1. Props and State

Props (Properties)

- Props are used to pass data from a parent component to a child component.
- They are immutable (read-only) and cannot be modified within the child component.
- Props help in component reusability and maintainability.

Example:

```
function Greeting(props)
{
  return <h1>Hello, {props.name}!</h1>;
}

function App()
{
  return <Greeting name="Dulal" />;
}
```

State

- State is used to manage data that can change over time within a component.
- Unlike props, state is mutable and managed using the `useState` hook.
- When state updates, the component re-renders to reflect the changes.

Example:

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

function Counter()
{
  const [count, setCount] = useState(0);

  return (

  <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

2. Event Handling

- React handles events similarly to JavaScript but uses camelCase syntax (e.g., `onClick` instead of `onclick`).
- Event handlers can be defined as inline functions or separate functions.

Example:

```
function Button()
 {
  function handleClick()
 {
    alert("Button clicked!");
  }
  return <button onClick={handleClick}>Click Me</button>;
}
```

Using Inline Function:

```
<button onClick={() => console.log("Clicked!")}>Click Me</button>
```

3. Conditional Rendering

- Conditional rendering in React allows components to render dynamically based on conditions.
- Common techniques include:
    - Using `if` statements
    - Ternary operators
    - Logical && operator

Example:

```
function UserStatus(props)
{
  if (props.isLoggedIn)
 {
    return <h1>Welcome Back!</h1>;
  } else
 {
    return <h1>Please Log In</h1>;
  }
}
```

Using Ternary Operator:

```
<h1>{isLoggedIn ? "Welcome Back!" : "Please Log In"}</h1>
```

Using Logical && Operator:

```
{isLoggedIn && <h1>Welcome Back!</h1>}
```

4. List Rendering

- Lists in React can be rendered using the `map()` function.
- Each list item should have a unique `key` prop to optimize performance.

Example:

```
function ListItems()
{
  const items = ["Apple", "Banana", "Cherry"];
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

5. Key Prop Importance

- Keys help React identify which elements have changed, added, or removed.
- Using `index` as a key should be avoided if list order may change.
- Prefer unique and stable keys.

Correct Usage:

```
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" },
];

<ul>
  {users.map(user => (
    <li key={user.id}>{user.name}</li>
  ))}
</ul>
```

Avoid Using Index as Key (If List Order May Change):

```
<li key={index}>{item}</li> // Not recommended
```

Summary:

- **Props** are immutable and passed from parent to child.
- **State** is mutable and used for component-specific data.
- **Event Handling** follows a camelCase convention and uses inline or function handlers.
- **Conditional Rendering** enables dynamic UI updates based on conditions.
- **List Rendering** uses the `map()` function, and **keys** are crucial for efficient rendering.