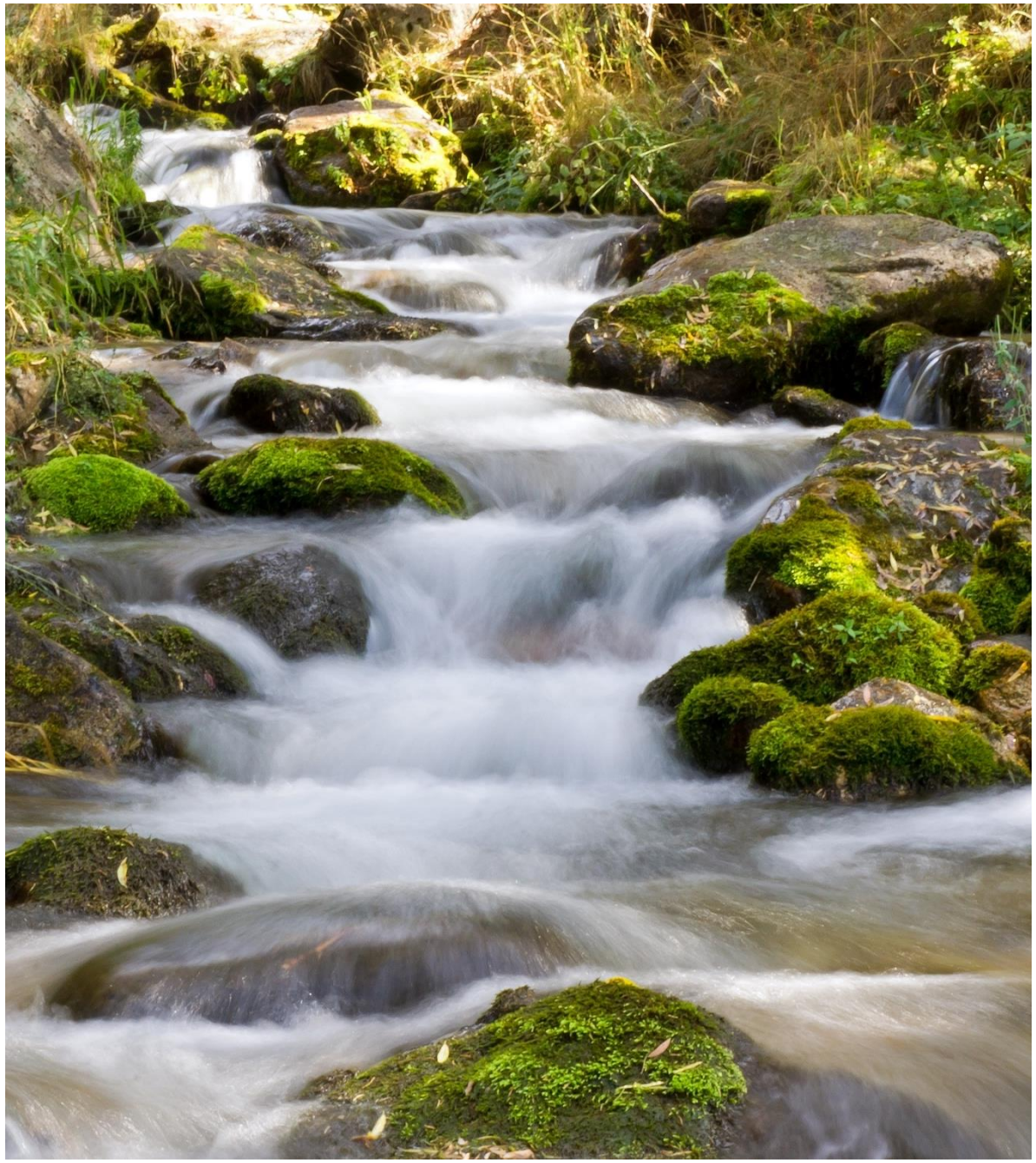
RxJS Terms and Syntax

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/



Processing Observable Streams

Start the stream

- Emit items into the stream

Items pass through a set of operations

As an observer

- Next item, process it
- Error occurred, handle it
- Complete, you're done

Stop the stream

Processing Observable Streams

Apple Factory	RxJS
Start the stream - Emits items	**Subscribe** - Emits items
Items pass through a set of operations	**Pipe through a set of operators**
As an observer - Next item, process it - Error occurred, handle it - Complete, you're done	**Observer** - next() - error() - complete()
Stop the stream	**Unsubscribe**

