# Reactive Programming with RxJS

## Overview

This is a quick introduction into the concept of Reactive Programming in Angular with RxJS.

The subject of *Reactive Programming* is large enough to warrant it’s own course and indeed there are a number of books and courses which deal with nothing other than reactive programming with RxJS.

So the goal of this section *isn’t* to give you a *complete* understanding, the goal is to *demystify* the concepts so you understand what it is and *where* you can learn more.

Angular uses RxJS for some parts of it’s internal functioning. If you want you *can* also choose to use RxJS but you don’t need to at all.

You do *not* need to know Reactive Programming or RxJS in order to build even the

*most complex* applications with Angular. It can however make some types of

applications easier to architect.

At the end of this section you will:

* Understand the terms *Stream* & *Reactive Programming*.
* Know what *Observables* are and how they are related to *RxJS*.
* Know how to write reactive code using pure RxJS.
* Know what operators are, where to read up on the breadth of available operators and how to understand them by using marble diagrams.
* Know the places you can use reactive programming in Angular and code up a simple form using RxJS.

## Streams & Reactive Programming

**Learning Objectives**

* Know what streams are and how to think about things that happen in an application as streams.
* Know what reactive programming is and how to start transforming your way of thinking from imperative to reactive.

**What are Streams?**

Streams are a *sequence of values over time*, that’s it.

For example a number that goes up by 1 every second might have a *stream* that looks like [0,1,2,3,4]

Another stream might be a sequence of x and y positions of mouse click events, like so:

[(12,34), (345,22), (1,993)]

We could even have a stream to represent a user filling in a form on a website.

We could have a stream to represent each keypress, like so:

[

"A",

"s",

"i",

"m"

]

Or we could have a stream which contains a json representation of the whole form as the user enters data, like so:

[

{ "name": "A" },

{ "name": "As" },

{ "name": "Asi" },

{ "name": "Asim" }

]

We could have a stream for:

* The x,y position of the mouse as it moves around the screen in a HTML5 game.
* The data returned from a real-time websockets connection.
* The chat windows opened by this user in a browser.

The more you think about it the more *everything* we do with a web application can be thought of as a stream.

**What is Reactive Programming?**

Reactive programming is the idea that you can create your entire program just by defining the different streams and the *operations* that are performed on those streams.

As a concept that is easy to write, but how can we actually *train* our mind to program *reactively*?

To explain this lets convert a simple imperative function into a reactive one.



Imperative programming is a programming paradigm that you probably have

been using so far in your career, it’s by far the most common and it’s involves

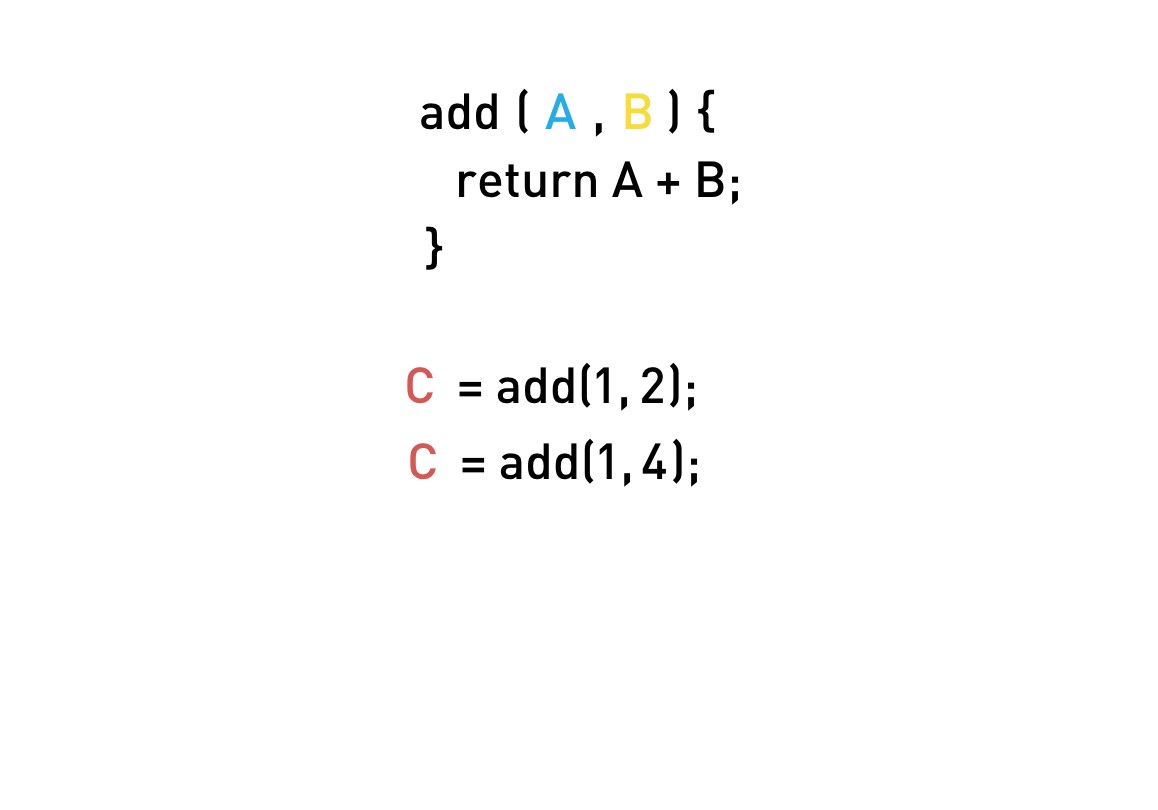
executing statements that change a programs state, i.e. call functions that change

