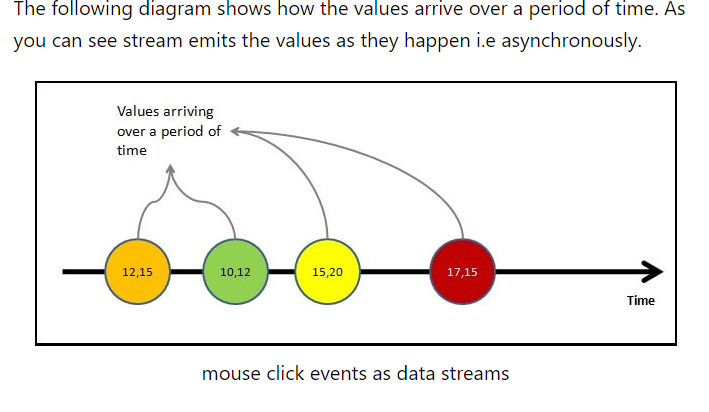
**Angular Observable**

**Angular Observable**, we hear a lot of terms like Reactive programming, data streams, Observable, Observers, RxJS, etc. It is very important to understand these terms before we start using the observables.

Rx stands from Reactive programming. It is defined as programming with asynchronous **data streams**.

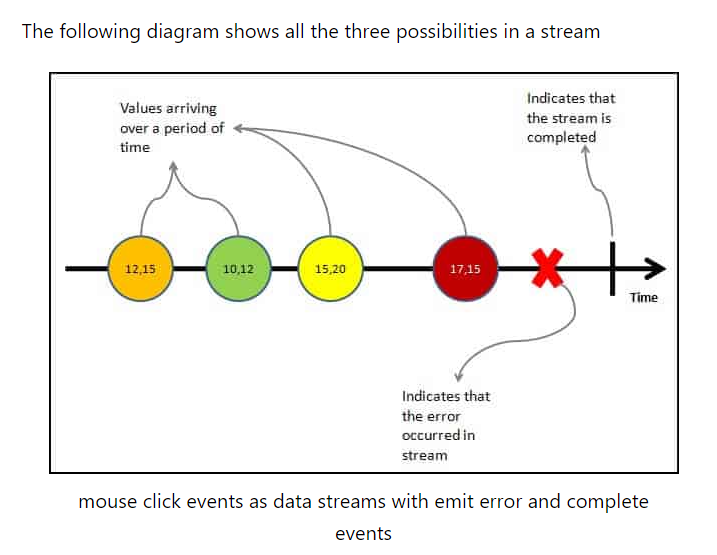
**What is a data stream**

A stream is a data, which arrives over a period of time. The stream of data can be anything. Like variables, user inputs, properties, caches, data structures, and even failures, etc



Value is not the only thing that stream emits. The stream may complete as the user closes the window or app. Or an error may happen which results in the closure of the stream. At any point in time stream may emit any of the following three things

**Value:** i.e the next value in the stream  
**Complete**: The stream has ended  
**Error**: The error has stopped the stream.



As said earlier the stream of data can be anything. For Example

* Mouse click or Mouse hover events with x & y positions
* Keyboard events like keyup, keydown, keypress, etc
* Form events like value changes etc
* Data which arrives after an HTTP request
* User Notifications
* Measurements from any sensor

Important Points regarding streams can

* emit zero, one or more values of any time.
* can also emit errors.
* must emit the complete signal, when completes (finite streams).
* can be infinite, that they never complete

## Reactive Programming

The reactive programming is all about creating the stream, emitting value, error or complete signals, manipulate, transfer or do something useful with the data streams.

This is where the RxJs comes into the picture

**What is RxJS**

The [RxJS](https://rxjs.dev/guide/overview) (Reactive Extensions Library for JavaScript) is a javascript library, that allows us to work with asynchronous data streams

The Angular uses the RxJS library heavily in its framework to implement Reactive Programming. Some of the examples where reactive programming used are

* Reacting to an [HTTP request in Angular](https://www.tektutorialshub.com/angular/angular-httpclient/)
* [Value changes](https://www.tektutorialshub.com/angular/valuechanges-in-angular-forms/) / [Status Changes](https://www.tektutorialshub.com/angular/statuschanges-in-angular-forms/) in Angular Forms
* The Router and Forms modules use observables to listen for and respond to user-input events.
* You can define custom events that send observable output data from a child to a parent component.
* The HTTP module uses observables to handle AJAX requests and responses.

The RxJs has two main players

1. Observable
2. Observers ( Subscribers)

## What is an Observable in Angular

Observable is a function that converts the **ordinary stream of data** into an **observable stream of data**. You can think of Observable as a wrapper around the **ordinary stream of data**.

**Observable stream** or simple Observable emits the **value from the stream** asynchronously. It emits the **complete** signals when the stream completes or an **error** signal if the stream errors out.

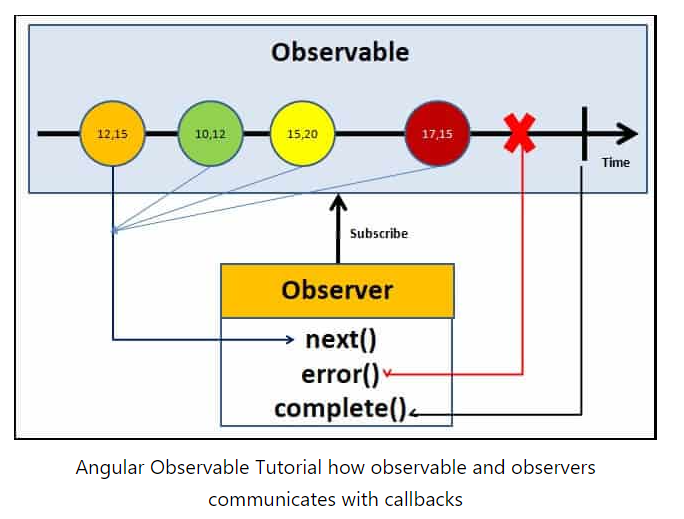
Observables are declarative. You define an observable function just like any other variable. The observable starts to emit values only when **someone subscribes to it**.

