

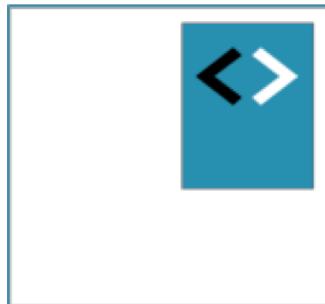


# Angular Fundamentals

## Module 4 – Observables



A CANON COMPANY



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

The image shows a screenshot of the ReactiveX website. On the left, the homepage features a dark background with a blurred image of a road at night. The ReactiveX logo (a stylized orange 'R') is at the top, followed by navigation links: Introduction, Docs ▾, Languages ▾, Resources ▾, and Community ▾. A large 'ReactiveX' title with a subtitle 'An API for asynchronous programming with observable streams' is centered. A pink button labeled 'Choose your platform' is visible. At the bottom, the URL <http://reactivex.io/> is displayed. On the right, a separate window shows the 'Languages' page with the same navigation bar. The main content area is titled 'Languages' and lists various platforms: 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, and Swift: RxSwift.

<b>DOCUMENTATION</b>	<b>LANGUAGES</b>	<b>RESOURCES</b>	<b>COMMUNITY</b>
<a href="#">Observable</a>	<a href="#">RxJava<sup>♂</sup></a>	<a href="#">Tutorials</a>	<a href="#">GitHub<sup>♂</sup></a>
<a href="#">Operators</a>	<a href="#">RxJS<sup>♂</sup></a>		<a href="#">Twitter<sup>♂</sup></a>
<a href="#">Single</a>	<a href="#">Rx.NET<sup>♂</sup></a>		<a href="#">Others</a>
<a href="#">Cancellable</a>	<a href="#">Rx.Cocoa</a>		

# 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.*

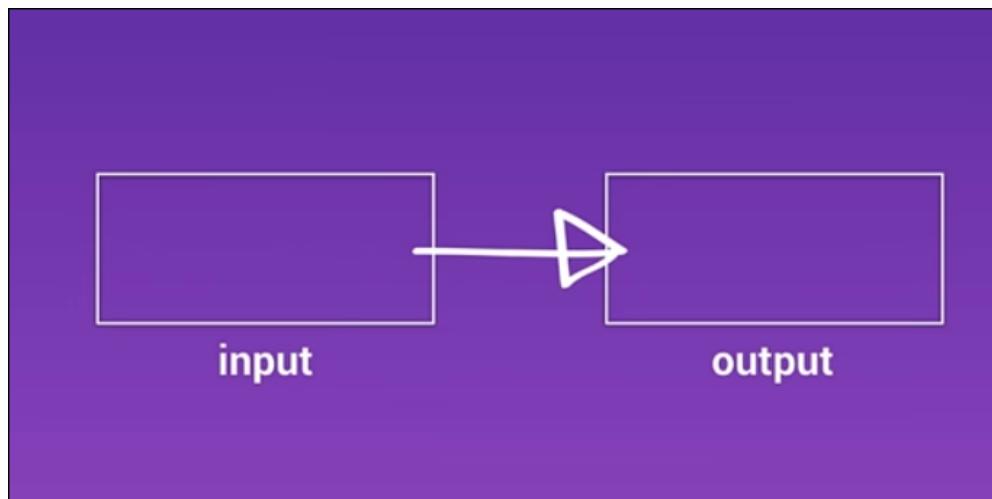
<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>

