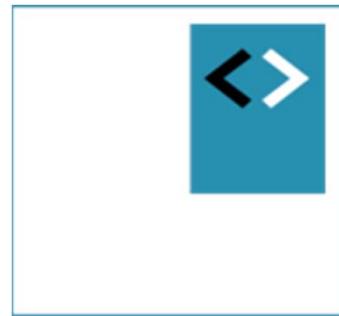




# Angular Fundamentals

## Module 4 – Observables

Lendex powered by  NIBC



Peter Kassenaar  
[info@kassenaar.com](mailto:info@kassenaar.com)



# Async services with RxJS/Observables

Reactive programming  
with asynchronous streams

# Async Services

- Fetching static data: *synchronous* action
- Working via HttpClient: *asynchronous* action
- Angular 1: Promises
- Angular 2: Observables

Angular : ReactiveX library RxJS



An API for asynchronous  
programming with observable streams

Choose your platform

<http://reactivex.io/>

## Languages

- Java: RxJava
- JavaScript: RxJS
- C#: Rx.NET
- C#(Unity): UniRx
- Scala: RxScala
- Clojure: RxClojure
- C++: RxCpp
- Ruby: Rx.rb
- Python: RxPY
- Groovy: RxGroovy
- JRuby: RxJRuby
- Kotlin: RxKotlin
- Swift: RxSwift

## ReactiveX for platforms and frameworks

- RxNetty
- RxAndroid
- RxCocoa

### DOCUMENTATION

Observable

Operators

Single

Combining

### LANGUAGES

RxJava<sup>®</sup>

RxJS<sup>®</sup>

Rx.NET<sup>®</sup>

RxClojure

### RESOURCES

Tutorials

### COMMUNITY

GitHub<sup>®</sup>

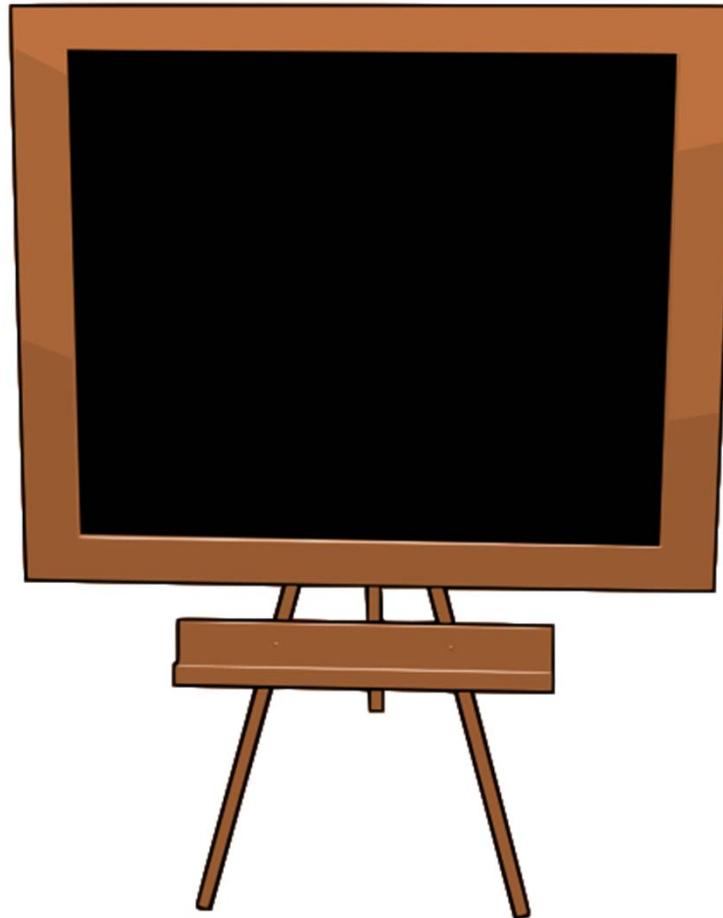
Twitter<sup>®</sup>

Others

# *The observable design pattern*

*“Understanding the  
observable stream”*

# Explanation



# Why Observables?

*"We can do **much more** with observables than with promises.*

*With observables, we have a whole bunch of operators to pull from, which let us customize our streams in nearly any way we want."*

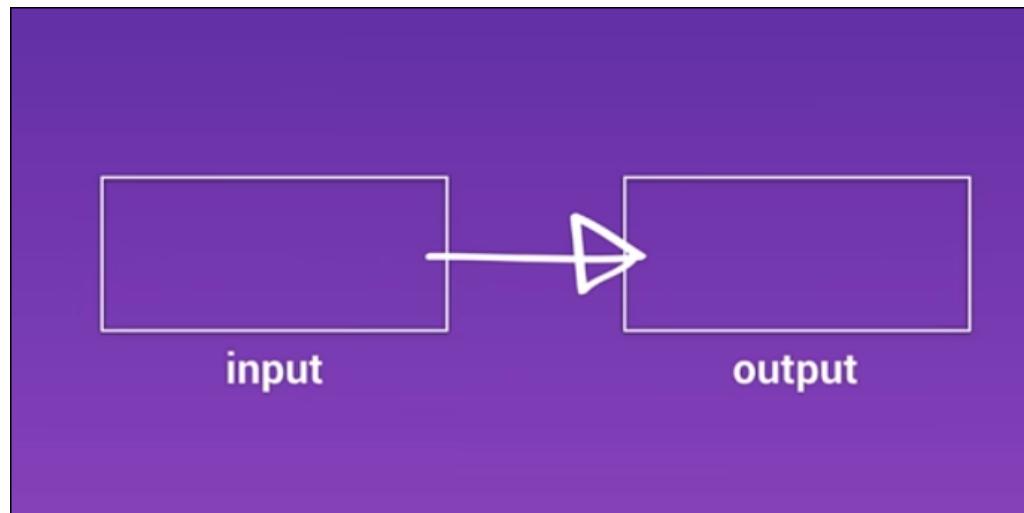
<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>