## Who are observers (subscribers)

The Observable on its own is useless unless someone consumes the value emitted by the observable. We call them observers or subscribers.

The observers communicate with the Observable using callbacks

The observer must subscribe with the observable to receive the value from the observer. While subscribing it optionally passes the three callbacks. next(), error() & complete()



The observable starts emitting the value as soon as observer or consumer subscribes to it.

The observable invokes the next() callback whenever the value arrives in the stream. It passes the value as the argument to the next callback. If the error occurs, then the error() callback is invoked. It invokes the complete() callback when the stream completes.

* Observers/subscribers subscribe to Observables
* Observer registers three callbacks with the observable at the time of subscribing. i .e next(), error() & complete()
* All three callbacks are optional
* The observer receives the data from the observer via the next() callback
* They also receive the errors and completion events from the Observable via the error() & complete() callbacks

## Angular Observable tutorial

Now, we have learned the basics of the RxJs Observable, let us now see how it works using an example.

### Import the required libraries

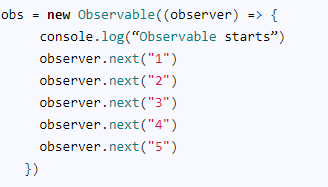
RxJs library is installed automatically when you create the Angular project. Hence there is no need to install it.

Import the Observable from the rxjs library

|  |  |
| --- | --- |
|  |  |

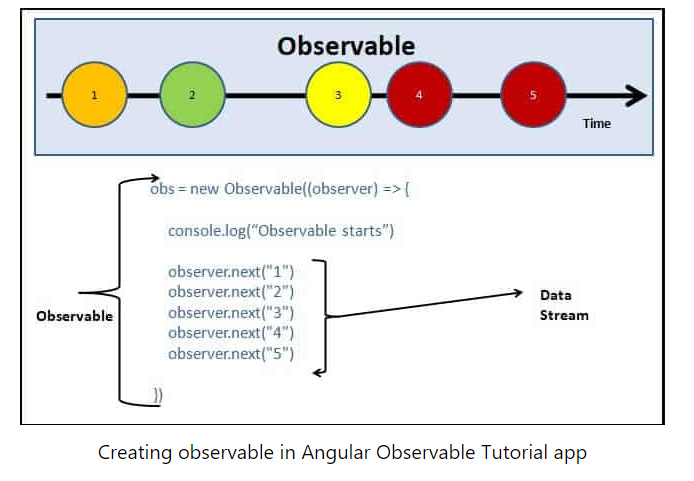
### Observable Creation

There are few ways in which you can create observable in angular. Simplest is to use the Observable constructor. The observable constructor takes observer (or subscriber) as its argument. The subscriber will run when this observable’s subscribe() method executes.



The variable obs is now of the type of observable.

The above example declares the obs as the observable but does not instantiate it. To make the observable to emit values, we need to subscribe to it.

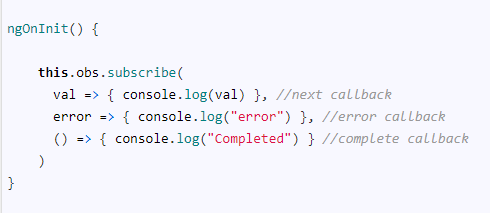


There are many operators available with the RxJS library, which makes the task of creating the observable easy. These operators help us to create observable from an array, string, promise, any iterable, etc. Here are list some of the commonly used operators

* [create](https://www.tektutorialshub.com/angular/rxjs-observable-using-create-of-from-in-angular/)
* defer
* empty
* [from](https://www.tektutorialshub.com/angular/rxjs-observable-using-create-of-from-in-angular/)
* [fromEvent](https://www.tektutorialshub.com/angular/create-observable-from-event-using-fromevent-in-angular/)
* interval
* [of](https://www.tektutorialshub.com/angular/rxjs-observable-using-create-of-from-in-angular/)
* range
* [throwError](https://www.tektutorialshub.com/angular/using-throwerror-in-angular-observable/)
* timer

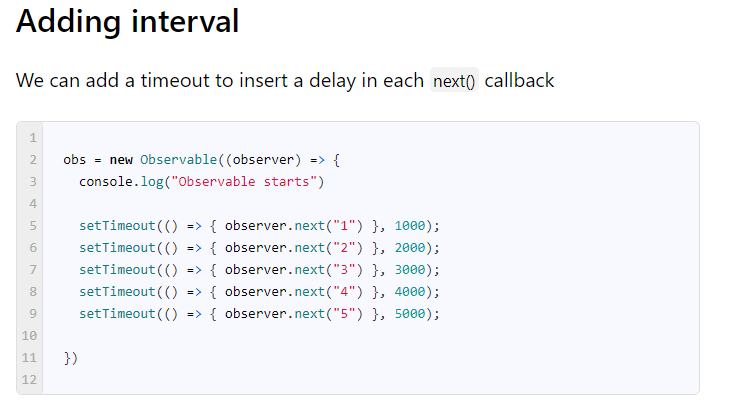
### Subscribing to the observable

We subscribe to the observable, by invoking the subscribe method on it. We can optionally, include the three callbacks next(), error() & complete()



