Context Provider:

Here <**Welcome/>** is the parent component which is shown below. It has a child component called **<Counter/>**.

```
import React from 'react';
import Counter from './Counter';
import CounterChild from './CounterChild';

const Welcome = (props) => {
    console.log('Welcome - Parent of all components')
    return(
        <div>
        <h1>Welcome to our Counter Application!</h1>
        <Counter render={<CounterChild/>}> //The component is passed as render prop
        </Counter>
        </div>
    );
}
export default Welcome;
```

The **<Counter/>** component is shown below. This accesses state from the context provider.

```
import React, {useContext} from 'react';
import {CounterContext} from './CounterContext';
import CounterChild from './CounterChild';

const Counter = (props) => {
console.log('Counter - Child of Welcome. I use context.');
const [count, setCount] = useContext(CounterContext);

const incrementCount = (e) => {
    setCount(prevCount => prevCount + 1);
}

const resetCount = (e) => {
    setCount(0);
}
return(
    <div>
    <p>The count is: {count}</p>
    <button onClick={incrementCount}>Increment Count</button>
    <button onClick={resetCount} style={{'margin':'10px'}}>Reset Count</button>
    <CounterChild/>
    </div>
);
};
```

```
export default Counter;
```

Below shown is **<CounterChild/>**, the child of the **<Counter/>** component. We console log inside these components to understand whether the child component would re-render when the context state is modified from the parent component <**Counter/>.**

```
import React from 'react';

const CounterChild = () => {
    console.log('Child of Counter');
    return(
        <p>Hi I'm a child of counter.</p>
    );
}

//export default React.memo(CounterChild);
export default CounterChild;
```

Below shown is the **<ContextProvider/>** which is a wrapper component that provides the values to be accessed by children using Context API:

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

// creating context
export const CounterContext = createContext();

// context provider
export const CounterProvider = (props) => {
    const [count, setCount] = useState(0);

    return(<CounterContext.Provider value={[count, setCount]}>
        {props.children}
        </CounterContext.Provider>);
}
```

In the **<App/>** component, we can see that only the **<Welcome/>** component is written since it's the parent component for the other components and wraps them.

```
import "./App.css";
import Welcome from "./components/Welcome";
import { CounterProvider } from "./components/CounterContext";
function App() {
  return (
    <CounterProvider>
      <div className="App">
        <Welcome/>
      </div>
    </CounterProvider>
  );
```

```
}
export default App;
```

The react render is written in the index.js file:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const container = document.getElementById('root');
ReactDOM.render(
<App/>,container

);
```

Video Url: Child re-render on context change. It can be seen from the recording that when the context is updated the child component also re-renders even though it doesn't use context. This is as expected since React causes the child components to re-render when the state of the parent component changes. It is described well in this article: Re-rendering in React.

So there are two ways of avoiding re-render:

- Memoize the child component using React.memo(ChildComponent).

```
export default React.memo(CounterChild);
```

- Instead of writing the child component inside the parent pass it as render props to the parent component.

```
import React, {useContext} from 'react';
import {CounterContext} from './CounterContext';
const Counter = (props) => {
console.log('Counter - Child of Welcome. I use context.');
const [count, setCount] = useContext(CounterContext);
const incrementCount = (e) => {
    setCount(prevCount => prevCount + 1);
}
const resetCount = (e) => {
    setCount(0);
}
return(
    <div>
    <p>The count is: {count}</p>
    <button onClick={incrementCount}>Increment Count</button>
    <button onClick={resetCount} style={{'margin':'10px'}}>Reset
Count</button>
    {props.render} //This render child component passed as render prop
    </div>
);
};
```

```
export default Counter;
```

It can be seen below that the child component is passed as render props inside the parent component.

```
import React from 'react';
import Counter from './Counter';
import CounterChild from './CounterChild';

const Welcome = (props) => {
    console.log('Welcome - Parent of all components')
    return(
        <div>
        <h1>Welcome to our Counter Application!</h1>
        <Counter render={<CounterChild/>}>
        </Counter>
        </div>
    );
}
export default Welcome;
```

In the below video it is clearly seen that the child component is not re-rendered on context change in the parent when the child component is passed as render props to the parent.

Child Components passed as render props.