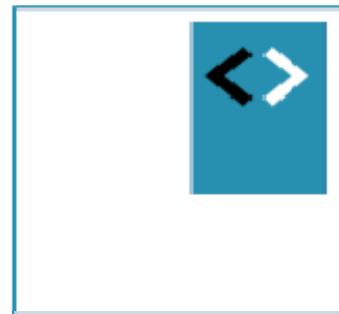




# Angular Fundamentals Module – Observables



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

## 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

<http://reactivex.io/>

### 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

# 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.*

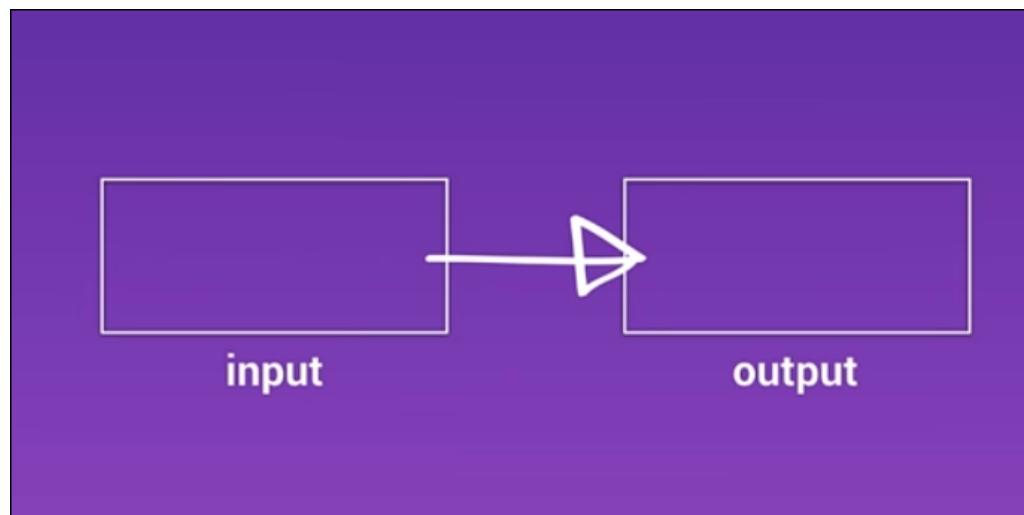
<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>