the values of variables.

To get a good overview of the different programming paradigms read this

articl

[e](https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms)



We have a function called add and some state variables, A, B and C.

To add A and B together and *change the state* of C to be the sum we call the function add.

Later on the value of B changes to 4.

1. First we need a way of simply *knowing* that B has changed, that’s hard enough to figure out by itself.
2. Secondly we need to know that because B has changed we need to recalculate C.

In a web application our inputs are constantly changing over time, via things like network events or a user interacting with a mouse.

Most of the logic we end up writing is just to figure out *what* functions need to be called for each of these changes to our inputs.

Applications can be thought of as just a huge pile of *variables* (which we call application state) as well *logic* to decide which *functions* to call, and in what order,

when any of those variables change.

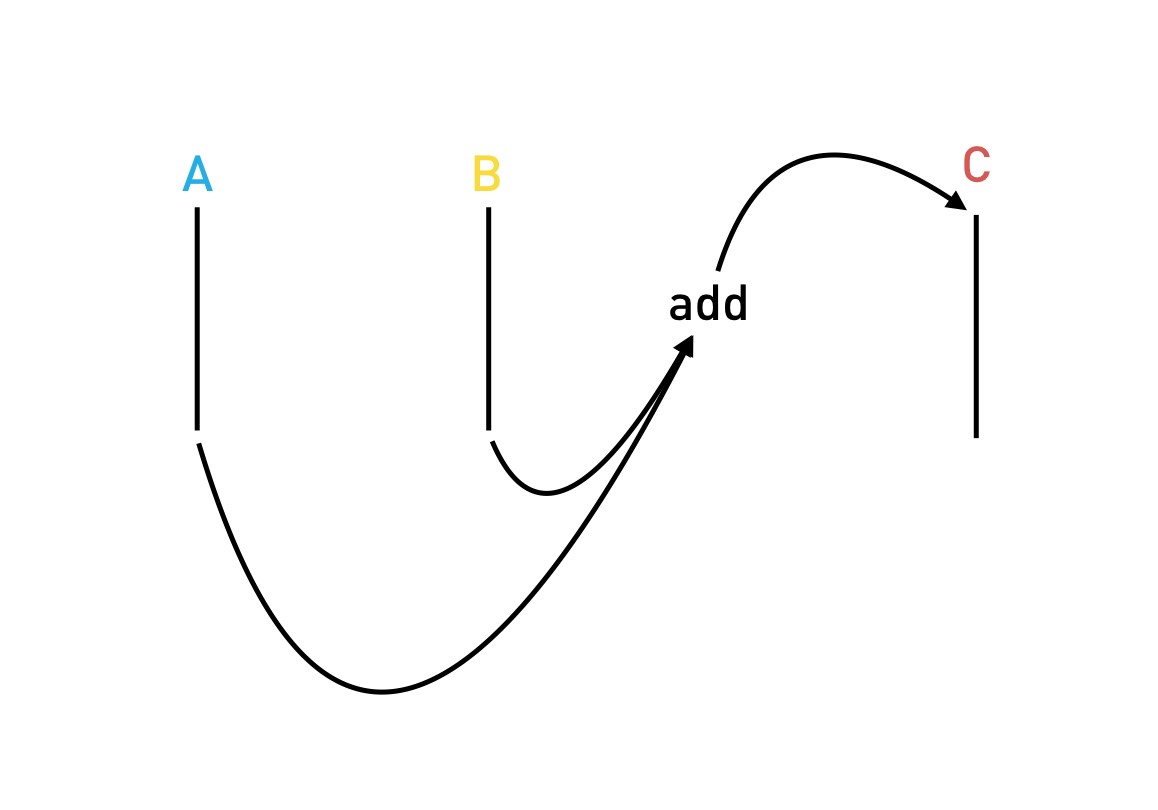
Calling those functions then also changes the values of variables, for which we need additional logic to figure out what *other* functions to call… it’s endless!

