**More RxJS Interview Questions and Answers (Simplified)**

**6. What is an Observer?**

**Answer:**  
An Observer is the "listener" for an Observable. It's an object containing callbacks that know how to handle the values, errors, and completion signals sent by the Observable.

It can have three methods:

* next(value): Called for each value emitted.
* error(err): Called if something goes wrong.
* complete(): Called when the Observable has no more values to emit.

**In simple words:** The Observer is the set of instructions you give to .subscribe() telling it what to do with the data.

**7. What is a Subscription and why is it important to unsubscribe?**

**Answer:**  
A **Subscription** is an object that represents the execution of an Observable. When you call .subscribe(), you get a Subscription object back. Its most important job is to let you clean up and stop the execution by calling its .unsubscribe() method.

**Why is it important to unsubscribe?**  
To prevent **memory leaks**. If you don't unsubscribe, long-lived Observables (like timers or DOM event listeners) will continue running in the background and emitting values even after the component that created them is destroyed. This consumes memory and can cause unexpected bugs.

**In simple words:** Subscribing is like turning on a tap. Unsubscribing is turning it off. If you don't turn it off, it will keep running forever.

**8. Explain the difference between a Cold and a Hot Observable.**

**Answer:**  
This describes how an Observable produces its values for subscribers.

* **Cold Observable:** Starts producing values **only when you subscribe**. Each new subscriber gets their own private, fresh stream of values starting from the very beginning.
  + **Analogy:** Watching a movie on **Netflix**. You press play (.subscribe()), and the movie starts from the beginning, just for you. Another person pressing play gets their own separate movie playback.
* **Hot Observable:** Is already running and producing values, **whether anyone is subscribed or not**. Subscribers tap into the ongoing stream and only receive values that are emitted *after* they subscribe.
  + **Analogy:** Tuning into a **live TV broadcast**. You join mid-stream and see whatever is happening *right now*. You missed what came before.

**9. What is a Subject?**

**Answer:**  
A **Subject** is a special hybrid that acts as both an **Observable** and an **Observer**.

* Like an **Observable**, you can subscribe to it to listen for values.
* Like an **Observer**, you can push new values into it by calling .next(value).

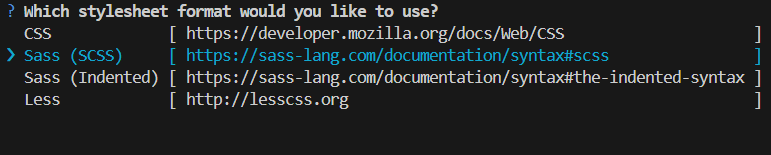
It **multicasts** values, meaning every subscriber receives the exact same value. Subjects are **Hot Observables** and are excellent for sharing a single data source with multiple parts of your application.

**In simple words:** A Subject is like a megaphone. You can speak into it (.next()) and anyone listening (.subscribe()) will hear the same message at the same time.

**10. What are the key differences between a Promise and an Observable?**

**Answer:**  
This is a classic interview question. Here are the main differences in simple terms:

|  |  |  |
| --- | --- | --- |
| Feature | Promise | Observable |
| **Values** | Emits a **single** value. | Emits **multiple** values over time. |
| **Execution** | **Eager**: Runs immediately once created. | **Lazy**: Runs only when subscribed to. |
| **Cancellable** | **No**. You cannot cancel a Promise once it starts. | **Yes**. You can cancel by unsubscribing. |
| **Operators** | Very few (.then, .catch). | A rich library of operators (map, filter, etc.). |





First view content on Oberservable and Promise ( Async and sync method) 