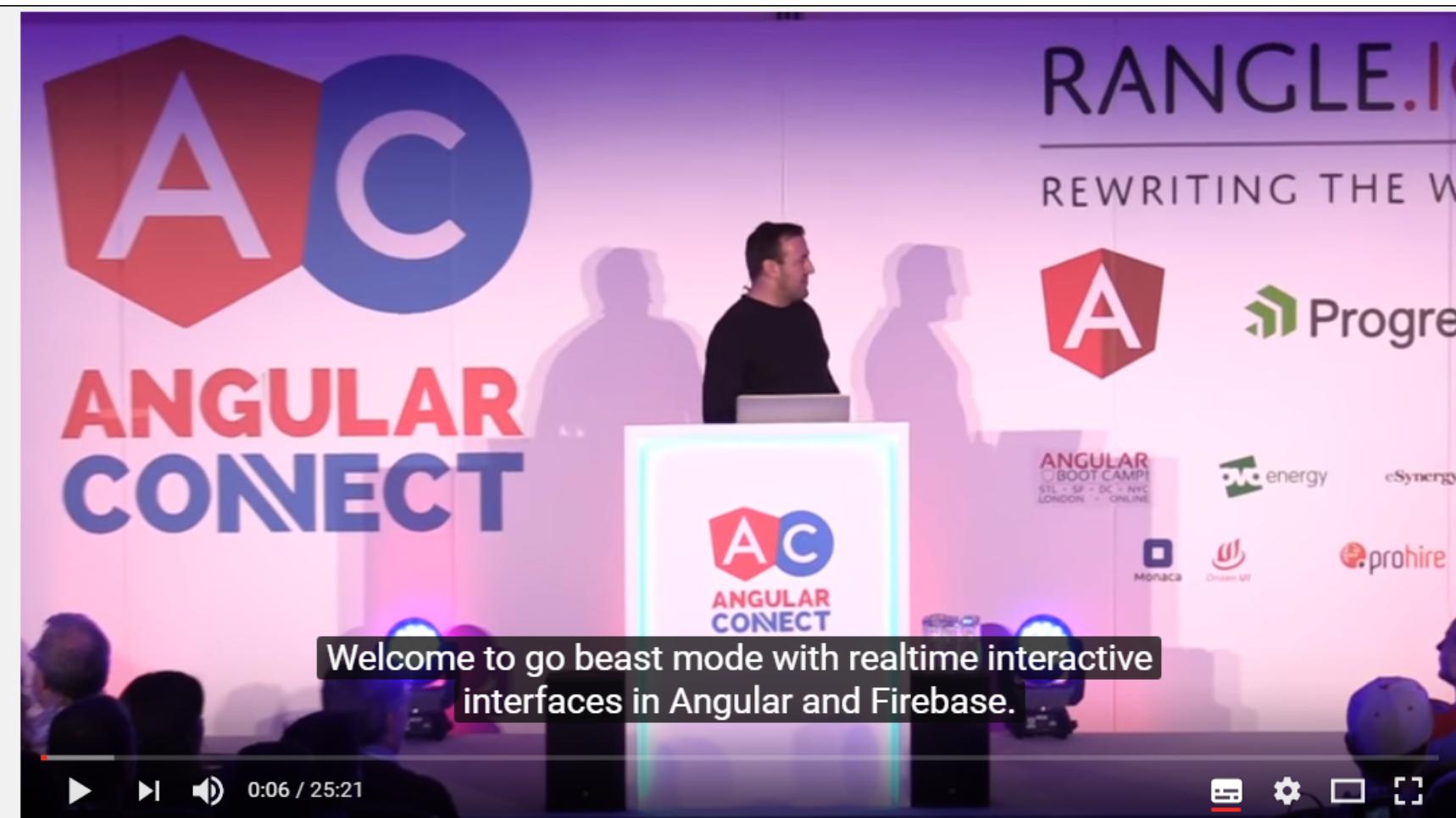
# Observables and RxJs

- “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:



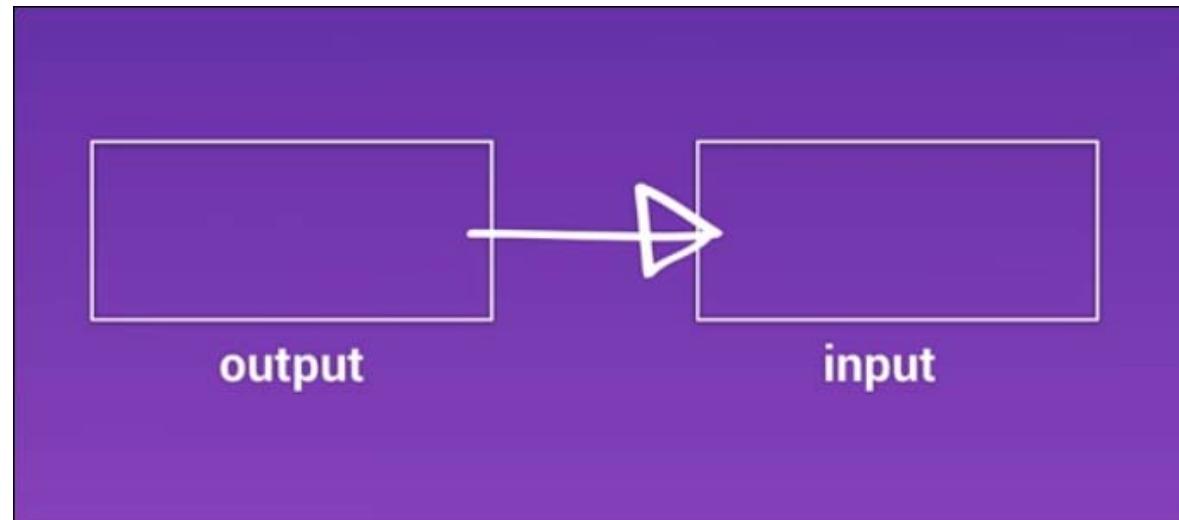


Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

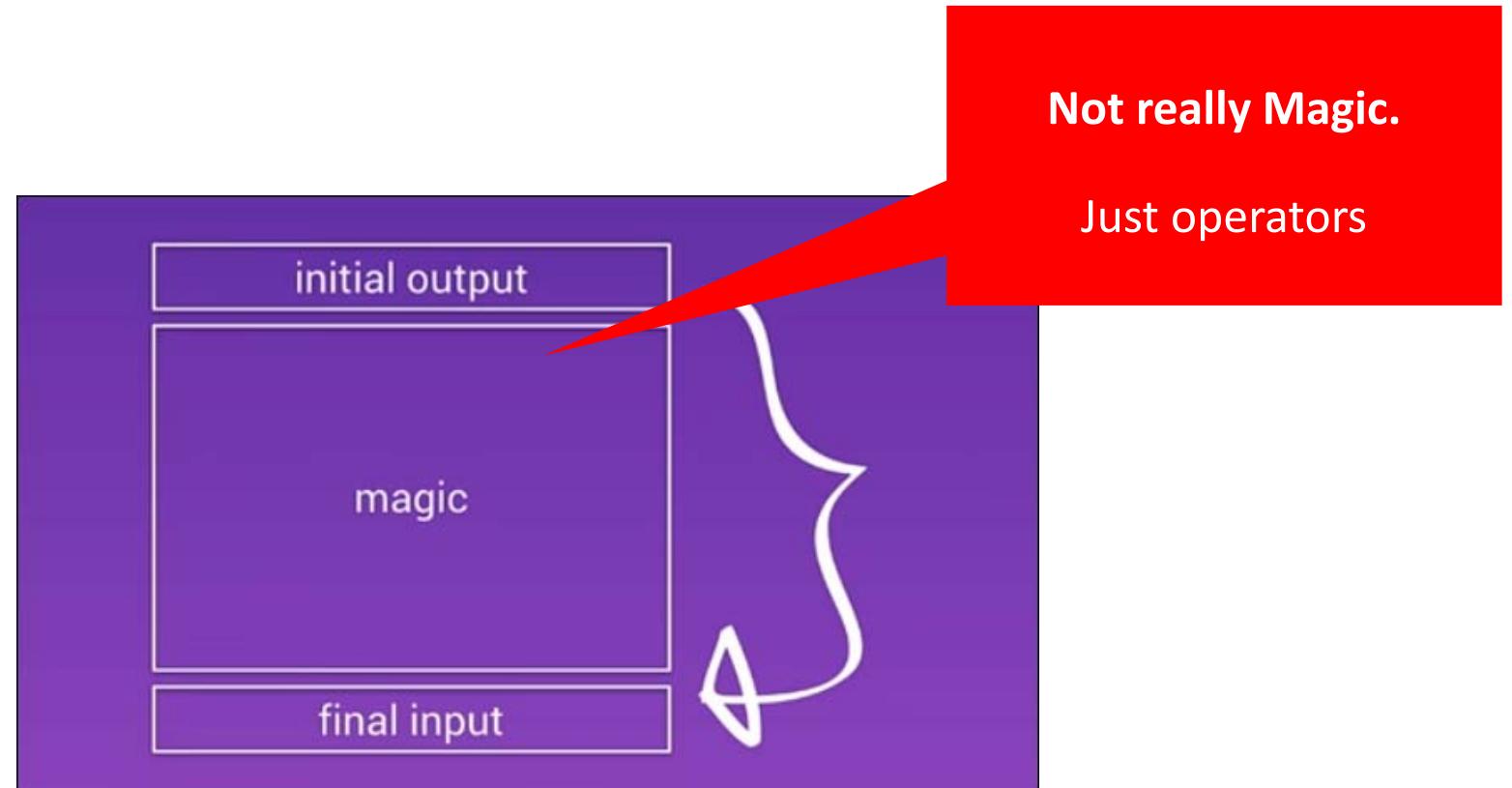
<https://www.youtube.com/watch?v=5CTL7aqSvJU>

