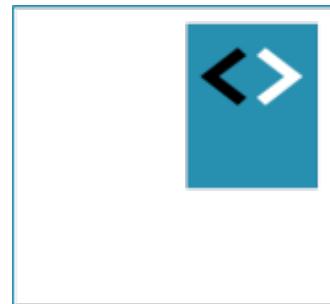




Angular Fundamentals Module – Observables



Peter Kassenaar –
info@kassenaar.com



Async services met RxJS/Observables

Reactive programming with asynchronous streams

Async Services

- Statische data ophalen: *synchrone* actie
- Werken via `Http`: *asynchrone* actie
- Werken via `HttpClient`: *Angular 4.3+*
- Angular 1: Promises
- Angular 2: Observables

Bovendien in Angular 2: ReactiveX library RxJS

The image shows the ReactiveX website homepage. At the top, there's a dark navigation bar with the ReactiveX logo and links for Introduction, Docs, Languages, Resources, and Community. Below the header is a large banner featuring the ReactiveX logo and the text "An API for asynchronous computation with observable streams". A pink button labeled "Choose your platform" is visible. To the right of the banner, the main content area has a dark background with a blurred image of a road at night. The main heading "Languages" is followed by a list of supported platforms. Below this, another section titled "ReactiveX for platforms and frameworks" lists several additional technologies.

<http://reactivex.io/>

ReactiveX Introduction Docs Languages Resources Community

ReactiveX

An API for asynchronous computation 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

DOCUMENTATION

- Observable
- Operators
- Single
- Combining

LANGUAGES

- RxJava[♂]
- RxJS[♂]
- Rx.NET[♂]
- RxScala

RESOURCES

- Tutorials

COMMUNITY

- GitHub[♂]
- Twitter[♂]
- 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.

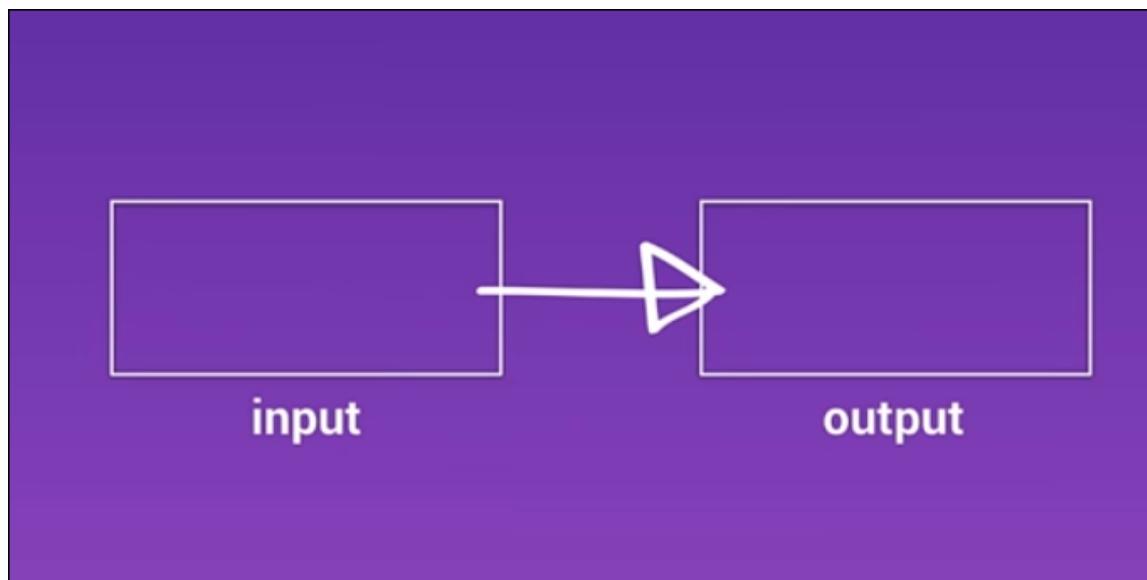
<https://auth0.com/blog/2015/10/15/angular-2-series-part-3-using-http/>

