**Exercise 13**

### Explaining Various Ways of Conditional Rendering

Conditional rendering in React allows you to display different elements or components based on certain conditions. Here are the most common ways to achieve this:

1. **if...else statements:** This is the most basic way to conditionally render components. You can use a standard JavaScript if statement within your component's render method to return different JSX based on a condition.  
   JavaScript  
   function Greeting({ isLoggedIn }) {  
    if (isLoggedIn) {  
    return <h1>Welcome back!</h1>;  
    } else {  
    return <h1>Please sign up.</h1>;  
    }  
   }
2. **Ternary Operator:** This is a more concise way to handle simple conditional rendering. It's often used inline within JSX.  
   JavaScript  
   function Greeting({ isLoggedIn }) {  
    return (  
    <div>  
    {isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}  
    </div>  
    );  
   }
3. **Logical && Operator:** If you want to render something only when a condition is true, you can use the logical && operator. If the condition is false, React will ignore the part after &&.  
   JavaScript  
   function Mailbox({ unreadMessages }) {  
    return (  
    <div>  
    <h1>Hello!</h1>  
    {unreadMessages.length > 0 &&  
    <h2>You have {unreadMessages.length} unread messages.</h2>  
    }  
    </div>  
    );  
   }
4. **switch statements:** For more complex conditions with multiple cases, you can use a switch statement to return the appropriate JSX.  
   JavaScript  
   function UserProfile({ userRole }) {  
    switch (userRole) {  
    case 'admin':  
    return <AdminPanel />;  
    case 'editor':  
    return <EditorDashboard />;  
    default:  
    return <UserDashboard />;  
    }  
   }

### Explaining How to Render Multiple Components

You can render multiple components in React by placing them next to each other within a parent element. However, React components must always return a single root element. This means you need to wrap the multiple components in a parent container like a <div>, <span>, or a React.Fragment.

**Using a div or other HTML element:**

JavaScript

function App() {  
 return (  
 <div>  
 <Header />  
 <MainContent />  
 <Footer />  
 </div>  
 );  
}

**Using React.Fragment (or <>...</>):**

React.Fragment is a special component that lets you group a list of children without adding extra nodes to the DOM. This is useful when you want to avoid unnecessary divs.

JavaScript

import React from 'react';  
  
function App() {  
 return (  
 <>  
 <Header />  
 <MainContent />  
 <Footer />  
 </>  
 );  
}

### Defining a List Component

A list component in React is a component that renders a list of items, typically by iterating over an array of data. The data for the list can be passed down as a prop.

Here is an example of a simple list component:

JavaScript

// A component for a single list item  
function ListItem({ value }) {  
 return <li>{value}</li>;  
}  
  
// The main list component  
function MyList({ items }) {  
 // Use the map() function to create a list of ListItem components  
 const listItems = items.map((item) =>  
 <ListItem key={item.id} value={item.name} />  
 );  
  
 return (  
 <ul>{listItems}</ul>  
 );  
}  
  
// Example usage  
const data = [  
 { id: 1, name: 'Apple' },  
 { id: 2, name: 'Banana' },  
 { id: 3, name: 'Cherry' }  
];  
  
function App() {  
 return <MyList items={data} />;  
}

### Explaining About Keys in React Applications

Keys are a special string attribute that you must include when creating lists of elements. React uses keys to identify which items have changed, are added, or are removed. This helps React efficiently update the UI and prevent performance issues.

**Why are keys important?**

* **Identity:** Keys provide a stable identity to each item in a list. When a list is re-rendered, React can use the keys to match the new items with the old ones.
* **Performance:** By using keys, React can avoid re-rendering the entire list. Instead, it can identify exactly which items need to be updated, added, or removed, which is much more performant.
* **Preventing Bugs:** Without keys, React might re-render components in the wrong order, leading to unexpected behavior and bugs, especially with components that maintain their own state or have side effects.

**Rules for keys:**

* **Keys must be unique:** The key for each item in a list must be unique among its siblings.
* **Keys should be stable:** A good key is a unique ID from your data, like a post.id or user.id. Avoid using the array index as a key if the list can be reordered, filtered, or grow over time, as this can lead to performance issues and bugs.