# 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:

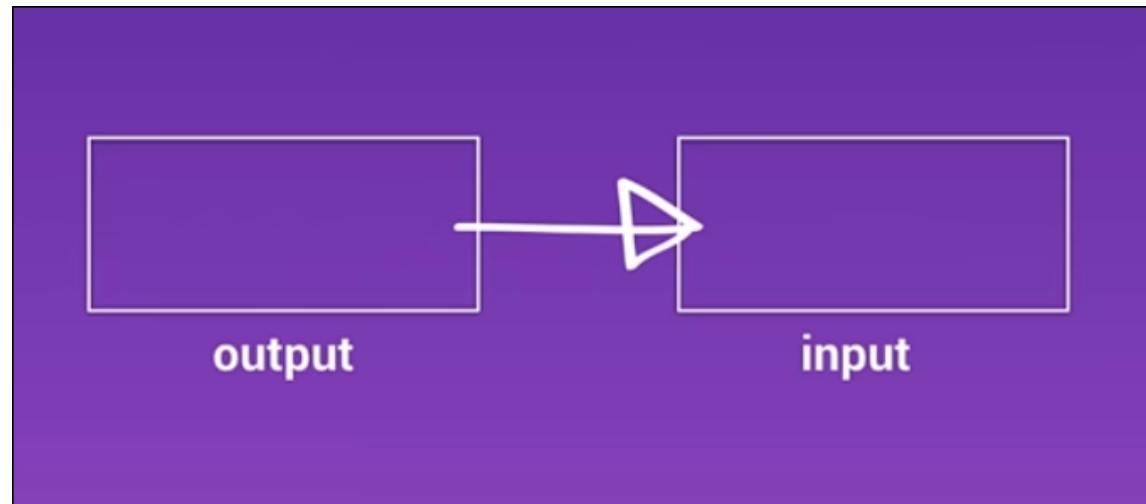




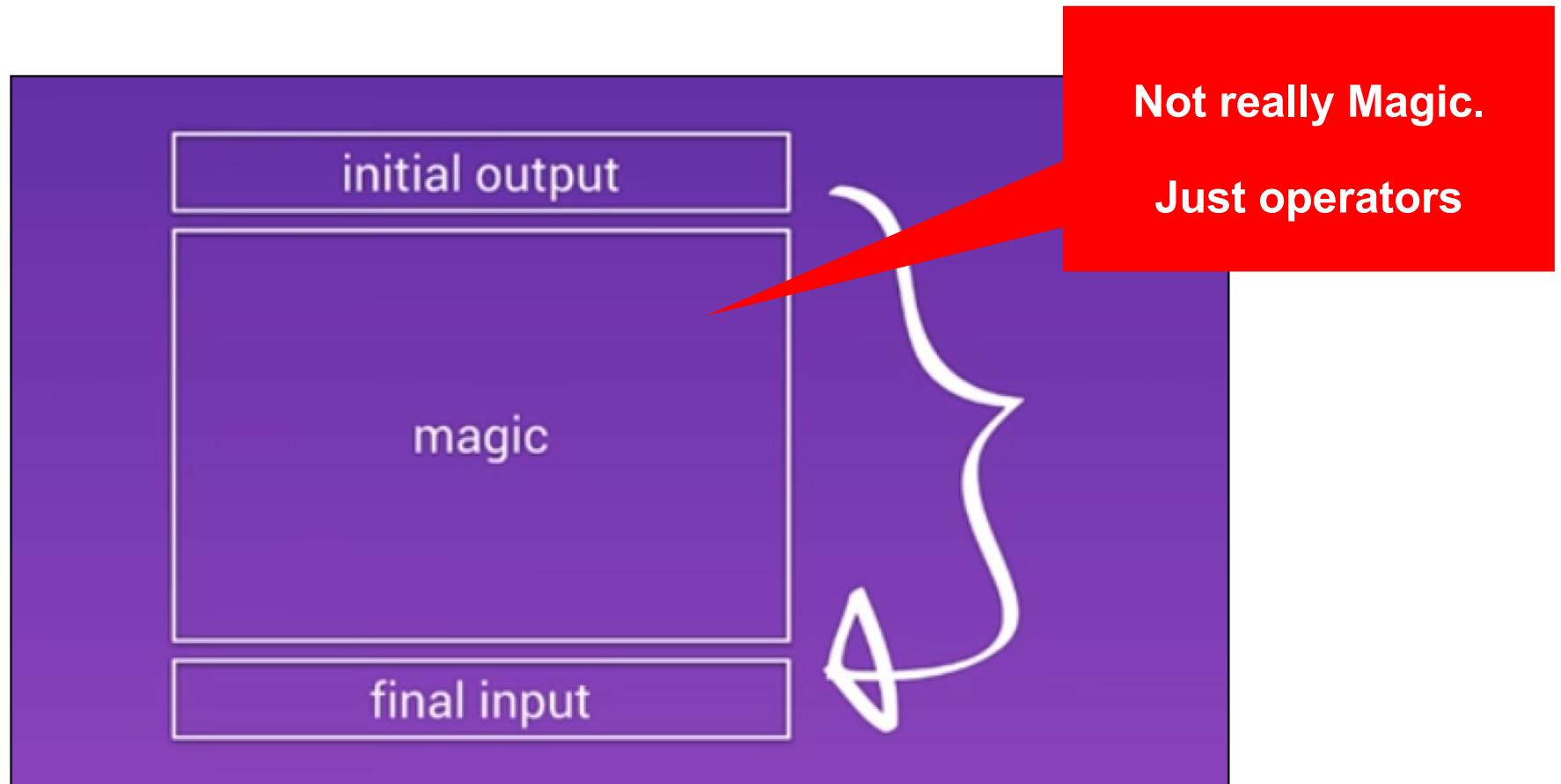
Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