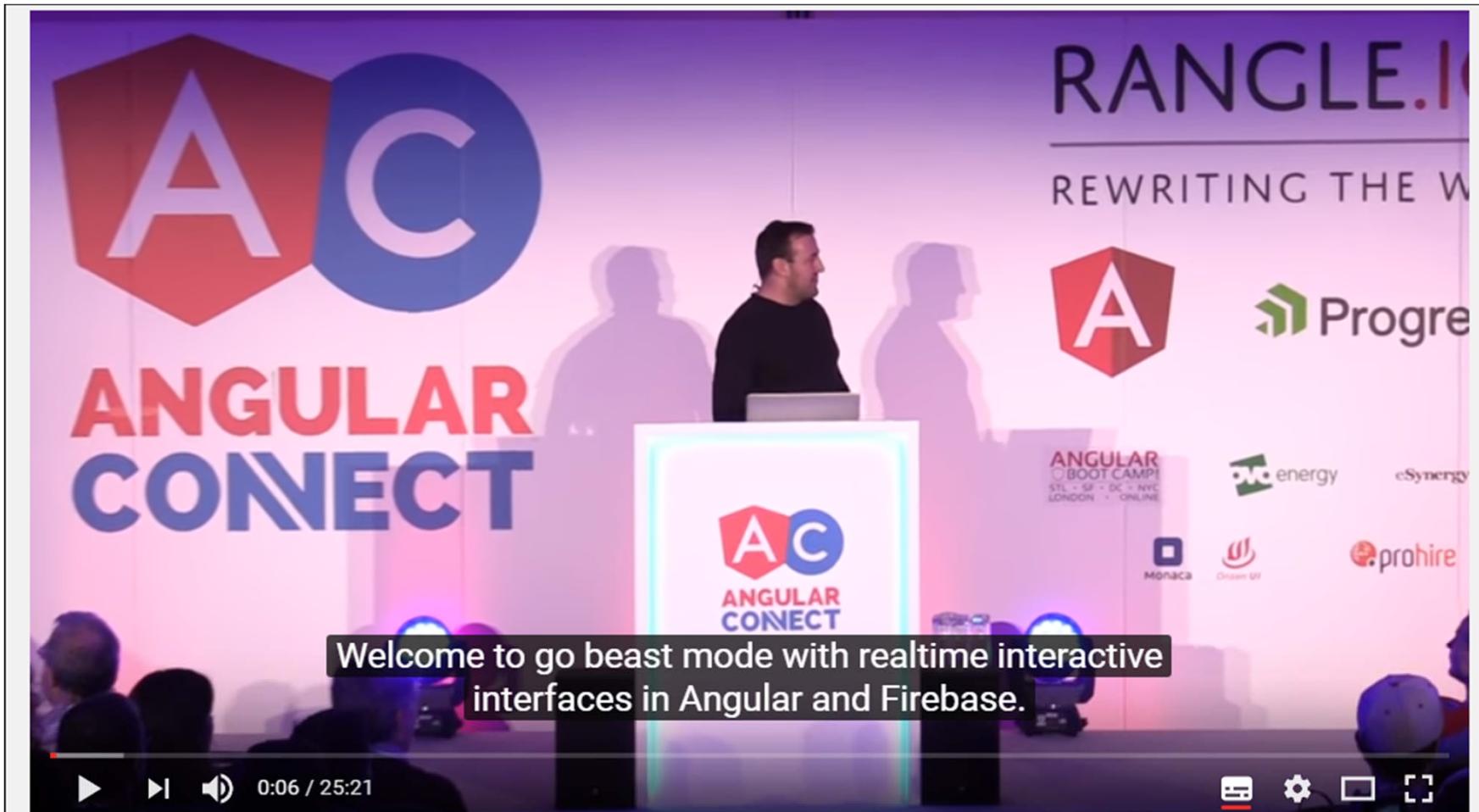
# Observables and RxJs

- “Reactive Programming”
  - *“Reactive programming is programming with asynchronous data streams.”*
  - <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- Observables have a lot of extra options, as opposed to Promises
  - Mapping
  - Filtering
  - Combining
  - Cancel
  - Retry
  - ...
- Consequence: no more `.success()`, `.error()` and `.then()` chaining!

# How do observables work

- First - The Observable Stream
- Later - all 10.000 operators...
- Traditionally:

