1. Reactive forms
2. Dependency injection
3. Pipe, directive, decorators
4. RxJS operators
5. Observable vs Subject vs BehaviorSubject:

**Think of a Subject like a radio station.**

* Imagine a radio station that broadcasts music. If you start listening to the station right now, you will hear whatever song is currently playing. But you won't hear the songs that played before you started listening. You only hear what happens from the moment you tune in.
* In technical terms, Subject is used when you want to broadcast values (data) to many places (subscribers). It doesn't store the past values; it only "sends out" data from the moment you start listening.

In Angular, Subject and BehaviorSubject are special types of Observables provided by the RxJS library. They help in managing data flow and state changes, especially in communication between components. Let's explore each one in simple terms.

**1. Subject**

* **What it is:** A Subject is like an Observable, but it can emit values to its subscribers directly. It's both an Observable and an Observer, meaning it can be used to create new data streams and subscribe to them.
* **Use Case:** It is used when you want to share a stream of data (events) among multiple components. For example, broadcasting a message to multiple components or emitting an event whenever a button is clicked.
* **Behavior:** When a new subscriber subscribes to a Subject, it doesn't get any previously emitted values; it only receives new emissions after it subscribes.

import { Subject } from 'rxjs';

const subject = new Subject<number>();

// Subscriber 1

subject.subscribe(value => console.log('Subscriber 1:', value));

// Emitting values

subject.next(1); // Output: Subscriber 1: 1

subject.next(2); // Output: Subscriber 1: 2

// Subscriber 2

subject.subscribe(value => console.log('Subscriber 2:', value));

subject.next(3); // Output: Subscriber 1: 3, Subscriber 2: 3

2. BehaviorSubject

What it is: A BehaviorSubject is a special type of Subject that requires an initial value and emits its current value to new subscribers. When a new subscriber subscribes to a BehaviorSubject, it receives the last emitted value (the "current state") immediately.

Use Case: It's helpful for state management or situations where you need to have an initial value and ensure that new subscribers receive the latest state. For example, managing user authentication status or current settings.

Behavior: A BehaviorSubject always holds a value. When a new subscriber joins, it gets the latest value immediately, even if it was emitted before the subscriber joined.

import { BehaviorSubject } from 'rxjs';

const behaviorSubject = new BehaviorSubject<number>(0); // Initial value is 0

// Subscriber 1

behaviorSubject.subscribe(value => console.log('Subscriber 1:', value));

// Emitting values

behaviorSubject.next(1); // Output: Subscriber 1: 1

behaviorSubject.next(2); // Output: Subscriber 1: 2

// Subscriber 2

behaviorSubject.subscribe(value => console.log('Subscriber 2:', value));

behaviorSubject.next(3); // Output: Subscriber 1: 3, Subscriber 2: 3

**Key Differences Between Subject and BehaviorSubject**

* **Initial Value:** Subject doesn't have an initial value, while BehaviorSubject requires one.
* **Subscription Behavior:** When a new subscriber joins a Subject, it only gets new values emitted after subscription. A BehaviorSubject, however, will immediately emit the last emitted value to any new subscriber.
* **Use Case Suitability:** Use Subject for event streams (clicks, notifications). Use BehaviorSubject for state management where you want to provide the current state to new subscribers.

1. Parent-child data sharing
2. Sibling data sharing
3. Promise vs Observable
4. Hoisting
5. Closures
6. var, let, const
7. Output questions
8. Object reference
9. What are Promises in JS, and how do they work?  
   What is async and await in JS?  
   How does setTimeout() work in a single-threaded environment?  
   Explain currying in JS.  
   Difference between arrow functions and normal functions.