Assignment 11: Data is the new Oil

1. **What is Prop Drilling?**

   Answers: In the React app, there can be lots of nested components. It can be very complex to manage all the data which is passed from one prop to another in such a big hierarchy.

   - React has one way data flow. React components can only pass props from parent to children.

   - If we have props W -> X -> Y -> Z the if we want to pass some prop to Z which is in W then we need to pass it through X & Y. **It is called Prop Drilling**.

   - Sometimes we have data in a component in one hierarchy and we want to access that data somewhere else in the app in another hierarchy then it is an issue.

2. **What is Lifting the state up?**

   Answer: If we have some state variable in a component and we have to remove the same from the component and move it to the parent component and pass as the prop to the component so that we can manage the state variable from the parent, that is known as **lifting the state up**.

   - For building accordion items where we open one item and that will close the others that are open, **we need to lift the state up of the expanded state variable**.

   - **Lighting the state up means that we need to move the logic from child component to parent componen**t.

3. **What are Context Provider and Context consumer?**

   Answer: In React, the Context API provides a way to share data (state) between components without having to pass it manually through props. Two essential components of the Context API are:

   1. Context Provider:
   - The `Context Provider` is a component that makes data (state) available to its child components.

- It is defined using `React.createContext` and typically wraps a portion of your component tree.
- The data it provides can be accessed by any descendant component that subscribes to the context.

```
const MyContext = React.createContext(); // Create a context

function App() {
  const sharedData = "Hello from Context!";

  return (
    <MyContext.Provider value={sharedData}>
      {/* Child components can access sharedData */}
    </MyContext.Provider>
  );
}
```

2. Context Consumer:
- The `Context Consumer` is a component that subscribes to the context and can access the data provided by the `Context Provider`.
- It uses the `MyContext.Consumer` component or the `useContext` hook to access the context's data.

```
const MyContext = React.createContext();

function ChildComponent() {
  return (
    <MyContext.Consumer>
      {value => <div>{value}</div>} {/* Access sharedData */}
    </MyContext.Consumer>
  );
}
```

or with `useContext`:

```
const MyContext = React.createContext();

function ChildComponent() {
  const value = useContext(MyContext); // Access sharedData
  return <div>{value}</div>;
}
```

The Context Provider and Context Consumer work together to create a data-sharing mechanism in React, allowing you to pass values from a higher-level component to any descendant component that needs access to that data. This is particularly useful for managing global state or theme settings in your application.

4. **If you don’t pass a value to a Provider does it take the default value?**

   Answers: Yes, if you don't pass a value to a `Context.Provider` in React, it will take the default value that you specified when creating the context using React.createContext(defaultValue).

   Here's how it works:
   const MyContext = React.createContext("Default Value");

```
function App() {
  // No value provided to the Provider
  return (
    <MyContext.Provider>
      {/* Child components can access the default value
*/}
      <ChildComponent />
    </MyContext.Provider>
  );
}

function ChildComponent() {
  const value = useContext(MyContext);
  return <div>{value}</div>; // Will display "Default
Value"
}
```

   In the example above, since we didn't provide a specific value to the `MyContext.Provider`, it uses the default value "Default Value." This default value is then accessible to any descendant component that consumes the context, such as **ChildComponent**.