What is hook?

React Hooks are special functions in React, a popular JavaScript library for building user interfaces, that allow you to use state and other React features in functional components. Introduced in React 16.8, Hooks enable you to manage state, lifecycle events, and side effects without needing to write class components.

Here are some commonly used React Hooks:

1. **useState**: Allows you to add state to a functional component.
   ```javascript
   import { useState } from 'react';

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

     return (
       <div>
         <p>You clicked {count} times</p>
         <button onClick={() => setCount(count + 1)}>
           Click me
         </button>
       </div>
     );
   }
   ```

2. **useEffect**: Allows you to perform side effects in functional components, such as data fetching or subscribing to events.
   ```javascript
   import { useEffect, useState } from 'react';

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

     useEffect(() => {
       document.title = `You clicked ${count} times`;
     }, [count]);

     return (
       <div>
         <p>You clicked {count} times</p>
         <button onClick={() => setCount(count + 1)}>
           Click me
         </button>
       </div>
     );
   }
   ```

3. **useContext**: Allows you to use context to pass data through the component tree without passing props down manually.
   ```javascript
   import { useContext } from 'react';
   const ThemeContext = React.createContext('light');

   function Example() {
     const theme = useContext(ThemeContext);
     return <div>The current theme is {theme}</div>;
   }
   ```

4. **useReducer**: An alternative to `useState` for more complex state logic.
   ```javascript
   import { useReducer } from 'react';

   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 Example() {
     const [state, dispatch] = useReducer(reducer, { count: 0 });

     return (
       <div>
         <p>Count: {state.count}</p>
         <button onClick={() => dispatch({ type: 'increment' })}>+</button>
         <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
       </div>
     );
   }
   ```

5. **useRef**: Allows you to persist values between renders without causing re-renders, often used for accessing DOM elements directly.
   ```javascript
   import { useRef } from 'react';

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

     const focusInput = () => {
       inputRef.current.focus();
     };

     return (
       <div>
         <input ref={inputRef} type="text" />
         <button onClick={focusInput}>Focus the input</button>
       </div>
     );
   }
   ```

Hooks simplify the logic and readability of functional components, making them a powerful tool in modern React development.