***RxJS*** stands for **Reactive Extension for Javascript.**It is a javascript implementation of **ReactiveX** (an API for asynchronous programming with observable streams).

**What is RxJS?**

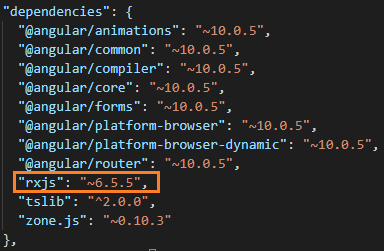
**Reactive programming** is an asynchronous programming paradigm that deals with streams of data and change propagation.

***"RxJS****is a****javascript library****that uses****observables****to work with reactive programming that deals with****asynchronous data calls, callbacks****and****event based programs."***

RxJS has **revolutionised**the way modern developers think about building apps. In RxJS ***"Everything is a stream"***. RxJS also provides many utility functions for different purposes such as:

* To convert existing code for asynchronous operations into Observables
* To Iterate over the values in a data stream
* Mapping Values of different types
* Filtering data from a stream
* Composing two or more data streams together

RxJS has become one of the most popular libraries to handle async operations and hence angular too prefers to use RxJS. In any new angular app, RxJS library comes preinstalled. The same can be seen in the **package.json**of an angular app as well:



We can use RxJS in any project involving a lot async task handling.

# ****RxJS Features****

The key features available in RxJS to handle async operations are as follows:

**1. Observables:-**An Observable is a ***'collection that arrives over time'***. Observable produces data which a subscriber can subscribe to and then use it. An observable object can be used to represent asynchronous requests.

**2. Observer:-**The subscriber of an observables that receives the data from Observable stream is known as Observer. Basically, It's a collection of callbacks that knows how to listen to values delivered by the **Observable**.

**3. Subscription:-**The process of an Observer listening to an observable for data stream is known as **Subscription.**It mainly represents the execution of an observable and can also be used to cancel the subscription.

**4. Operators:-**Operators are pure functions that take an observable as an input and also returns an observable. Some of the commonly used operators are map(), filter(), concat(), reduce(), etc...

**5. Subject:-**A**Subject**is an Observable which is capable of multicasting i.e. it can talk to multiple observers.

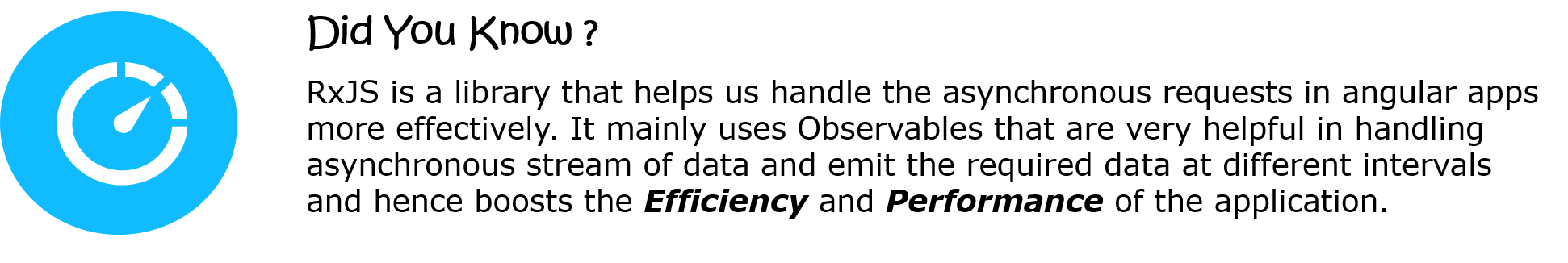
**6. Schedulers:- Schedulers**are a way to control the timing strategy used to execute tasks. It also helps to control concurrency and allows us to co-ordinate when the computation happens.

# RxJS Advantages

**1.**It is supported by Javascript as well as Typescript and can be used with any javascript based library or framework such as angular, react, vuejs.etc...

**2.**It very nicely handles the async operations by the help of observables and reactive programming together.

**3.**It provides a huge collection of operators for mathematical, transformation, filtering, error handling and many more such tasks.