Complete example:





### Error event

As mentioned earlier, the observable can also emit an error. This is done by invoking the error() callback and passing the error object. The observables stop after emitting the error signal. Hence values 4 & 5 are never emitted.

obs = new Observable((observer) => {

    console.log("Observable starts")

    setTimeout(() => { observer.next("1") }, 1000);

    setTimeout(() => { observer.next("2") }, 2000);

    setTimeout(() => { observer.next("3") }, 3000);

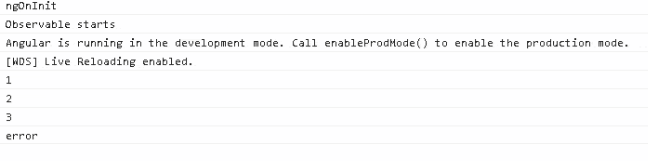
    setTimeout(() => { observer.error("error emitted") }, 3500);    //sending error event. observable stops here

    setTimeout(() => { observer.next("4") }, 4000);          //this code is never called

    setTimeout(() => { observer.next("5") }, 5000);

  })

Output:



### Complete Event

Similarly the complete event. The observables stop after emitting the complete signal. Hence values 4 & 5 are never emitted.

    obs = new Observable((observer) => {

    console.log("Observable starts")

    setTimeout(() => { observer.next("1") }, 1000);

    setTimeout(() => { observer.next("2") }, 2000);

    setTimeout(() => { observer.next("3") }, 3000);

    setTimeout(() => { observer.complete() }, 3500);   //sending complete event. observable stops here

    setTimeout(() => { observer.next("4") }, 4000);    //this code is never called

    setTimeout(() => { observer.next("5") }, 5000);

  })

Output:



## Observable Operators

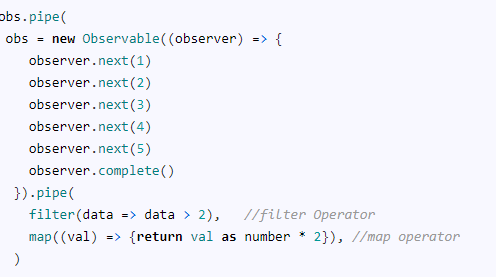
The Operators are functions that operate on an Observable and return a new Observable.

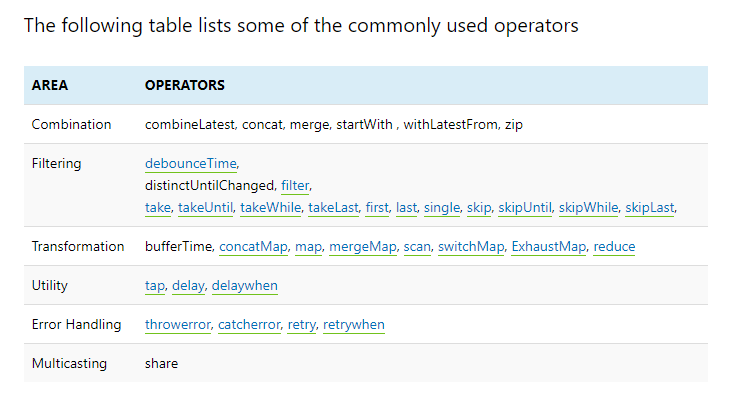
The power of observable comes from the [operators](https://angular.io/guide/rx-library#operators). You can use them to manipulate the incoming observable, filter it, merge it with another observable, alter the values or subscribe to another observable.

You can also chain each operator one after the other using the [pipe](https://www.tektutorialshub.com/angular/angular-observable-pipe/). Each operator in the chain gets the observable from the previous operator. It modifies it and creates a new observable, which becomes the input for the next observable.

The following example shows the [filer](https://www.tektutorialshub.com/angular/filter-operator-in-angular-observable/) & [map](https://www.tektutorialshub.com/angular/angular-observable-map-operator/) operators chained inside a [pipe](https://www.tektutorialshub.com/angular/angular-observable-pipe/). The filter operator removes all data which is less than or equal to 2 and the map operator multiplies the value by 2.

The input stream is [1,2,3,4,5] , while the output is [6, 8, 10].

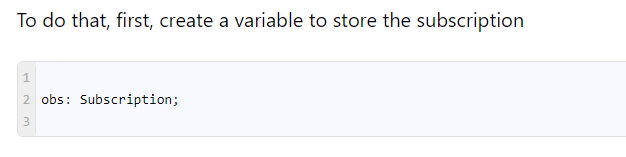




## Unsubscribing from an Observable

We need to [unsubscribe to close the observable](https://www.tektutorialshub.com/angular/unsubscribing-from-an-observable-in-angular/) when we no longer require it. If not it may lead to memory leak & Performance degradation.

To Unsubscribe from an observable, we need to call the Unsubscribe() method on the subscription. It will clean up all listeners and frees up the memory.





When we destroy the component, the observable is unsubscribed and cleaned up.

But, you do not have to unsubscribe from every subscription. For Example, the observables, which emits the complete signal, close the observable.

# **Create observable from a string, array & object in angular**

There are many ways to create observable in Angular. You can make use of Observable Constructor as shown as above.

