let's go through each of these hooks andexamples for each.

1. **useState**:

The `useState` hook allows functional components to manage state.

Example 1: Simple Counter using useState

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

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

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

  const decrement = () => {
    setCount(count - 1);
  };

  return (
    <div>
      <h2>Counter</h2>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
    </div>
  );
}

export default Counter;
```

Example 2: Form Input using useState

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

function Form() {
  const [value, setValue] = useState('');

  const handleChange = (e) => {
```

```
    setValue(e.target.value);
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('Submitted value:', value);
    // Perform form submission logic here
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={value} onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
}

export default Form;
```

2. **useEffect**:

The `useEffect` hook allows functional components to perform side effects.

Example 1: Fetching data with useEffect

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

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

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Error fetching data:', error));
  }, []);

  return (
    <div>
      <h2>Data Fetching</h2>
      {data ? (
        <ul>
```

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

export default DataFetching;
```

Example 2: Subscribing to window events with useEffect

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

function WindowResize() {
  const [windowSize, setWindowSize] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => {
      setWindowSize(window.innerWidth);
    };

    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  return (
    <div>
      <h2>Window Resize</h2>
      <p>Window width: {windowSize}px</p>
    </div>
  );
}

export default WindowResize;
```

3. **useContext**:

The `useContext` hook allows functional components to consume context.

Example 1: Creating and consuming a context

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

const ThemeContext = React.createContext('light');

function ThemeProvider({ children }) {
  return <ThemeContext.Provider value="dark">{children}</ThemeContext.Provider>;
}

function ThemedComponent() {
  const theme = useContext(ThemeContext);

  return (
    <div>
      <h2>Themed Component</h2>
      <p>Current theme: {theme}</p>
    </div>
  );
}

export { ThemeProvider, ThemedComponent };
```

Example 2: Using multiple contexts in a component

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

const ThemeContext = React.createContext('light');
const UserContext = React.createContext(null);

function UserProfile() {
  const theme = useContext(ThemeContext);
  const user = useContext(UserContext);

  return (
    <div>
      <h2>User Profile</h2>
      <p>Username: {user ? user.username : 'Guest'}</p>
```

```
      <p>Theme: {theme}</p>
    </div>
  );
}

export { ThemeContext, UserContext, UserProfile };
```

4. **Custom hooks**:

Custom hooks allow you to extract reusable logic from components.

Example 1: Custom hook for fetching data

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

function useDataFetching(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        const result = await response.json();
        setData(result);
        setLoading(false);
      } catch (error) {
        setError(error);
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error };
}

export default useDataFetching;
```

Example 2: Custom hook for debouncing user input

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

function useDebounce(value, delay) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const timerId = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(timerId);
    };
  }, [value, delay]);

  return debouncedValue;
}

export default useDebounce;
```

In each example, we've provided two distinct use cases for the given hook to demonstrate its versatility and applicability in different scenarios.