useContext

Understanding useContext in React

The useContext hook is a built-in **React Hook** that allows components to **access global data** without prop drilling. It helps in **state management** by providing a way to share data between components without passing props manually.

What is useContext?

- It allows a **component to consume data from a Context** without passing props down multiple levels.
- Works together with **React Context API** (React.createContext()).
- Useful for **theme switching, authentication, language settings, etc.**

Syntax:

```
const value = useContext(MyContext);
```

- MyContext is created using React.createContext().
- value holds the **context value** provided by the Context.Provider.

Example: Using useContext for Theme Switching

Step 1: Create a Context

```
import { createContext } from "react";

export const ThemeContext = createContext(null); // Create a Context
```

- createContext(null) initializes a **Context object**.

- ThemeContext will **hold and provide** the theme state.

Step 2: Create a Provider Component

```
import { useState } from "react";
import { ThemeContext } from "./ThemeContext";

export default function ThemeProvider({ children }) {
  const [theme, setTheme] = useState("light");

  function toggleTheme() {
    setTheme(theme === "light" ? "dark" : "light");
  }

  return (
    <ThemeContext.Provider value={{ theme,
toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}
```

- ThemeProvider **wraps child components** and provides theme & toggleTheme function.
- ThemeContext.Provider **makes** theme **and** toggleTheme **accessible**.

Step 3: Consume the Context in a Component

```
import { useContext } from "react";
import { ThemeContext } from "./ThemeContext";

export default function ThemeSwitcher() {
  const { theme, toggleTheme } = useContext(ThemeContext);
// Consume context
```

```
    return (
        <div style={{ background: theme === "light" ? "#fff" :
"#333", color: theme === "light" ? "#000" : "#fff", padding:
"20px", textAlign: "center" }}>
            <p>Current Theme: {theme}</p>
            <button onClick={toggleTheme}>Toggle
Theme</button>
        </div>
    );
}
```

- useContext(ThemeContext) **accesses the theme state**.
- Clicking the button **toggles between "light" and "dark" themes**.

Step 4: Wrap the App with the Provider

```
import ThemeProvider from "./ThemeProvider";
import ThemeSwitcher from "./ThemeSwitcher";

export default function App() {
    return (
        <ThemeProvider>
            <ThemeSwitcher />
        </ThemeProvider>
    );
}
```

- ThemeProvider **wraps** the ThemeSwitcher, making theme and toggleTheme accessible.

---

useContext VS. Prop Drilling

Feature	Prop Drilling	useContext
Data Access	Passed manually via props	Accessed globally via useContext
Scalability	Hard to manage in large apps	Simplifies data flow
Performance	Unnecessary re-renders in child components	Prevents deep nesting issues

When to Use useContext?

✅ **Global state management** (e.g., themes, authentication, language settings).
✅ **Avoiding "prop drilling"** (passing props through multiple components).
✅ **Providing shared data** across deeply nested components.