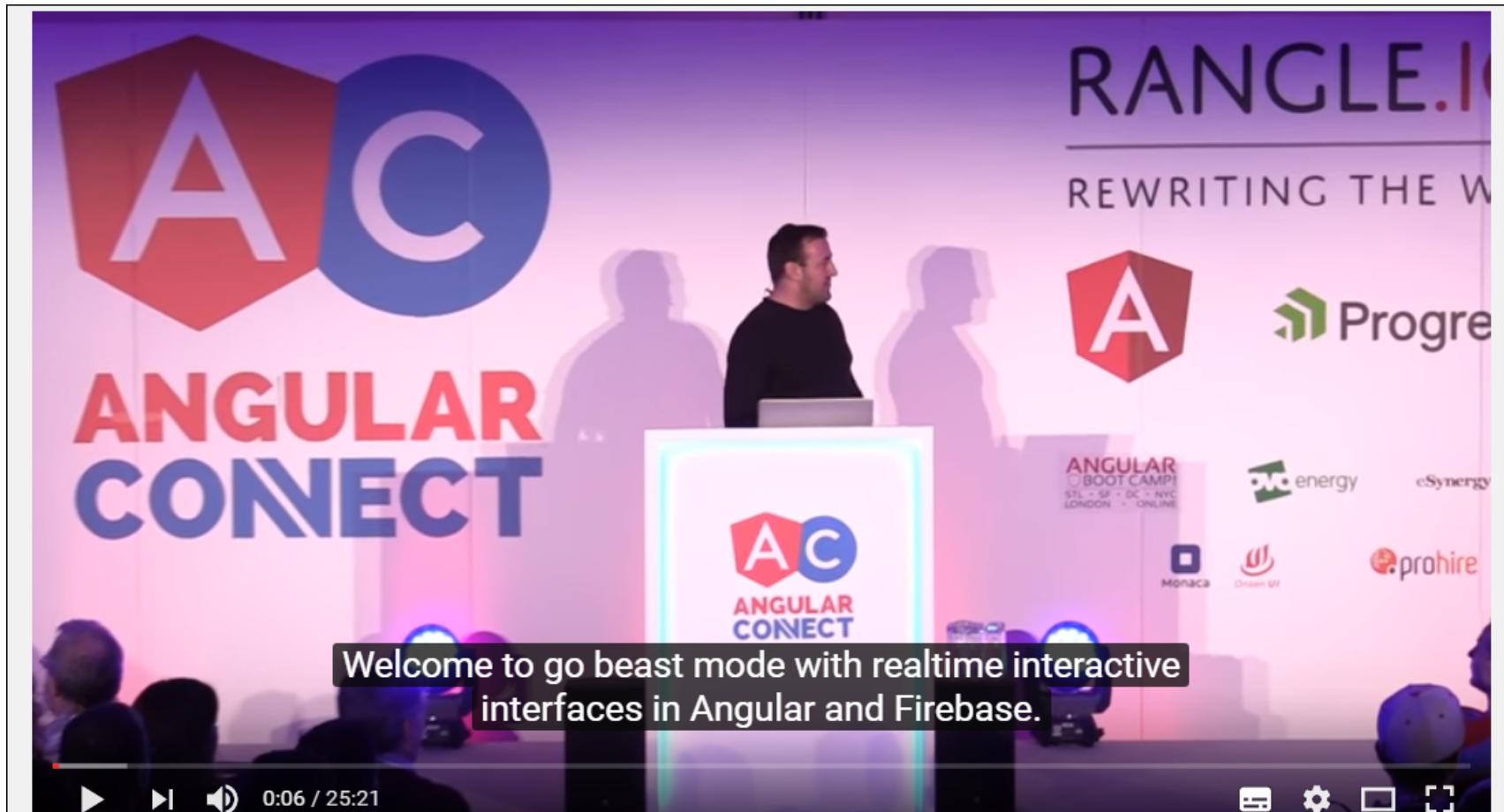
Observables en RxJs

- “Reactive Programming”
 - *“Reactive programming is programming with asynchronous data streams.”*
 - <https://gist.github.com/staltz/868e7e9bc2a7b8c1f754>
- Observables hebben extra mogelijkheden ten opzichte van Promises
 - Mapping
 - Filtering
 - Combining
 - Cancel
 - Retry
 - ...
- Gevolg: géén `.success()`, `.error()` en `.then()` chaining meer!