Module Overview

Observer/Subscriber

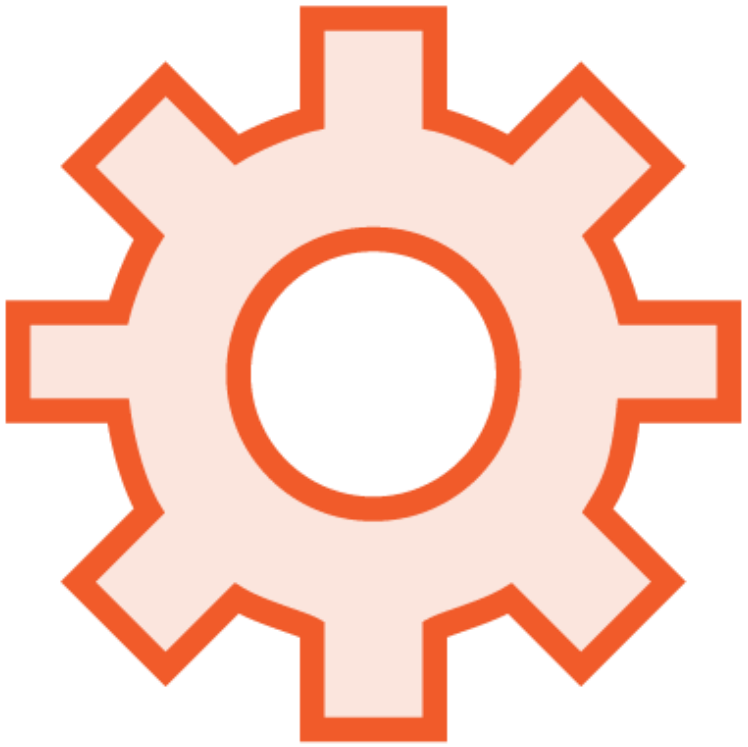
Observable stream (Observable)

Starting the Observable stream / Subscription

Stopping the Observable stream

Creation functions

RxJS Features



```
of

from
```

Observer

As an observer

- Next item, process it
- Error occurred, handle it
- Complete, you're done

Observer

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Observer

next()

error()

complete()

Observes the stream and responds to its notifications

Observer

"Observer: is a collection of callbacks that knows how to listen to values delivered by the Observable."

"A JavaScript object that defines the handlers for the notifications you receive."

In RxJS, an Observer is also defined as an interface with next, error, and complete methods.

Observer

- next()
- error()
- complete()

Observes the stream and responds to its notifications

Subscriber

Subscriber

- next()
- error()
- complete()

Observer that can unsubscribe from an Observable

Observer

- next()
- error()
- complete()

Observes the stream and responds to its notifications

Observer

```
const observer = {
  next: apple => console.log(`Apple was emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

Observable Stream

Stream of apples moving on a conveyor

Observable Stream

Stream of apples moving on a conveyor

Any stream of data, optionally produced over time

- Numbers
- Strings
- Events
- Object literals
- Response returned from an HTTP request
- Other Observable streams

Observable Stream

Also called:
- An Observable sequence
- An Observable
- A stream

Observables can be synchronous or asynchronous

Observables can emit a finite or infinite number of values

Any stream of data, optionally produced over time

- Numbers
- Strings
- Events
- Object literals
- Response returned from an HTTP request
- Other Observable streams

Observable

```
const observer = {
  next: apple => console.log(`Apple was emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

```
const appleStream = new Observable(appleObserver => {
  appleObserver.next('Apple 1');
  appleObserver.next('Apple 2');
  appleObserver.complete();
});
```

