⚛️ Props & Props Drilling in React – Complete Notes

🔹 1️⃣ What Are Props?

Props (short for "Properties") are read-only inputs passed from a **parent component to a child component**.

They allow components to be:

- 🔁 Reusable
- 🔗 Dynamic
- 🧩 Composable

Think of props like **function arguments**, but for components.

📌 Basic Example of Props

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return <Welcome name="Kousik" />;
}
```

✅ Explanation

- `App` is the parent.
- `Welcome` is the child.
- `name="Kousik"` is passed as a prop.
- Props are accessed via `props.name`.

🔹 2️⃣ Props with Destructuring (Best Practice)

```
function Welcome({ name }) {
  return <h1>Hello, {name}</h1>;
}
```

✓ Cleaner
✓ More readable

🔹 3️⃣ Props Can Be Any Data Type

🟢 String

```
<Greeting message="Good Morning" />
```

🔵 Number

```
<Counter count={10} />
```

🟡 Boolean

```
<Button disabled={true} />
```

🟣 Array

```
<List items={["Apple", "Banana"]} />
```

🟠 Object

```
<User user={{ name: "Kousik", age: 22 }} />
```

🔴 Function (Very Important)

```
function Parent() {
  const handleClick = () => alert("Clicked");

  return <Child onClick={handleClick} />;
}
```

👉 This allows child → parent communication.

🔹 4️⃣ Props Are Read-Only

❌ Wrong:

```
props.name = "New Name";
```

React enforces **unidirectional data flow**:

Parent → Child

If data needs to change → use state in parent.

🔷 5️⃣ Passing Props Through Multiple Components

Example:

```
function App() {
  return <Parent name="Kousik" />;
}

function Parent({ name }) {
  return <Child name={name} />;
}

function Child({ name }) {
  return <GrandChild name={name} />;
}

function GrandChild({ name }) {
  return <h1>{name}</h1>;
}
```

This leads to…

🚨 6️⃣ What is Props Drilling?

Props Drilling happens when you pass props through multiple intermediate components that don't actually use them.

App → Parent → Child → GrandChild
Only GrandChild needs the data.

🎯 Why Props Drilling is a Problem

- ❌ Hard to maintain
- ❌ Redundant code
- ❌ Tight coupling
- ❌ Difficult scaling

In large apps, it becomes messy.

📉 Visual Representation of Props Drilling

App
⬇
Parent

⬇
Child
⬇
GrandChild

Only GrandChild uses the prop, but everyone passes it.

🔥 7️⃣ How to Solve Props Drilling

✅ Solution 1: Context API (Best for Global Data)

Step 1: Create Context

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

export const UserContext = createContext();
```

Step 2: Provide Context

```
<UserContext.Provider value="Kousik">
  <App />
</UserContext.Provider>
```

Step 3: Consume Context

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

function GrandChild() {
  const user = useContext(UserContext);
  return <h1>{user}</h1>;
}
```

✔ No need to pass through Parent and Child.

✅ Solution 2: Component Composition

Instead of passing data, pass components.

```
function Parent({ children }) {
  return <div>{children}</div>;
}

function App() {
  return (
    <Parent>
      <GrandChild name="Kousik" />
    </Parent>
  );
}
```

✅ Solution 3: State Management Libraries

For large applications:

- Redux
- Zustand
- Recoil
- Jotai

🔹 8️⃣ When Props Drilling Is OK

Props drilling is NOT always bad.

It’s fine when:

- Small app
- Only 1–2 levels deep
- Clear component structure

Overengineering with Context for small apps is worse.

🔹 9️⃣ Real-World Example

Imagine you’re building:

- 🔐 Authentication system
- 🎨 Theme switcher
- 🌍 Language selector

These are global data → Use Context
Not props drilling.

🔹 🔟 Props vs State

Feature	Props	State
Owned by	Parent	Component itself
Mutable	❌ No	✅ Yes
Direction	Parent → Child	Internal
Re-render trigger	Yes	Yes

🧠 Interview Questions

1. What are props in React?
2. Are props mutable?
3. What is props drilling?
4. How do you avoid props drilling?
5. When should you use Context instead?