How do observables work

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



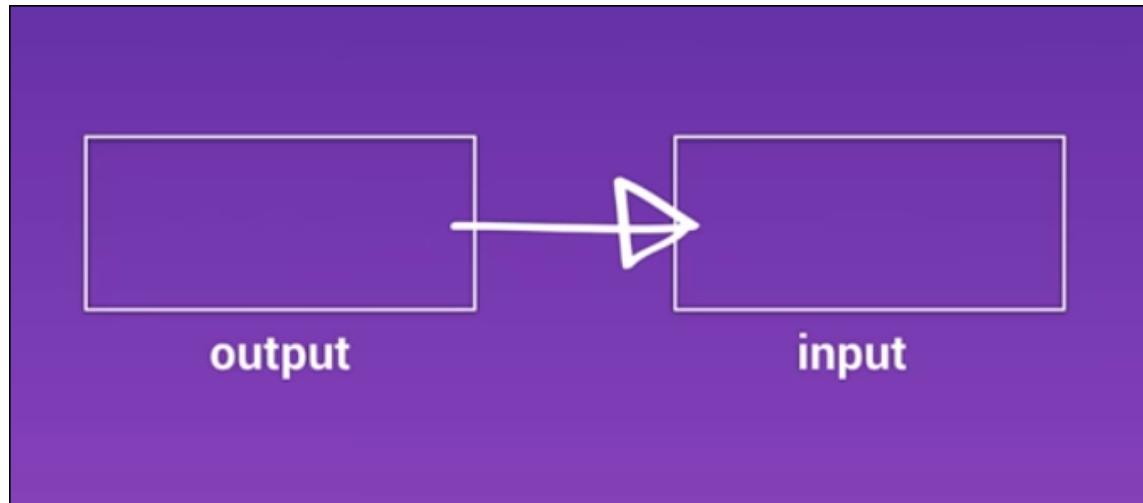


Go beast mode with realtime reactive interfaces in Angular 2 & Firebase | Lukas Ruebbelke

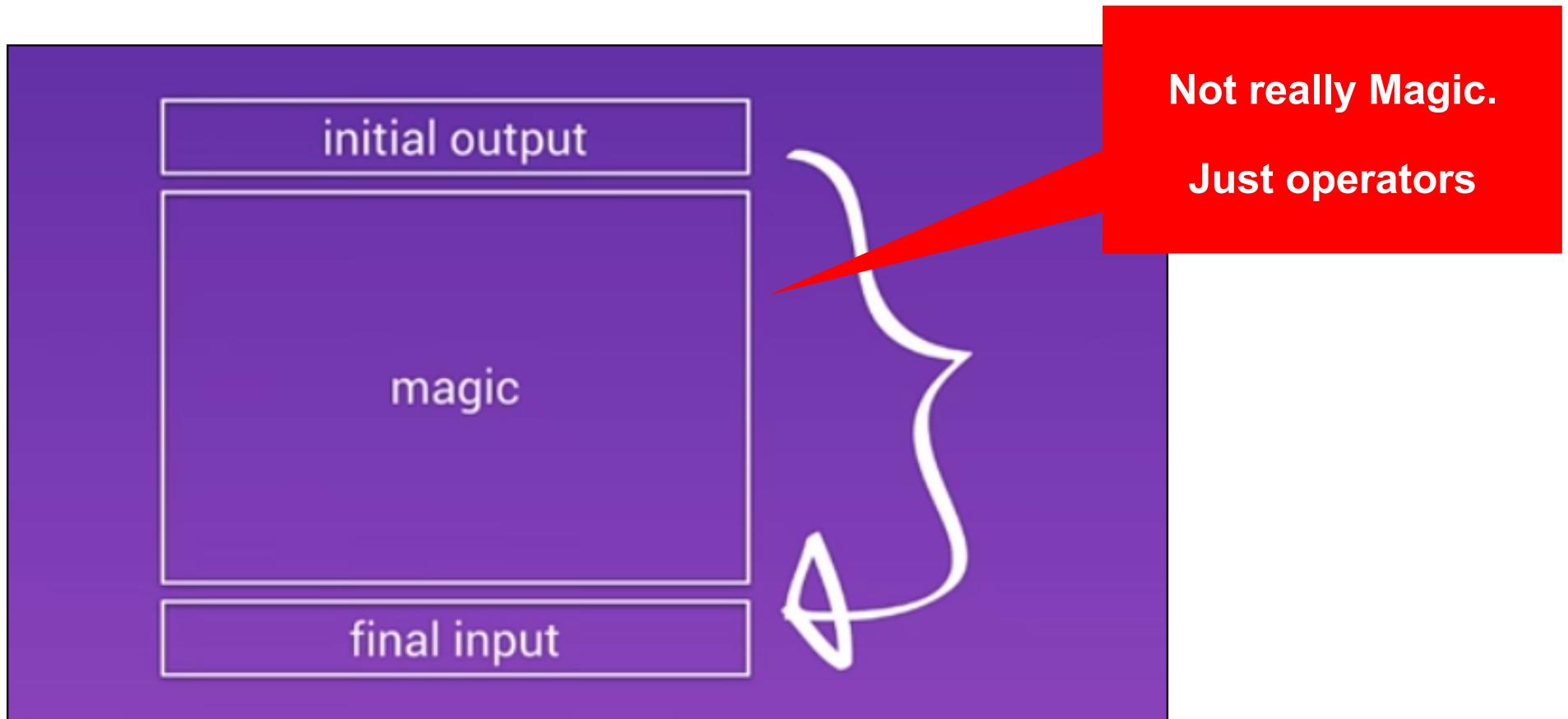
<https://www.youtube.com/watch?v=5CTL7aqSvJU>

