# 230111_2반_실습

## Hook

### React의 Hook 이란?

- 리액트 v16.8 에 새로 도입된 기능
- 함수형태의 컴포넌트에서 사용되는 기술
- 함수형 컴포넌트에서도 상태 관리를 할 수 있는 `useState`, 그리고 렌더링 직후 작업을 설정하는 `useEffect` 등의 기능 등을 제공

### Hook의 기능

- 함수형 컴포넌트가 클래스형 컴포넌트의 기능을 사용할 수 있도록 해주는 기능
- useState와 useEffect를 사용하여 특징적으로 state와 lifecycle과 같은 기능을 사용 가능하게 해준다.

### side effect 란?

- 컴포넌트가 화면에 1차로 렌더링된 이후에 **비동기로 처리되어야 하는 부수적인 효과**들

## props

### props 란?

- 프로퍼티, props(properties의 줄임말) 라고 한다.
- 상위 컴포넌트가 하위 컴포넌트에 값을 전달할때 사용한다.
- 프로퍼티는 수정할 수 없다는 특징이 있다.

### component 정의

1. 함수를 사용하여 정의

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

2. class를 사용하여 정의

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

## 기본값 설정

- `defaultProps`를 사용하여 props의 기본값을 설정

### Hello.js

```
import React from 'react';

function Hello({ color, name }) {
  return <div style={{ color }}>안녕하세요 {name}</div>
}

Hello.defaultProps = {
  name: '이름없음'
}

export default Hello;
```

App.js

```
import React from 'react';
import Hello from './Hello';

function App() {
  return (
    <>
      <Hello name="react" color="red"/>
      <Hello color="pink"/>
    </>
  );
}

export default App;
```

## props.children

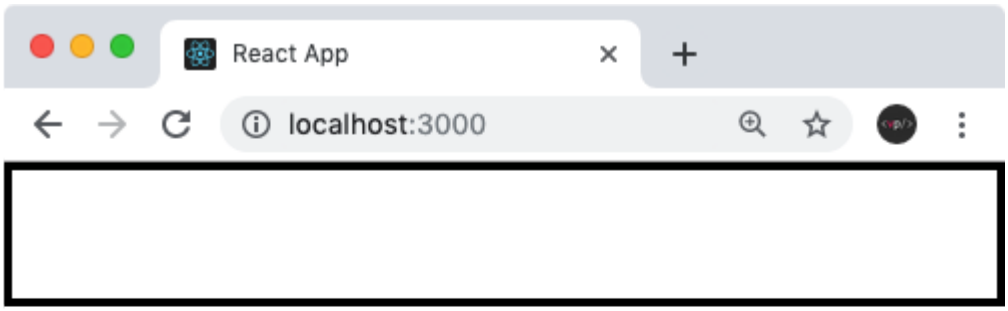
- 컴포넌트 태그 사이에 넣은 값을 조회할 때 사용

### Wrapper.js

```
import React from 'react';

function Wrapper() {
  const style = {
    border: '2px solid black',
    padding: '16px',
  };
  return (
    <div style={style}>

    </div>
  )
}

export default Wrapper;
```

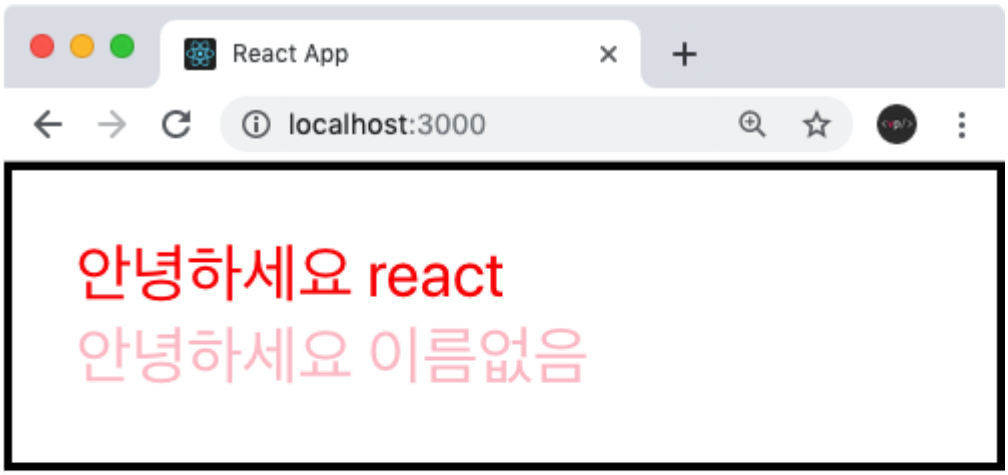
### App.js

```
import React from 'react';
import Hello from './Hello';
import Wrapper from './Wrapper';

function App() {
  return (
    <Wrapper>
      <Hello name="react" color="red"/>
      <Hello color="pink"/>
    </Wrapper>
  );
}

export default App;
```