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

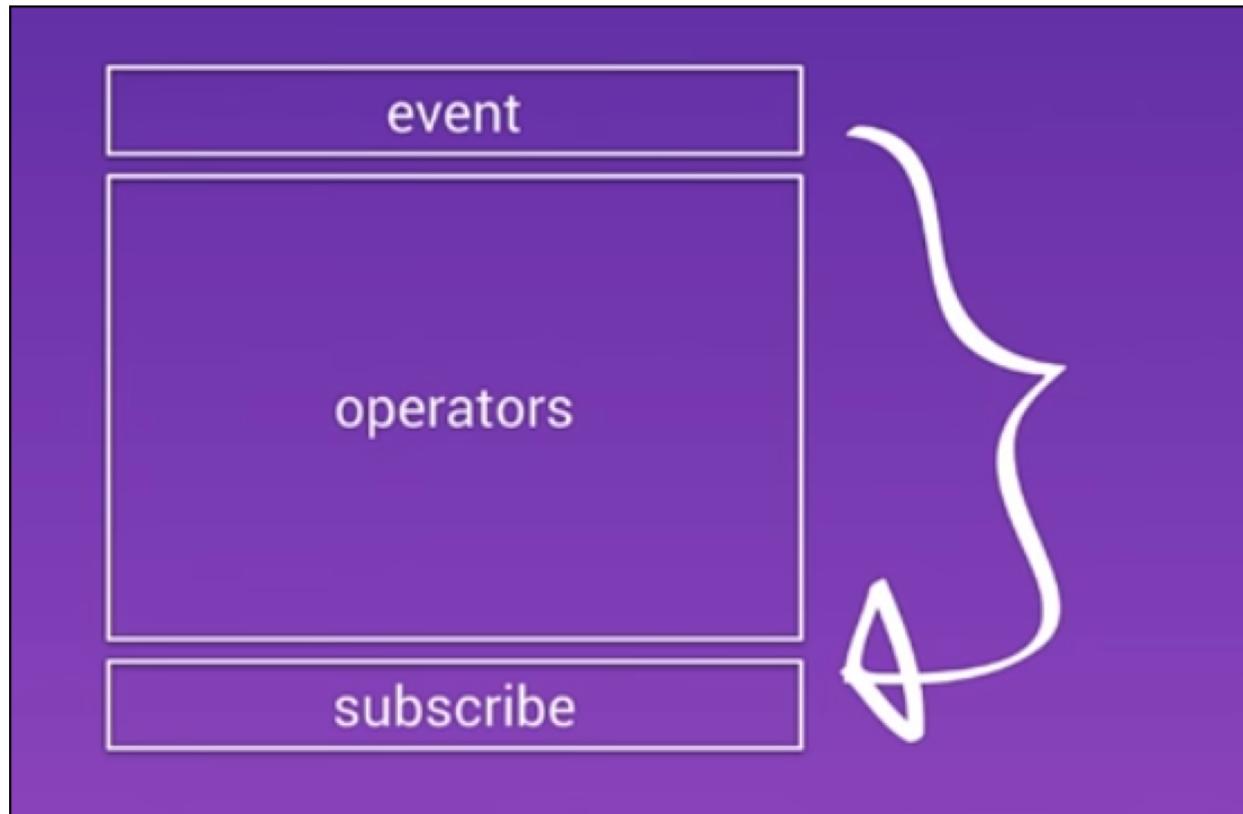
- With Observables -
  - a system, already outputting data,
  - Subscribe to that data
- "trade Output for Input"
- "Push vs. Pull"