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

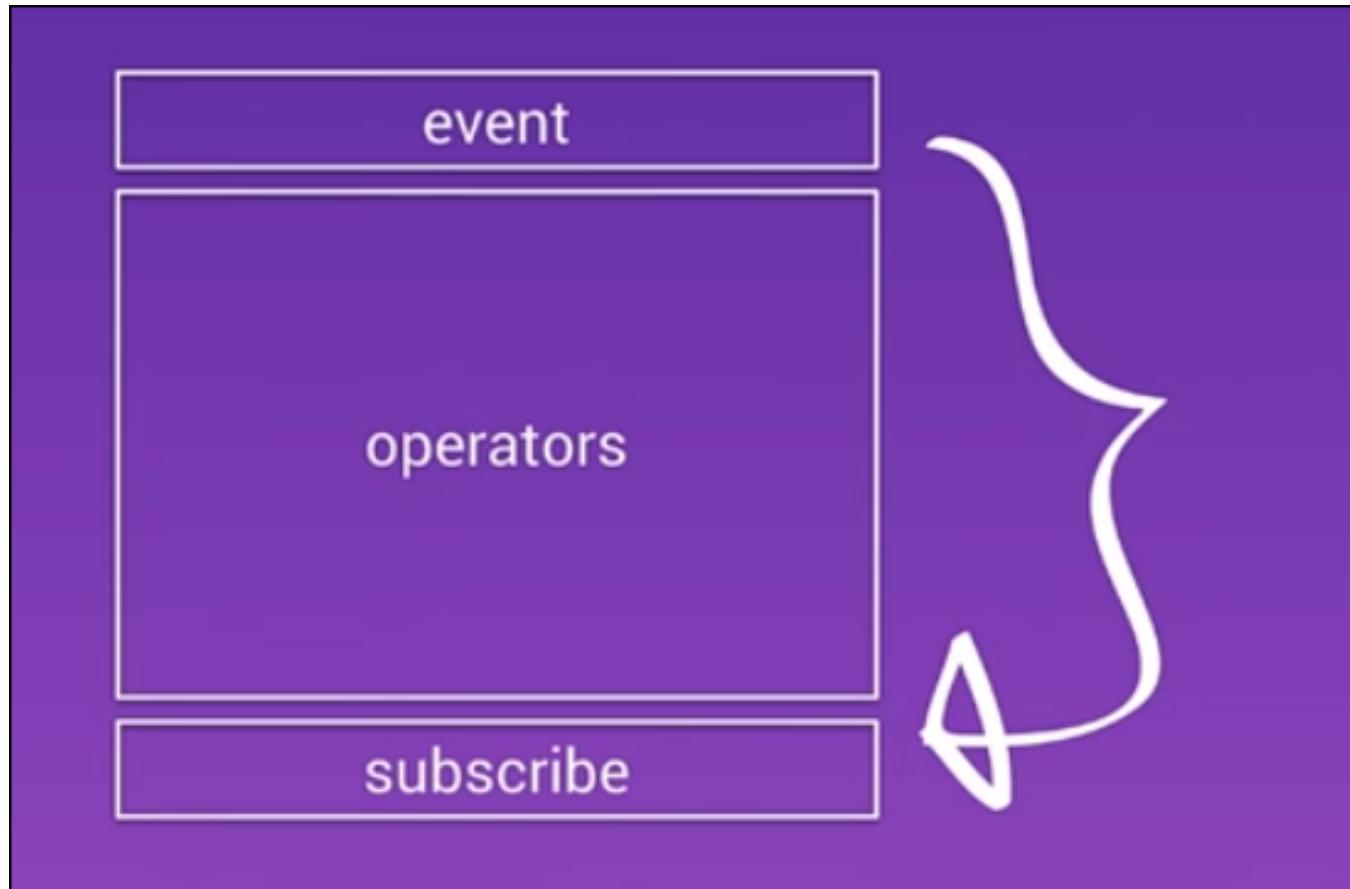
- With Observables -
 - a system, already outputting data,
 - Subscribe to that data
- "trade Output for Input"
- "Push vs. Pull"