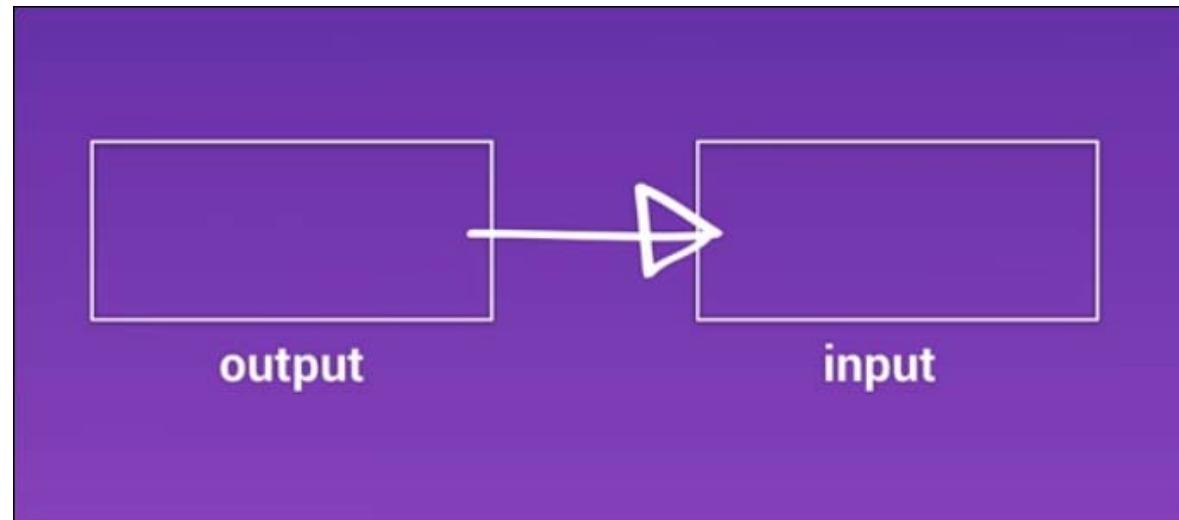




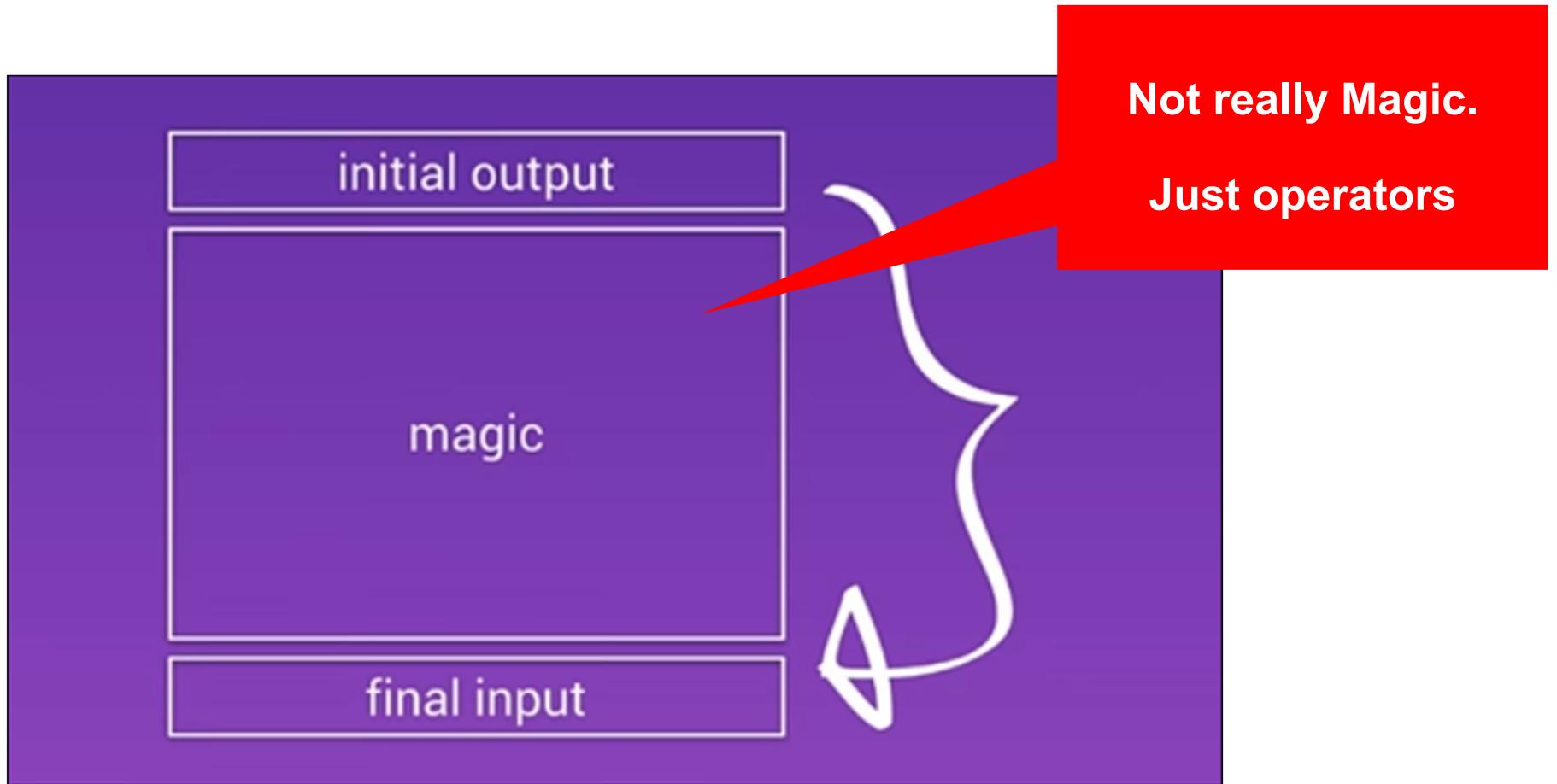
Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

