 For this lesson, we're going to create our own observable by creating an observable from scratch.

[00:07] An observable is really nothing more than a function -- we'll call it "My Observable" -- that takes an observer. An observer isn't anything more than an object that has a Next method that gets a value, and Error method that gets an error, and a Complete method that takes no argument.

[00:36] Let's have this do something. I'll start off by just creating a nice For loop inside here, that goes up to 9 amidst these values, through the observer's Next method, and when it's done, calls Complete.

[00:57] When I call this function, this is the equivalent of calling Subscribe on an observable, and I'll show you that in a little bit. I'm going to reuse this observer that we had up here, and I'll have it log out, and we'll have it log something when it's complete.

[01:15] Let's go ahead and try it. You see it logs out our 10 digits, and then completes.

[01:26] That's your basic observable, but what happens when we want to do something asynchronous, like, I don't know, use a set Timeout in here? In that case, I'm going to do a set Timeout. I'm just going to have it log out. Anything, "Hello" will do.

[01:50] We want to be able to cancel this, so what we need to be able to do is return some cancellation semantic, like so, and I'm going to make this last half a second. Let's go ahead and run it. It waits half a second, and then logs "Hello." I don't have it completing, so let's add that, and run it again -- "Hello," and then Complete.

[02:20] What if I wanted to cancel this before this time out? That's easy. My observable is now returning an Unsubscription method, and if I wanted to, I could call this on a Timeout that was a little bit shorter. I could shorten this up, and when I run this, nothing happens, because my Unsub method was called, which is right here. I'll log that out, so you can see.

[02:55] See, that was called. It cleared our Timeout, so this is never fired, but we've gotten a little problem with our observer.

[03:07] If I was to take this out of here and go back to my regular loop, and then in here, I'm going to again Next Out that value. Afterwards, I'm going to complete it. I don't need to worry about the Unsubscription anymore. I run this. It does the same thing it did before, but what happens if right after it, I observe another value?

[03:50] That's not OK. I just told you that this is complete, and there you can see, it's not at all complete. We need to prevent this from happening. How do we do that?

[04:02] To do that, we need to create some way to wrap that observer, and we're going to call this a Safe Observer. I'm going to call the observer we're going to pass into it, the observer we're wrapping, the Destination. This is not something you would ever have to do in your own code, because libraries like RxJS do this for you.

[04:30] It's, of course, going to have a Next method that accepts a value, and that's really just going to forward values onto the destination, but we want to check to see if it's unsubscribed, so we're going to add this little bit in here.

[04:58] Now, how do we know when it's unsubscribed? That's tricky, right? We want to unsubscribe it when Complete is fired, so we'll go ahead and add that in here. The Error method won't look much different than this, other than it's passing an error around, so I'll just copy and paste that in.

[05:33] What we're going to have to do is, inside of my observable, we're going to have to take this observer and wrap it in a Safe Observer and use that internally. I'll even leave this bit in here, because we shouldn't see this logged out now. I'm also going to have to move this Class up above where we were using it. Let's give that a whirl.

[06:03] You see it's not logging out "HaHa" anymore, but now we've got another problem. This is kind of gross. You don't want to have to do this every time you implement an observable, so maybe we could take observable, and I know it's just a function, but let's make it into a Class, so we can handle some of this for you.

[06:22] Here's what we're going to do. Create a Class called "Observable." It's going to have a constructor that takes that -- this exact method right here. I'm going to call this a Subscriber method, or a Subscribe method, and it's going to have its own Subscribe method that when you call it with an observer, it just wraps that in a Safe observer and then calls Subscribe.

[07:00] We can change this around so this will be an actual object. We don't need this, a Safe observer, anymore. Whoo, I just spotted a bug ahead in there before, but that's OK. The real thing I wanted to show you is down here. I wouldn't have caught that beforehand. That's the benefits of watching somebody else code, I suppose.