"The observable sandwich"



Subscribe to events



In code

```
this.http.get('assets/data/cities.json')  
    .map(cities => cities.json())  
    .subscribe((result) => {  
        //... Do something  
    });
```

Initial Output

Operator(s)

Final Input

Ook: importeren HttpModule in @NgModule

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

Angular 4.3/5+: HttpClientModule

- In je @NgModule: imports : [HttpClientModule]
- Niet meer .map(res => res.json()).
 - Json is de standaard!
- Nieuwe optie: Interceptors
- <https://alligator.io/angular/httpclient-intro/> en
- <https://alligator.io/angular/httpclient-interceptors/>
- Is de standaard in Angular 5
 - HttpModule wordt in toekomstige versies verwijderd

Met HttpClientModule – geen mapping .json()

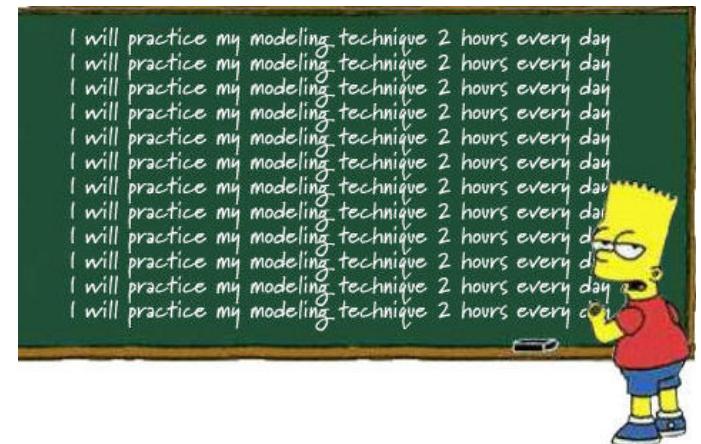
```
this.http.get<City[]>('assets/data/cities.json')  
.subscribe(result => {  
    //... Do something  
});
```

Wel: Type opgeven en casting bij de .get() call

Oefening

- Bekijk het voorbeeld in /201_services_rxjs
- Maak een eigen .json-bestand en importeer dit in je applicatie.
- Oefening 5c) , 5d)

Exercise....