There are a number of functions that are available which you can use to create new observables. These operators help us to create observable from an array, string, promise, any iterable, etc. Here are some of the operators

* create
* defer
* empty
* from
* fromEvent
* interval
* of
* range
* throw
* timer

All the creation related operators are part of the RxJs core library. You can import it from the ‘rxjs’ library

**Create**

The Create method is one of the easiest. The create method calls the observable constructor behind the scene. Create is a method of the observable object, Hence you do not have to import it.

|  |
| --- |
| 1 |

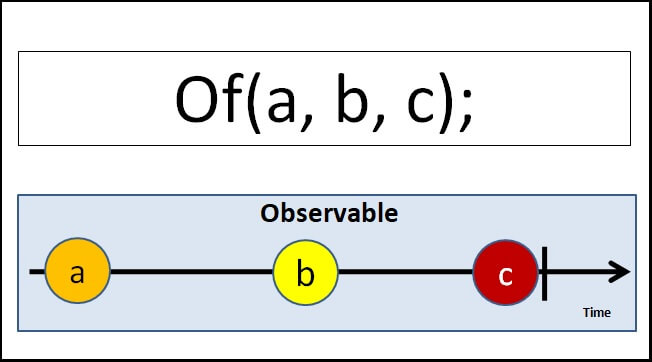


Note:

There is no difference between the Observable.create method and observable constructor. The Create method calls the constructor behind the scene.

## Of Operator

The Of creates the observable from the arguments that you pass into it. You can pass any number of arguments to the Of. Each argument emitted separately and one after the other. It sends the Complete signal in the end.

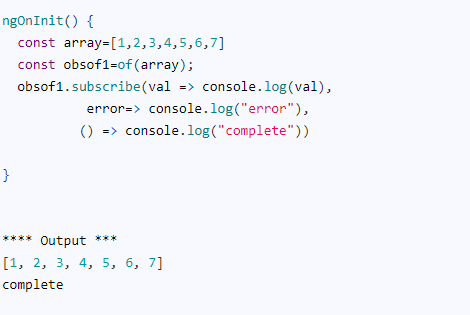


To use of you need to import it from rxjs library as shown below.



### observable from an array

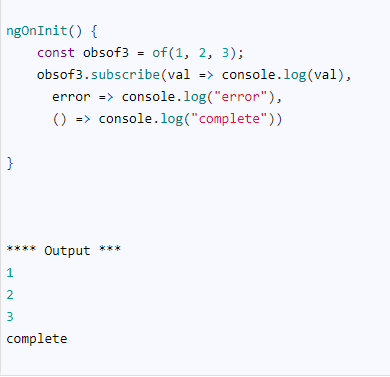
Example of sending an array. Note that the entire array is emitted at once.





### observable from a sequence of numbers

In the following example, we pass 1,2 & 3 as the argument to the from. Each emitted separately.



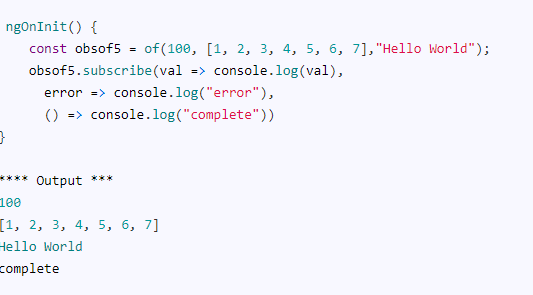
### observable from string

We pass two strings to the of method. Each argument is emitted as it is.



### observable from a value, array & string

We can pass anything to the Of operator. It justs emits it back one after the other.



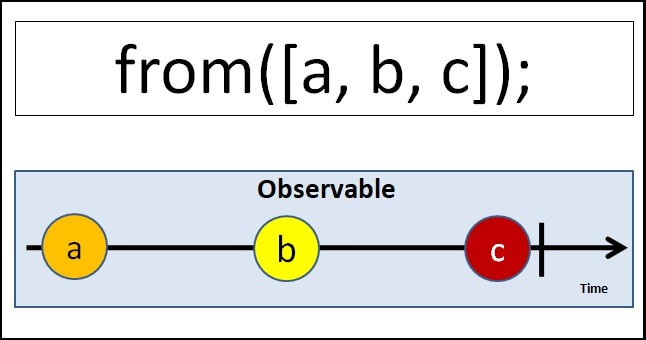
**From Operator**

From Operator takes only one argument that can be iterated and converts it into an observable.

You can use it to convert

* an Array,
* anything that behaves like an array
* Promise
* any iterable object
* collections
* any observable like object

It converts almost anything that can be iterated to an Observable.

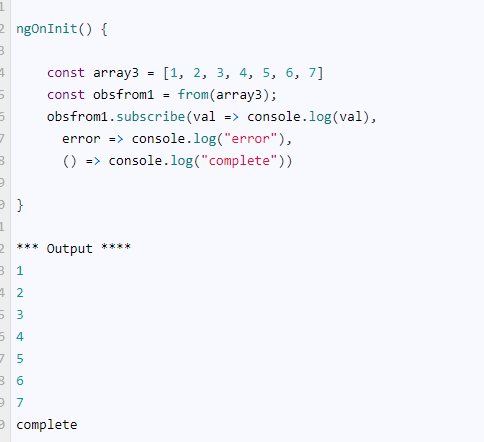


To use from you need to import it from rxjs library as shown below.



### observable from an array

The following example converts an array into an observable. Note that each element of the array is iterated and emitted separately



### Observable from string

The from operator iterates over each character of the string and then emits it. The example is as shown below.



### Observable from collection

Anything that can be iterated can be converted to observable. Here is an example using a collection.



