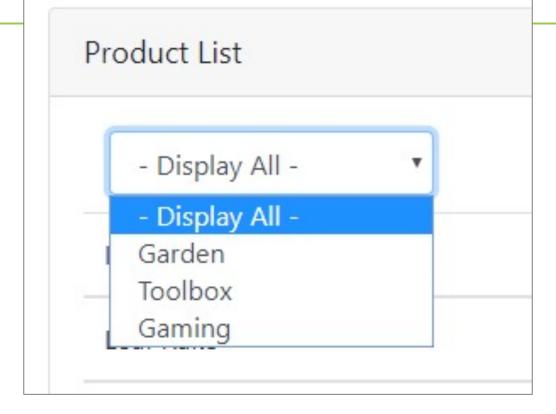


Creating an Action Stream

```
private categorySelectedSubject = new Subject<number>();
categorySelectedAction$ = this.categorySelectedSubject.asObservable();
```

```
onSelected(categoryId: string): void {
  this.categorySelectedSubject.next(+categoryId);
}
```

</select>



Reacting to Actions



Create an action stream (Subject/BehaviorSubject)



Combine the action stream and data stream to react to each emission from the action stream



Emit a value to the action stream when an action occurs

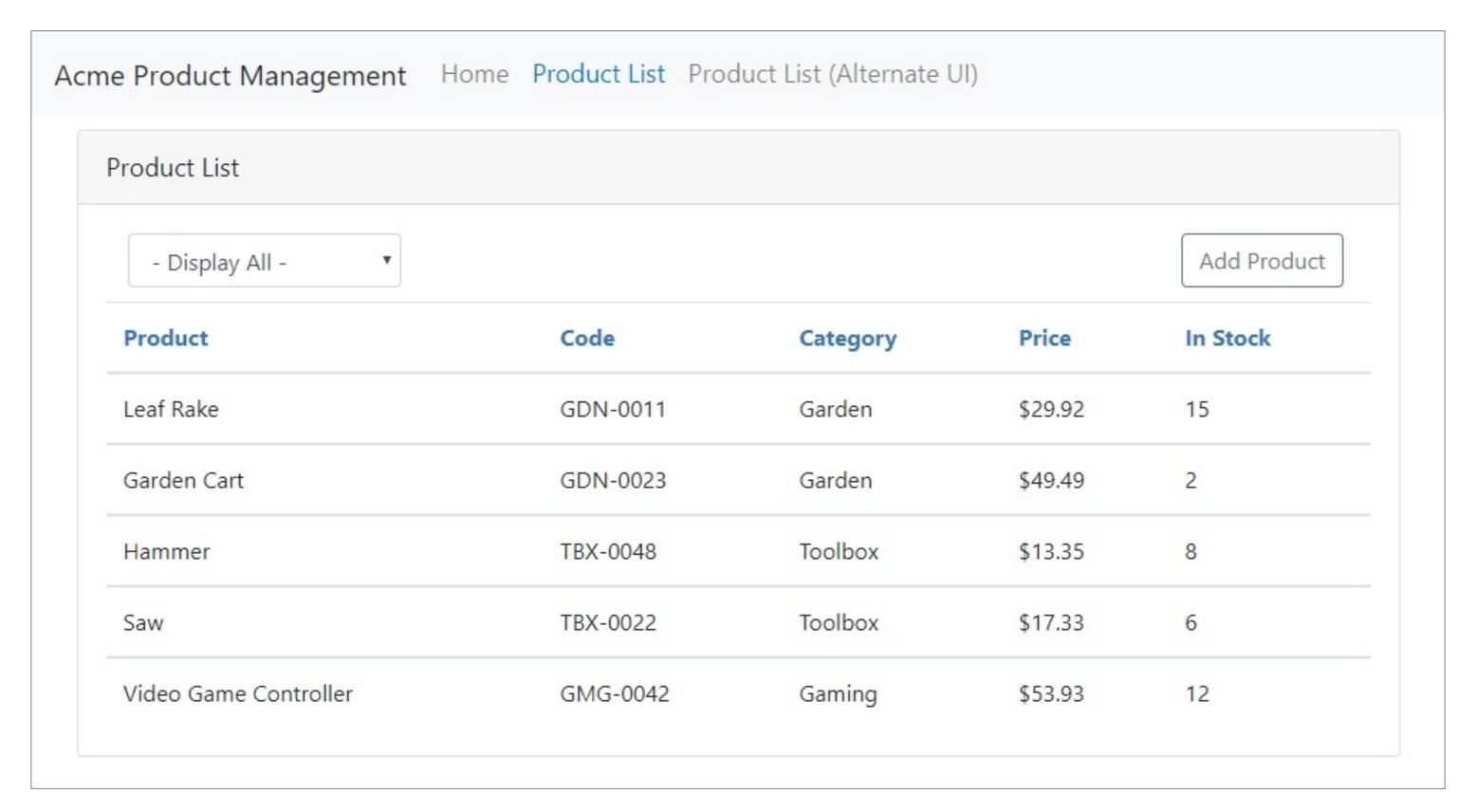
Reacting to Actions

```
private categorySelectedSubject = new Subject<number>();
categorySelectedAction$ = this.categorySelectedSubject.asObservable();
```

```
onSelected(categoryId: string): void {
  this.categorySelectedSubject.next(+categoryId);
}
```

```
products$ = combineLatest([
    this.productService.products$,
    this.categorySelectedAction$
])
    .pipe(
    map(([products, categoryId]) =>
        products.filter(product =>
        categoryId ? product.categoryId === categoryId : true
    ))
);
```

