State 상태 & Props 속성

React State

React에서의 state란 컴포넌트 내부에서 관리되는 상태 값을 의미합니다. 컴포넌트가 생성되고, 갱신될 때마다 변경될 수 있는 값이며, 이 값이 변경될 때마다 화면이 다시 렌더링됩니다. state는 useState Hook을 사용하여 컴포넌트 내부에서 관리할 수 있으며, setState 함수를 통해 값을 업데이트할 수 있습니다.

React에서의 state는 컴포넌트의 상태를 저장하고, 필요에 따라 다시 렌더링하는 데 사용됩니다. 사용자 인터랙션에 따라 컴포넌트 내의 상태가 변경되면, React는 이를 감지하고 변경된 상태를 바탕으로 화면을 다시 렌더링합니다. 이를 통해 동적으로 변하는 UI를 만들 수 있습니다.

React Props

Props는 React 컴포넌트에게 데이터를 전달하는 방법 중 하나입니다.

Props는 부모 컴포넌트로부터 자식 컴포넌트로 전달되며, 컴포넌트 내부에서 변경할 수 없는 읽기 전용 데이터입니다.

Props를 사용하면 컴포넌트 간의 데이터 전달이 간단하고 유지보수하기 쉬워집니다. 또한 컴포넌트의 재사용성을 높일 수 있습니다.

state lifting up : 상태 끌어올리기

React에서는 상위 컴포넌트가 하위 컴포넌트의 상태를 직접 변경할 수 없습니다. 따라서 React에서는 하위 컴포넌트에서 발생한 이벤트를 상위 컴포넌트에서 처리하도록 하는 상태 끌어올리기 패턴을 사용합니다.

예를 들어, 다음과 같은 코드가 있다고 가정해봅시다.

src > Exam3.js 를 만들고 작성합니다.

```
// Exam3.js
const Id = () => {
    const [id, setId] = React.useState("");

    return (
      <>
        <div className="wrapper">
          <label htmlFor="id">ID: </label>
          <input id="id" />
        </div>
      </>
    );
  };

const Pw = () => {
  const [pw, setPw] = React.useState("");

  return (
    <>
      <div className="wrapper">
        <label htmlFor="pw">PW: </label>
        <input type="password" id="pw" />
      </div>
    </>
  );
};

const Exam3 = () => {
  return (
    <>
      <Id />
      <Pw />
      <div className="wrapper">
          <button disabled={true}>
              Login
          </button>
      </div>
    </>
  );
};

export default Exam3;
```

위 예제는 아이디와 패스워드를 입력해 로그인을 할 수 있는 기능을 하는 **부모(Exam3)** 컴포넌트와 **자식(Id, Pw)** 컴포넌트 그리고 로그인 버튼으로 이루어져있습니다.

로그인이 되었을 때 Login 버튼의 disabled를 false로 바꾸기 위해서는 상태 끌어올리기를 적용시켜야합니다. 현재 상태로는 Id와 Pw의 부모인 Exam3이 상태를 알 수 있는 방법이 없

습니다. 하나의 컴포넌트에서 다른 컴포넌트의 state를 필요로 하면, **가장 가까운 공통 조상으로 state를 들어 올려** 사용할 수 있다는 개념입니다.

```
...

const Exam3 = () => {
    const [id, setId] = React.useState("");
    const [pw, setPw] = React.useState("");

    const onChangeId = (event) => {
      setId(event.target.value);
    };

    const onChangePw = (event) => {
      setPw(event.target.value);
    };

    return (
      <>
        <Id onChangeId={onChangeId} />
        <Pw onChangePw={onChangePw} />
        <div className="wrapper">
          <button disabled={id.length === 0 || pw.length === 0}>
            Login
          </button>
        </div>
      </>);
  };

export default Exam3;
```

자식 컴포넌트인 Id와 Pw에 있는 아이디값, 패스워드 값, 그리고 함수를 부모 컴포넌트인 Exam3 컴포넌트로 끌어올립니다. 다음 각 자식 컴포넌트에 **필요한 값이나 함수를 프로퍼티**로 넘겨줍니다.