<https://youtu.be/5CTL7aqSvJU?t=4m31s>

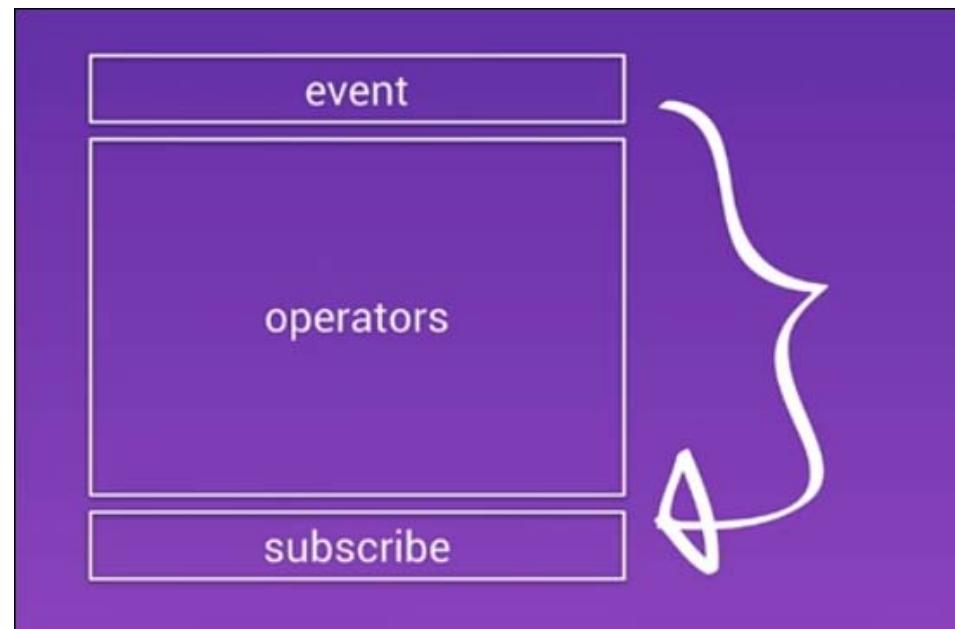
- With Observables -
  - a system, already outputting data,
  - Subscribe to that data
- "trade Output for Input"
- "Push vs. Pull"