# "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 {  
}

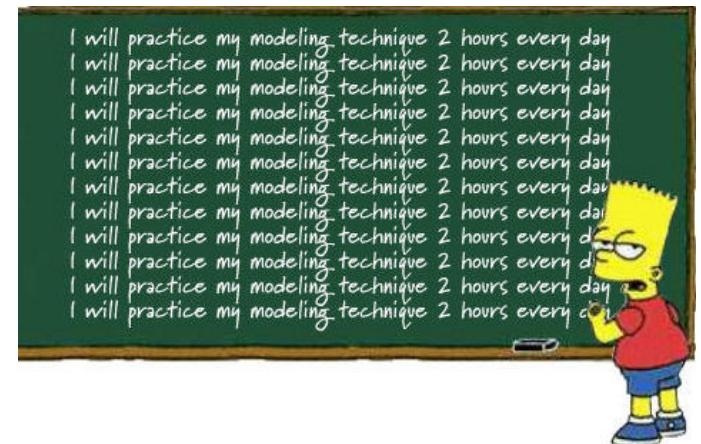
# **OLD/Deprecated: Angular < 4.3: HttpModule**

- 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 also 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....

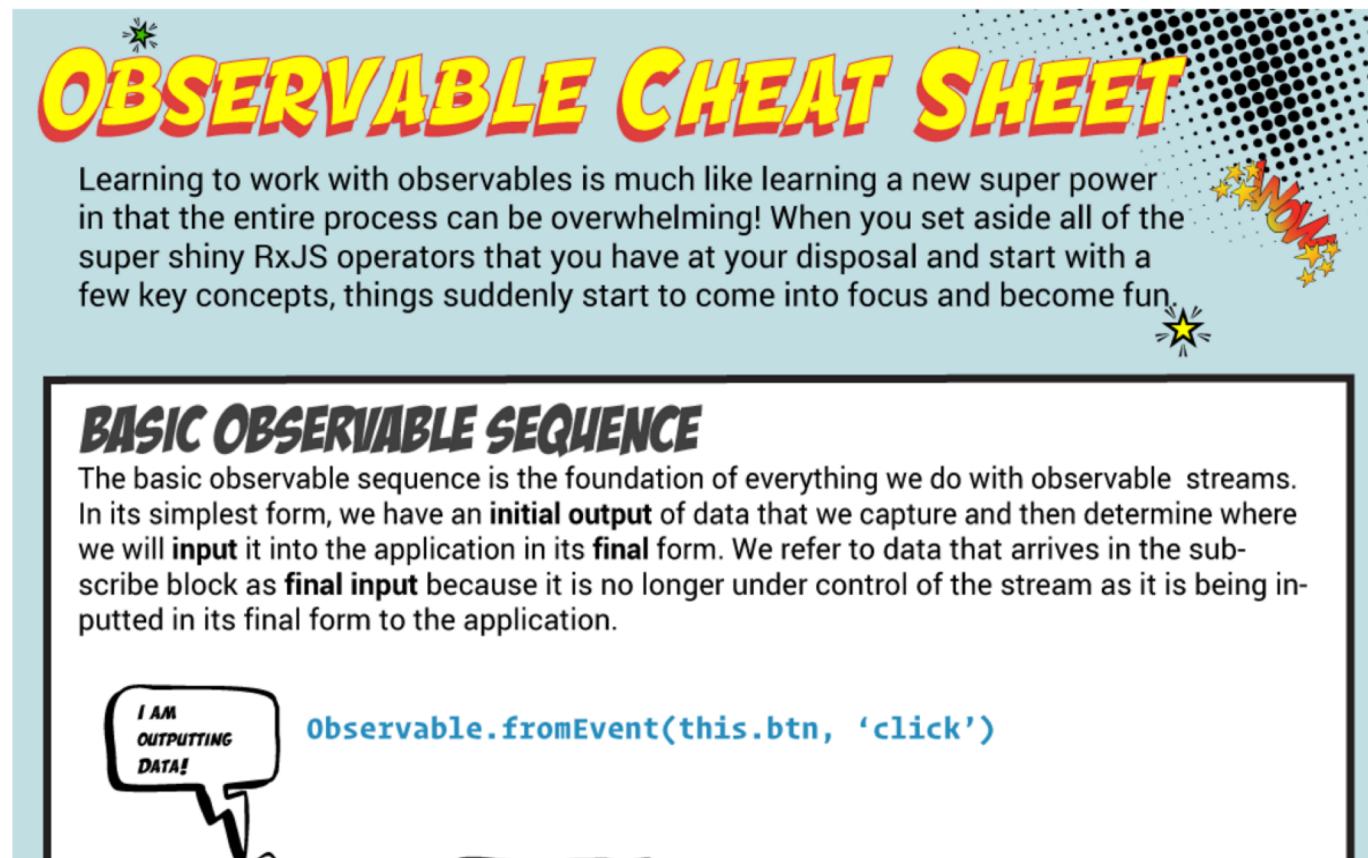


# Observable Cheat Sheet

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



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

# Hello RxJS

Free online training

The screenshot shows the 'Hello RxJS' course page from the Ultimate Angular platform. On the left, there's a sidebar with the course title 'Hello RxJS' and a progress bar indicating '5% COMPLETE'. Below that are links for 'Class Curriculum' and 'Your Instructor'. The main content area is titled 'Class Curriculum' and features a purple button 'Start next lecture >' followed by the text 'Presentation: Realtime Observable Streams'. Below this, a list of lessons is displayed in a grid format:

Lesson Title	Action
Presentation: Realtime Observable Streams	Start
Slides: Realtime Observable Streams	Start
The Basic Observable Sequence (2:09)	Start
Lab: The Basic Observable Sequence	Start
Mapping Values (2:22)	Start
Lab: Mapping Values	Start
Maintaining State (3:25)	Start
Lab: Maintaining State	Start
Merging Streams (1:57)	Start
Lab: Merging Streams	Start
Mapping to Functions (5:18)	Start
Lab: Mapping to Functions	Start

<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
  },
  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) {

