## useState

## useEffect

## useContext

## useReducer

## useRef

The useRef hook in React is a versatile tool that allows you to persist values across renders without triggering a re-render. Here are some common use cases for useRef:

### 1. Accessing DOM Elements

* Use useRef to directly access and manipulate DOM nodes or elements.

**Example:**

import React, { useRef } from "react";

const FocusInput = () => {

const inputRef = useRef();

const focusInput = () => {

inputRef.current.focus(); // Directly focuses the input element

};

return (

<div>

<input ref={inputRef} type="text" />

<button onClick={focusInput}>Focus Input</button>

</div>

);

};

**Use Case:** Setting focus, scrolling, or measuring DOM elements.

### 2. Persisting Mutable Values Across Renders

* Use useRef to persist a value that does not trigger a re-render when updated.

**Example:**

const CounterWithRef = () => {

const countRef = useRef(0);

const increment = () => {

countRef.current += 1; // Updates the value without re-rendering

console.log(countRef.current); // Logs the current count

};

return <button onClick={increment}>Increment</button>;

};

**Use Case:** Tracking mutable state, such as counters, timers, or flags.

### 3. Storing Previous State

* Use useRef to store the previous value of a state or prop for comparison purposes.

**Example:**

const PreviousStateExample = ({ value }) => {

const previousValue = useRef();

useEffect(() => {

previousValue.current = value; // Update ref after render

});

return (

<div>

<p>Current Value: {value}</p>

<p>Previous Value: {previousValue.current}</p>

</div>

);

};

**Use Case:** Comparing the current and previous state or prop values.

### 4. Avoiding Re-Creation of Objects

* Use useRef to store a value that you don't want to be re-created on every render, such as a function or an object.

**Example:**

const Timer = () => {

const timerId = useRef();

const startTimer = () => {

timerId.current = setInterval(() => console.log("Tick"), 1000);

};

const stopTimer = () => {

clearInterval(timerId.current); // Clears the timer

};

return (

<div>

<button onClick={startTimer}>Start</button>

<button onClick={stopTimer}>Stop</button>

</div>

);

};

**Use Case:** Managing timers, intervals, or callbacks without reinitializing them.

### 5. Avoiding State Update Loops

* Use useRef to track a value across renders without triggering a re-render, which can help avoid infinite update loops.

**Example:**

const AvoidInfiniteLoop = () => {

const renderCount = useRef(0);

useEffect(() => {

renderCount.current += 1; // Updates without triggering re-render

console.log(`Rendered ${renderCount.current} times`);

});

return <p>Check the console</p>;

};

**Use Case:** Counting renders or tracking a condition without affecting the UI.

### 6. Optimizing Expensive Computations

* Use useRef to cache expensive computations or objects to avoid recalculating them on every render.

**Example:**

const ExpensiveCalculation = () => {

const resultRef = useRef(doExpensiveCalculation());

return <p>Result: {resultRef.current}</p>;

};

function doExpensiveCalculation() {

// Simulate heavy computation

return 42;

}

**Use Case:** Memoizing values that don't need to change during the component's lifecycle.

### 7. Debouncing or Throttling Events

* Use useRef to store references to timers or intervals when implementing debounced or throttled functions.

**Example:**

const DebouncedInput = () => {

const debounceTimeout = useRef();

const handleChange = (event) => {

clearTimeout(debounceTimeout.current);

debounceTimeout.current = setTimeout(() => {

console.log("Value:", event.target.value);

}, 300);

};

return <input type="text" onChange={handleChange} />;

};

**Use Case:** Improving performance of event handling.

**Key Characteristics of useRef:**

* Updates to the .current property **do not trigger re-renders**.
* Values stored in a useRef persist across renders.
* Useful for working with mutable values or avoiding re-renders when managing references to DOM elements or other objects.

## userImperaiveHandle, forwardRef

## useLayoutEffect

## useInsertionEffect

## useId

## useTransition