Day - 4

✡ How re-rendering works with Context

→ When a component consumes some context value and the value of this context changes, that component re-renders.

→ When it comes to the default behavior of React rendering, if a component renders, React will recursively re-render all its children regardless of props or context.

e.g. Top level component injects a context provider at the top.

⇒
```
function App() {
return (
    <AppContext.Provider>
          <CompA />
    </AppC.Pro.>
)
}
```

```
const CompA = () => <CompB />
const CompB = () => <CompC />
const CompC = () => null
```
} functions same as func CompA(){...}

Now, if App re-renders, A, B, C all three will re-render as well.

App(Context Provider) ⟶ A ⟶ B ⟶ C

If the components are complex in nature, this could result in performance hit.

→ How to solve this issue?
⇓
React.memo() & useMemo hook

✱ React.memo()

→ If your component renders the same result given the same props, you can wrap it in a call to React.memo() for a performance boost by memoizing the result.

→ Memoization is a programming technique that accelerates performance by caching the return values of expensive func. calls.

→ This means that React will skip rendering the component, and reuse the last rendered result.

✱ useMemo()

→ Used for caching a specific calculation or value.

acceps 2 arguments

→ const memoized = useMemo(compute, dependencies)

(More about hooks in next module)