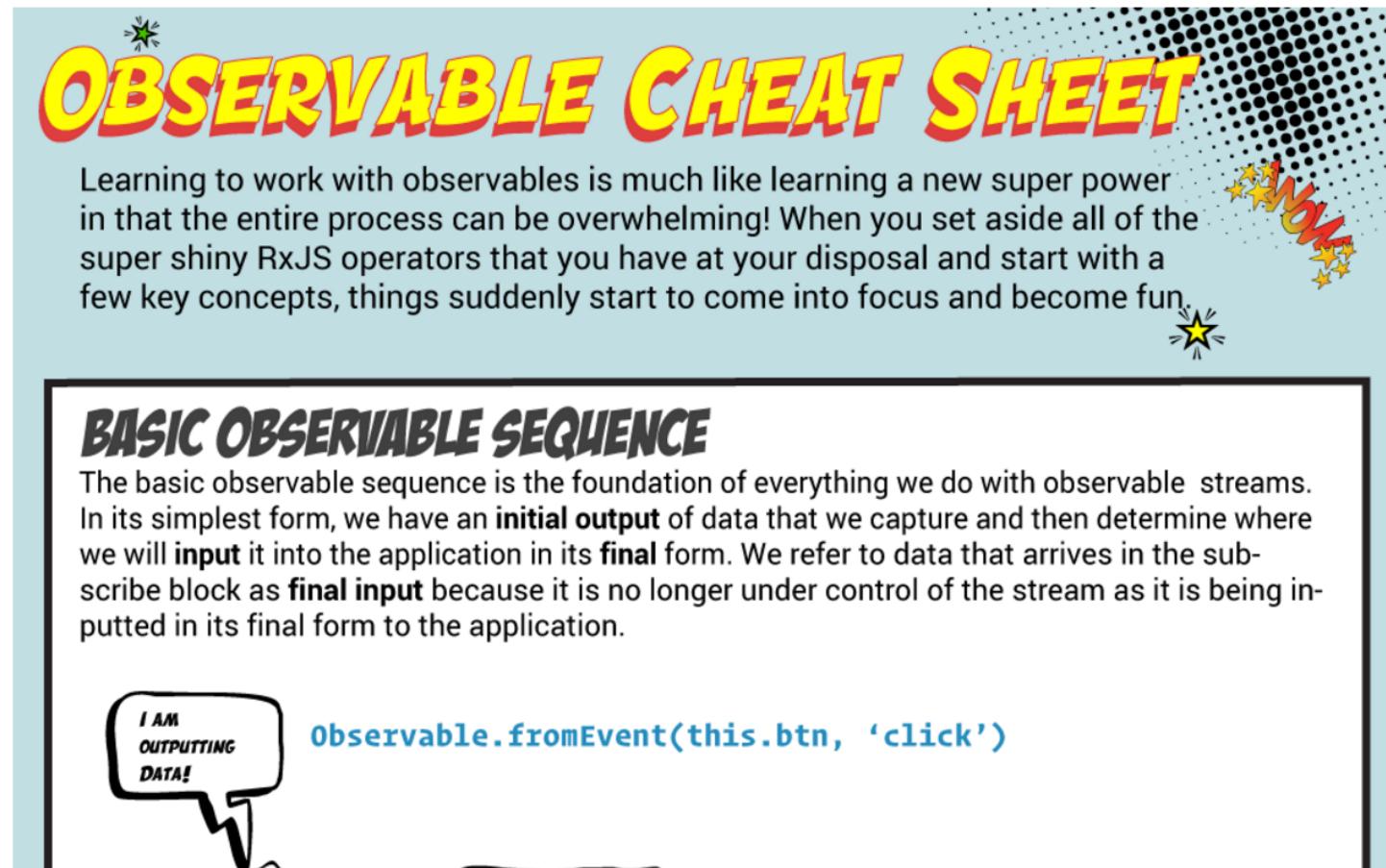


Observable Cheat Sheet

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

Gratis online training

The screenshot shows a course interface for 'ULTIMATE ANGULAR' with the specific 'Hello RxJS' micro-course selected. The left sidebar displays the course title and a progress bar indicating '5% COMPLETE'. The main content area is titled 'Class Curriculum' and lists the following modules:

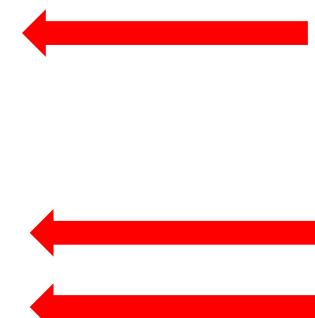
Module	Description	Action
1	Presentation: Realtime Observable Streams	Start
2	Slides: Realtime Observable Streams	Start
3	▶ The Basic Observable Sequence (2:09)	Start
4	📄 Lab: The Basic Observable Sequence	Start
5	▶ Mapping Values (2:22)	Start
6	📄 Lab: Mapping Values	Start
7	✓ ▶ Maintaining State (3:25)	
8	📄 Lab: Maintaining State	Start
9	▶ Merging Streams (1:57)	Start
10	📄 Lab: Merging Streams	Start
11	▶ Mapping to Functions (5:18)	Start
12	📄 Lab: Mapping to Functions	Start

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

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...'))
```



Observables in een Angular 2-applicatie

- Importeer Rx in de applicatie
 - Geheel, of alleen de benodigde onderdelen

