Handling Events and Conditional Rendering

Handling Events in React

In React, handling events is similar to handling events in plain HTML, but there are some syntactic differences:

1. Events are named using camelCase, rather than lowercase.
2. With JSX, you pass a function as the event handler, rather than a string.

Here's an example of handling a button click event:

```jsx
import React from 'react';

function ActionButton() {
  function handleClick() {
    alert('Button was clicked!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}

export default ActionButton;
```

In this example, handleClick is a function that will be called when the button is clicked.

Conditional Rendering

In React, you can create different components that encapsulate the behavior you need. Then, you can render only some of them depending on the state of your application.

Example 1: Using if-else statements

```jsx
import React, { useState } from 'react';

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

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

function Greeting() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  return (
    <div>
      {isLoggedIn ? <UserGreeting /> : <GuestGreeting />}
      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>
        {isLoggedIn ? 'Logout' : 'Login'}
      </button>
    </div>
  );
}

export default Greeting;
```

In this example, the Greeting component renders either the UserGreeting or GuestGreeting component based on the value of isLoggedIn.

Example 2: Using logical && operator

```jsx
import React, { useState } from 'react';

function Mailbox(props) {
  const [messages, setMessages] = useState(['React', 'Re: React', 'Re:Re: React']);

  return (
    <div>
      <h1>Hello!</h1>
      {messages.length > 0 && <h2>You have {messages.length} unread messages.</h2>}
    </div>
  );
}

export default Mailbox;
```

In this example, the message count is displayed only if there are unread messages.

Lists and Keys

In React, you can build collections of elements and include them in JSX using curly braces {}.

Example: Rendering a list of items

```jsx
import React from 'react';

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );

  return (
    <ul>
      {listItems}
    </ul>
  );
}

export default NumberList;
```

In this example, NumberList component takes a list of numbers and returns an unordered list of items. Each list item needs a unique "key" prop.

Keys

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:

```jsx
import React from 'react';

function TodoList() {
  const todos = ['Learn React', 'Build a React App', 'Deploy the App'];

  return (
    <ul>
      {todos.map((todo, index) => (
        <li key={index}>{todo}</li>
      ))}
    </ul>
  );
}

export default TodoList;
```

In this example, each todo item is given a unique key based on its index in the array.

Summary

1. **Handling Events**: Use camelCase for event names and pass a function as the event handler.
2. **Conditional Rendering**: Use if-else statements or logical operators to render components conditionally.
3. **Lists and Keys**: Use the map function to render lists and provide a unique key to each list item to maintain stability.