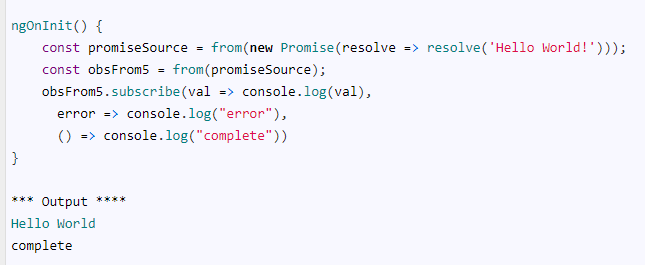
### Observable from iterable

Any Iterable types like Generator functions can be converted into an observable using from the operator.



### Observable from promise

Use it to convert a Promise to an observable

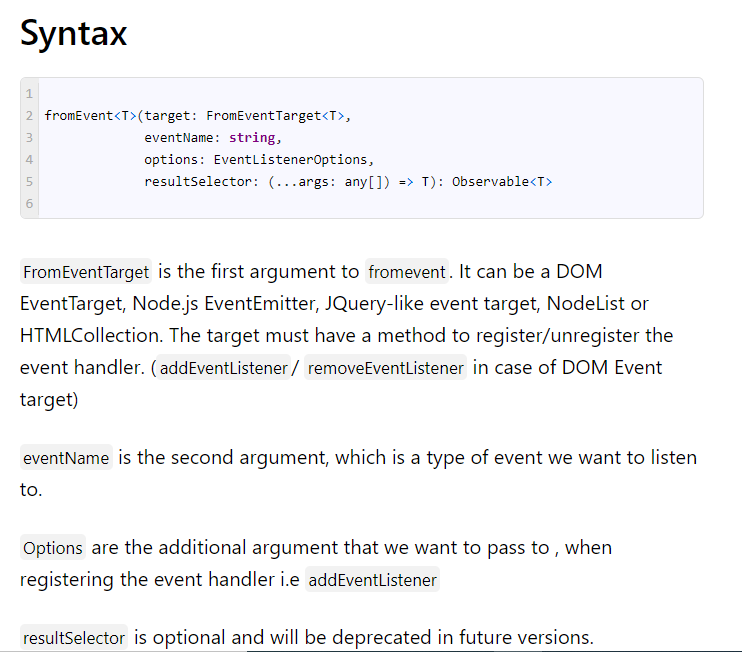


## Of Vs From

| **Of** | **from** |
| --- | --- |
| Accepts variable no of arguments | Accepts only one argument |
| emits each argument as it is without changing anything | iterates over the argument and emits each value |

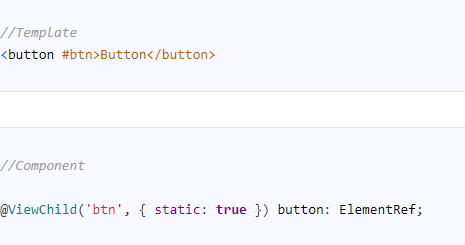
# **Create Observable from Event using FromEvent in Angular**

Angular provides FromEvent method to create an observable from DOM events directly. In this article let us learn how to do that by creating an observable from the button click event, keyup even & scroll events.

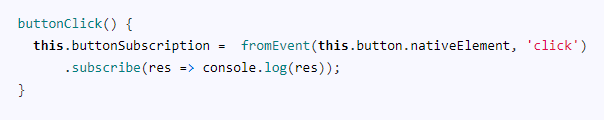


## Example of fromEvent

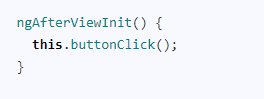
To create an observable from any event, first, we need to get the reference to DOM element using the viewchild & ElementRef



The code this.button.nativeElement returns the native DOM element. We pass this as the first argument to the fromEvent to create an observable to the click event.



We can invoke the above method from the ngAfterViewInit method. Note that the @ViewChildwill not initialize the btn element until the [ngOnInit](https://www.tektutorialshub.com/angular/angular-ngoninit-and-ngondestroy/) Hence we are using the ngAfterViewInit here.

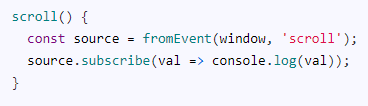


### How it works

When we subscribe to an observable, which we created using the fromEvent method, it registers the event handler using the addEventListener in the DOM element. Whenever the user clicks on the button, fromevent captures the value and emits it to the subscriber as the first argument. When we unsubscribe, it unregisters the event handler using the removeEventListener.

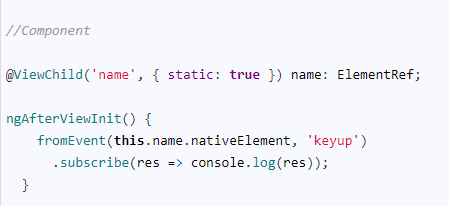
## fromevent from scroll

The following code shows how to create observable from the window scroll event



## fromevent from keyup

The following code shows how to create observable from a keyUp event.



# **Using Angular observable pipe with example**

The pipe method of the [Angular Observable](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/) is used to chain multiple operators together. We can use the pipe as a standalone method, which helps us to reuse it at multiple places or as an instance method.