```
import 'rxjs/Rx';  
  
import {Observable} from "rxjs";  
  
import 'rxjs/add/operator/map';
```

- Gebruik observable functies als .map(), .filter() etc.
- Piping. Resultaat van de een functie dient als invoer voor de volgende functie.

RxJS-operators in de service

```
import {Injectable} from '@angular/core';
import {Http, Response} from "@angular/http";
import {Observable} from "rxjs";
import 'rxjs/add/operator/map';

@Injectable()
export class CityService {

    constructor(private http: Http) {

        // retourneer alle cities
        getCities(): Observable<Response> {
            return this.http.get('app/cities.json')
                .map(cities => cities.json());
        }
    }
}
```

Import operator

Transform stream
in de service

```
getCities() {  
  if (!this.cities) {  
    this.cityService.getCities()  
      .map(res => res.json())  
      .delay(3000)  
      .subscribe(cityData => {  
        this.cities = cityData;  
      },  
      err => console.log(err),  
      ()=> console.log('Getting cities complete...')  
  )  
}  
}
```

RxJs-functies

Useful operators

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

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

The screenshot shows the homepage of the Learn RxJS website. On the left, there's a sidebar with a search bar and a sidebar menu. The sidebar menu includes sections for 'learn-rxjs' and 'LEARN RXJS', followed by a detailed list of operators under 'Operators'. 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 and a 'But...' section. At the bottom of the main content is a 'Content' section. The top right of the page has social sharing icons and user statistics: 'Star 733', 'Watch 54', and links to Twitter, Facebook, and GitHub.

Type to search

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



Async pipe

Automatische `.subscribe()` en `.unsubscribe()`

Async Pipe

- Bij `.subscribe()`, eigenlijk ook `.unsubscribe()` aanroepen.
 - Netjes!
 - Bij HTTP-requests niet beslist nodig, bij andere subscriptions wel, in verband met memory leaks.
- Niet meer zelf `.subscribe()` en `.unsubscribe()` aanroepen:
 - **Gebruik `async pipe` van Angular**

- In de component:

```
Cities$: Observable<City[]>; // Nu: Observable naar Type
```

```
...
```

```
ngOnInit() {  
    // Call naar de service, retourneert Observable  
    this.cities$ = this.cityService.getCities()  
}
```

- In de view:

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

Werken met Live API's

- MovieApp
- Oefeningen\09-services-live



Voorbeeld API's

- <https://pokeapi.co/> - Pokemon API
- <http://openweathermap.org/API> (weerbericht)
- <http://filltext.com/> (random NAW-gegevens)
- <http://ergast.com/mrd/> - Ergast Motor (F1) API
- <http://www.omdbapi.com/> - Open Movie Database
- <http://swapi.co/> - Star Wars API
- Zie ook JavaScript APIs.txt met meer voorbeelden

JSON Schema to typescript – server sided

Necessitates Proper Modularity

npm Enterprise features pricing documentation support

npm find packages sign up or log in

★ **json-schema-to-typescript** public

Compile json schema to typescript typings

Example

Input:

