Conditional Rendering + Lists and Keys

Conditional rendering

Conditional rendering in React works the same way conditions work in JavaScript.

Point 1

```
function UserGreeting(props) {
 return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
 return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
 const isLoggedIn = props.isLoggedIn;
 if (isLoggedIn) {
   return <UserGreeting />;
 }
 return <GuestGreeting />;
}

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Greeting isLoggedIn={false} />);
```

Point 2

```
class LoginControl extends React.Component {
```

```
constructor(props) {
  super(props);
  this.handleLoginClick = this.handleLoginClick.bind(this);
  this.handleLogoutClick = this.handleLogoutClick.bind(this);
  this.state = { isLoggedIn: false };
}

handleLoginClick() {
  this.setState({ isLoggedIn: true });
}

handleLogoutClick() {
  this.setState({ isLoggedIn: false });
}

render() {
  const isLoggedIn = this.state.isLoggedIn;
  let button;
  if (isLoggedIn) {
    button = <LogoutButton onClick={this.handleLogoutClick} />;
  } else {
    button = <LoginButton onClick={this.handleLoginClick} />;
  }

  return (
    <div>
      <Greeting isLoggedIn={isLoggedIn} />
      {button}
    </div>
  );
```

```
 }
}
```

Inline If with Logical && Operator

```
function Mailbox(props) {
 const unreadMessages = props.unreadMessages;
 return (
   <div>
     <h1>Hello!</h1>
     {unreadMessages.length > 0 && (
       <h2>You have {unreadMessages.length} unread messages.</h2>
     )}
   </div>
 );
}

const messages = ["React", "Re: React", "Re:Re: React"];

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Mailbox unreadMessages={messages} />);
```

It works because in JavaScript, true && expression always evaluates to expression, and false && expression always evaluates to false

Inline If-Else with Conditional Operator

Another method for conditionally rendering elements inline is to use the JavaScript conditional operator `condition ? true : false`

Point 1

```
render() {
```

```
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

Point 2

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LoginButton onClick={this.handleLoginClick} />
      }
    </div>
  );
}
```

Preventing Component from Rendering

So we can pretty much use conditional rendering to hide a component even if it gets rendered.

```
function WarningBanner(props) {
 if (!props.warn) {
   return null;
 }
 return <div className="warning">Warning!</div>;
}
```

Rendering Multiple Components

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) => <li>{number}</li>);
<ul>{listItems}</ul>;
```

Basic List Component

```
function NumberList(props) {
 const numbers = props.numbers;
 const listItems = numbers.map((number) => <li>{number}</li>);
 return <ul>{listItems}</ul>;
}

const numbers = [1, 2, 3, 4, 5];
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<NumberList numbers={numbers} />);
```

When you run this code, you'll be given a warning that a key should be provided for list items. A "key" is a special string attribute you need to include when creating lists of elements. Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) => (
 <li key={number.toString()}>{number}</li>
));
```

When you don't have stable IDs for rendered items, you may use the item index as a key as a last resort:

```
const todoItems = todos.map((todo, index) => (
 // Only do this if items have no stable IDs
 <li key={index}>{todo.text}</li>
));
```

Using indexes as keys is not advisable as it may negatively impact performance (article).

Extracting Components with Keys

```
function ListItem(props) {
 const value = props.value;
 return (
   // Wrong! There is no need to specify the key here:
   <li key={value.toString()}>{value}</li>
 );
}

function NumberList(props) {
 const numbers = props.numbers;
 const listItems = numbers.map((number) => (
   // Wrong! The key should have been specified here:
   <ListItem value={number} />
 ));
 return <ul>{listItems}</ul>;
}
```

Keys Must Only Be Unique Among Siblings