### Explaining How to Extract Components with Keys

When you create a list of components, it's good practice to extract the list items into their own components. This makes your code more readable and reusable. The key should be specified on the component you are creating inside the map() function.

JavaScript

// Data  
const posts = [  
 { id: 1, title: 'Hello World', content: 'Welcome to learning React!' },  
 { id: 2, title: 'Installation', content: 'You can install React from npm.' }  
];  
  
// A single component for a blog post  
function BlogPost({ id, title, content }) {  
 return (  
 <div className="blog-post">  
 <h3>{title}</h3>  
 <p>{content}</p>  
 </div>  
 );  
}  
  
// The component that renders the list of blog posts  
function Blog({ posts }) {  
 const postItems = posts.map((post) =>  
 // The key is placed on the BlogPost component  
 <BlogPost key={post.id} id={post.id} title={post.title} content={post.content} />  
 );  
 return (  
 <div>  
 {postItems}  
 </div>  
 );  
}  
  
// Example usage  
function App() {  
 return <Blog posts={posts} />;  
}

In this example, the key is assigned to the BlogPost component. This ensures that React can correctly identify and manage each individual post within the list.

### Explaining React Map, map() function

The map() function is a standard JavaScript array method that is fundamental to rendering lists in React. It allows you to iterate over an array and transform each item into a new value. In the context of React, you use map() to transform an array of data into an array of JSX elements or components.

**How map() works:**

The map() function takes a callback function as an argument. This callback function is executed once for each element in the array. The callback function can take three arguments:

1. currentValue: The current element being processed in the array.
2. index (optional): The index of the current element.
3. array (optional): The array map() was called upon.

The map() function then returns a new array containing the results of the callback function for each element.

**Example in React:**

JavaScript

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

In this example:

1. The numbers array contains [1, 2, 3, 4, 5].
2. map() iterates over each number in the array.
3. For each number, it returns a <li> element.
4. The key is set to number.toString() to provide a unique identifier for each list item.
5. The listItems constant becomes a new array of JSX <li> elements.
6. Finally, this new array of JSX is rendered inside the <ul> element.

**BlogDetails.js**

import React from 'react';

const BlogDetails = ({ blog }) => {

  return (

    <div>

      <h2>Blog Details</h2>

      {blog ? (

        <div>

          <h3>{blog.title}</h3>

          <p>Author: {blog.author}</p>

          <p>{blog.content}</p>

          {blog.isPublished ? (

            <p style={{ color: 'blue' }}>Status: Published</p>

          ) : (

            <p style={{ color: 'gray' }}>Status: Draft</p>

          )}

        </div>

      ) : (

        <p>No blog post selected.</p>

      )}

    </div>

  );

};

export default BlogDetails;

**BookDetails.js**

import React from 'react';

const BookDetails = ({ book }) => {

  if (!book) {

    return <p>No book selected.</p>;

  }

  return (

    <div>

      <h2>Book Details</h2>

      {book.title && <h3>Title: {book.title}</h3>}

      {book.author && <p>Author: {book.author}</p>}

      {book.available && <p style={{ color: 'green' }}>Status: Available</p>}

      {!book.available && <p style={{ color: 'red' }}>Status: Not Available</p>}

    </div>

  );

};

export default BookDetails;

**CourseDetails.js**

import React from 'react';

const CourseDetails = ({ course }) => {

  const renderCourseLevel = () => {

    if (!course) {

      return <p>No course selected.</p>;

    }

    switch (course.level) {

      case 'beginner':

        return <p style={{ color: 'green' }}>Level: Beginner</p>;

      case 'intermediate':

        return <p style={{ color: 'orange' }}>Level: Intermediate</p>;

      case 'advanced':

        return <p style={{ color: 'red' }}>Level: Advanced</p>;

      default:

        return <p>Level: Not specified</p>;

    }

  };

  return (

    <div>

      <h2>Course Details</h2>

      {course ? (

        <>

          <h3>{course.title}</h3>

          <p>Instructor: {course.instructor}</p>

          {renderCourseLevel()}

        </>

      ) : (

        <p>No course selected.</p>

      )}

    </div>

  );