```
{  
  "title": "Example Schema",  
  "type": "object",  
  "properties": {  
    "firstName": {  
      "type": "string"  
    },  
    "lastName": {  
      "type": "string"  
    },  
    "age": {  
      "description": "Age in years",  
      "type": "integer",  
      "minimum": 0  
    }  
  }  
}
```

Manage permissions for the whole team
Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

npm i json-schema-to-typescript
[how?](#) [learn more](#)

bcherny published 16 hours ago

4.3.0 is the latest of 33 releases

github.com/bcherny/json-schema-to-typescript

MIT 

Collaborators [list](#)


Stats

<https://www.npmjs.com/package/json-schema-to-typescript>

Data Mocken - 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, 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 plans starting at \$50/year.

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 below the table. At the bottom of the page, there are controls for "# Rows" (set to 1000), "Format" (set to CSV), "Line Ending" (set to Unix (LF)), and "Include" options (header checked, BOM unchecked). There are also buttons for "Download Data" (green), "Preview", and "More". A note at the bottom asks if you want to save the data for later and provides a link to "Sign up for free".

<http://mockaroo.com/>

Official documentation...

The screenshot shows the RxJS documentation interface. At the top, there's a navigation bar with links for Home, Manual, Reference, Source, Test, and a theme switcher (dark theme/light theme). A search icon is also present. On the left, a sidebar lists various RxJS components categorized by type (e.g., C for Class, T for Type):

- C AsyncSubject
- C BehaviorSubject
- C Notification
- C Observable
- C ReplaySubject
- C Scheduler
- C AnonymousSubject
- C Subject
- C SubjectSubscriber
- C Subscriber
- C Subscription
- I ObservableInput
- I Observer
- I SubscribableOrPromise
- I TeardownLogic
- T Rx.Scheduler
- T Rx.Symbol
- observable
- C ConnectableObservable
- observable/dom
- C AjaxError
- C AjaxResponse
- C AjaxTimeoutError
- I AjaxRequest
- operator
- C GroupedObservable
- scheduler
- C Action

The main content area is titled "Observable". It contains sections for "Direct Subclass" (ConnectableObservable, GroupedObservable, Subject) and "Indirect Subclass" (AnonymousSubject, AsyncSubject, BehaviorSubject, es6/operator/windowTime.js~CountedSubject, ReplaySubject). Below these is a descriptive paragraph: "A representation of any set of values over any amount of time. This is the most basic building block of RxJS." There's also a "Test:" section with links to Observable, Observable.create, and Observable.lift.

At the bottom, there's a "Static Method Summary" section with a table for "Static Public Methods". The first method listed is bindCallback, which takes a function, selector, and scheduler as parameters and returns an Observable. A note below it says: "Converts a callback API to a function that returns an Observable." The second method listed is bindNodeCallback, which also takes a function, selector, and scheduler as parameters and returns an Observable.

At the very bottom right, there's a note: "Generated by ESDoc(0.4.8)".

<http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html>

More documentation

The screenshot shows a web browser displaying the ReactiveX documentation at <http://reactivex.io/documentation/operators.html>. The page title is "Operators By Category". Below the title, there are two main sections: "Creating Observables" and "Transforming Observables". Each section contains a bulleted list of operators and their descriptions.

Creating Observables

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

Transforming Observables

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>

Article - 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 that is the author's profile picture and name, 'Netanel Basal', with a 'Follow' button. The article title is 'RxJS—Six Operators That you Must Know'. The main content is a code snippet in a dark-themed code editor:

```
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);  
    }  
  }  
}  
// Usage:  
const subscriber = new TakeSubscriber({ total: 3 }, console.log);  
interval(1000).pipe(take(3)).subscribe(subscriber);
```

At the bottom, there's a call-to-action: 'Never miss a story from **NetanelBasal**, when you sign up for Medium. [Learn more](#)' with a 'GET UPDATES' button.

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

Some Examples

- /09a-services-live
- /09b-services-mapping
- /17c-forms-typeahead

```
this.videos = this.searchYouTube.valueChanges
    .debounceTime(600)                      // wait for 600ms to hit the API
    .map(query => makeURL(query))          // turn keyword into a real youtube-URL
    .switchMap(url => this.http.get(url))   // wait for, and switch to the Observable the
                                              switchmap)
    .map((res: Response) => res.json())     // map its response to json
    .map(response => response.items);       // unwrap the response and return only the items
```

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 slide with code in a code editor. The code defines several functions: `nextCallback`, `errorCallback`, `completeCallback`, and `giveMeSomeData`. The `giveMeSomeData` function uses `document.addEventListener` to handle a 'click' event. On the right, there is a video feed of André Staltz speaking at a podium with a laptop displaying the Angular logo. The video player has standard controls at the bottom: play/pause, volume, and a progress bar showing 5:11 / 22:44.

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

function errorCallback(err) {
}

function completeCallback() {
}

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

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

<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. Raw

The introduction to Reactive Programming you've been missing

introrx.md

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>

Also by Andre Stalz - RxMarbles

RxMarbles Interactive diagrams of Rx Observables

Fork me on GitHub

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; TopStream[20, 40, 60, 80, 100] --> Merge((merge)); BottomStream[1, 1] --> Merge; Merge --> MergedStream[20, 40, 60, 1, 80, 100, 1];
```

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

<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                     results: customers,
39                     totalRecords
40                 };
41             })
42             .catch(this.handleError);
43     }
44 }
```

52:13 / 1:24:02

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



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>