**176. Understanding Operators**

* Instructor: Now it's all nice what we learned .
* But there is one super important thing about observables and the RxJS library that you absolutely have to know and that would be Operators .
* **Operators:**
* Operators are the magic feature of the RxJS library and they are the thing that really turn observables into awesome constructs .
* If we have an observable and an observer we of course get data and we listen to that with a subscription .
* That is what you learn .
* Now, however, sometimes you don't need the raw data .
* You might want to transform it or filter out certain data points and of course you could do all of that inside of your subscription or in the function you pass to your subscription .
* But there is a more elegant way .

Timeline

Description automatically generated

* Instead of setting up that subscription like this you can use builtin operators in between .
* That means that the data points, first of all reach these operators that do something to the data and that something can be anything .
* *There are tons of builtin operators and then you subscribe to the result of these operators .*
* **Example:**
* Now let me show you a concrete example that makes that clearer .
* Let's say here, in our application, we of course get our values 0,1,2,3 before we get an error .
* And let's say I'm not interested in these values but I want to have some text that says something like round one, round two, round three, and so on .
* So in the end I want to use that count add one and add the word round in front of it .
* Now obviously what we can do is we can simply log, round plus data plus one here, add extra parenthesis around that so that it is really used as a calculation .
* And if we do that well we see round one, two, three and so on being locked here .
* Now nothing wrong with that and for simple transformations like this one, of course that makes sense but the more complex your logic gets the nicer it might be to not care about that here in the place where you subscribe .
* But in an earlier place .
* So here I'll revert this to just logging the data and I wanna change the format in which that data arrives .
* And obviously we could do that directly here where we create our observable .
* **Operators:**
* But what if you don't own the observable? What if you wanted to change the route prints, for example? You can't change the original code and that is where operators are important .
* You can use them on any observable like our custom interval observable by calling a method called pipe .
* Now every observable has a pipe method .
* The pipe method is built into RxJS .

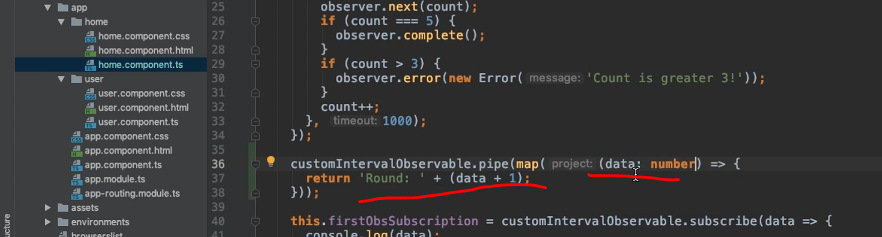
Text

Description automatically generated

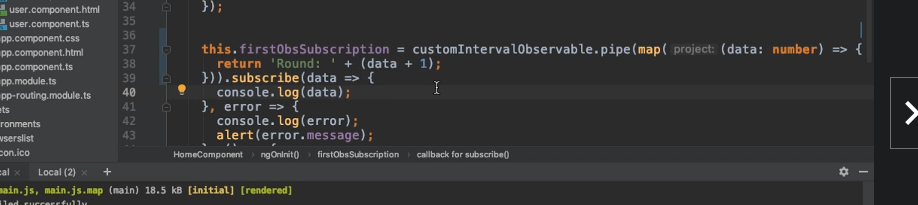
* Now you can import observables not from the RxJS package but from RxJS slash operators and there are tons of builtin operators like the map operator which is one of the operators you'll use more often .
* As a side note, if you want to learn more about RxJS and operators hands on you can visit my homepage academind .
* com and there on the learn JavaScript you'll find a complete series on RxJS which introduces you to RxJS and to operators as well .
* It uses slightly older syntax .
* RxJS was changed in the meantime, which by the way all the cover on my page, here in RxJS six what changed, but the core idea and the explanation about these operators still applies .
* That can be very useful for learning more about RxJS but that's just a side note back to our code here .