Starting with an Initial Value

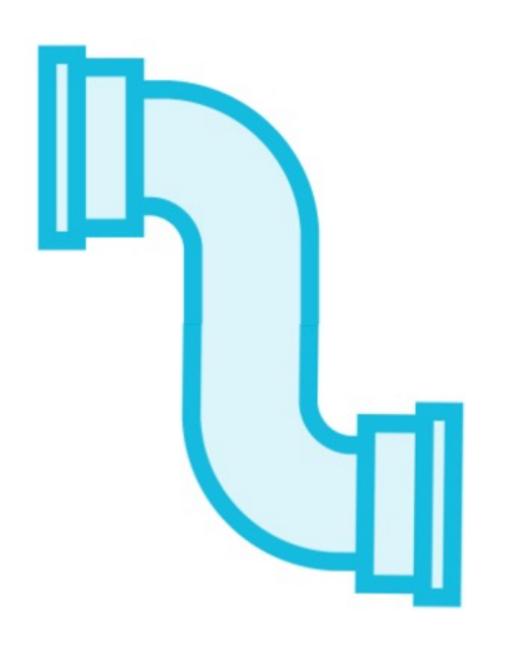


Starting with an Initial Value

```
this.categorySelectedAction$
   .pipe(
        startWith(0)
    )
```

```
private categorySelectedSubject = new BehaviorSubject<number>(0);
categorySelectedAction$ = this.categorySelectedSubject.asObservable();
```

RxJS Operator: startWith



Provides an initial value

startWith('Orange')

Emits its argument (in order)

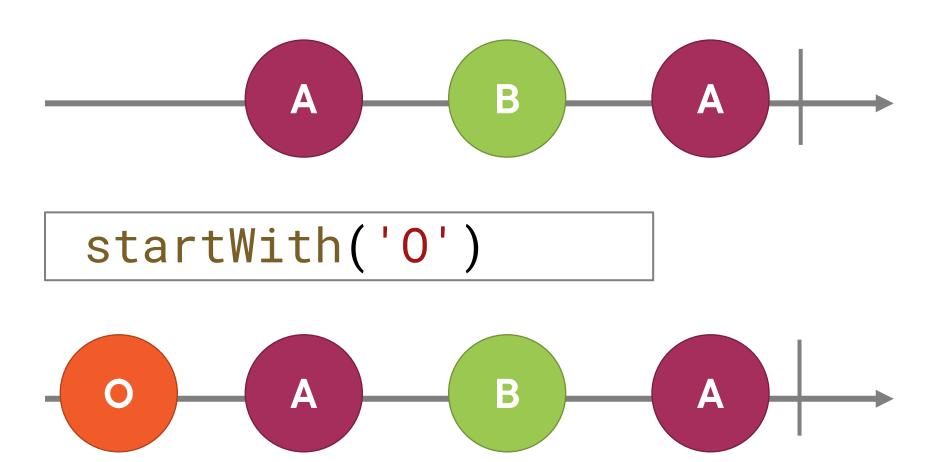
Then emits from the source

Used for

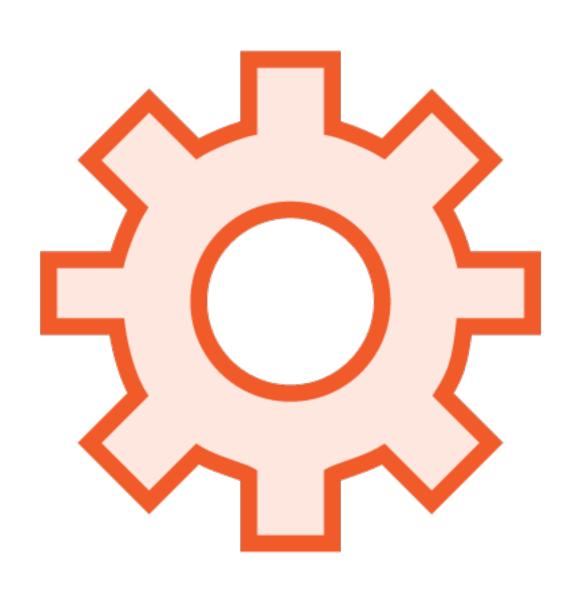
- Emitting initial item(s)

Marble Diagram: startWith

```
of('A', 'B', 'A')
  .pipe(
    startWith('O')
  )
  .subscribe(x => console.log(x));
```



RxJS Operator: startWith



startWith is a combination operator

- Subscribes to its input Observable
- Creates an output Observable
- When subscribed, synchronously emits all provided values

When an item is emitted

Item is emitted to the output Observable

Initial value(s) must be the same type as the input Observable emissions



RxJS Checklist: Subject + Behavior Subject



Subject: Special type of Observable that is both

- An Observable with a subscribe() method
- An Observer with next(), error(), and complete() methods

```
actionSubject = new Subject<string>();
```

BehaviorSubject: Special type of Subject that

- Buffers its last emitted value
- Emits that value to any late subscribers
- Requires a default value
- Emits that default value if it hasn't yet emitted any items

```
actionSubject = new BehaviorSubject<number>(0);
```

RxJS Checklist: Subject vs Behavior Subject



Use Subject if you don't need an initial value

Use BehaviorSubject if you want an initial value

- Important when using combineLatest



RxJS Checklist: Reacting to Actions



Create an action stream (Subject/BehaviorSubject)

```
private actionSubject = new Subject<number>();
action$ = this.actionSubject.asObservable();
```

Combine the action and data streams

```
products$ = combineLatest([
    this.products$,
    this.action$
]).pipe(...);
```

Emit a value to the action stream when an action occurs

```
onSelected(categoryId: string): void {
  this.actionSubject.next(+categoryId);
}
```

RxJS Checklist: Features



filter: Emits items that match specified criteria

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

startWith: Emits specified values, then the source values

```
startWith('Orange')
```



Coming up next...

Reacting to Actions: Examples