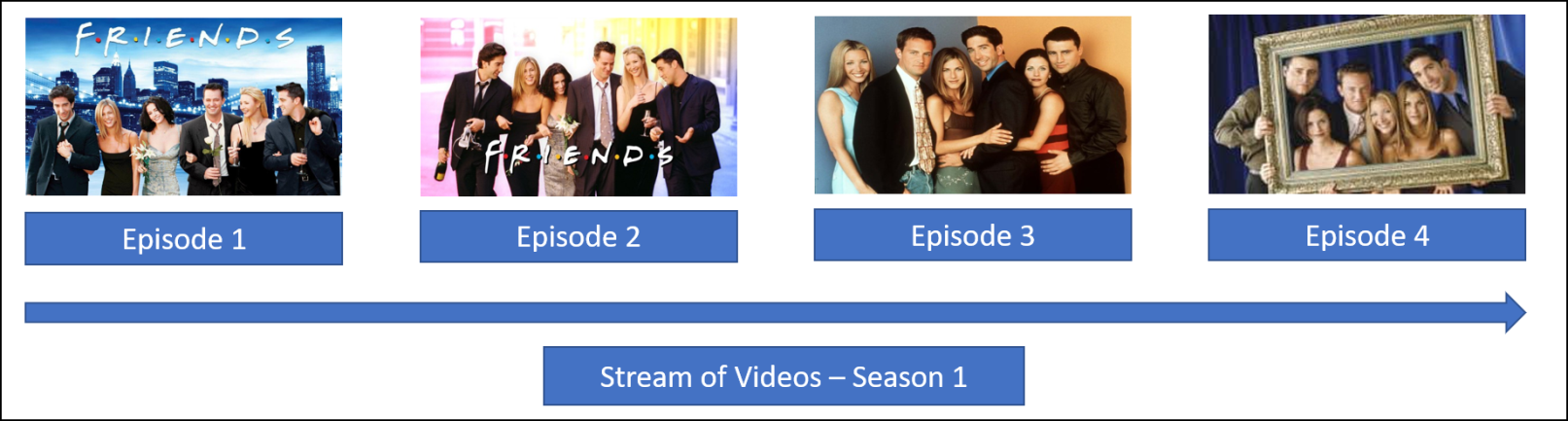


We already saw the various features that RxJS provides such as Observables, Observers and Subscription. Lets learn more about them and see how do they all work together.

As we know, **Observable** is a wrapper around some asynchronous data source (a stream of data sources that keeps on emitting data one after other continuously). **Observers**are there to execute some block of code whenever some data is received from the observable or when some error happens or when Observable reports that the data stream is over or completed. The **Observer**will execute all these blocks of code only when we **subscribe**to the Observable.

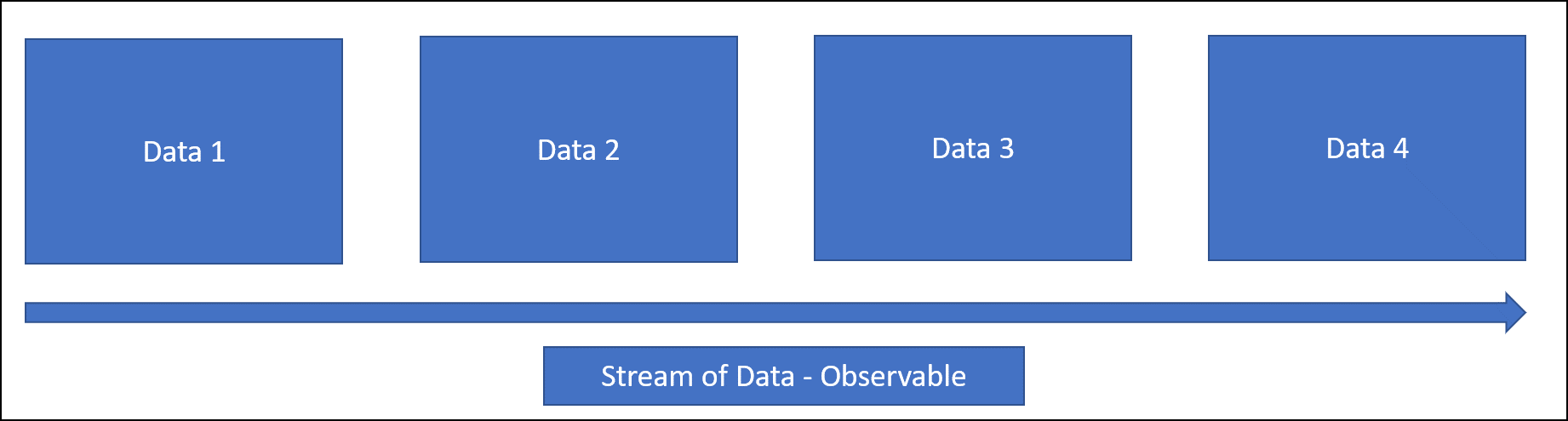
Lets understand this with an example,

Consider any of your favourite **Web Series, like "FRIENDS".** Now, ***FRIENDS***, does not show everything in a single video, it is divided across multiple videos which are grouped together as a Season.



