📌 React Component 개념과 활용

🚀 1. React Component란?

📖 개념

- **React Component**는 UI를 구성하는 **독립적이고 재사용 가능한 코드 조각**입니다.
- 컴포넌트는 HTML과 JavaScript 로직을 하나의 단위로 결합하여 관리할 수 있도록 합니다.
- React에서는 모든 UI 요소가 **컴포넌트 단위**로 만들어지며, 이를 조합하여 웹 애플리케이션을 구축합니다.
- 컴포넌트는 **재사용성(Reusability)**, **유지보수성(Manageability)**, **가독성(Readability)**을 높여줍니다.

💡 2. React Component의 종류

React에서는 크게 **함수형 컴포넌트(Function Component)** 와 **클래스형 컴포넌트(Class Component)** 두 가지 방식으로 컴포넌트를 정의할 수 있습니다.

✅ 함수형 컴포넌트 (Function Component)

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

- 함수형 컴포넌트는 `return` 문을 사용하여 UI를 렌더링하는 **순수 함수(Pure Function)** 입니다.
- `props`를 받아와서 UI를 동적으로 변경할 수 있습니다.
- **React Hooks (useState, useEffect 등)를 사용하여 상태 관리 가능**

✅ 클래스형 컴포넌트 (Class Component)

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

- 클래스형 컴포넌트는 `React.Component`를 상속받아 구현됩니다.
- `render()` 메서드에서 JSX를 반환합니다.
- 이전에는 `state`와 라이프사이클 메서드를 사용하기 위해 많이 사용되었으나, **React Hooks의 등장 이후 함수형 컴포넌트가 주로 사용됩니다.**
-

🔥 함수형 컴포넌트 vs. 클래스형 컴포넌트 비교

비교 항목	함수형 컴포넌트	클래스형 컴포넌트
선언 방식	함수(Function)	클래스(Class)
`this` 사용 여부	불필요	필요
상태 관리	`useState()` 활용	`this.state` 활용
생명주기 관리	`useEffect()` 활용	`componentDidMount()` 등 활용
성능	가볍고 빠름	상대적으로 무거움
최신 React 트렌드	✅ 권장됨	❌ 비권장됨

🛠 3. React 컴포넌트의 주요 개념

📌 1) Props (Properties)

- **Props는 부모 컴포넌트가 자식 컴포넌트에게 데이터를 전달하는 방법**입니다.
- Props는 **읽기 전용(Read-Only)** 이므로, **자식 컴포넌트에서 직접 변경할 수 없습니다.**

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

// 사용 예시
<Greeting name="John" />
```

📌 2) State (상태 관리)

- **State는 컴포넌트 내부에서 관리되는 데이터**입니다.
- State가 변경되면 해당 컴포넌트가 자동으로 리렌더링됩니다.

✅ 함수형 컴포넌트에서 `useState()` 활용

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

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

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

✅ 클래스형 컴포넌트에서 `this.state` 활용

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>Increment</button>
      </div>
    );
  }
}
```

📌 3) 컴포넌트 생명주기 (Lifecycle Methods)

✅ 클래스형 컴포넌트에서의 라이프사이클 메서드

단계	메서드	설명
마운트	`componentDidMount`	컴포넌트가 처음 화면에 나타날 때 실행
업데이트	`componentDidUpdate`	props 또는 state가 변경될 때 실행
언마운트	`componentWillUnmount`	컴포넌트가 화면에서 사라질 때 실행

✅ **함수형 컴포넌트에서 `useEffect()` 활용**

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

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

  useEffect(() => {
    console.log('Component Mounted!');
    return () => {
      console.log('Component Unmounted!');
    };
  }, []);

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

✅ `useEffect()`를 사용하면 `componentDidMount`, `componentDidUpdate`, `componentWillUnmount`를 하나로 관리할 수 있습니다.

🎯 4. React 컴포넌트 실전 적용

✅ **함수형 컴포넌트를 기본적으로 사용하며, 필요한 경우 Hooks로 상태 관리**

✅ **Props를 활용하여 부모 → 자식 데이터 전달**

✅ **State를 이용하여 동적인 UI 구현**

✅ **`useEffect()`를 활용하여 컴포넌트 생명주기 관리** ✅ **컴포넌트를 작은 단위로 나누어 재사용성을 높이기**

🏁 5. 적용

- **React의 핵심은 컴포넌트 기반 개발이며, 이를 활용하여 유지보수성과 재사용성을 극대화할 수 있음**
- **React Hooks (`useState`, `useEffect` 등)를 활용하면 더욱 직관적인 코드 작성 가능**
- **실무에서는 상태 관리 라이브러리(Redux, Recoil 등)와 조합하여 대규모 애플리케이션을 구축 가능**