<https://www.youtube.com/watch?v=5CTL7aqSvJU>

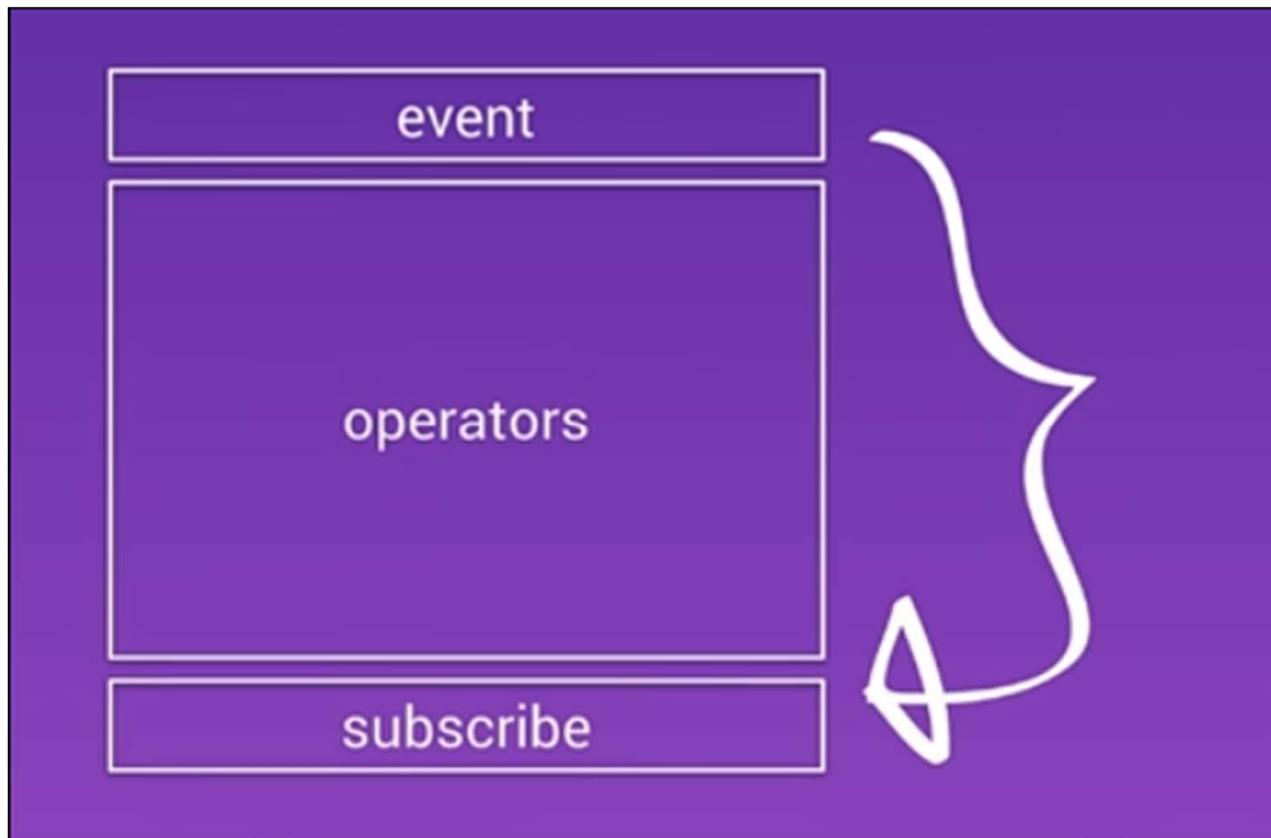
- With Observables -
  - a system, already outputting data,
  - Subscribe to that data
- "trade Output for Input"
- "Push vs. Pull"



# "The observable sandwich"



# "The Observable Sandwich"



In code, for http-call:

```
this.http.get<City[]>('assets/data/cities.json')
  .pipe(
    filter(...),
    map(...)
  )
  .subscribe((result) => {
    //... Do something
});
```

Initial Output

Optional:  
operator(s)

Final Input

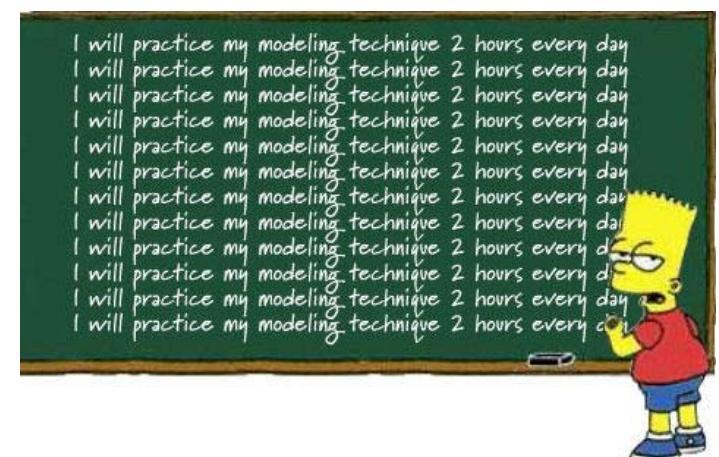
# Import HttpClientModule in @NgModule – don't forget

```
• // Angular Modules  
...  
• import {HttpClientModule} from '@angular/common/http';  
// Module declaration  
  
@NgModule({  
    imports : [BrowserModule, HttpClientModule],  
    declarations: [AppComponent],  
    bootstrap : [AppComponent],  
    providers : [CityService] // DI voor service  
})  
export class AppModule {  
}
```

# Exercise

- See the example in /201\_services\_http
- Create your own .json-file and import it in your application.
- Exercise 5c ), 5d )