With reactive programming we *stop* thinking about variables, instead we think in terms of *streams* and how those streams are *connected* together.

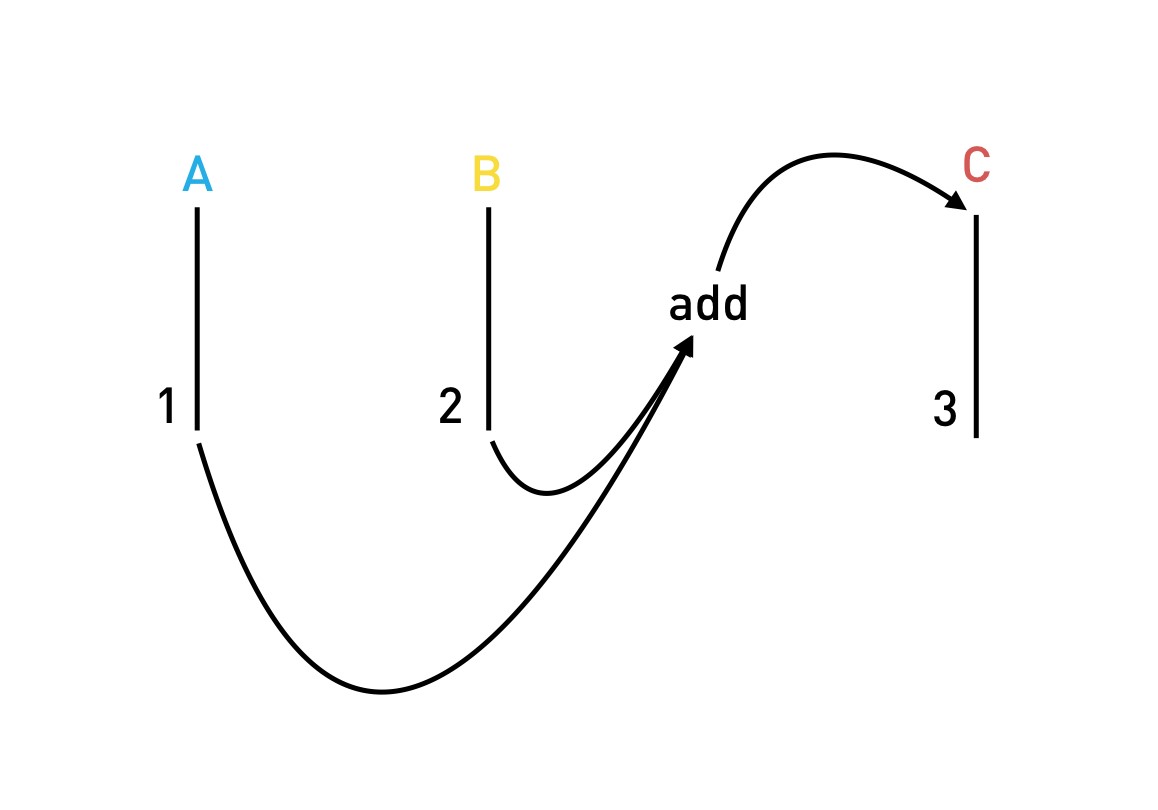
Going back to our example, we convert the variables A, B and C into *streams*.

