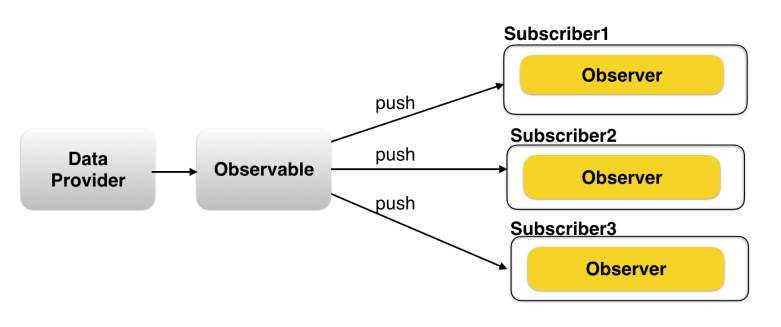
# Angular

# RxJS



An observable gets data from some data source (a socket, an array, UI events) one element at a time. To be precise, an observable knows how to do three things:

1. Emit the next element to the observer
2. Throw an error on the observer
3. Inform the observer that the stream is over

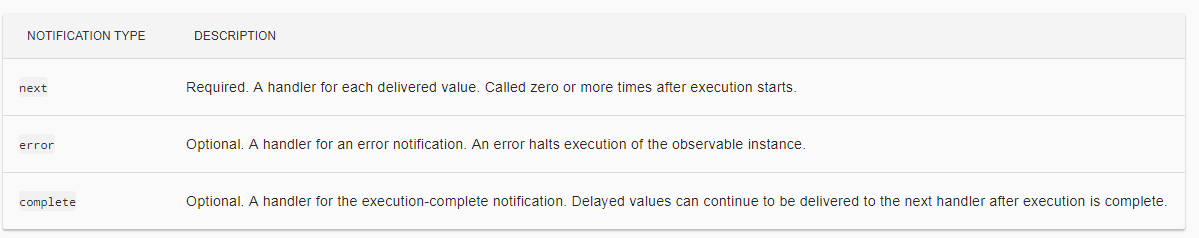
Accordingly, an observer object provides up to three callbacks:

1. The function to handle the next element emitted by the observable
2. The function to handle errors thrown by the observable
3. The function to handle the end of stream

The subscriber connects an observable and observer by invoking the method subscribe() and disconnects them by invoking unsubscribe().

## Q) Observer

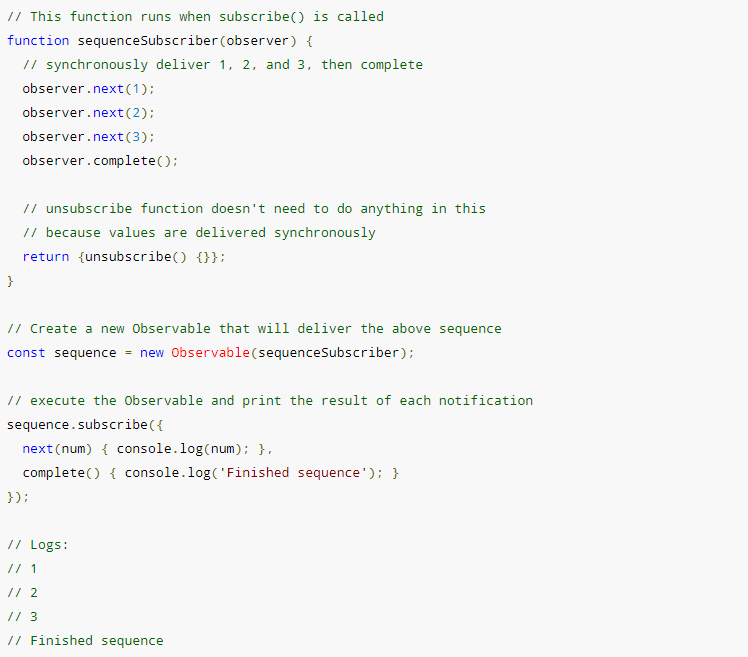
A handler for receiving observable notifications implements the Observer interface. It is an object that defines callback methods to handle the three types of notifications that an observable can send:

