1️⃣What is useRef?

Definition: `useRef` is a React hook that provides a persistent, mutable reference which survives re-renders.

- Unlike `useState`, updating a ref does **NOT** trigger re-render.

Common Uses: - Accessing DOM elements directly - Storing mutable values across renders without causing re-renders - Holding previous state values

Syntax:

```
const ref = useRef(initialValue);
```

- Returns a ref object: `{ current: initialValue }` - Can be updated: `ref.current = newValue`

2️⃣Internal Working

- React stores a ref object in the fiber for the component
- `ref.current` persists across re-renders
- Updating `ref.current` **does not trigger a render**
- Ideal for values to read/write across renders without re-rendering

3️⃣Accessing DOM Elements

Example:

```
import { useRef, useEffect } from "react";

function InputFocus() {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus(); // focus input after mount
  }, []);

  return <input ref={inputRef} />;
}
```

- `ref` attached to JSX element → `current` points to DOM node - Can be used for scrolling, measuring size, animations, focus

---

4️⃣Storing Mutable Values

Example:

```
function Timer() {
  const countRef = useRef(0);

  const handleClick = () => {
    countRef.current += 1;
    console.log(countRef.current); // persists across renders
  };

  return <button onClick={handleClick}>Increment</button>;
}
```

- Unlike `useState`, updating `countRef.current` does not trigger re-render

5️⃣Storing Previous Values

Example:

```
import { useEffect, useRef } from "react";

function PrevCounter({ count }) {
  const prevCountRef = useRef();

  useEffect(() => {
    prevCountRef.current = count;
  }, [count]);

  const prevCount = prevCountRef.current;
  return <div>Previous: {prevCount}, Current: {count}</div>;
}
```

- No re-render triggered - Useful in animation, diffing, logging, side-effect conditions

---

6️⃣Combining with useEffect for Timers / Intervals

Example:

```
function IntervalCounter({ delay }) {
  const countRef = useRef(0);

  useEffect(() => {
    const interval = setInterval(() => {
      countRef.current += 1;
      console.log(countRef.current);
    }, delay);

    return () => clearInterval(interval);
  }, [delay]);
}
```

- Key pattern in dynamic timers - Avoids stale closures without forcing re-render

---

7️⃣Avoiding Re-renders

- Perfect when mutable state **does not trigger render**
- Example: storing previous scroll position, animation frame IDs, timeouts, WebSocket references

  ```
  const timeoutRef = useRef();

  function handleClick() {
    clearTimeout(timeoutRef.current);
    timeoutRef.current = setTimeout(() => console.log("Done"), 1000);
  }
  ```

8️⃣Difference Between useRef and useState

Feature	useRef	useState
Triggers render on update	❌No	✅Yes
Persists across renders	✅Yes	✅Yes
Ideal for	DOM nodes, mutable values, previous values, timers	UI state that affects render

Feature	useRef	useState
Functional updates needed?	No	Yes for prev-dependent state
Access in callbacks	Direct via `.current`	Must pass state value

9️⃣ Advanced / Production Patterns

1️⃣ Tracking previous props / state

```
const prevProp = useRef();
useEffect(() => { prevProp.current = propValue; }, [propValue]);
```

2️⃣ Stable mutable callbacks

```
const callbackRef = useRef();
useEffect(() => { callbackRef.current = someCallback; }, [someCallback]);
useEffect(() => {
  const tick = () => callbackRef.current();
  const id = setInterval(tick, 1000);
  return () => clearInterval(id);
}, []);
```

- Avoids re-subscribing on every render

3️⃣ Integrating with third-party libraries

```
const chartRef = useRef();
useEffect(() => {
  chartRef.current = new ChartLibrary(chartContainer);
}, []);
```

- Storing DOM nodes, animation refs, or chart instances

🔟 Senior-Level Mental Model

- `useRef` = persistent, mutable container
- Updating `.current` **does not trigger render** → ideal for non-UI state
- Common uses:
- DOM access
- Previous values
- Timers / intervals / subscriptions

- Stable callback references
- Works with `useEffect` to avoid stale closures
- Avoid using refs for values that affect render → use `useState`

1️⃣1️⃣Summary Table

Use Case	useRef	useState
DOM element reference	✅	❌
Mutable value across renders	✅	✅(triggers re-render)
Previous value storage	✅	✅(triggers re-render)
Trigger render on update	❌	✅
Async timers / intervals	✅	✅

✅**Key Takeaways:** - `useRef` is not state → changing `.current` does not re-render component - Great for mutable values, DOM nodes, intervals, previous values - Works with `useEffect` to avoid stale closures - Always use state for values that need to render UI