A is now *not* an individual value at one point in time, it’s a *stream* of values over time.

The function add we think of an an *operation* we perform which connects the output of streams A and B to the input of stream C, a visual representation would be something like the below:



Now if we push some numbers onto stream A and B, the add operation is *automatically* called, calculating the sum of 4 and pushing it onto stream C.



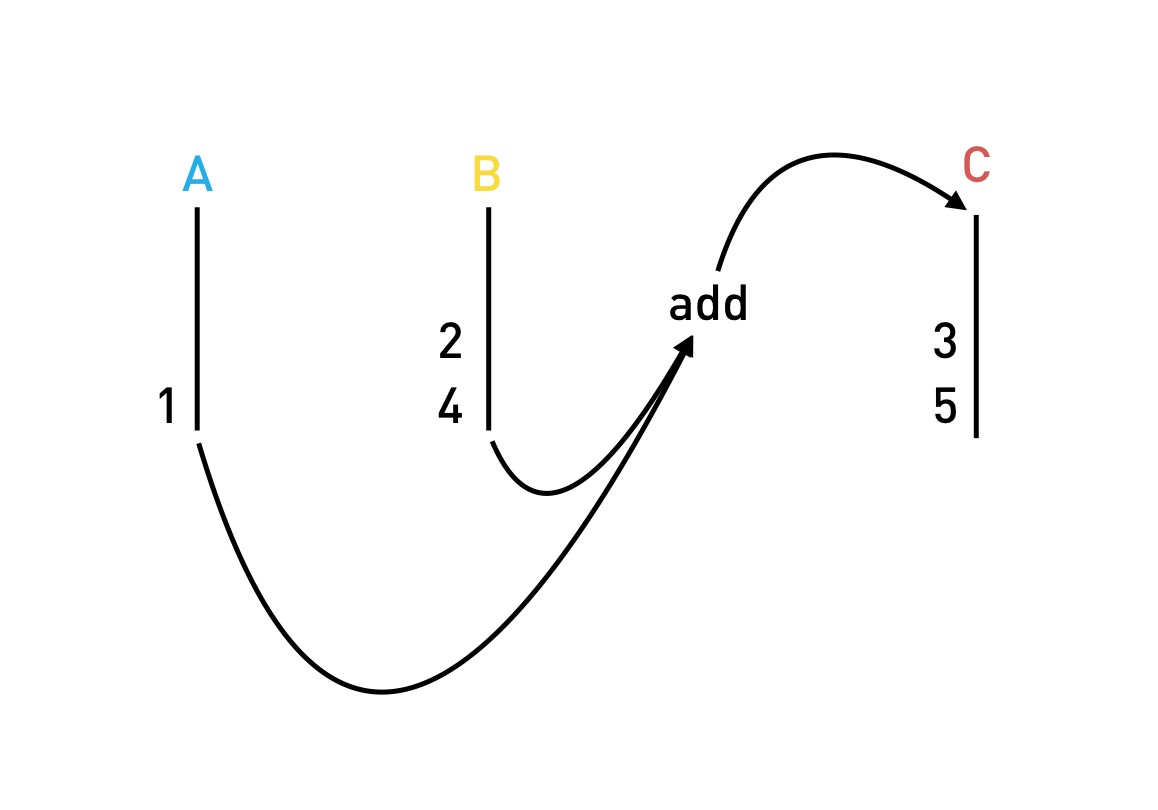
If stream C was connected to another stream via another operation, that operation would then be called automatically as well.

An analogy which works for me is to think about reactive programming as

*plumbing*. We decide which *pipes* we need in our application, we decide *how* those pipes are connected together and then we *turn on* the water and sit back.

With reactive programming *we don’t call functions*, we just define how our application is plumbed together and start pushing values onto streams and let the plumbing and operations handle the rest.

So if later on the value of B changes, we simply push the new value onto the stream B and then let the plumbing handle the rest, like so:



**Summary**

Streams are just a *sequence of values* over time.

Reactive programming is the idea we can define an application as a series of different streams with operations that connect the different streams together and which are automatically called when new values are pushed onto those streams.

In this this lecture we just covered the idea, the concept of reactive programming, in the next lecture we’ll cover how to actually program reactively using observables and the RxJS library.