[07:31] Now we could use what appears to be the observer directly, because whatever comes in here is going to be a Safe observer. Use the same observer, and then down here, I have to call Subscribe instead. Try that. Oh, Clear it and try it again. The exact same result, so we can see this is working, and it's not allowing this. I'll go ahead and remove this, because it doesn't matter anymore.

[08:00] Let's try our Async. For our Async version, let's go ahead and do an interval this time. We'll have that only go to 9, and then complete. We have to have it return some Teardown logic. Let's try to run this.

[08:33] We're not incrementing. Do over. What? We're still not incrementing. Also, it'd be nicer if this actually ticked -- make it tick a little faster. There we go, so now it's working, but we still need to go ahead and try out the Teardown, so let's do that. Let me give this a little more room. We'll delay this for half a second or so, and then call our Unsub.

[09:11] We made it that far, and then we unsubscribed. There's one last little bit that we want out of this. We don't want to accidentally allow our observer to have a value nexted through it when you've already unsubscribed, so if someone calls this Teardown logic, you need to know ahead of time so you can flag our Safe observer as "Is unsubscribed."

[09:54] After that's done, make sure that you can't Next Through. You have to flip this flag here, so that means that our Safe observer needs to know. It basically needs to wrap this Teardown logic that we're passing in, so how do we do that?

[10:11] That's pretty simple, actually. All I'm going to do here is I'm going to provide that. I'm going to return a new function that calls a Safe observer, Unsubscribe, with no underscore, because this guy is going to call this guy. Let's go up here and implement that.

[10:36] First things first. It sets "Is unsubscribed," and then it's going to call Unsubscribe, that we pass to it, if it exists, because sometimes, we might not get any value back here, so it's important. There we go.

[11:00] The only other thing that's left is we don't want to simply flag this. Now, we'll actually call Unsubscribe, so when our Complete logic fires, it actually fires our Teardown logic, because that's really what we want.

[11:18] The same thing here, and we'll try to demonstrate that here in a second, too. I'm going to add in a little bit of logging, just so that we can witness this happen. I'll try out our code, see that it's unsubbing right here, but what about the case of our Complete? Is it still going to unsub? If we ran this before, we added this. This wouldn't have happened.

[11:49] Let's try it out now. It unsubscribes, and completes. The order of that actually isn't quite right, because we might want an operator later on that says, "Hey, let's repeat this. Let's not unsubscribe just yet. Let's wrap this around," so I'm going to go ahead, and I'm going to change the order of that. Try it again. There we go -- Complete, Unsub.

[12:17] This is a really rudimentary and basic observable implementation. There are a lot of other things that RxJS is going to do for you. For example, what happens when this user-provided function throws an error? You should probably tear down all of the Unsubscription logic that you've been passed from here.

[12:40] The same thing with errors here and here. You want to make sure that those are thrown, and finally, there are concerns around what happens if one observable in a chain of observables throws an error during unsubscription. You still want to unsubscribe and tear down your resources from those other observables, otherwise, you'll get memory leaks.

[13:02] I don't really recommend implementing your own observable and using it. Even though this looks nice and fun and terse, there's a lot more to it than just this.

[13:13] At the end of this lesson, the thing I want you to remember the most is that an observable is nothing more than a function. It's made basically to wrap this Subscribe call, and make sure that a few things happen safely inside of it. This function here, this is your observable.

[13:33] When you call Subscribe, it doesn't do anything but call this function, and the observable's on its way. The observer is really what's doing all the work for you.

[13:42] The final thing I wanted to say about this implementation is that I was returning Unsubscribe functions. In RxJS, we returned subscription objects that have an Unsubscribe method. There are a variety of reasons for this, not the least of which is I could name this anything I wanted, and then use it to unsubscribe later, and that's not very readable.

[14:05] If this is an object with an Unsubscribe, it's easy to tell that this was a subscription object, because of how we're calling it. There's more reasons than that, but that's probably the biggest one.