**Pipe and Map:**

* So we have that pipe method and we're importing map here and map, as I mentioned is one of the more important operators .
* Now we simply call map as a function inside of pipe here .
* And map in turn takes a function as an argument .
* So an anonymous function here if you want that however gets an argument that anonymous function gets an argument and that is the data you would otherwise get here in your first subscribe callback .
* So in that first function you pass to subscribe it's the current data of your observable that's being emitted by that observable .
* So we get that data here and in map we now have to return new data that can of course be the unchanged old data, in which case that map operator is totally useless .
* But it can also be transformed data like for example, round plus data plus one .



* So what we previously did right in console lock now we're doing it here .
* We just have to be clear about the format of data here as it seems .
* You can of course do that by wrapping data and parenthesis here in the argument list, adding a colon after it and making it clear that this will be a number and now if you're safe that it will compile .
* But what you'll see in your project now is still the old output .
* The reason for data is let's wait for this to complete .
* The reason for data is that this is applied, but we still subscribe to this observable and this does not change data inside of the observable .

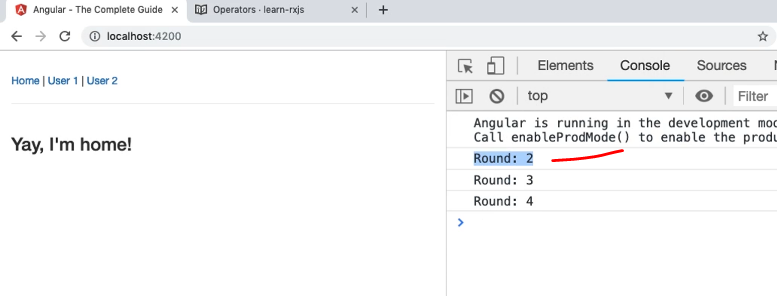


* Instead, it only changes the data we get after pipe .
* So what we would have to do here is simply we have to take that and replace our observable here with it .
* So the concrete observable, we're subscribing to you .
* Now if we do that, so if we have our observable then we pipe the data .
* So we add a couple of operators in this case the map operator .
* Then when we subscribe thereafter we'll get that mapped .
* So that changed data and now when we save this, we'll see that that we get round 1, 2, 3 and so on .
* So this is the map operator .
* And again, just before this is kind of a constructed example here it is of course more useful .
* For example, if you're fetching complex data from a web server and you want to transform that data before you use it in a component and that is something we'll do later too but the core idea is the same with pipe, we can add one or more operators .
* If you have more than one you'll simply add them as arguments here to the pipe method .
* So the pipe method takes an unlimited amount of arguments and each argument would be an operator imported from RxJS operators and you can add as many as you want and they will execute after each other and do different kinds of stuff on the data .
* This is map of course, for example, you could also filter .
* That's our operator indeed .

Text

Description automatically generated

* So let's use that just to demonstrate one more .
* Now there are dozens by the way, besides using my page here .
* You can also visit learn RxJS .
* io where you find a long list of the operators with examples .
* So that's really useful .
* But back to our code .
* Now let's dive into filter .
* And if we wanted to add that too we would simply add it here, separate by a comma in front of map as an extra argument here and filter all that gets the data of course .
* And inside of that anonymous function which we pass to filter, where we do get that data we now have to return true or false, which decides whether that data point will continue in that chain .
* So whether it will reach map and thereafter subscription or whether it will be dropped in which case it will never reach map nor the subscription .
* Now of course we could hard code true or false here but that's not too useful .
* Instead we could say, "Yeah return true if data is greater than zero .
* " And that effectively means that we will return every value .
* But the very first one and therefore instance we bridge round data plus one will not start with round one because we're dropping this zero value .
* Instead we'll actually start with round two .
* Cause only then this condition is true .
* So if we now go back to our application here, let this reload, we see nothing, but then round two round three and so on, so round one, indeed is skipped and that shows you hopefully the power of operators that they allow you to build up a chain of steps you want to funnel your observable data through .



* That can be really helpful when it comes to transforming data, filtering out data, and so on .