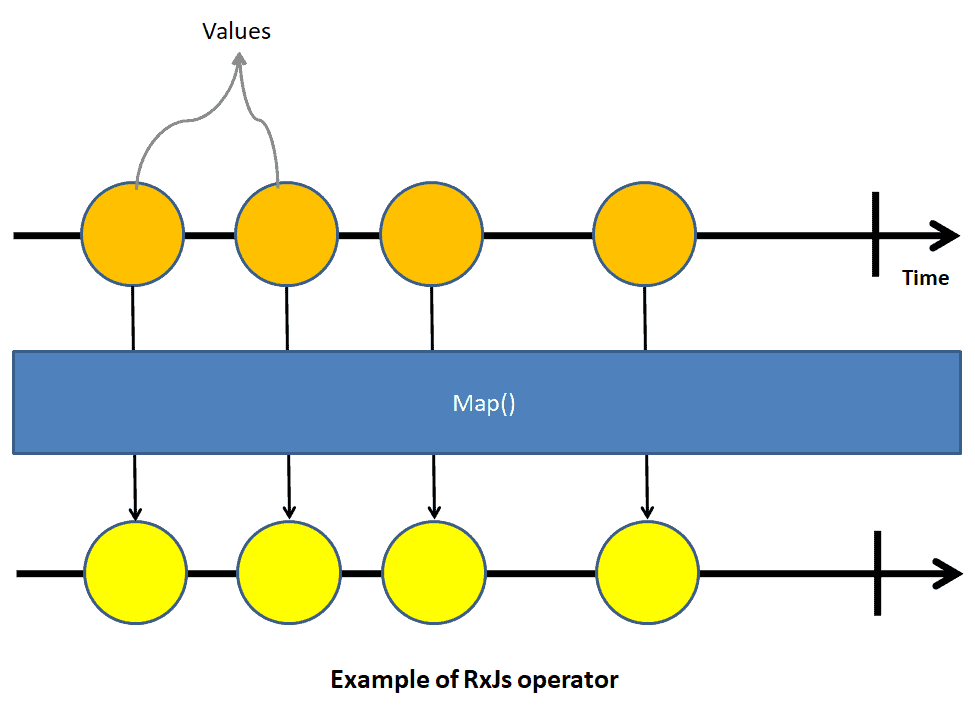
## RxJs Operators

The operators are very important components of the Rxjs library. They are functions that take an observable as input and transform it into a new observable and return it. We use them to manipulate the observable data stream.

For Example.

Map operator applies a given project function to each value emitted by the source Observable and emits the resulting values as an Observable.

Filter operator filter items from the source observable based on some condition and returns the filtered value as a new observable

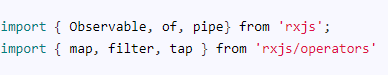


## Using pipe to combine operators

The pipe method accepts operators such as filter, map, as arguments. Each argument must be separated by a comma. The order of the operators is important because when a user subscribes to an observable, the pipe executes the operators in a sequence in which they are added.

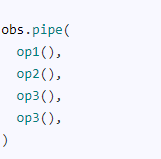
There are two ways we can use the pipe. One as an instance of observable and the other way is to use if as standalone method

To use observable we need it to import from the rxjs library. If you are intend to use the pipe standalone function, then you also need to import it as well. All the operators are available in the library rxjs/operators.

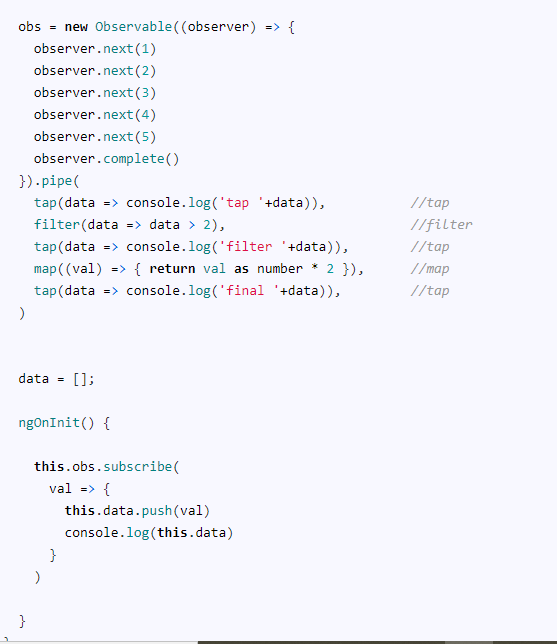


### Pipe as an instance method

The pipe as an instance method is used as below. We the operators op1, op2 etc are passed as the argument to pipe method. The output of op1 method becomes input of the op2 operator and so forth.



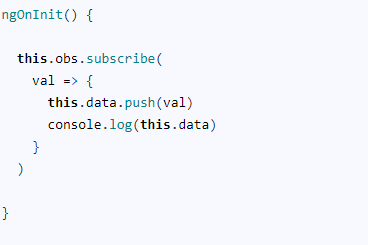




### Pipe as stand alone method

We can also use the pipe as a standalone function to compose operators and re use the pipe at other places.







# **Using Map operator in Angular**

The [Angular observable](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/) Map operator takes an **observable source** as input. It applies a project function to each of the values emitted by the source [observable](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/) and transforms it into a new value. It then emits the new value to the subscribers.

map<T, R>(project: (value: T, index: number) => R, thisArg?: any): OperatorFunction<T, R>

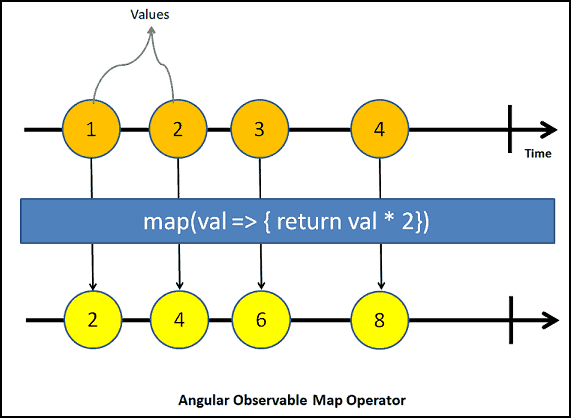
project: is a function that we use to manipulate the values emitted by the source observable. The project can accept two arguments. one is value i.e. the value emitted by the observable. The second argument is index number. The index number starts from 0 for the first value emitted and incremented by one for every subsequent value emitted. It is similar to the index of an array.

thisArg: is optional and default is [undefined](https://www.tektutorialshub.com/typescript/typescript-null-undefined-strict-null-checks/).It defines what this is in the project function.

