Conditional Rendering

- React doesn’t have if/else statements inside JSX.
- Instead, we use conditional rendering to decide what to show.
- Conditional rendering means displaying UI elements based on certain conditions or states.

Using if/else:

```
function App() {
  const isLoggedIn = true;

  if (isLoggedIn) {
    return <h1>Welcome back!</h1>;
  }
  return <h1>Please log in</h1>;
}
```

React components can return different JSX depending on logic.

Note:

Don’t trying to use if inside JSX tags —it is not allowed.

Using Ternary Operator (Inline Rendering)

Syntax:

```
condition ? <ComponentA /> : <ComponentB />
```

Example:

```
function App() {
  const isLoggedIn = true;

  return (
    <div>
      {isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Log In</h1>}
    </div>
  )
}
```

Note:

Avoid complex ternaries with multiple ? : ? : ? — it reduces readability.

Logical AND (&&) Operator

```
SYNTAX:

condition && <Component />
```

```
EXAMPLE:

function Notification({ hasMessage }) {
  return (
    <div>
      {hasMessage && <p>You have new messages!</p>}
    </div>
  );
}
```

Note:

- ❑ If the condition is true → it renders the element.
- ❑ If false → renders nothing.
- ❑ Use case → When you only want to show something (not alternate options).

Conditional Rendering with Variables

```jsx
function Dashboard({ option }) {
  let content;

  if (option === "a") {
    content = <ComponentA />;
  } else if (option === "b") {
    content = <ComponentB />;
  } else {
    content = <ComponentC />;
  }

  return <div>{content}</div>;
}
```

- Store JSX in a variable.
- Helps keep return statement clean and readable.

Conditional CSS or ClassName

```
function Button({ isActive }) {
  return (
    <button className={isActive ? "btn-active" : "btn-inactive"}>
      Click Me
    </button>
  );
}
```

Conditional Rendering with State

```jsx
import { useState } from "react";

function ToggleMessage() {
  const [show, setShow] = useState(true);

  return (
    <div>
      <button onClick={() => setShow(!show)}>
        {show ? "Hide" : "Show"} Message
      </button>
      {show && <p>This is a secret message!</p>}
    </div>
  );
}
```

Returning null (Hide Completely)

```
function Alert({ show }) {
  if (!show) return null;
  return <div className="alert">This is an alert!</div>;
}
```

- Returning null means “render nothing”.
- Common in modal or popup components.

Advanced: Conditional Components

```
function Layout({ type }) {
  const Component = type === "home" ? HomePage : AboutPage;
  return <Component />;
}
```

- You can even conditionally select components themselves.
- Useful for routing-like patterns or dynamic layouts.

Using switch Statement

When you have 3 or more conditions, if/else becomes messy. That's when switch makes your code cleaner and more readable.

```
function StatusMessage({ status }) {
  switch (status) {
    case "loading":
      return <p>Loading...</p>;

    case "success":
      return <p>Data loaded successfully ✅</p>;

    case "error":
      return <p>Something went wrong ❌</p>;

    default:
      return <p>Idle state...</p>;
  }
}
```

Using Mapping (Object Lookup)

For even more elegant and scalable conditional rendering —you can use object maps to store your UI patterns.

```
function StatusMessage({ status }) {
  const statusMap = {
    loading: <p>Loading...</p>,
    success: <p>Data loaded successfully ✅</p>,
    error: <p>Something went wrong ❌</p>,
  };

  return statusMap[status] || <p>Idle state...</p>;
}
```

Common Beginner Mistakes

❌ Writing if inside JSX → use ternary or logical AND.

❌ Nesting too many ternaries → makes code unreadable.

❌ Forgetting to handle “else” case → empty UI.

❌ Using == instead of === → unexpected behavior.

❌ Forgetting return null → causes unnecessary rendering.

Tips

- [x] Extract conditional logic into small functions.
- [x] Keep JSX clean — logic outside, UI inside.
- [x] Prefer readability over compactness.
- [x] For multiple conditions, use switch or mapping.

Thank You