# "The observable sandwich"



# Subscribe to events



In code:

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

Initial Output

Optioneel:  
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 {
}
```

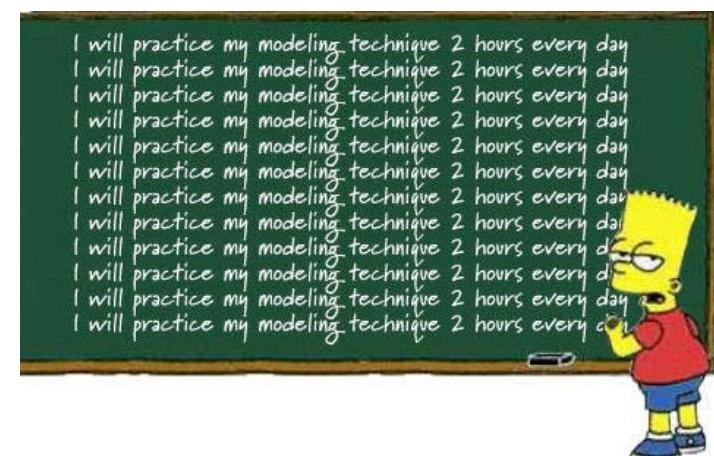
## OLD: Angular < 4.3: `HttpModule`

- In `@NgModule: imports : [HttpModule]`
- Using map-operator `.map(res => res.json())`.
  - Json has to be extracted by using `.map()`.
- `HttpModule` will be removed in future versions!
- Now we have Interceptors (in `HttpClientModule`)
- <https://alligator.io/angular/httpclient-intro/> and
- <https://alligator.io/angular/httpclient-interceptors/>

# Exercise

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

# Exercise....

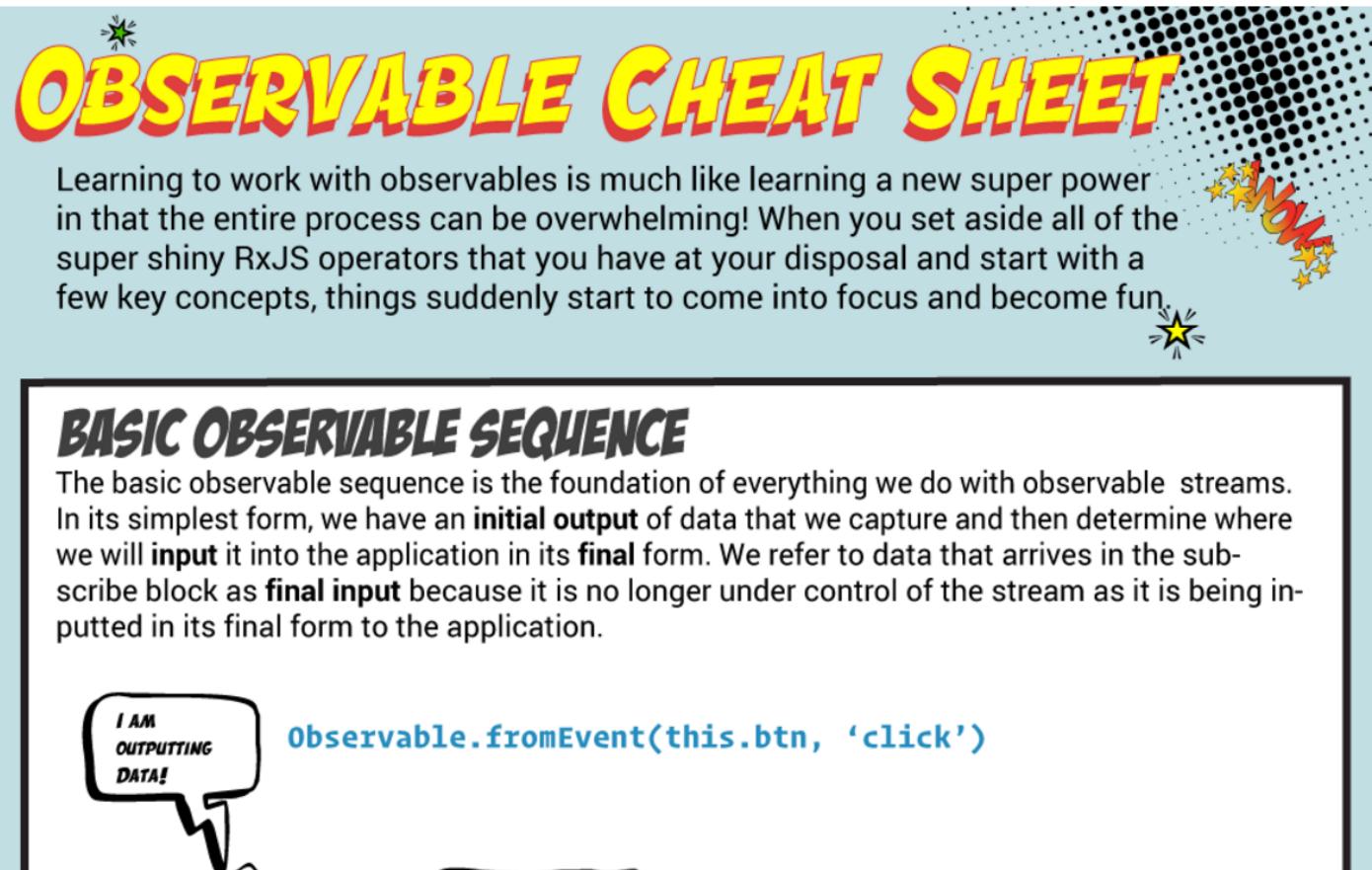


# Observable Cheat Sheet

genius to understand.

You can download the full-sized infographic at <http://bit.ly/observable-cheat-sheet>.

I really hope that you find the infographic helpful. Be sure to drop me a line below if you have any questions or comments. #highFive



The image is a screenshot of the Observable Cheat Sheet infographic. At the top, it features a large, bold title "OBSERVABLE CHEAT SHEET" in yellow with a red shadow effect. Below the title is a text block that reads: "Learning to work with observables is much like learning a new super power in that the entire process can be overwhelming! When you set aside all of the super shiny RxJS operators that you have at your disposal and start with a few key concepts, things suddenly start to come into focus and become fun." To the right of this text is a decorative graphic of several yellow stars and sparkles. Below this section is a box with a black border containing the heading "BASIC OBSERVABLE SEQUENCE" in bold. The text inside the box explains: "The basic observable sequence is the foundation of everything we do with observable streams. In its simplest form, we have an **initial output** of data that we capture and then determine where we will **input** it into the application in its **final** form. We refer to data that arrives in the subscribe block as **final input** because it is no longer under control of the stream as it is being inputted in its final form to the application." To the left of the text in the box is a speech bubble icon with the text "I AM OUTPUTTING DATA!" inside. To the right of the text is a line of code: `Observable.fromEvent(this.btn, 'click')`.

<http://onehungrymind.com/observable-cheat-sheet/>

# Hello RxJS

Free online training

The screenshot shows a web-based learning platform interface for a 'Hello RxJS' micro course. On the left, there's a sidebar with a course box icon, a progress bar showing '5% COMPLETE', and links for 'Class Curriculum' and 'Your Instructor'. The main content area has a dark header with a back arrow and a user profile icon. Below the header, the title 'Class Curriculum' is displayed. A purple button labeled 'Start next lecture >' is followed by the text 'Presentation: Realtime Observable Streams'. The main content area lists 14 items under the heading 'Hello RxJS', each with a small icon, a title, and a 'Start' button. The items are:

- Presentation: Realtime Observable Streams
- Slides: Realtime Observable Streams
- The Basic Observable Sequence (2:09)
- Lab: The Basic Observable Sequence
- Mapping Values (2:22)
- Lab: Mapping Values
- Maintaining State (3:25) (marked with a checkmark)
- Lab: Maintaining State
- Merging Streams (1:57)
- Lab: Merging Streams
- Mapping to Functions (5:18)
- Lab: Mapping to Functions

<http://courses.ultimateangular.com/>

# 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...)  
)
```