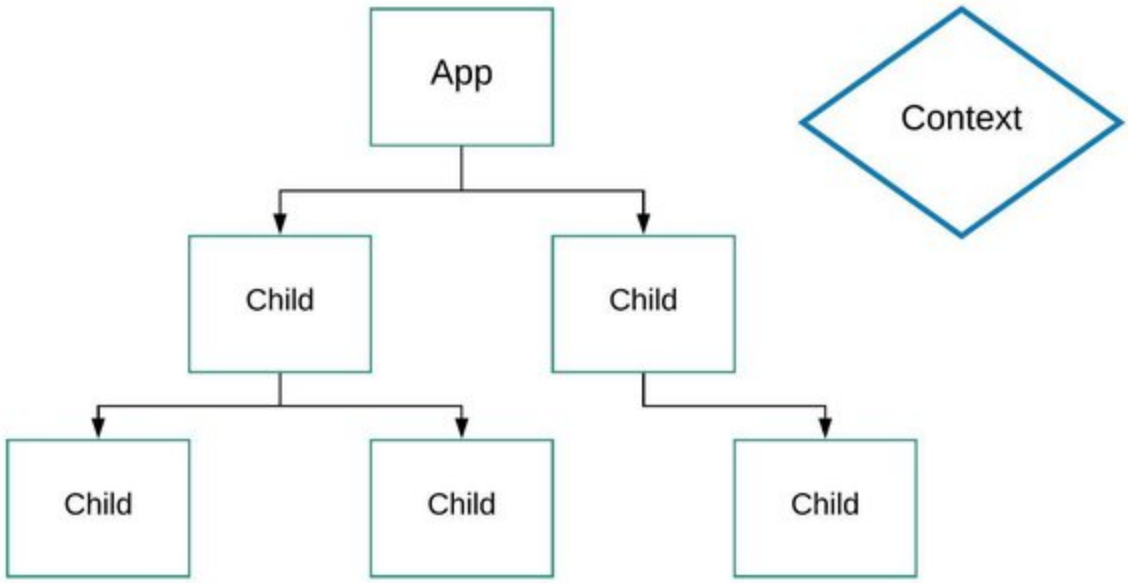
```
const Id = ({ onChangeId }) => {
  return (
    <>
      <div className="wrapper">
        <label htmlFor="id">ID: </label>
        <input id="id" onChange={onChangeId} />
      </div>
    </>);
};

const Pw = ({ onChangePw }) => {
```

```
  return (
    <>
      <div className="wrapper">
        <label htmlFor="pw">PW: </label>
        <input type="password" id="pw" onChange={onChangePw} />
      </div>
    </>);
};

...
```

```
// Exam.js 상태끌어올리기 예제 전체코드
import React from "react";

const Id = ({ onChangeId }) => {
    return (
      <>
        <div className="wrapper">
          <label htmlFor="id">ID: </label>
          <input id="id" onChange={onChangeId} />
        </div>
      </>);
  };

  const Pw = ({ onChangePw }) => {
    return (
      <>
        <div className="wrapper">
          <label htmlFor="pw">PW: </label>
          <input type="password" id="pw" onChange={onChangePw} />
        </div>
      </>);
};

const Exam3 = () => {
    const [id, setId] = React.useState("");
    const [pw, setPw] = React.useState("");

    const onChangeId = (event) => {
      setId(event.target.value);
    };

    const onChangePw = (event) => {
      setPw(event.target.value);
    };

    return (
      <>
        <Id onChangeId={onChangeId} />
        <Pw onChangePw={onChangePw} />
        <div className="wrapper">
          <button disabled={id.length === 0 || pw.length === 0}>
            Login
```

```
          </button>
        </div>
      </>);
  };

export default Exam3;
```

Props Drilling : 상태 내리꽂기



Props drilling은 React에서 사용되는 용어로, **Props를 통해 데이터를 전달할 때, 하위 컴포넌트에서 필요하지 않은 Props를 계속해서 전달하는 것**을 의미합니다. 이는 코드의 가독성을 떨어뜨리고, 유지보수를 어렵게 만들 수 있습니다. 따라서 Props drilling을 최소화하기 위해서는, 필요한 Props만을 전달하고, 필요하지 않은 Props는 하위 컴포넌트에서 직접 접근하는 것이 좋습니다. 이를 위해서는 **React Context나 Redux와 같은 상태 관리 라이브러리**를 사용하는 것도 좋은 방법입니다.