Also, When we watch a season of FRIENDS, it again does not play all the videos together, rather we watch the stream of videos one after the other in a sequence.

Like a season, which is a stream of multiple videos, ***Observable***is a stream of data as shown:



The episodes of a Season are played in an order only when we start playing the videos. Likewise the data is emitted from the Observable only when we subscribe to it. When an observable is subscribed, the use of data emitted is controlled by the Observer.

Next we will learn more about Observers.

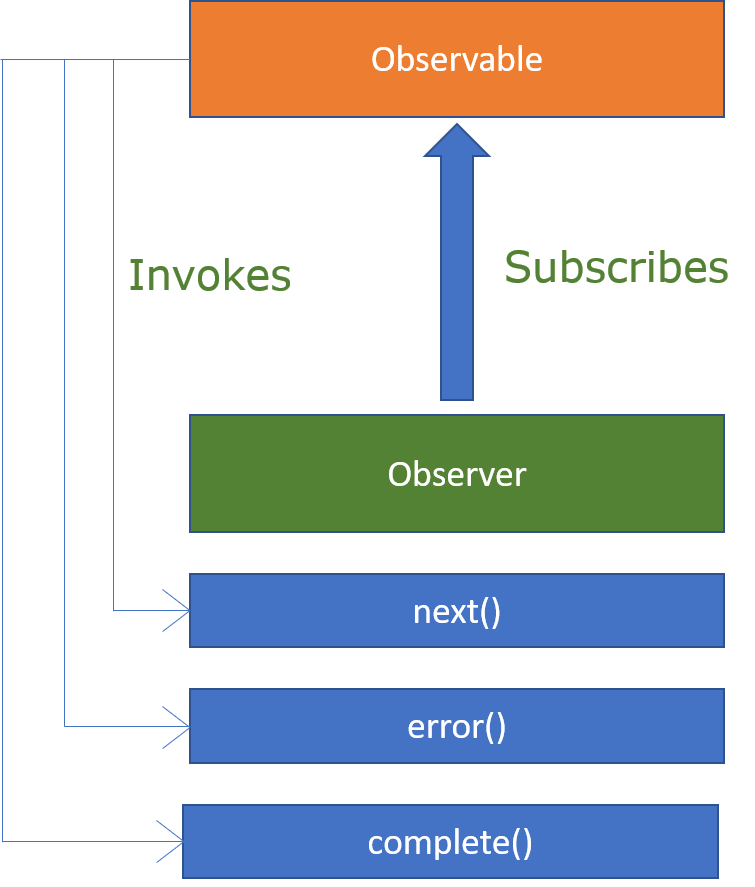
As we know, An Observer is a consumer of the data emitted by the Observable. The Observers are able to use these emitted data only after subscription.

In RxJS, The observers are a set of callback functions that help us handle the different types of values that an Observable emits. It includes:

* **next:-**It sends any value such as numbers, arrays or objects when subscribed
* **error:-**It sends a javascript error object when subscribed
* **complete:-**It does not send any value, rather it marks the completion of the data stream. No further values can be sent after complete function has been called

**next()** method is the most commonly invoked observer method as they actually deliver the data to its subscribers. During the execution of an Observable, **next()** method can be called any number of times but the execution stops and no more data can be emitted from the Observable as soon as**error()**or **complete()**method is invoked.

The entire data flow while using an observable happens as shown:



**Displosing/Unsubscribing observables:-**

An Observable is capable to run infinite times and emits values, but we may not want that to happen always. At certain point of time, we might need to stop emitting values otherwise it would lead to unnecessary waste of memory and computing power. We can prevent such issues by unsubscribing to an observable using the **unsubscribe()** method.

Next, We will set up rxjs environment and see examples on Observables.

To use RxJS library we must first install it.

1. Create a folder**rxjs-demo** and setup the project using the command in rxjs-demo folder:

1. npm init --yes

2. **Install *rxjs***locally in rxjs-demo folder using the command:

1. npm install rxjs --save

3. Confirm the installation by checking the **package.json** file having the dependency rxjs added to it.

https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_auth_013036379973525504115_shared/web-hosted/assets/packagerxjs.PNG

Next, We will see how to create and use Observables using a demo.

**Problem Statement :-**Create an **Observable**, use the observer callbacks **next(), error()**and **complete()**and**subscribe** to the created Observable.