# Subscribe - only once per block!

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

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



# RxJS-operators in the service

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {map, delay, takeUntil, ...} from "rxjs/operators";

@Injectable()
export class CityService {

    constructor(private http: HttpClient) {

    }

    // retourner alle cities
    getCities(): Observable<Response> {
        return this.http.get('shared/data/cities.json')
            .pipe(...);
    }
}
```

The result of the first operator is the input of the next operator in the pipeline

```
getCities() {  
    if (!this.cities) {  
        this.cityService.getCities()  
            .pipe(  
                delay(3000),  
                retry(3)  
                map(result => ...),  
                takeUntil(...condition...)  
            )  
            .subscribe(cityData => {  
                this.cities = cityData;  
            })  
    }  
}
```



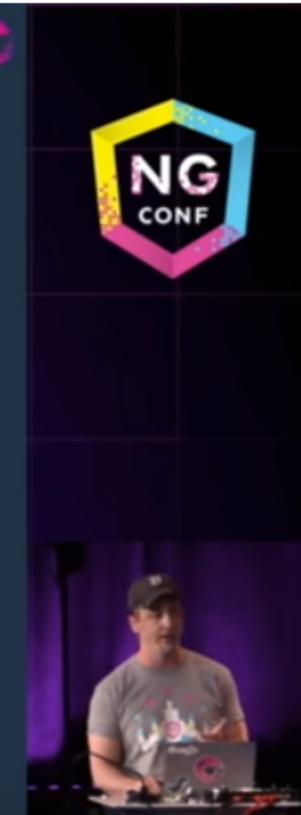
Operators in .pipe()

# 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

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

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

The screenshot shows the homepage of the Learn RxJS website. The page has a sidebar on the left containing navigation links for 'learn-rxjs', 'LEARN RXJS' (with 'Introduction' selected), 'Operators' (with 'Combination' expanded, showing 'combineAll', 'combineLatest', 'concat', 'concatAll', 'forkJoin', 'merge', 'mergeAll', 'pairwise', 'race', 'startWith', 'withLatestFrom', and 'zip'), and 'Conditional'. The main content area features a large title 'Learn RxJS' and a subtitle 'Clear examples, explanations, and resources for RxJS.' Below this is a section titled 'Introduction' with a paragraph about RxJS being one of the hottest libraries in web development. There is also a 'But...' section with text about learning RxJS being hard due to its complexity and shift in mindset. A 'Content' section is visible at the bottom.

Type to search

☰ EDIT THIS PAGE A

Star 733 Watch 54

learn-rxjs

LEARN RXJS

Introduction

Operators

Combination

- combineAll
- combineLatest
- concat
- concatAll
- forkJoin
- merge
- mergeAll
- pairwise
- race
- startWith
- withLatestFrom
- zip

Conditional

# Learn RxJS

Clear examples, explanations, and resources for RxJS.

## Introduction

RxJS is one of the hottest libraries in web development today. Offering a powerful, functional approach for dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to utilize your knowledge across [nearly any language](#), having a solid grasp on reactive programming and what it can offer seems like a no-brainer.

**But...**

Learning RxJS and reactive programming is [hard](#). There's the multitude of concepts, large API surface, and fundamental shift in mindset from an [imperative to declarative style](#). This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the [official docs](#) and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and tackle the pain points. Learning Rx may be difficult but it is certainly worth the effort!

## Content



# Using the `async` pipe

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

# Async Pipe

- On `.subscribe()`, you actually need to
  - `.unsubscribe()`
  - Best practice
  - Not \*really\* necessary on HTTP-requests, but you do in other subscriptions well, to avoid memory leaks.
- 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">
```

# Working with Live API's

- MovieApp
- examples\210-services-live



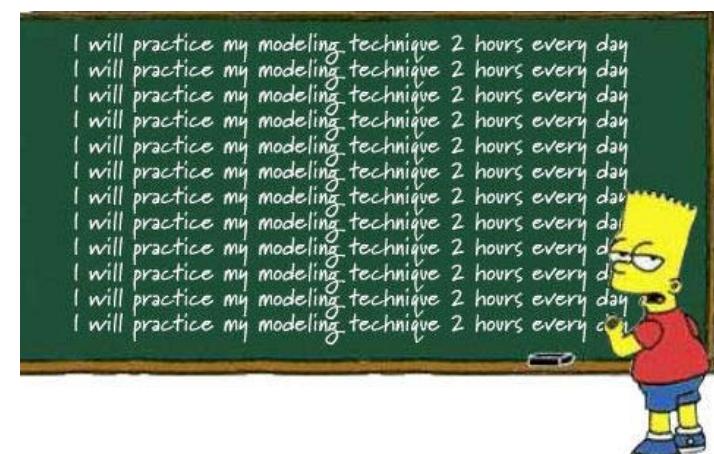
# Example API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weather forecast)
- <http://randomuser.me/> (random user data)
- <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
  - ( 502-forms-typeahead (We'll cover forms later on))
- 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

