Observables, Subjects,

The foundation of Angular is built upon the RxJS library. You may not need extensive knowledge of this library to write an Angular App, but understanding some key features will make your life a lot easier. The three items which you will come across in your Angular application are Subjects, BehaviorSubjects, and Observables. It is imperative to understand their uses as you begin to learn Angular.

## **Observables**

The Observable is the [core type](https://rxjs.dev/guide/overview) of the RxJS library. Its primary use is to be “listened” to or “observed” for future events. Listening to these events is done via calling the subscribe() function of the observable where you can access the value that is being emitted. There is no way to invoke the event or value change using an observable alone, so you can kind of think of it as a “read-only” type. Due to this, it is usually best practice to expose Observables in cases where you do not want other parts of the application to invoke events, i.e., you just want them to listen for changes.

e.g., the HTTP client uses an Observable as the way to expose the event of an HTTP request.

/\*\*

 \* get function returns an Observable that will emit an event

 \* when the response is received

 \*/

this.httpClient.get<Inventory[]>('https://sample-api/items').subscribe(items => {

  // Do something with response

});

## **Subjects**

Subjects **are a type of Observable.** However, unlike an Observable, Subjects can emit events/values to its subscribers using the next() function. Therefore, you can publish changes (using next()) to a Subject and listen for changes (using subscribe()). You can also cast Subjects to an Observable for instances where you want to conceal the Subject-like behavior and only expose the ability to subscribe to changes.

The way to differentiate a Subject from a BehaviorSubject is:

a) Subjects have no initial value

b) Subscribers will only be notified and receive events/values after the subscription is made – i.e., Subscribers will not receive the last emitted value upon subscription.

e.g., say we want to use Subjects to notify subscribers when an event has occurred:

const subject = new Subject();

subject.next('event 0');

subject.subscribe(event => console.log(event));

subject.next('event 1');

subject.next('event 2');

subject.next('event 3');

/\*\*

 \* Expected output:

 \* event 1

 \* event 2

 \* event 3

 \*/

Since event 0 was emitted before the subscription was made, the subscriber will not receive that value. If the use case for a subject requires the Subject to emit that initial value, a BehaviorSubject would be a better choice.