Understanding useMemo and useCallback

React keeps our UI in sync w/ our application state. Code: "re-renders".
useMemo and useCallback are tools built to help us optimize re-renders.
In Javascript, we only have one main thread.

useMemo takes two arguments:
1) A chunk of work to be performed, wrapped up in a function.
2) A list of dependencies.

→ it is like a small cache, and the dependencies are the cache invalidation strategy.

Memoization

React.memo → pure components
→ export default React.memo(Component)

useMemo use cases: 1) Heavy computations
2) Preserve references

From the documentation

useId is a React hook for generating unique IDs that are stable across the server and client, while avoiding hydration mismatches. It is not for generating keys in a list. Keys should be generated from your data.

same for objects

Problem: every time React re-renders, we're producing a new array. They're equivalent in terms of value, but not in terms of reference.

useMemo to preserve a reference to a particular array/object.

useMemo → for arrays/objects ; useCallback → for functions.
Functions are also compared by reference, not by value.

```
React.useCallback(function helloWorld(){}, []);
```

<=>

```
React.useMemo(() => function helloWorld(){}, []);
```

When to use these hooks "for sure": 1) Inside generic custom hooks

2) Inside context providers

↙

Since it's common to pass a big object as the 'value' attribute, it's generally a good idea to memoize this object. Example:

```
const AuthContext = React.createContext({});

function AuthProvider({ user, status, children }){
    const memoizedValue = React.useMemo(() => {
        return {
            user,
            status,
        };
    }, [user, status]);

    return (
        <AuthContext.Provider value={memoizedValue}>
            {children}
        </AuthContext.Provider>
    );
}
```