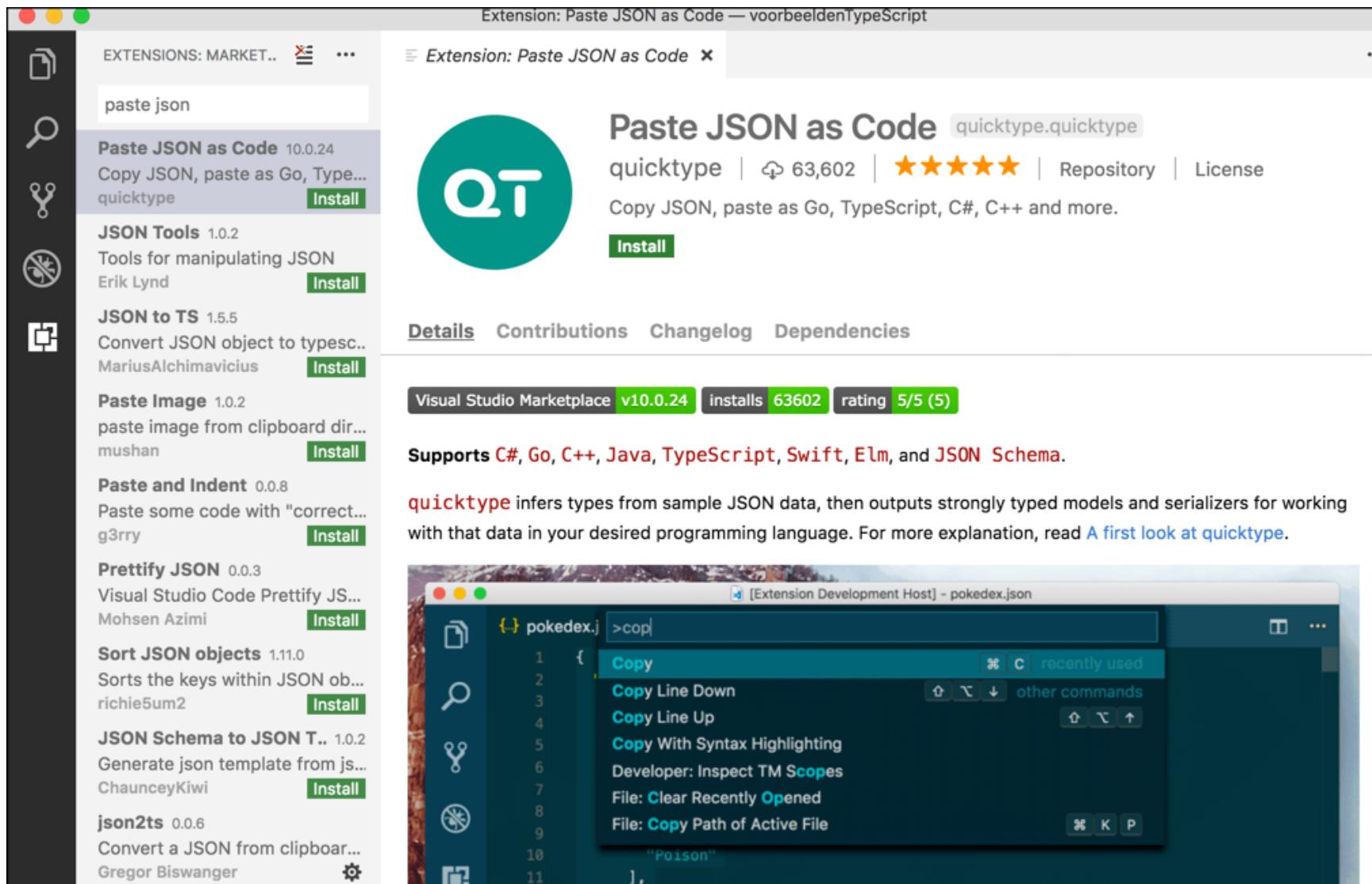
# Online JSON to TypeScript converter

The screenshot shows the json2ts website interface. At the top, the title "json2ts" is displayed in large white letters on a blue header bar, 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 array 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 web application interface. At the top, there is 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 generating up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats. It also mentions a \$50/year plan.

The main area contains a table for defining data 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 at the bottom of this section.

Below the table are settings for the generated data:

- # Rows: 1000
- Format: CSV
- Line Ending: Unix (LF)
- Include:  header  BOM

At the 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



[Sign in / Sign up](#)

[Netanel Basal](#) [Follow](#)  
Jan 24 · 3 min read

## RxJS—Six Operators That you Must Know

```
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);  
    }  
  }  
}  
class TakeUntilSubscriber<T> extends Subscriber<T> {  
  private source: Observable<T>;  
  private until: Observable<any>;  
  private subscription: Subscription;  
  
  constructor(source: Observable<T>, until: Observable<any>) {  
    super();  
    this.source = source;  
    this.until = until;  
  }  
  
  protected _next(value: T): void {  
    this.subscription = this.until.subscribe(() => {  
      this.source.unsubscribe();  
    });  
  }  
}
```

 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>

# Documentation at `reactivex.io`

The screenshot shows a web browser displaying the `reactivex.io` documentation. The top navigation bar includes links for `ReactiveX`, `Introduction`, `Docs`, `Languages`, `Resources`, and `Community`. Below the navigation, a breadcrumb trail reads "Operators By Category > Creating Observables". The main content title is "Operators By Category". Under the heading "Creating Observables", it says "Operators that originate new Observables." followed by a bulleted list of operators: `Create`, `Defer`, `Empty / Never / Throw`, `From`, `Interval`, `Just`, `Range`, `Repeat`, `Start`, and `Timer`. Further down, under the heading "Transforming Observables", it says "Operators that transform items that are emitted by an Observable." followed by a bulleted list of operators: `Buffer`, `FlatMap`, and `GroupBy`.

Operators that originate new Observables.