    }

    // retourneer 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;  
            })  
    }  
}
```



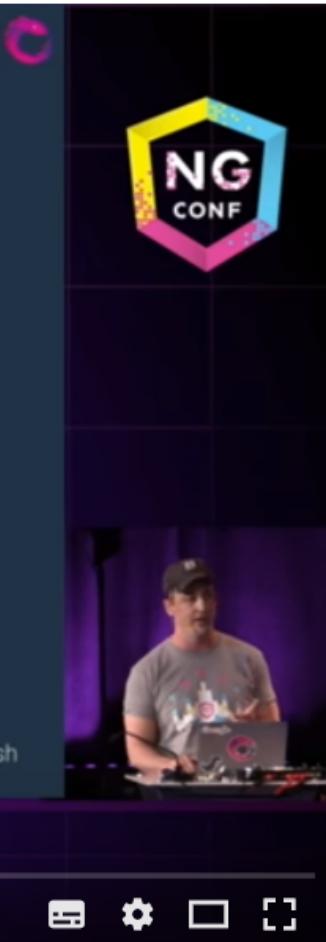
Operators in .pipe()

# 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 header with a search bar, edit options, and social sharing links. The main title is "Learn RxJS" with a subtitle "Clear examples, explanations, and resources for RxJS". On the left, there's a sidebar with navigation links for "learn-rxjs", "LEARN RXJS", "Introduction", "Operators", "Combination", "Conditional", and "Content". The "Combination" section lists various operators: combineAll, combineLatest, concat, concatAll, forkJoin, merge, mergeAll, pairwise, race, startWith, withLatestFrom, and zip. The "Content" section contains two main paragraphs. The first paragraph discusses RxJS as a powerful, functional approach for dealing with events and integration points, mentioning its appeal across nearly any language. The second paragraph, under the heading "But...", explains that learning RxJS and reactive programming is hard due to the large API surface and fundamental shift in mindset, but promises to make it approachable through clear examples and references to existing material.

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">
```

<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



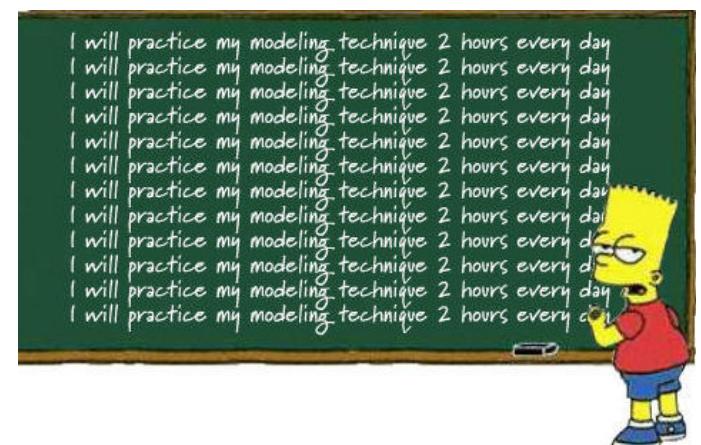
# Example API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weather forecast)
- <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