};

export default CourseDetails;

**App.js**

import React, { useState } from 'react';

import BookDetails from './components/BookDetails';

import BlogDetails from './components/BlogDetails';

import CourseDetails from './components/CourseDetails';

import './App.css';

const allData = {

  book: {

    title: 'The React Handbook',

    author: 'John Doe',

    available: true,

  },

  blog: {

    title: 'Getting Started with React Hooks',

    author: 'Jane Smith',

    content: 'Hooks are a new addition in React 16.8...',

    isPublished: true,

  },

  course: {

    title: 'Advanced React Patterns',

    instructor: 'Chris Evans',

    level: 'advanced',

  },

};

const componentMap = {

  book: BookDetails,

  blog: BlogDetails,

  course: CourseDetails,

};

function App() {

  const [activeComponent, setActiveComponent] = useState('book');

  const ComponentToRender = componentMap[activeComponent];

  const propsToPass = { [activeComponent]: allData[activeComponent] };

  return (

    <div className="App">

      <header className="App-header">

        <h1>Blogger App</h1>

        <nav>

          <button onClick={() => setActiveComponent('book')}>Book Details</button>

          <button onClick={() => setActiveComponent('blog')}>Blog Details</button>

          <button onClick={() => setActiveComponent('course')}>Course Details</button>

        </nav>

      </header>

      <main className="App-main">

        {ComponentToRender && <ComponentToRender {...propsToPass} />}

      </main>

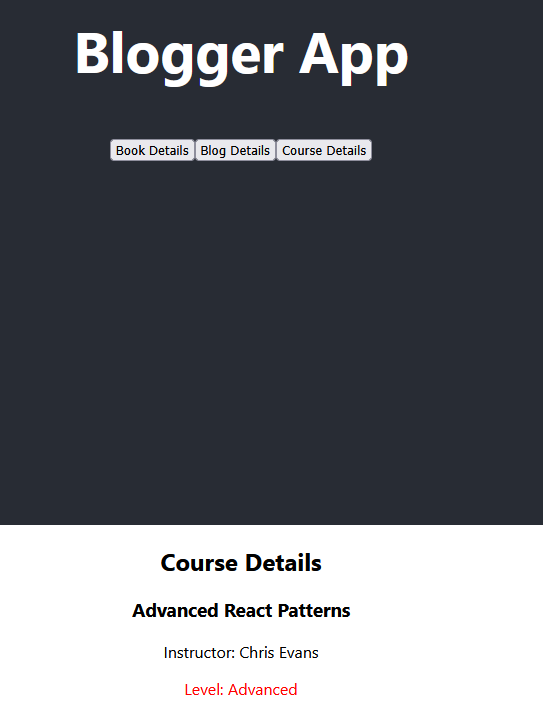
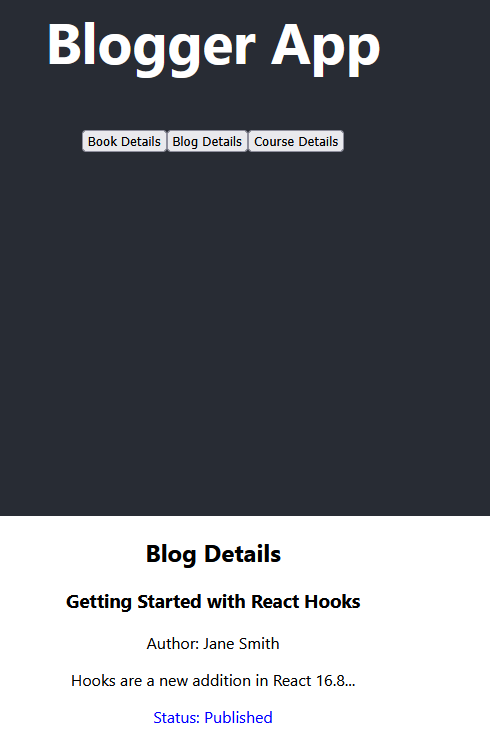
    </div>

  );

}

export default App;

**Output:**

********