npm install –g rxjs

import { Component, OnInit } from '@angular/core';

import {Observable} from 'rxjs';

@Component({

    selector: 'app-root’,

    templateUrl: './app.component.html',

    styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

    constructor() { }

    ngOnInit() {

        const sqnc = new Observable(countOnetoTen);

  sqnc.subscribe({

            next(num) { console.log(num); }

        });

        function countOnetoTen(observer) {

            for(var i = 1; i <= 10; i++) {

                observer.next(i);

            }

            return {unsubscribe(){}};

        }

    }

}

Observer : Any object that wishes to be notified when the state of another object changes.

Observable : **Any object whose state may be of interest, and in whom another object may register an interest**.

**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.

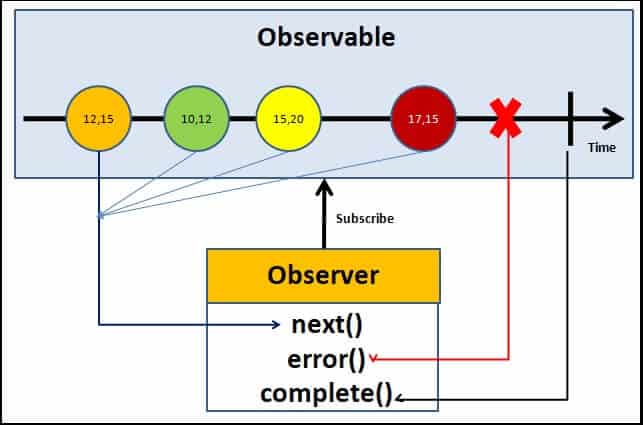
Observables are declarative. You define an observable function just like any other variable. The observable starts to emit values only when **someone subscribes to it**.

**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()

Angular Observable Tutorial how observable and observers communicates with callbacks

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.

Create a new project in angular. Remove the contents from app.component.html. Open the app.component.ts

**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

|  |  |
| --- | --- |
| 1  2  3 | import { Observable } from 'rxjs'; |

**Observable Creation**

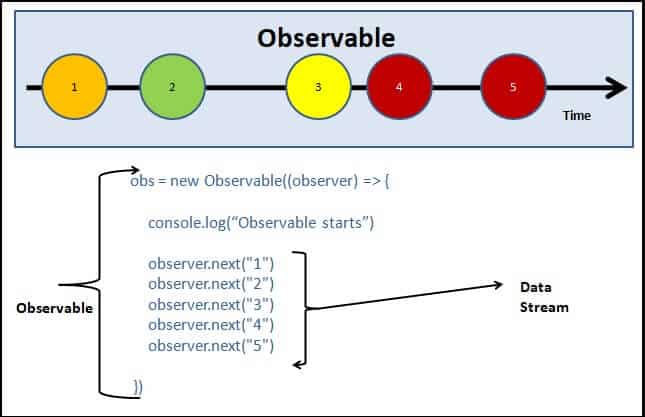
There are few ways in which you can create observable in angular. Simplest is to use the Observable constructor. The observable constructor takes observer (or subscriber) as its argument. The subscriber will run when this observable’s subscribe() method executes.

The following example creates an observable of a stream of numbers 1, 2, 3, 4, 5

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | obs = new Observable((observer) => {       console.log(“Observable starts”)       observer.next("1")       observer.next("2")       observer.next("3")       observer.next("4")       observer.next("5")     }) |

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.

Creating observable in Angular Observable Tutorial app

In the above example, we used the Observable Constructor to create the Observable. 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() as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ngOnInit() {        this.obs.subscribe(        val => { console.log(val) }, //next callback        error => { console.log("error") }, //error callback        () => { console.log("Completed") } //complete callback      )  } |

The complete app.component.ts code is as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import { Component, OnInit } from '@angular/core';  import { Observable } from 'rxjs';   @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent implements OnInit {    title = 'Angular Observable using RxJs - Getting Started';     obs = new Observable((observer) => {      console.log("Observable starts")        observer.next("1")        observer.next("2")        observer.next("3")        observer.next("4")        observer.next("5")    })        ngOnInit() {        this.obs.subscribe(        val=> { console.log(val) },        error => { console.log("error")},        () => {console.log("Completed")}      )    }  } |

Now, run the code and watch the debug window.

**Adding interval**

We can add a timeout to insert a delay in each next() callback

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | 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.next("4") }, 4000);      setTimeout(() => { observer.next("5") }, 5000);      }) |

1

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

title = 'Angular Observable using RxJs - Getting Started';

obs = new Observable((observer) => {

console.log("Observable starts")

observer.next("1")

observer.next("2")

observer.next("3")

observer.next("4")

observer.next("5")

})

data=[];

ngOnInit() {

this.obs.subscribe(

val=> { console.log(val) },

error => { console.log("error")},

() => {console.log("Completed")}

)

}

}

2.

import { Component, OnInit } from '@angular/core';

import { Observable, of} from 'rxjs';

import { map, filter, tap } from 'rxjs/operators'

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

obs = new Observable((observer) => {

observer.next(1)

observer.next(2)

observer.next(3)

observer.next(4)

observer.next(5)

observer.complete()

}).pipe(

filter(data => data > 2), //filter Operator

map((val) => {return val as number \* 2}), //map operator

)

data = [];

ngOnInit() {

this.obs1.subscribe(

val => {

console.log(this.data)

}

)

}

}

3.

import { Component } from "@angular/core";

import { filter } from "rxjs/operators";

import { interval, of, timer } from "rxjs";

@Component({

  selector: "my-app",

  template: `

    <h1>Filter Example</h1>

  `,

  styleUrls: ["./app.component.css"]

})

export class AppComponent {

  ngOnInit() {

    of(1,2,3,4,5,6,7,8,9,10)

      .pipe(

        filter(val => {

          return val %2==0;

        }),

      )

      .subscribe(val => console.log(val));

  }

}

4.

import { Component } from "@angular/core";

import { filter } from "rxjs/operators";

import { interval, of, timer } from "rxjs";

import { Subject } from "rxjs";

@Component({

selector: "app-root",

template: `<h1>Filter Example</h1>`,

styleUrls: ["./app.component.css"]

})

export class AppComponent {

ngOnInit() {

of(1,2,3,4,5,6,7,8,9,10)

.pipe(

filter(val => {

return val %2==0;

}),

)

.subscribe(val => console.log(val));

//2

const subject = new Subject();

//First Observer

subject.subscribe({

next: (data) => console.log('First observer prints '+ data)

});

subject.next(1);

//Second Observer

subject.subscribe({

next: (data) => console.log('Second observer prints '+ data)

});

subject.next(34);

subject.next(14);

}

}