# Exercise....





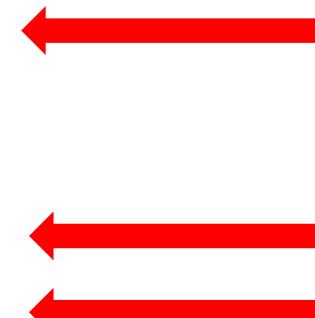
# More on subscriptions

Using parameters inside the subscriber

# Subscribe - only once per block!

- Part of RxJs
- Three parameters:
  - success( )
  - error( )
  - complete( )

```
this.cityService.getCities()
  .subscribe(cityData => {
    this.cities = cityData
  },
  err => console.log(err),
  ()=> console.log('Getting cities complete...'))
)
```



# Pipeable operators

- In RxJS 6.x and up: all operators inside `.pipe()` function
- The parameters of pipe function are the operators!
- Comma-separate different operators.
  - Don't forget `import {...} from 'rxjs/operators';`

```
.pipe(  
    delay(3000),  
    retry(3),  
    map(result => ...),  
    takeUntil(..condition...)  
)
```

# Background info: Ben Lesh on observables in RxJS 6.0

The two you care about

- rxjs
  - **Types:** Observable, Subject, BehaviorSubject, etc.
  - **Creation methods:** fromEvent, timer, interval, delay, concat, etc.
  - **Schedulers:** asapScheduler, asyncScheduler, etc.
  - **Helpers:** pipe, noop, identity, etc
- rxjs/operators
  - **All operators:** map, mergeMap, takeUntil, scan, and so one.

@benlesh

INTRODUCING RXJS6!

Ben Lesh

14:47 / 23:50

Introducing RxJS6! - Ben Lesh

<https://www.youtube.com/watch?v=JCXZhe6KsxQ>

<https://www.learnrxjs.io/>

The screenshot shows a web browser window displaying the [Operators](https://www.learnrxjs.io/learn-rxjs/operators) page from the [Learn RxJS](https://www.learnrxjs.io) website. The page has a sidebar on the left with a navigation menu. The main content area features a heading "Operators" and a section titled "Contents (By Operator Type)" listing various RxJS operators categorized by type.

**Sidebar Navigation:**

- Introduction
- LEARN RXJS
  - Operators
  - Combination
  - Conditional
  - Creation
  - Error Handling
  - Multicasting
  - Filtering
  - Transformation
  - Utility
  - Full Listing
- Subjects
- Recipes
- Concepts

**Main Content Area:**

## Operators

A complete list of RxJS operators with clear explanations, relevant resources, and executable examples.

*Prefer a complete list in alphabetical order?*

### Contents (By Operator Type)

- Combination
  - combineAll
  - combineLatest ★
  - concat ★
  - concatAll
  - endWith
  - forkJoin
  - merge ★
  - mergeAll
  - pairwise
  - race
  - startWith ★
  - withLatestFrom ★
  - zip
- Conditional
  - defaultIfEmpty
  - every
  - iif
  - sequenceEqual
- Creation



# Using the `async` pipe

Automagically `.subscribe()` and `.unsubscribe()`

# Async Pipe

- On `.subscribe()`, you actually need to`.unsubscribe()` to avoid memory leaks
  - Best practice
  - Not \*really\* necessary on HTTP-requests, but you do in other subscriptions.
- No more manually `.subscribe()` and`.unsubscribe()`:
  - **Use Angular async pipe**

- Component:

```
cities$: Observable<City[ ]>; // Now: Observable to Type
```

...

```
ngOnInit() {  
    // Call service, returns an Observable  
    this.cities$ = this.cityService.getCities()  
}
```

- View:

```
<li *ngFor="let city of cities$ | async">
```

<https://blog.angularindepth.com/angular-question-rxjs-subscribe-vs-async-pipe-in-component-templates-c956c8c0c794>  
(Background information)

# Working with Live API's

- MovieApp
- examples\210-services-live



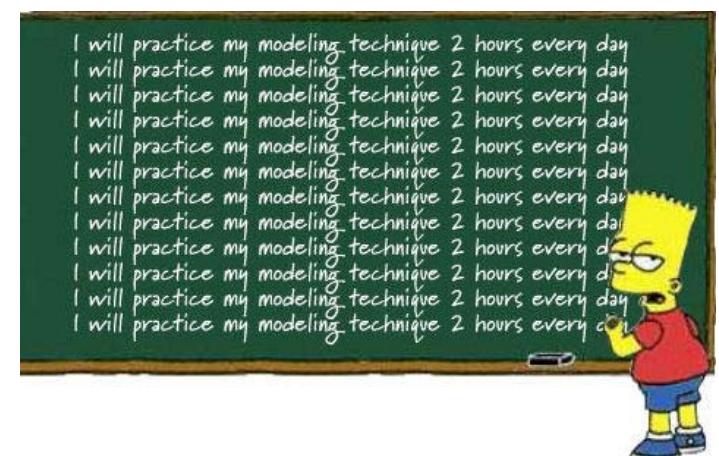
# Example API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weather forecast)
- <https://jsonplaceholder.typicode.com/> random users, posts, photos
- <http://ergast.com/mrd/> - Ergast Motor (F1) API
- <http://www.omdbapi.com/> - Open Movie Database
- <http://swapi.co/> - Star Wars API
- See also `JavaScript APIs.txt` with other API's.