- `Create` — create an Observable from scratch by calling observer methods programmatically
- `Defer` — do not create the Observable until the observer subscribes, and create a fresh Observable for each observer
- `Empty / Never / Throw` — create Observables that have very precise and limited behavior
- `From` — convert some other object or data structure into an Observable
- `Interval` — create an Observable that emits a sequence of integers spaced by a particular time interval
- `Just` — convert an object or a set of objects into an Observable that emits that or those objects
- `Range` — create an Observable that emits a range of sequential integers
- `Repeat` — create an Observable that emits a particular item or sequence of items repeatedly
- `Start` — create an Observable that emits the return value of a function
- `Timer` — create an Observable that emits a single item after a given delay

Operators that transform items that are emitted by an Observable.

- `Buffer` — periodically gather items from an Observable into bundles and emit these bundles rather than emitting the items one at a time
- `FlatMap` — transform the items emitted by an Observable into Observables, then flatten the emissions from those into a single Observable
- `GroupBy` — divide an Observable into a set of Observables that each emit a different group of items from the original Observable, organized by key

<http://reactivex.io/documentation/operators.html>

# Creating Observables from scratch

## - André Staltz

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

```
function nextCallback(data) {
  console.log(data);
}

function errorCallback(err) {
}

function completeCallback() {
}

function giveMeSomeData(nextCB, errCB) {
  document.addEventListener('click', () => {
    nextCB();
    errCB();
  });
}

giveMeSomeData(
  nextCallback,
  errorCallback,
  completeCallback
);
```

The video player interface includes controls for play/pause, volume, and progress bar, along with a timestamp of 5:11 / 22:44.

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

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

 **staltz / introrx.md** Last active an hour ago

**Code** Revisions 259 Stars 10812 Forks 1203 Embed <script src="https://gist.github.com/staltz/868e7e9bc2a7b8c1f754">

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(( )) --> B(( )); A --> C(( )); A --> D(( )); A --> E(( )); A --> F(( )); B --> G(( )); B --> H(( )); G --> I(( )); H --> I;
```

v1.4.1 built on RxJS v2.5.3 by @andrestaltz

<http://rxmarbles.com/>

# Dan Wahlin on Modules and Observables

Integrating Angular with RESTful Services using RxJS and Observables

```
15  baseUrl: string = '/api/customers';
16
17  constructor(private http: Http) {
18
19  }
20
21  getCustomers(): Observable<ICustomer[]> {
22    return this.http.get(this.baseUrl)
23      .map((res: Response) => {
24        let customers = res.json();
25        this.calculateCustomersOrderTotal(customers);
26        return customers;
27      })
28      .catch(this.handleError);
29  }
30
31  getCustomersPage(page: number, pageSize: number): Observable<IPagedResults<ICustomer[]> {
32    return this.http.get(`${this.baseUrl}/page/${page}/${pageSize}`)
33      .map((res: Response) => {
34        const totalRecords = +res.headers.get('x-inlinelcount');
35        let customers = res.json();
36        this.calculateCustomersOrderTotal(customers);
37        return {
38          totalRecords,
39          customers
40        };
41      })
42      .catch(this.handleError);
43  }
44
45  calculateCustomersOrderTotal(customers: ICustomer[]): void {
46    // ...
47  }
48
49  handleError(error: any): Observable<never> {
50    // ...
51  }
52
53  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
54    // ...
55  }
56
57  private handleError(error: any): Observable<never> {
58    // ...
59  }
60
61  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
62    // ...
63  }
64
65  private handleError(error: any): Observable<never> {
66    // ...
67  }
68
69  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
70    // ...
71  }
72
73  private handleError(error: any): Observable<never> {
74    // ...
75  }
76
77  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
78    // ...
79  }
80
81  private handleError(error: any): Observable<never> {
82    // ...
83  }
84
85  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
86    // ...
87  }
88
89  private handleError(error: any): Observable<never> {
90    // ...
91  }
92
93  private calculateCustomersOrderTotal(customers: ICustomer[]): void {
94    // ...
95  }
96
97  private handleError(error: any): Observable<never> {
98    // ...
99  }
100
101 private calculateCustomersOrderTotal(customers: ICustomer[]): void {
102   // ...
103 }
104
105 private handleError(error: any): Observable<never> {
106   // ...
107 }
```

<https://www.youtube.com/watch?v=YxK4UW4UfCk>



THOUGHTRAM

Time

TRAINING

CODE REVIEW

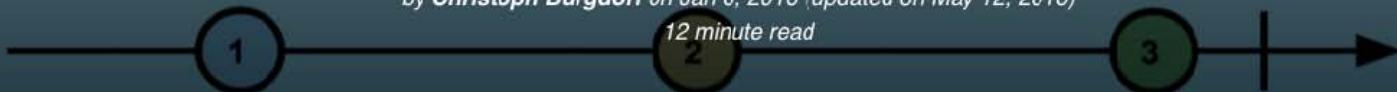
BLOG



# 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 *lazyness* 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>