React Hooks

React Hooks are a feature introduced in React 16.8 that allows you to use state and other React features in functional components, without the need for class components. Hooks provide a more concise and straightforward way to manage state, handle side effects, and access other React features within functional components.

Here are some commonly used React Hooks:

1. useState: The `useState` hook allows you to add state to a functional component. It returns an array with two elements: the current state value and a function to update the state. Example:

```jsx
import React, { useState } from 'react';

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

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
};
```

2. useEffect: The `useEffect` hook allows you to perform side effects in functional components, such as fetching data, subscribing to events, or manipulating the DOM. It replaces the lifecycle methods in class components. Example:

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

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

  useEffect(() => {
    const fetchData = async () => {
      const response = await fetch('https://api.example.com/data');
      const result = await response.json();
      setData(result);
    };

    fetchData();
  }, []);

  return (
    <div>
      {data ? (
        <ul>
          {data.map(item => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      ) : (
        <p>Loading data...</p>
      )}
    </div>
  );
};
```

3. useContext: The `useContext` hook allows you to access the value of a React context within a functional component. It eliminates the need for wrapping components with a context consumer. Example:

```jsx
import React, { useContext } from 'react';
import ThemeContext from './ThemeContext';

const ThemeButton = () => {
  const theme = useContext(ThemeContext);

  return (
    <button style={{ backgroundColor: theme.primaryColor, color: theme.textC
      Click me
    </button>
  );
};
```

These are just a few examples of React Hooks. React provides several other built-in hooks like `**useReducer**`, `**useCallback**`, `**useMemo**`, etc., which can be used to manage complex state logic, optimize performance, and improve code readability within functional components.

By leveraging React Hooks, you can write more concise and reusable code, as functional components with hooks tend to be shorter and easier to understand compared to class components with lifecycle methods.