# Exercise

- Pick one of your own projects, or see for examples:
  - 210-services-live
- Create a small application using one of the API's in the file JavaScript API's.txt, using RxJS-calls, for example
  - Star Wars API
  - OpenWeatherMap API
  - ...
- Exercise 5e)

# Exercise....





# More info on observables

# Fetching a collection of observables

<https://blog.angularindepth.com/practical-rxjs-in-the-wild-requests-with-concatmap-vs-mergemap-vs-forkjoin-11e5b2efe293>

The screenshot shows a blog post on the Angular In Depth website. The header features the 'Angular In Depth' logo with 'by AG-Grid' and navigation links for HOME, ANGULAR, RXJS, NGRX, ABOUT, SUPPORT US, and AG-GRID: THE BEST ANGULAR GRID IN THE WORLD. A 'Follow' button is also present. The main title of the post is 'Practical RxJS In The Wild' followed by a lion emoji. Below the title, the subtitle reads 'Requests with concatMap() vs mergeMap() vs forkJoin()' followed by a boxing glove emoji. The author's profile picture shows a man with a beard, and his name is Tomas Trajan. The post was published on Dec 19, 2017, and it is a 7 min read.

M |  **Angular In Depth** by 

HOME ANGULAR RXJS NGRX ABOUT SUPPORT US | AG-GRID: THE BEST ANGULAR GRID IN THE WORLD [Follow](#)

## Practical RxJS In The Wild —

### Requests with concatMap() vs mergeMap() vs forkJoin()

 Tomas Trajan [Follow](#)  
Dec 19, 2017 · 7 min read

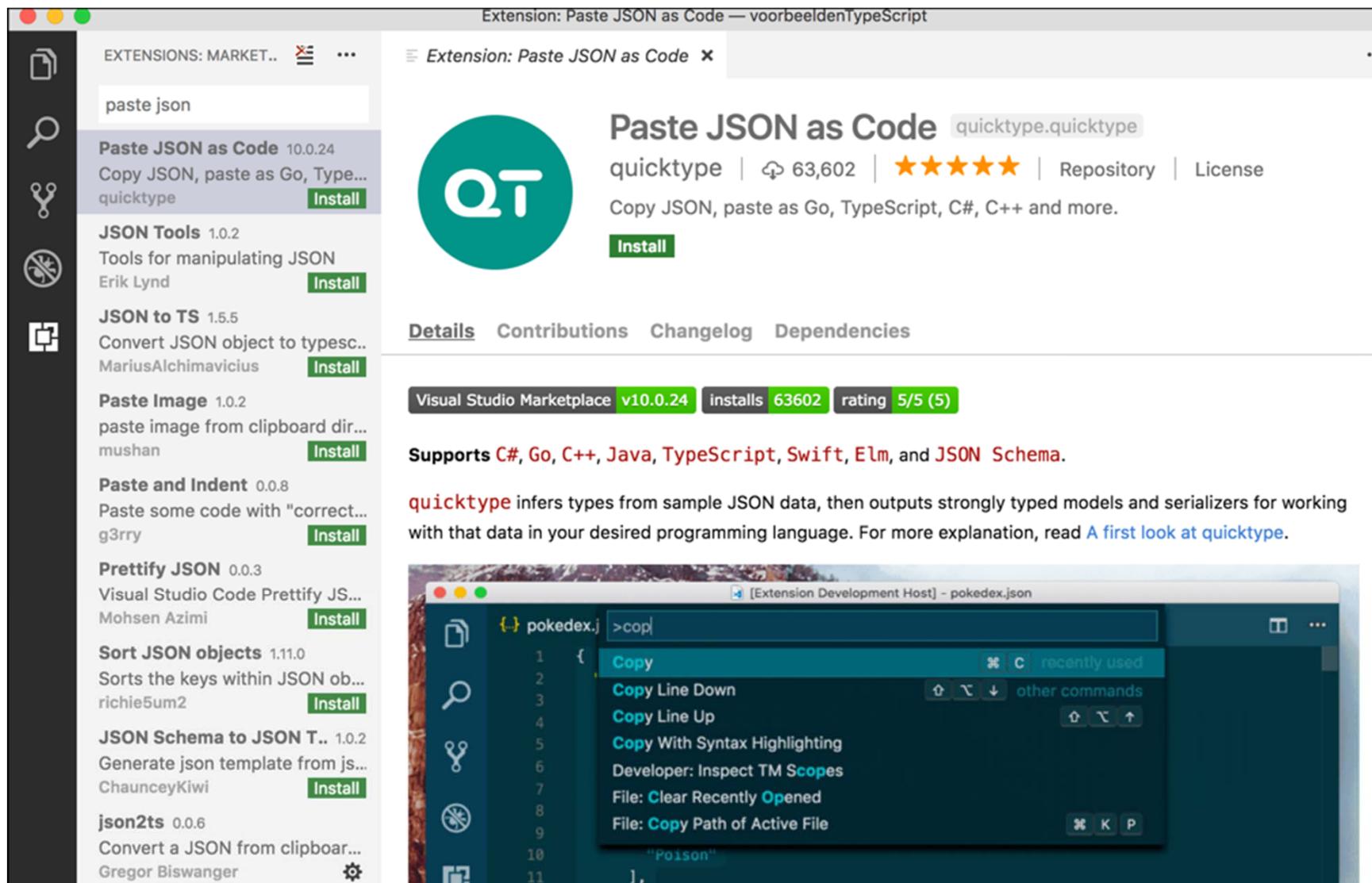
# Online JSON to TypeScript converter

The screenshot shows the json2ts website interface. At the top, the logo "json2ts" is displayed in large white letters on a blue background, with the subtitle "generate TypeScript interfaces from JSON" below it. On the right side of the header are links for "email", "feedback", and "help". The main content area contains a JSON string representing movie data. Below the JSON is a green button labeled "generate TypeScript". Underneath the button, the generated TypeScript code is shown:

```
declare module namespace {  
  
    export interface Search {  
        Title: string;  
        Year: string;  
        imdbID: string;  
        Type: string;  
        Poster: string;  
    }  
}
```