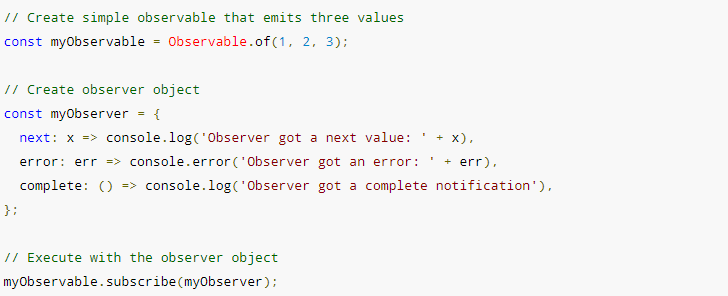


## Q) Observable

Use the Observable constructor to create an observable stream of any type.

A subscriber function receives an Observer object, and can publish values to the observer's next() method.





### Q) **Creating observables**

RxJS offers multiple ways of creating an observable depending on the type of the data producer.

* Observable.of(1,2,3) – turns the sequence of numbers into an Observable
* Observable.create(myObserver) – returns an Observable that can invoke  methods on myObserver that you will create and supply as an argument
* Observable.from(myArray) – converts an array represented by the variable myArray into an Observable. You can also use any an iterable data collection or a generator function as an argument of from().
* Observable.fromEvent(myInput, ‘keyup’) – converts the keyup event from some HTML element represented by myInput into an Observable
* Observable.interval(1000) – emits a sequential integer (0,1,2,3…) every second

### Q) Async Observable

By default, the operator from() returns a synchronous observable, but if you want an asynchronous one, use a second argument specifying an async scheduler:

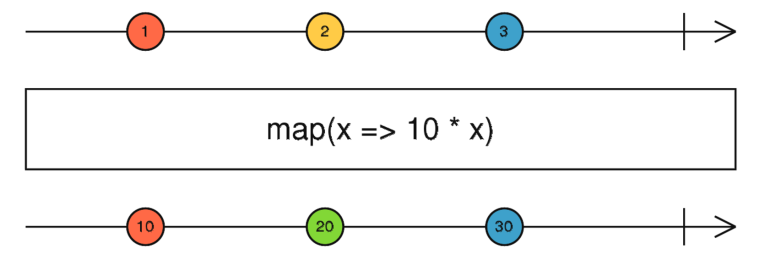


## Q) **Subscriber**

An Observable instance begins publishing values only when someone subscribes to it. You subscribe by calling the subscribe() method of the instance, passing an observer object to receive the notifications.

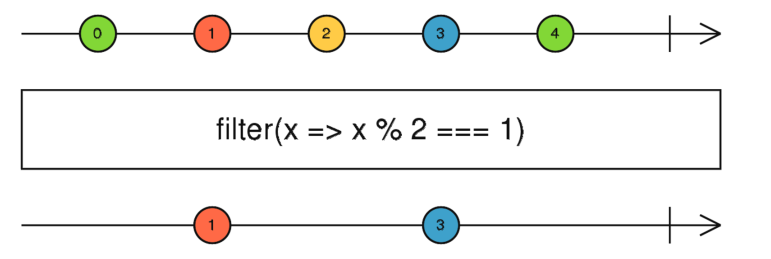
## **Q) map()**

The map() operator transforms one value to another. It takes a given value from the observable stream and applies the provided transforming function to it.



## **Q) filter()**

The filter() operator takes a function predicate as an argument, which returns true if the emitted value meets the criteria, or false otherwise. Only the values that meet the criteria will make it to the observer.



## **Q) reduce()**

The operator reduce() that allows you aggregate values emitted by an observable.

As you see from the above diagram, the accumulator function also has two arguments:

* acc stores the currently accumulated value, which is available for each emitted element
* curr stores the currently emitted value.