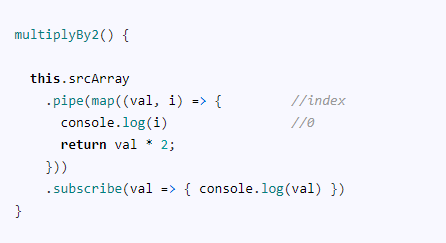
## Using Observable Map

  The output is 2,4,6,8

The following image explains how values from the source observable ( i.e.1,2,3,4 ) go through the map which transforms it into new values by multiplying it by 2.

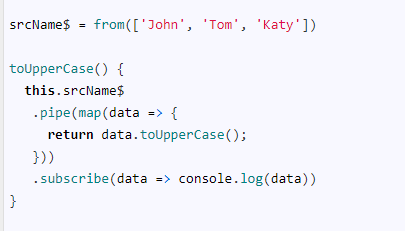


You can also access the second argument index as shown below. It starts as 0 for the first value and gets incremented for every subsequent value



## Map Examples

### Convert input to upper case





### Using Map with [HTTP](https://www.tektutorialshub.com/angular/angular-httpclient/) Request

The following code gets the list of dogs breeds from the https://dog.ceo/api/breeds/list/all API and uses the [keyValue](https://www.tektutorialshub.com/angular/angular-keyvalue-pipe/) pipe to transform the object into an array of key-value pairs.



### Using with event

|  |
| --- |
| You can [create observable from event](https://www.tektutorialshub.com/angular/create-observable-from-event-using-fromevent-in-angular/) and use the map to transform the values.     map with filter  Multiple map The following examples shows use of multiple map functions. The first map adds 10, while the second mad multiplies by 2.   **Tap operator in Angular observable** The Angular Tap RxJs operator returns an observable that is identical to the source. It **does not modify** the stream in any way. Tap operator is useful for logging the value, debugging the stream for the correct values, or perform any other side effects.   Tap Operator Example In the following example, [we create an observable using the of operator](https://www.tektutorialshub.com/angular/rxjs-observable-using-create-of-from-in-angular/#of-operator). We use the [pipe](https://www.tektutorialshub.com/angular/angular-observable-pipe/) to chain the tap operator, which just logs the values of the source observable into the console.    if we simply pass the console.log function to the tap operator and the results will be same. |

