React Hooks

React hooks are functions that let you use state and other React features in functional components. Here are some of the most commonly used hooks with examples:

### 1. useState

useState is a hook that lets you add state to functional components.

jsx

Copy code

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

export default Counter;

### 2. useEffect

useEffect is a hook that lets you perform side effects in functional components, such as data fetching, subscriptions, or manually changing the DOM.

jsx

Copy code

import React, { useState, useEffect } from 'react';

function DataFetcher() {

const [data, setData] = useState(null);

useEffect(() => {

fetch('https://jsonplaceholder.typicode.com/posts/1')

.then(response => response.json())

.then(data => setData(data));

}, []); // Empty dependency array means this effect runs once after the initial render.

return (

<div>

{data ? (

<div>

<h1>{data.title}</h1>

<p>{data.body}</p>

</div>

) : (

<p>Loading...</p>

)}

</div>

);

}

export default DataFetcher;

### 3. useContext

useContext is a hook that lets you subscribe to React context without introducing nesting.

jsx

Copy code

import React, { useContext, createContext } from 'react';

const MyContext = createContext();

function MyComponent() {

const value = useContext(MyContext);

return <div>{value}</div>;

}

function App() {

return (

<MyContext.Provider value="Hello from context!">

<MyComponent />

</MyContext.Provider>

);

}

export default App;

### 4. useReducer

useReducer is usually preferable to useState when you have complex state logic that involves multiple sub-values or when the next state depends on the previous one.

jsx

Copy code

import React, { useReducer } from 'react';

const initialState = { count: 0 };

function reducer(state, action) {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

default:

throw new Error();

}

}

function Counter() {

const [state, dispatch] = useReducer(reducer, initialState);

return (

<div>

Count: {state.count}

<button onClick={() => dispatch({ type: 'increment' })}>

Increment

</button>

<button onClick={() => dispatch({ type: 'decrement' })}>

Decrement

</button>

</div>

);

}

export default Counter;

### 5. useRef

useRef is a hook that lets you persist values between renders and access DOM elements directly.

jsx

Copy code

import React, { useRef } from 'react';

function TextInputWithFocusButton() {

const inputEl = useRef(null);

const onButtonClick = () => {

inputEl.current.focus();

};

return (

<div>

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

<button onClick={onButtonClick}>Focus the input</button>

</div>

);

}

export default TextInputWithFocusButton;

### 6. useMemo

useMemo is a hook that lets you memoize expensive calculations.

jsx

Copy code

import React, { useState, useMemo } from 'react';

function ExpensiveCalculationComponent({ num }) {

const computeExpensiveValue = (num) => {

console.log('Computing expensive value...');

return num \* 2;

};

const memoizedValue = useMemo(() => computeExpensiveValue(num), [num]);

return <div>Expensive value: {memoizedValue}</div>;

}

function App() {

const [count, setCount] = useState(0);

return (

<div>

<button onClick={() => setCount(count + 1)}>Increment</button>

<ExpensiveCalculationComponent num={count} />

</div>

);

}

export default App;

### 7. useCallback

useCallback is a hook that returns a memoized callback function.

jsx

Copy code

import React, { useState, useCallback } from 'react';

const Button = React.memo(({ handleClick }) => {

console.log('Button re-rendered');

return <button onClick={handleClick}>Click me</button>;

});

function App() {

const [count, setCount] = useState(0);

const handleClick = useCallback(() => {

setCount(count + 1);

}, [count]);

return (

<div>

<p>Count: {count}</p>

<Button handleClick={handleClick} />

</div>

);

}

export default App;

These examples cover the most commonly used hooks in React. They provide a powerful way to use state and other React features in functional components, making your code more readable and easier to maintain.