Observables comes along with the **rxjs**library, so we must import it to use Observables:

1. import { Observable } from 'rxjs';

**Creating an Observable:**

To create an observable, create an instance of the Observable class imported from rxjs in a typescript file (**.ts** extension):

1. const observable = new Observable();

The Observable instance will take a callback function as a parameter. The parameter to the callback function is the **observer.**The observer has 3 methods **next(), error()**and **complete()** which can be used to pass a value, error or to complete the Observable.

1. const observable = new Observable(observer => {
2. observer.next(1);
3. observer.next(2);
4. observer.complete();
5. });

The created Observable will emit values 1 and 2 and then end emitting values, when subscribed.

**Subscribing an Observable:-**

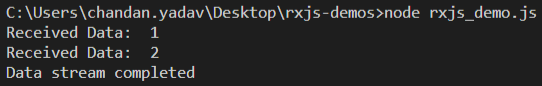
The created observable will not emit any values, until we subscribe to it. To subscribe an Observable, we use the method **subscribe().**The **subscribe()** method takes ***3 callbacks*** as parameters. The first callback is to handle the **data**emitted by the Observable, second callback is to handle the error thrown by the Observable where as the third one executes when the Observable completes emitting data stream.

1. observable.subscribe(
2. data => console.log("Recieved Data: ", data),
3. error => console.log("Error Occured: ", error),
4. () => console.log("Data stream completed"),
5. )

The Complete code for the created observable and its subscription is below:

1. import { Observable } from 'rxjs';
2. const observable = new Observable(observer => {
3. observer.next(1);
4. observer.next(2);
5. observer.complete();
6. });
7. observable.subscribe(
8. data => console.log("Received data: ", data),
9. error => console.log("Error occurred: ", error),
10. () => console.log("Data stream completed"),
11. )

Transpile the typescript code to javascript and execute the javascript file to get the output:

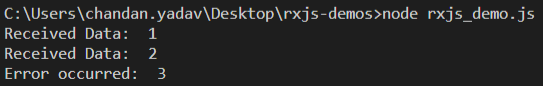


Next, lets see more on Observables with another demo.

**Problem Statement :-** Create an Observable, use the observer callbacks **next(), error()**and **complete()**and **subscribe**to the created Observable.

1. import { Observable } from 'rxjs';
2. const observable = new Observable(observer => {
3. observer.next(1);
4. observer.next(2);
5. observer.error(3);
6. observer.complete();
7. });
8. observable.subscribe(
9. data => console.log("Received Data: ", data),
10. error => console.log("Error occurred: ", error),
11. () => console.log("Data stream completed"),
12. )

Execute the code and observe the output:



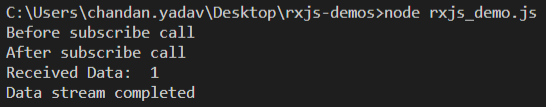
In this case, it did not display **"Data stream completed",**because whenever either of **error()**or **complete()**method gets executed, the Observable stops executing there.

Next, lets see more on Observables with asynchronous block of code.

**Problem Statement :-** Create an **Observable**, use the observer callbacks **next(), error()**and **complete()** with **setTimeout.**

1. import { Observable } from 'rxjs';
2. const observable = new Observable(observer => {
3. setTimeout(() => {
4. observer.next(1);
5. }, 1000);
6. setTimeout(() => {
7. observer.next(2);
8. }, 2000);
9. setTimeout(() => {
10. observer.complete();
11. }, 1000);
12. });
13. console.log('Before subscribe call');
14. observable.subscribe(
15. data => console.log("Received Data: ", data),
16. error => console.log("Error occurred: ", error),
17. () => console.log("Data stream completed"),
18. )
19. console.log('After subscribe call');

Execute the code and observe the output:



In this case, the value 2 is not displayed in console as it was supposed to be emitted after 2 seconds whereas just after 1 second, the complete method was already invoked.

We have seen three different methods i.e. **error(), complete()**and **unsubscribe()**which when executed, stop the Observable from emitting values further. If all do the same work, then why 3 different methods? Lets try to understand the difference between them with an example.

Consider you go to a restaurant and order for unlimited food buffet, Once you start getting food, it can stop in three possible ways, either the restaurant can say the stock of food is over, or it suddenly gives bad-tasting food or your stomach is full and you cannot have anything more.

On the same Lines, The Observables are like the restaurant which serves infinite amount of data (as in a buffet).

**error()**and **complete()** methods are observer methods. **error()** method is invoked to return an error and stop emitting values. **complete()** method does not return any value and signifies no more data is left for emitting from the observable.

**unsubscribe()**method can be invoked by the user to stop receiving any values further from the observable. When unsubscribe is invoked,**complete()** or **error()** may not be invoked. But when **error()**or **complete()**is invoked, it automatically unsubscribes to the observable.