|  |  |
| --- | --- |
| Debugging the Observable One of the use cases for the tap operator is using it to debug the Observable for the correct values.  The [map operator](https://www.tektutorialshub.com/angular/angular-observable-map-operator/) in the following example, adds 5 to the source observable. To debug it, we can add the two tap operators. One before and one after it and inspect the values.   Error & Complete callbacks We can also use the tap operator to log the error and complete callbacks as shown in the example below.     **Using ThrowError in Angular Observable** Angular ThrowError operator returns an observable, which on subscription immediately errors out. It does not emit any results. ThrowError ThrowError creates a new observable. Hence we must subscribe to it. The following example creates an ThrowError observable and then subscribes to it.    First, we create an observable using throwError. The first argument to the throwError is the error object. This error object is passed to the consumers when it raises the error notification. We, [subscribe](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/#subscribing-to-the-observable) to it in the [ngOnInit](https://www.tektutorialshub.com/angular/angular-ngoninit-and-ngondestroy/) method. The observable immediately raises the error notification and completes. The error callback is invoked and we will see the error message in the console window. Throw Error Vs ThrowError It is very easy confuse between the Throw Error With ThrowError.  Throw Error throws an error. It is a JavaScript construct and is not part of the RxJs. We need to use the try/catch block to catch the errors thrown from the Throw Error. The RxJS uses the try/catch block to catch any errors thrown from the observables. And when they catch one, **they emit an error notification (raises the error callback), and then the observable stops**.  ThrowError does not throw errors like throw Error. It returns a new observable, which **emit an error notification (raises the error callback), and then stops**. Throw Error Example     The observable emits values 2 & 4.  When map operators receive the value A it uses throw Error to throw an error. The observable catches this error and raises the error notification and terminates.  The last value 8 is never emitted. ThrowError Now, let us replace the throw Error with return throwError      The observable emits values 2 & 4.  When the[map operator](https://www.tektutorialshub.com/angular/angular-observable-map-operator/) receive the value A it returns throwError. Remember throwError returns an observable. It will raise the error notification, only if you subscribe to it.  The map operator does not subscribe to the observable. It just returns it to the subscriber.  Hence the subscriber receives the throwError observable as value. Hence you see [object Object] in the console.  Since there is no error raised, the observable continues and emits the next value 8 and then completes. Using ThrowError The throwError needs to be subscribed for it to emit error notification. We can use it to compose with other Observables such as [mergeMap](https://www.tektutorialshub.com/angular/using-mergemap-in-angular/),  [switchMap](https://www.tektutorialshub.com/angular/using-switchmap-in-angular/), [catchError](https://www.tektutorialshub.com/angular/angular-catcherror/) etc. Using with [catchError](https://www.tektutorialshub.com/angular/angular-catcherror/)     The code throws the error using throw error in [map operator](https://www.tektutorialshub.com/angular/angular-observable-map-operator/).  [CatchError](https://www.tektutorialshub.com/angular/angular-catcherror/) will catch this error. We use the [CatchError](https://www.tektutorialshub.com/angular/angular-catcherror/) to handle the errors thrown by the [Angular Observable](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/). Once we handle the error, we must return an observable. We can either return a replacement observable or return an error. The observable returned from CatchError is immediately subscribed.  Hence we can use the throwError here, which is immediately subscribed , which in turn emits an error notification   Using it with [MergeMap](https://www.tektutorialshub.com/angular/using-mergemap-in-angular/) The [Angular MergeMap](https://www.tektutorialshub.com/angular/using-mergemap-in-angular/) maps each value from the source observable into an inner observable, subscribes to it, and then starts emitting the values from it.  In the following example, we use throwError to return a observable, when we receive the value **3**. The [MergeMap](https://www.tektutorialshub.com/angular/using-mergemap-in-angular/)subscribes to this new observable and raises the error notification and stops.     **Using Catcherror Operator in Angular Observable** Angular CatchError is an RxJs Operator. We can use it to handle the errors thrown by the [Angular Observable](https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/). Like all other RxJs operators, the CatchError also takes an observable as input and returns an observable (or throws an error). We can use CatchError to provide a replacement observable or throw a user-defined error.  Catch operator was renamed as catchError in RxJs 5.5, Hence if you are using Angular 5 or prior version then use catch instead of catchError. Handling Errors in Observable We can handle the errors at two places.   1. Using the error callback of the subscribe method 2. Catch errors in the observable stream  Using Error Callback of Subscribe method We subscribe to an Observable by using the subscribe method. The subscribe method accepts three callback methods as arguments. They are the next value, error,  or complete event. We use the error callback to catch & handle the errors.  For Example, consider the following code. The obs observable multiplies the values (srcArray) by 2 using the [map](https://www.tektutorialshub.com/angular/angular-observable-map-operator/) operator. If the result is [NaN](https://www.tektutorialshub.com/typescript/nan-in-typescript/), then we throw an error using throw new Error("Result is NaN").      We subscribe and start to receive the values from the obs observable in the [ngOnInit](https://www.tektutorialshub.com/angular/angular-ngoninit-and-ngondestroy/) method. When the observable stream throws an error, it invokes the error callback. In the error callback, we decide what to do with the error.  Note that once the observable errors out it will not emit any values neither it calls the complete callback. Our subscription method will never receive the final value of 8. Catch errors in the observable stream Another option to catch errors is to use the CatchError Operator. The CatchError Operators catches the error in the observable stream as and when the error happens. This allows us to retry the failed observable or use a replacement observable. Using CatchError Operator  |  | | --- | | To use CatchError operator, we need to import it from the rxjs/operators as shown below |    Syntax The catchError is a [pipeable operator](https://www.tektutorialshub.com/angular/angular-observable-pipe/). We can use it in a [Pipe method](https://www.tektutorialshub.com/angular/angular-observable-pipe/) similar to the other operators like [Map](https://www.tektutorialshub.com/angular/angular-observable-map-operator/), etc.  The catchError operator gets two argument.  The first argument is err, which is the error object that was caught.  The second argument is caught, which is the source observable. We can return it back effectively retrying the observable.  The catchError ***must return a new observable*** or it can throw an error. Returning a new observable The following examples shows the use of catchError operator.    In the code above, the map emits the values 2 & 4, which is input to the catchError. Since there are no errors, catchError forwards it to the output. Hence the subscribers receive values 2 & 4.  The catchError comes into play, when the [map operator](https://www.tektutorialshub.com/angular/angular-observable-map-operator/) throws an error. The catchError handle the error and must return a new observable (or throw an error). In the example above we return a new observable i.e. of(0). You can also emit any observable for example return from(['A','B','C']) etc  **You can also return the original observable. Just use the return this.obs; instead of return of(0);. But beware, It will result in an**[**infinite loop**](https://stackblitz.com/edit/angular-catcherror-3-infinite-loop)**.**  The new observable is automatically subscribed and the subscriber gets the value 0. The new observable now finishes and emits the complete event.  Since the original observable ended in a error, it will never emit the the value 8. Throws a new Error catchError can also throw an error. In the following example, we use the throw new Error(error) to throw a JavaScript error. This error will propagate to the subscriber as shown in the example below.    We can also make use of throwError to return an observable. Remember that the throwError does not throw an error like throw new Error but returns an observable, which emits an error immediately.   Retrying You can also retry the observable using the Retry operator.    The catchError gets the source observable as the second argument. If we return it, it will get subscribed again effectively retrying the observable.  Ensure that you keep track of no of tries so that you can stop the observable after a few failed attempts. Otherwise, you may run into an infinite loop if the observable always emits an error.   **Filter Operator in Angular Observable** The Filter Operator in Angular filters the items emitted by the source Observable by using a condition (predicate). It emits only those values, which satisfies the condition and ignores the rest. Filter in Angular Filter is the simplest and most used RxJs Operator in Angular. The Filter Operator takes 2 arguments.  **Syntax**  filter<T>(predicate: (value: T, index: number) => boolean, thisArg?: any): MonoTypeOperatorFunction<T>  The first argument is the predicate function. This function is evaluated against each value of the source observable. Filter emits only those values which satisfies the the predicate. The predicate function takes 2 parameters. The first one is the value emitted by the source. The second argument is zero based index.  The second argument thisArg is Optional. It determines the value of this in the predicate function Filter Example The following example filter function returns true only if the value is an even number. |