Props는 다음과 같은 방식으로 사용할 수 있습니다.

src > Exam4.js를 작성합니다.

```
// Exam4.js
function MyComponent(props) {
  return <div>{props.name}</div>;
}
```

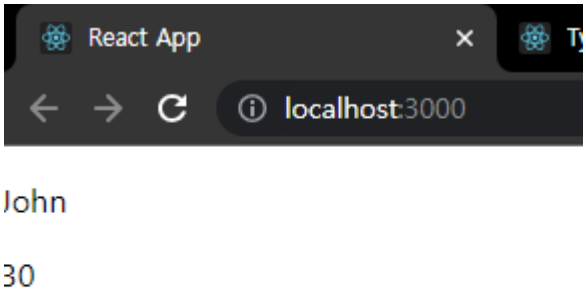
```
function Exam4() {
  return <MyComponent name="John" />;
}
```

위 예제에서는 `name`이라는 Props를 MyComponent에 전달하고 있습니다. MyComponent 내에서는 `props.name`을 통해 전달된 값을 사용할 수 있습니다.

Props는 객체 형태로 전달되며, 여러 개의 Props를 전달할 수도 있습니다.

```
function MyComponent(props) {
  return (
    <div>
      <p>{props.name}</p>
      <p>{props.age}</p>
    </div>
  );
}

function Exam4() {
  return <MyComponent name="John" age={30} />;
}
```

위 예제에서는 `name`과 `age`라는 두 개의 Props를 MyComponent에 전달하고 있습니다. MyComponent 내에서는 `props.name`과 `props.age`를 통해 전달된 값을 사용할 수 있습니다.



Props는 컴포넌트 내부에서 변경할 수 없기 때문에, Props를 변경하려면 부모 컴포넌트에서 Props를 업데이트해야 합니다.

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

function MyComponent(props) {
    return (
        <div>
            <p>{props.name}</p>
            <p>{props.age}</p>
        </div>
    );
  }

function Exam4() {
    const [name, setName] = useState("John");

    const handleClick = () => {
      setName("Jane");
    };

    return (
      <>
        <MyComponent name={name} />
        <button onClick={handleClick}>Change Name</button>
      </>
    );
}

export default Exam4;
```

위 예제에서는 Exam4 컴포넌트에서 `name`이라는 State를 관리하고 있습니다.

MyComponent에는 `name`이라는 Props로 전달되고 있습니다. Change Name 버튼을 클릭하면 `name` State가 업데이트되고, MyComponent는 새로운 `name` Props를 받아와서 화면에 표시합니다.

React Props를 사용하면 컴포넌트 간의 데이터 전달이 간단하고 유지보수하기 쉬워집니다. 이를 잘 활용하면 더욱 유연하고 재사용성이 높은 컴포넌트를 만들 수 있습니다.

props 예제 응용편

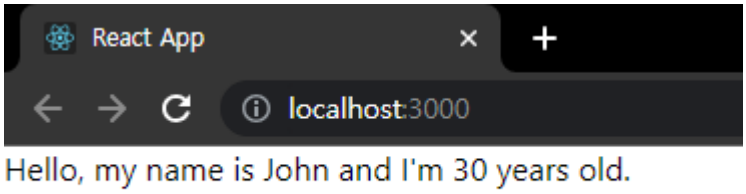
src > Exam5.js 에 작성합니다.

```
// Exam5.js
function MyComponent(props) {
    const { name, age } = props;
    const greeting = `Hello, my name is ${name} and I'm ${age} years old.`;
    return <div>{greeting}</div>;
}
```

```
function Exam5() {
    const userData = { name: 'John', age: 30 };
    return <MyComponent {...userData} />;
}

export default Exam5;
```

위 예제에서는 MyComponent에 `name`과 `age` Props를 전달하고 있습니다. 그러나 이전 예제와는 달리, 이번에는 Props를 객체 형태로 전달하고 있습니다. 이렇게 전달된 Props는 MyComponent에서 비구조화 할당을 통해 `name`과 `age`라는 변수에 할당됩니다. 이후에는 이 변수들을 자유롭게 사용할 수 있습니다.

Exam5 컴포넌트에서는 userData라는 객체를 생성하고, 이를 MyComponent에 전달하고 있습니다. 이 때, 전개 연산자(...)를 사용하여 객체를 Props로 전달하고 있습니다. 이는 객체 내부의 모든 값들을 Props로 전달하는 방법입니다.



이처럼 Props를 객체 형태로 전달하면, 코드의 가독성을 높일 수 있습니다. 또한 Props를 전개 연산자를 사용하여 한 번에 전달할 수 있기 때문에, 코드의 양도 줄일 수 있습니다.