REACT JS

Swipe >>>

Conditional Rendering

Conditional rendering allows you to dynamically display different UI elements based on certain conditions

Ternary Operator

A concise way to render different UI elements based on a boolean condition.

```
return isLogged ? <UserGreeting /> : <GuestGreeting />;
```

Short Circuit Evaluation

A technique to efficiently render UI elements based on conditional expressions.

```
return isLoggedIn && <UserGreeting />;
```

Swipe >>>

Event Handling

Handling Events

A way to define functions that respond to specific user interactions, such as clicks, form submissions, and input changes.

```
const handleClick = () => {
  // Handle the click event
};
```

useState with Event Handling

Integrating event handlers with useState to dynamically update component state based on user input.

```
const [inputValue, setInputValue] = useState("");

const handleChange = (e) => {
  setInputValue(e.target.value);
};
```

Swipe >>>

Hooks

useState

A hook for managing component state, allowing you to keep track of data within the component and update it as needed.

```
const [count, setCount] = useState(0);
```

useEffect

A hook for handling side effects, such as data fetching, subscriptions, and cleanup logic.

```
useEffect(() => {
// Effect logic here
}, [dependencies]);
```

useContext

A hook for sharing data across components without the need for prop drilling

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

Swipe >>>

Components and Props

Functional Components

The fundamental building blocks of UI construction in React.

```jsx
const MyComponent = (props) => {
  return <div>{props.message}</div>;
};
```

Destructuring Props

A concise way to access prop values directly within the functional comp.

```jsx
const MyComponent ({ message }) => {
  return <div>{message}</div>;
};
```

Default Props

A mechanism to provide default values for props.

```jsx
MyComponent.defaultProps = {
  message: "Default Message",
};
```

Swipe >>>

useReducer

A hook for managing complex state, providing a more structured approach to state updates.

```
const [state, dispatch] = useReducer (reducer, initialState);
```

useCallback

This hook memoizes a callback function, preventing unnecessary re-renders of child components.

```
const memoizedCallback = useCallback(() => {
// callback logic
}, [dependencies]);
```

Swipe >>>

useMemo

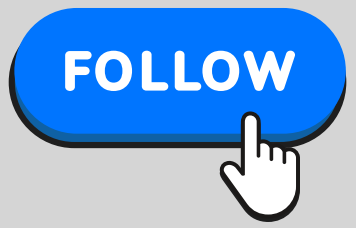
useMemo memoizes the result of a computation, preventing it from being recalculated on every render.

```
const value useMemo(() => computeValue(a, b), [a, b]);
```

useRef

useRef returns a ref object with a current property that can be used to hold a mutable value.

```
const myRef = useRef(initialValue);
```


FOLLOW