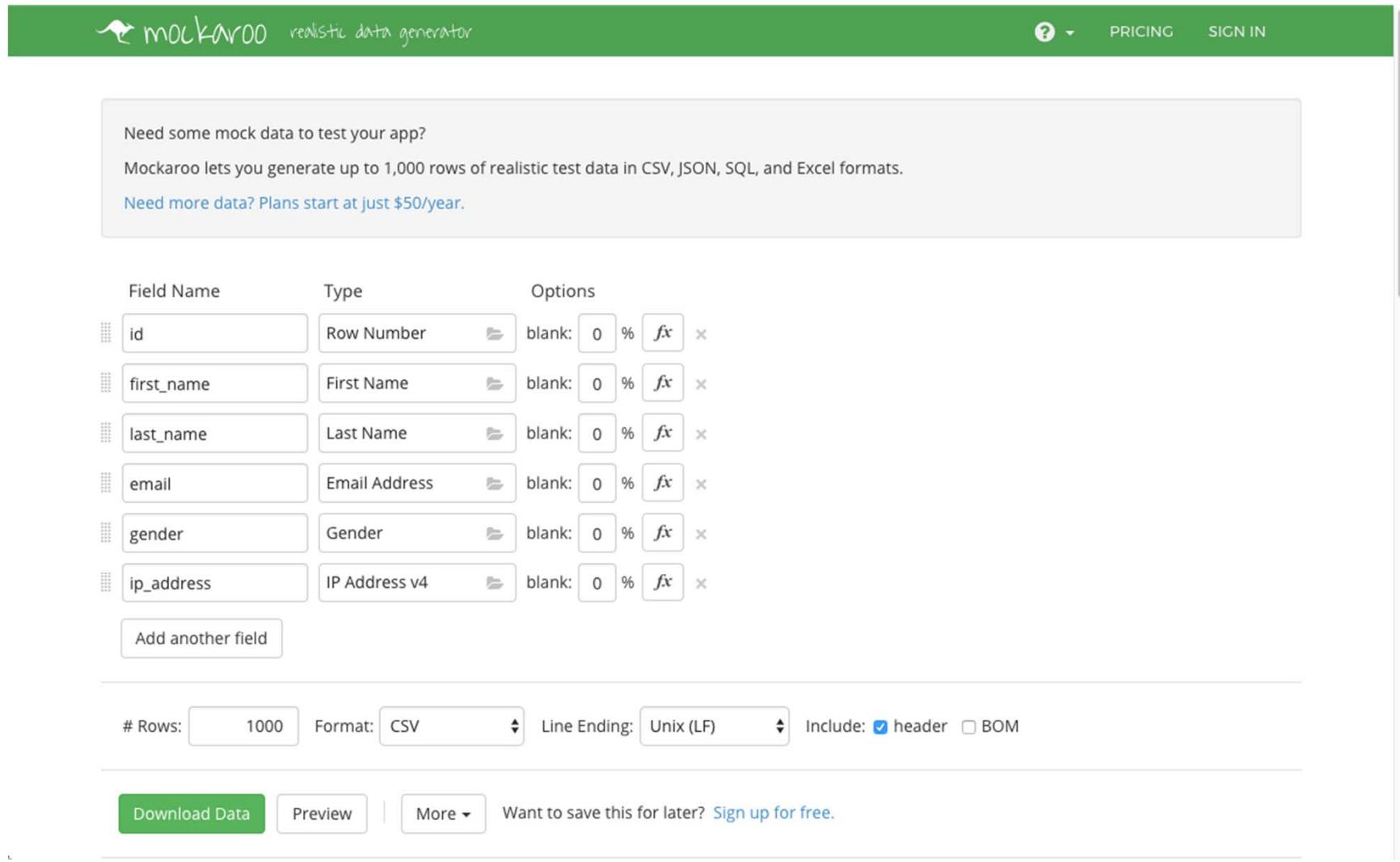
<http://json2ts.com/>

# In VS Code? Use this extension!



<https://marketplace.visualstudio.com/items?itemName=quicktype.quicktype>

# Data Mocking - Mockaroo



The screenshot shows the Mockaroo data generator interface. At the top, there's a green header bar with the Mockaroo logo, a search icon, and links for PRICING and SIGN IN. Below the header, a promotional message encourages users to generate up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats. It also mentions plans starting at \$50/year.

The main area contains a table for defining fields:

Field Name	Type	Options
id	Row Number	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>
first_name	First Name	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>
last_name	Last Name	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>
email	Email Address	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>
gender	Gender	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>
ip_address	IP Address v4	blank: 0 % <input type="button" value="fx"/> <input type="button" value="x"/>

A button labeled "Add another field" is located below the table. At the bottom, there are settings for "# Rows" (set to 1000), "Format" (set to CSV), "Line Ending" (set to Unix (LF)), and checkboxes for "Include:  header" and " BOM".

At the very bottom, there are three buttons: "Download Data" (green), "Preview", and "More". A link "Want to save this for later? [Sign up for free.](#)" is also present.

<http://mockaroo.com/>

# Useful operators

- RxJS operators are (mostly) just like Array operators
- Perform actions on a stream of objects
- Grouped by subject
  - Creation operators
  - Transforming
  - Filtering
  - Combining
  - Error Handling
  - Conditional and Boolean
  - Mathematical
  - ...