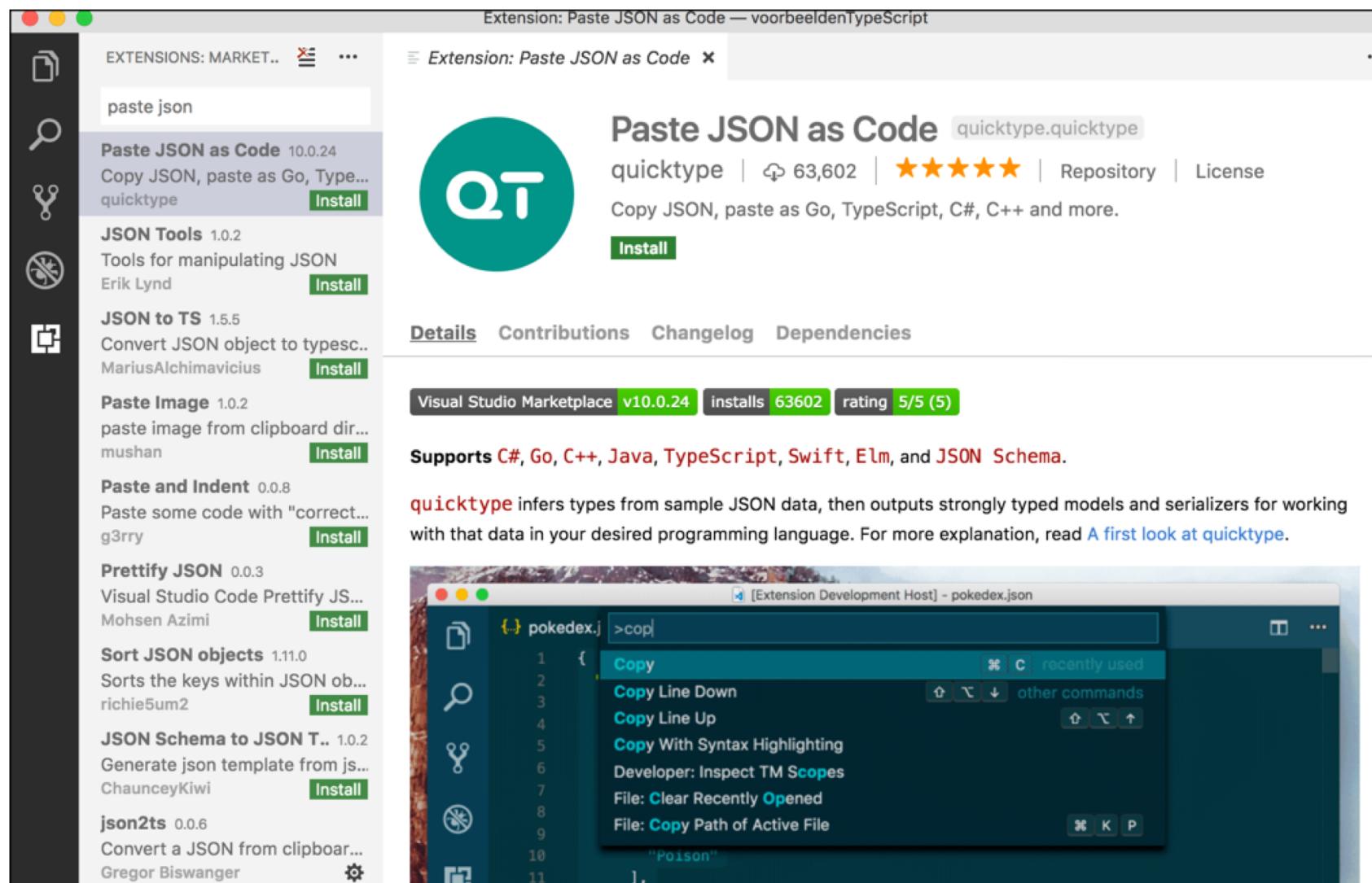
The screenshot shows the json2ts website interface. At the top, there's a blue header with the title "json2ts" in large white font, followed by "generate TypeScript interfaces from JSON" in smaller white font. On the right side of the header are links for "email", "feedback", and "help". Below the header is a large text area containing a JSON object representing movie data. At the bottom of this area is a green button labeled "generate TypeScript". Below the button is a code editor window displaying the generated TypeScript interface code.

```
300.jpg"}, {"Title": "The Amazing Captain Nemo", "Year": "1978", "imdbID": "tt0077156", "Type": "movie", "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTc4NzExNjcwN15BMi5BanBnXkFtZTYwMTM1Mjg5._V1_SX300.jpg"}, {"Title": "Nemo", "Year": "1984", "imdbID": "tt0087784", "Type": "movie", "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTY2NzlwMTgwN15BMi5BanBnXkFtZTcwMjlyMzMzMQ@._V1_SX300.jpg"}, {"Title": "Captain Nemo", "Year": "1975", "imdbID": "tt0453375", "Type": "movie", "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BM2JmOTRIMGQtODMxNy00YmRkLWI1OWEtMmQ2YjZiZmQzZGU5XkEyXkFqcGdeQXVyNDUxNjc5NjY@._V1_SX300.jpg"}, {"Title": "Finding Nemo", "Year": "2003", "imdbID": "tt0401422", "Type": "game", "Poster": "N/A"}, {"Title": "Making Nemo", "Year": "2003", "imdbID": "tt0387373", "Type": "movie", "Poster": "N/A"}, {"Title": "Finding Nemo Submarine Voyage", "Year": "2007", "imdbID": "tt1319713", "Type": "movie", "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMzAxMzMyODQtNWY0Yy00N2M3LWE5MDQtZDUzNjc1ZGFmMzA4XkEyXkFqcGdeQXVyMzkxMzc4Mw@._V1_SX300.jpg"}, {"Title": "Little Nemo: The Dream Master", "Year": "1990", "imdbID": "tt0206895", "Type": "game", "Poster": "N/A"}], "totalResults": "31", "Response": "True"}
```

```