### Wrapper.js

```
import React from 'react';

function Wrapper({ children }) {
  const style = {
    border: '2px solid black',
    padding: '16px',
  };
  return (
    <div style={style}>
      {children}
    </div>
  )
}

export default Wrapper;
```




# setState()

## state 란?

- 일반적으로 컴포넌트의 내부에서 변경 가능한 데이터를 관리해야할 때에 사용 한다.
- 프로퍼티(props)의 특징은 컴포넌트 내부에서 값을 바꿀 수 없다는 것이었다.
- 하지만 값을 바꿔야 하는 경우도 분명 존재하며, 이럴때 state라는 것을 사용한다.
- 값을 저장하거나 변경할 수 있는 객체로 보통 이벤트와 함께 사용된다.
- 컴포넌트에서 동적인 값을 상태(state)라고 부르며, 동적인 데이터를 다룰 때 사용된다 볼 수 있다.

> ※ 참고
>
> 리액트 16.8 이전 버전에서는 함수형 컴포넌트에서는 상태를 관리할 수 없었다.
>
> 즉 State, Life Cycle 등등 이 없기 때문에 클래스형 컴포넌트를 사용 했었다.
>
> 리액트 16.8부터 Hooks 라는 기능이 도입되면서 함수형 컴포넌트에서도 상태를 관리할 수 있게 되었다.

> useState Hook을 사용하여 State 사용이 가능하다.

## state와 props의 차이점

- `props`는 (함수 매개변수처럼) 컴포넌트에 전달되는 반면 `state`는 (함수 내에 선언된 변수처럼) 컴포넌트 안에서 관리

props와 state는 Javascript 객체입니다. 두 객체 모두 렌더링 결과물에 영향을 주는 정보를 갖고 있습니다.
props는 컴포넌트에 전달되며, state는 컴포넌트 안에서 관리되는 차이점이 있습니다.
더 자세히 알고 싶다면 여기를 참고하세요

`props` ("properties"의 줄임말) 와 `state` 는 일반 JavaScript 객체입니다. 두 객체 모두 렌더링 결과물에 영향을 주는 정보를 갖고 있는데, 한 가지 중요한 방식에서 차이가 있습니다. `props`는 (함수 매개변수처럼) 컴포넌트*에* 전달되는 반면 `state`는 (함수 내에 선언된 변수처럼) 컴포넌트 *안에서* 관리됩니다.

언제 `props`와 `state`를 사용하는지 더 알고 싶다면 아래의 자료를 확인해보세요.

- Props vs State
- ReactJS: Props vs. State

## setState() 란?

- setState()는 리액트의 함수형 컴포넌트 내에서 상태를 관리하기 위해 사용하는 hooks인 useState()를 통해 반환되는 함수이다.
  constructor 안에서 state값을 바꾸는 것은 가능 하지만 생성된 후 state값을 바꾸는 방법은 `setState`
  를 사용한다.

```
this.state.mode = 'welcome'
//안됨 -> 리액트 모르게 바꾸는 것 -> 렌더링이 안됨

this.setState({
  mode:'welcome'
})
//리액트에게 알려줘서 다시 렌더링된다.
```

## setState()의 특징

1. 기본적으로 비동기로 동작한다.
2. 연속적으로 호출했을 때 리액트 내부적으로는 BATCH 처리를 한다.
3. state 객체를 넘겨줄 수 있을 뿐만 아니라 새로운 state를 반환하는 함수를 인자로 넘겨줄 수 있다.

## setState 사용법

1. import

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

2. 선언 방법

```
const [state, setState] = useState(initialState);
const [ 데이터, 데이터변경함수 ] = useState(초기값(생략가능));
```

- react 모듈에서 { useState }를 불러오고 useState()를 선언해서 사용면 된다.
- useState의 변수값이 바뀌면 컴포넌트가 새롭게 렌더링 된다.
- 예제

```
const [count, setCount] = useState(0);
```

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

const Main = () => {
    const [ cnt, setCnt ] = useState(0)
    const updateCnt = () => setCnt(cnt + 1);
    const clearCnt = () => setCnt(0);
    return (
        <div>
            클릭한 횟수는 {cnt}번 입니다.
            <div>
                <button onClick={updateCnt}> 클릭해 보세요! </button>
                <button onClick={clearCnt}> 초기화 하기! </button>
            </div>
        </div>
    );
};

export default Main;
```

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

const Main = () => {
  // let myName = "Snoopy"; // useState를 사용하여 변경
  const [ myName, setMyName ] = useState("Snoopy");

  function changeName() {
      /*
  myName = myName === "Snoopy" ? "Woodstock" : "Chalri";
      console.log(myName);
      document.getElementById("name").indderText = myName;
  */
  setMyName(myName = myName === "Snoopy" ? "Woodstock" : "Chalri");
  }

    return (
        <div>
      <h1>안녕하세요. {myName} 입니다.</h1>
      {/* <button
        onClick={() => {
          setMyName(myName = myName === "Snoopy" ? "Woodstock" : "Chalri");
        }}
      >Change</button> */}
            <button onClick={changeName}>Change</button>
        </div>
    );
};

export default Main;
```