# 6 Operators you must know

The screenshot shows a Medium article page. At the top right are 'Sign in / Sign up' and social sharing icons for LinkedIn and Twitter. Below the header is the author's profile picture, Netanel Basal, followed by the name 'Netanel Basal' and a 'Follow' button, with a note 'Jan 24 · 3 min read'. The main title 'RxJS—Six Operators That you Must Know' is displayed prominently. A large black box contains the code for a 'TakeSubscriber' class, which extends 'Subscriber'. It includes a constructor that takes a 'destination' subscriber and a 'total' number, and a protected '\_next' method that checks if the current count is less than or equal to the total before emitting the value. At the bottom left is another profile picture, and at the bottom right are buttons for 'GET UPDATES' and a 'Medium' link.

```
class TakeSubscriber<T> extends Subscriber<T> {  
  private count: number = 0;  
  
  constructor(destination: Subscriber<T>, private total: number) {  
    super(destination);  
  }  
  
  protected _next(value: T): void {  
    const total = this.total;  
    const count = ++this.count;  
    if (count <= total) {  
      this.destination.next(value);  
    }  
  }  
}
```

Never miss a story from **NetanelBasal**, when you sign up for Medium. [Learn more](#)

[GET UPDATES](#)

<https://netbasal.com/rxjs-six-operators-that-you-must-know-5ed3b6e238a0#.11of73aox>

# Creating Observables from scratch

## - André Staltz

André Staltz (@andrestaltz): You will learn RxJS at ng-europe 2016



```
1 function nextCallback(data) {
2   console.log(data);
3 }
4 function errorCallback(err) {
5 }
6 function completeCallback() {
7 }
8
9 function giveMeSomeData(nextCB, errCB, completeCB) {
10   document.addEventListener('click', (ev: UIEvent) => {
11     nextCB(ev);
12     giveMeSomeData(
13       nextCallback,
14       errorCallback,
15       completeCallback
16     );
17   });
18 }
```



<https://www.youtube.com/watch?v=uQ1zhJHclvs>

**GitHub Gist** Search... All gists GitHub New gist 

 **staltz / introrx.md** Last active an hour ago ★ Star 10,812 Fork 1203 ⓘ

Code Revisions 259 Stars 10812 Forks 1203 Embed <script src="https://gist.github.com/staltz/868e7e9bc2a7b8c1f754.introrx.md" data-bbox="700 250 720 270"/> Download ZIP

The introduction to Reactive Programming you've been missing

 [introrx.md](#) Raw

---

## The introduction to Reactive Programming you've been missing

(by [@andrestaltz](#))

---

### This tutorial as a series of videos

If you prefer to watch video tutorials with live-coding, then check out this series I recorded with the same contents as in this article: [Egghead.io - Introduction to Reactive Programming](#).

---

So you're curious in learning this new thing called Reactive Programming, particularly its variant comprising of Rx, Bacon.js, RAC, and others.

Learning it is hard, even harder by the lack of good material. When I started, I tried looking for tutorials. I found only a handful of practical guides, but they just scratched the surface and never tackled the challenge of building the whole architecture

<https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>

# RxMarbles

Fork me on GitHub

## RxMarbles

Interactive diagrams of Rx Observables

TRANSFORMING OPERATORS

- [delay](#)
- [delayWithSelector](#)
- [findIndex](#)
- [map](#)
- [scan](#)
- [debounce](#)
- [debounceWithSelector](#)

COMBINING OPERATORS

- [combineLatest](#)
- [concat](#)
- [merge](#)
- [sample](#)
- [startWith](#)
- [withLatestFrom](#)
- [zip](#)

FILTERING OPERATORS

- [distinct](#)
- [distinctUntilChanged](#)
- [elementAt](#)
- [filter](#)
- [find](#)
- [first](#)

merge

```
graph LR; A((20, 40, 60, 80, 100)) --> B((1, 1)); B --> C((20, 40, 60, 1, 80, 100, 1))
```

v1.4.1 built on RxJS v2.5.3 by [@andrestaltz](#)

<http://rxmarbles.com/>



THOUGHTRAM

Time

TRAINING

CODE REVIEW

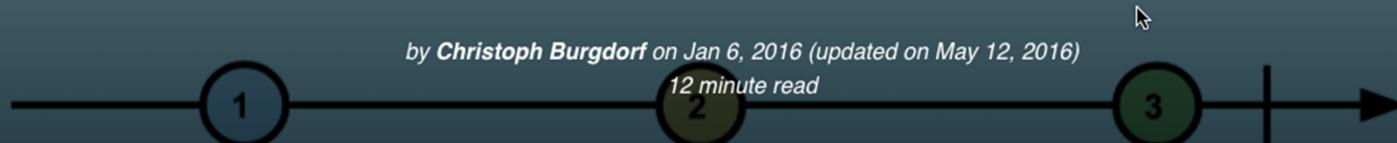
BLOG



# TAKING ADVANTAGE OF OBSERVABLES IN ~~distinctUntilChanged()~~ ANGULAR 2

by **Christoph Burgdorf** on Jan 6, 2016 (updated on May 12, 2016)

12 minute read



Some people seem to be confused why Angular 2 seems to favor the Observable abstraction over the Promise abstraction when it comes to dealing with async behavior.

There are pretty good resources about the difference between Observables and Promises already out there. I especially like to highlight this free [7 minutes video](#) by [Ben Lesh](#) on egghead.io. Technically there are a couple of obvious differences like the *disposability* and *laziness* of Observables. In this article we like to focus on some practical advantages that

<http://blog.thoughtram.io/angular/2016/01/06/taking-advantage-of-observables-in-angular2.html>