# **Observable in Angular**

The Observables in Angular, a popular framework and a platform in **Javascript** using which you can build tremendous single-page client-side applications using the bootlegs of [Typescript](https://www.javatpoint.com/typescript-tutorial)

Observable in Angular is a feature that provides support for delivering messages between different parts of your single-page application. This feature is frequently used in Angular because it is responsible for handling **multiple values, asynchronous programming** , and also event handling processes.

However, an observer or observable is a software paradigm constituting the design of an object which is called a subject that maintains a list of various dependencies that are called observers. These dependencies are automatically notified whenever you try to change states.

## Nature of Observable

Observables in Angular are generally declarative i.e., if you ought to define a function for value publication. It won't be executed until and unless it is subscribed. The subscriber is termed as a **consumer** who receives the notifications until the function completes itself or until they manually unsubscribe.

An observable can be used to deliver any type of multiple values be it literal events, messages depending upon the context provided. The receiving and delivering values are synchronously or asynchronously the same. Since the setup and breakdown are handled by the observable, you don't need to worry about your application code being subscribed to consumer values or unsubscribed. Be it an HTTP response or a timer of intervals, the interface for listening to events and stopping them will be the same. This adds an advantage of making use of observables frequently for development purposes.

## Usage

The basic usage of Observable in Angular is to create an instance to define a **subscriber function**. Whenever a consumer wants to execute the function the **subscribe()** method is called. This function defines how to obtain messages and values to be published.

To make use of the observable, all you need to do is to begin by creating notifications using subscribe() method, and this is done by passing observer as discussed previously. The notifications are generally Javascript objects that handle all the received notifications. Also, the unsubscribe() method comes along with subscribing () method so that you can stop receiving notifications at any point in time.

Consider the below example constituting geolocation updates whenever a subscriber subscribes.

### **Creating an observable**

1. const locations = new Observable((observer) =**>**
2. {
3. let watchId: number;

### **Making API integration to check values**

1. if ('geolocation' in navigator) {
2. watchId = navigator.geolocation.watchPosition((position: Position) =**>** {
3. observer.next(position);
4. }, (error: PositionError) =**>** {
5. observer.error(error);
6. });
7. } else {
8. observer.error('Geolocation not available'); }

Data is ready for cleanup for next subscription using the below command.

1. return {
2. unsubscribe() {
3. navigator.geolocation.clearWatch(watchId);
4. }
5. };
6. });

Now, use subscribe() function for getting continuous updates using the below code:

1. const locationslocationsSubscription = locations.subscribe({
2. next(position) {
3. console.log('Current Position: ', position);
4. },
5. error(msg) {
6. console.log('Error Getting Location: ', msg);
7. }
8. });

Set the time intervals for 5 seconds which the notifications will be stopped automatically through the help of API integration. Consider the below code for reference.

1. setTimeout(() =**>**
2. {
3. locationsSubscription.unsubscribe();
4. },
5. 5000;

## Types of Notifications and Description

1. **Next**: It is called after the execution starts for zero times or more than that. It is a mandatory notification for catching each value delivered.
2. **Error**: This is an optional handler for chasing error notifications. This notification halts execution while running instances of observable.
3. **Complete**: It is an optional handler that notifies completion of executions. The currently delayed values continue to be delivered even when the execution is marked as complete.