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's a green header bar with the Mockaroo logo, a search icon, a 'PRICING' link, and a 'SIGN IN' link. 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 pricing 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 of the page, there are settings for the data generation: '# Rows' set to 1000, 'Format' set to CSV, 'Line Ending' set to Unix (LF), and checkboxes for 'Include: header' (checked) and 'BOM'. There are also 'Download Data' (green button), 'Preview', and 'More' buttons, along with a note about saving the data for later.

<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, name 'Netanel Basal', a 'Follow' button, and a timestamp 'Jan 24 · 3 min read'. The main title 'RxJS—Six Operators That you Must Know' is displayed prominently. A large black rectangular box contains a portion of the article's code, specifically the implementation of the `TakeSubscriber` class. At the bottom left is the author's profile picture again, with the text 'Never miss a story from **NetanelBasal**, when you sign up for Medium. Learn more'. To the right is a blue 'GET UPDATES' button.

```
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);  
    }  
  }  
}  
export { TakeSubscriber };
```

<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

The screenshot shows a video player interface. On the left, there is a code editor window displaying a JavaScript file with the following code:

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

The status bar at the bottom of the code editor shows: master, 0 ▲ 0, — INSERT MODE —, Line 10, Col 36, Spaces: 2, UTF-8, LF, JavaScript.

On the right, there is a video feed of André Staltz speaking at a podium. The podium has a laptop displaying the Angular logo and the URL ngeurope.org. The video player has a progress bar at 5:11 / 22:44 and various control icons (play, pause, volume, etc.).

<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.  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/>

Time



TRAINING

CODE REVIEW

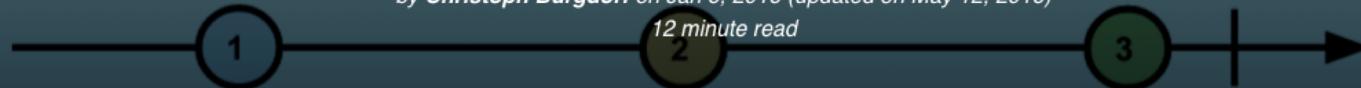
BLOG



# 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>