Subscription

Start the stream

- Emits items

Items pass through a set of operations

As an observer

- Next item, process it
- Error occurred, handle it
- Complete, you're done

Stop the stream

Subscription

Start the stream

Emits items

Items pass through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop the stream

Call subscribe() on the Observable

MUST subscribe to start the Observable stream

Subscription

```
const observer = {
  next: apple => console.log(`Apple was emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

```
const appleStream = new Observable(appleObserver => {
  appleObserver.next('Apple 1');
  appleObserver.next('Apple 2');
  appleObserver.complete();
});
```

```
const sub = appleStream.subscribe(observer);
```

Subscription

```
const observer = {
  next: apple => console.log(`Apple was emitted ${apple}`),
  error: err => console.log(`Error occurred: ${err}`),
  complete: () => console.log(`No more apples, go home`)
};
```

```
const sub = appleStream.subscribe(observer);
```

```
const sub = appleStream.subscribe(
  apple => console.log(`Apple was emitted ${apple}`),
  err => console.log(`Error occurred: ${err}`),
  () => console.log(`No more apples, go home`)
);
```

Subscription

```
const appleStream = new Observable(appleObserver => {
  appleObserver.next('Apple 1');
  appleObserver.next('Apple 2');
  appleObserver.complete();
});
```

```
const sub = appleStream.subscribe(
  apple => console.log(`Apple was emitted ${apple}`),
  err => console.log(`Error occurred: ${err}`),
  () => console.log(`No more apples, go home`)
);
```

Stopping the Stream

Start the stream

Emit items

Items pass through a set of operations

As an observer

Next item, process it

Error occurred, handle it

Complete, you're done

Stop the stream

Stopping an Observable Stream

Technique	Completes?

Unsubscribe

```
const sub = appleStream.subscribe(observer);
```

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

Properly unsubscribing from each Observable prevents memory leaks

Subscription

```
const appleStream = new Observable(appleObserver => {
  appleObserver.next('Apple 1');
  appleObserver.next('Apple 2');
  appleObserver.complete();
});
```

```
const sub = appleStream.subscribe(
  apple => console.log(`Apple was emitted ${apple}`),
  err => console.log(`Error occurred: ${err}`),
  () => console.log(`No more apples, go home`)
);
```

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

In Angular, we often work with Observables that Angular creates for us.

Creating an Observable

```
const appleStream = new Observable(appleObserver => {
  appleObserver.next('Apple 1');
  appleObserver.next('Apple 2');
  appleObserver.complete();
});
```

```
const appleStream = of('Apple1', 'Apple2');
```

```
const appleStream = from(['Apple1', 'Apple2']);
```

# of vs. from

```
const apples = ['Apple 1', 'Apple 2'];
```

```
of(apples);
// [Apple1, Apple2]
```

```
from(apples);
// Apple1 Apple2
```

```
of(...apples);
// Apple1 Apple2
```

Creating an Observable

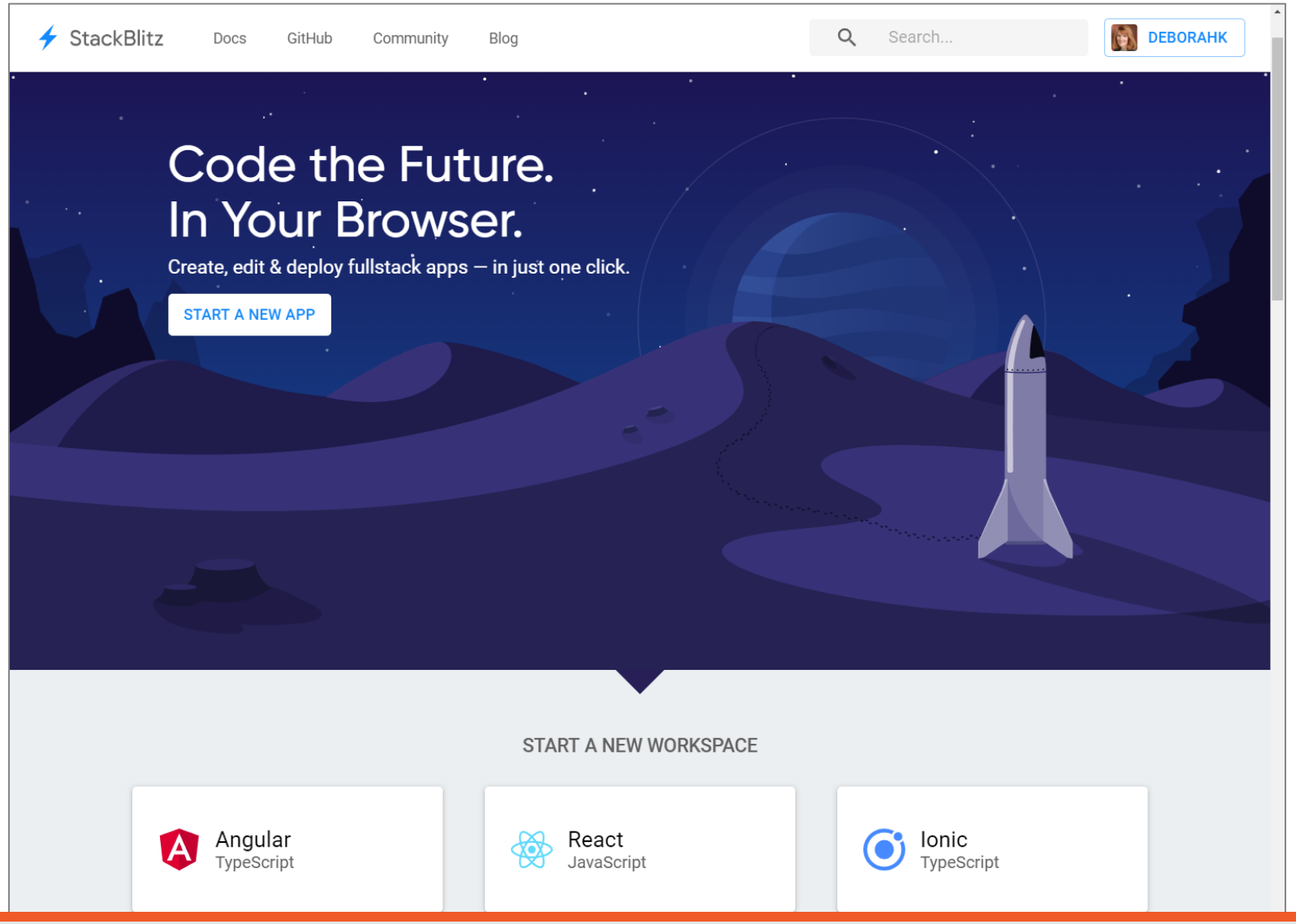
```
@ViewChild('para') par: ElementRef;

ngAfterViewInit() {
  const parStream = fromEvent(this.par.nativeElement, 'click')
                              .subscribe(console.log)
}
```

```
const num = interval(1000).subscribe(console.log);
```

Demo

Creation functions:

- `of`
- `from`



https://stackblitz.com

Terms

Observable

- Any stream of data

Observer

- Observes the stream
- Methods to process notifications from the stream: next(), error(), complete()

Subscriber

- An Observer that can unsubscribe

Subscription

- Represents the execution of an Observable
- subscribe() returns a Subscription

Creating an Observable

Constructor

Creation functions

- of, from, fromEvent, interval, ...
- Create an Observable from anything

Returned from an Angular feature

- Forms: valueChanges
- Routing: paramMap
- HTTP: get
- ...

Starting an Observable

Call subscribe!

Pass in an Observer

- next(), error(), complete()

```
const sub = appleStream.subscribe(
  apple => console.log(`Emitted: ${apple}`),
  err => console.log(`Error: ${err}`),
  () => console.log(`No more apples, go home`)
);
```

Stopping an Observable

Call complete() on the Observer

Use a creation function that completes

- of, from, ...

Use an operator that completes

- take, ...

Throw an error

Call unsubscribe() on the Subscription

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