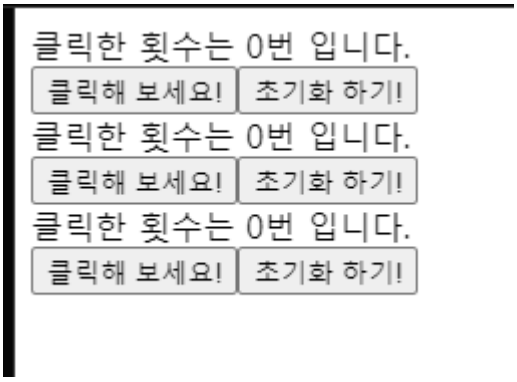
3. 객체(Object)도 상태변수(State Value)로 사용 가능하다.

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

const Main = () => {
    const [ state, setState ] = useState({cnt : 0})
    const updateCnt = val =>
    setState({
        ...state,
        [val] : state[val] + 1
    })
    const { cnt } = state
    return (
        <div>
            클릭한 횟수는 {cnt}번 입니다.
            <div>
                <button onClick={updateCnt.bind(null, 'cnt')}> 클릭해 보세요! </butto
n>
            </div>
        </div>
    );
};

export default Main;
```

4. useState를 호출하여 반환된 업데이트 함수는 setState와 유사하게 사용 가능 하다.

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

const Main = () => {
    const [ cnt, setCnt ] = useState(0)
    // const updateCnt = () => setCnt(cnt + 1);
    // const clearCnt = () => setCnt(0);
    return (
        <div>
            클릭한 횟수는 {cnt}번 입니다.
            <div>
            <button onClick={() => setCnt(prevCnt => prevCnt + 1)}> 클릭해 보세요! </
button>
            <button onClick={() => setCnt(0)}> 초기화 하기! </button>
            </div>
        </div>
    );
};

export default Main;
```

```
import React, { Component } from 'react';
import Main from './component/Main';
import Wrapper from './component/Wrapper';

function App() {
  return (
    <div>
      <Main />
      <Main />
      <Main />
    </div>
  );
}

export default App;
```



# useEffect

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });
}
```

- useEffect Hook을 이용하여 우리는 React에게 컴포넌트가 렌더링 이후에 어떤 일을 수행해야하는 지를 말합니다.
- 기본적으로 첫번째 렌더링과 이후의 모든 업데이트에서 수행

## 사용법

1. import 해오기

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

2. useEffect 선언하기
   - useEffect의 매개변수는 익명함수 ()=>{} 와 빈배열 [] 두가지가 들어간다.

```
useEffect(() => {
    document.title = '업데이트 횟수: ${count}';
}, []);
```

3. 예제

```
function myComponent(){
    const [count, setCount] = useState(0); // count라는 state 추가를 해주자

    useEffect(() => { // useEffect로 count가 업데이트될 때 마다 값을 보여주자
        document.title = '업데이트 횟수: ${count}';
    }, []); // 빈배열을 추가해서 불필요한 반복렌더링을 막자

    return // 버튼에 onClick이벤트가 발생하면 count를 1씩 더해주자
        <button onClick={() => setCount(count+1)}>
            increase
        </button>;
}
```

# Event

## React와 HTML에서 이벤트처리의 차이점

1. React 이벤트 이름은 소문자 대신 camelCase를 사용

```
onClick={changename} (x)
onClick={changeName} (o)
```

2. JSX에 문자열 대신 함수를 전달

```
onClick="changeName()" (x)
onClick={changeName} (o)
```

**html**

```
<button onclick="activateButton()">클릭하세요</button>
```

React

```
<button onClick={activateButton}>클릭하세요</button>
```

## 이벤트와 프로퍼티

| 이벤트 명 | JSX DOM 이벤트 프로퍼티 | 이벤트 호출 시점 |
|---|---|---|
| click | onClick | 엘리먼트에 마우스나 키보드가 클릭 된 경우 |
| change | onChange | 엘리먼트의 내용이 변경된 경우 |
| submit | onSubmit | 폼의 데이터가 전송될 때 |
| keydown | onKeyDown | 키보드 버튼이 눌린 경우( 값 입력전에 발생하며, shift, alt, ctrl 등 특수키에 동작한다. ) (한/영, 한자 등은 인식불가) |
| keyup | onKeyUp | 키보드 버튼이 눌렸다 뗀 경우( 값 입력후에 발생하며, onKeyDown 과 동일하게 동작한다. ) |
| keypress | onKeyPress | 키보드 버튼이 눌려져 있는 경우( 실제 글자가 작성될때 이벤트이며, ASCII 값으로 사용되어 특수키를 인식 못한다.) |
| focus | onFocus | 엘리먼트가 포커스 된 경우 |

| 이벤트 명 | JSX DOM 이벤트 프로퍼티 | 이벤트 호출 시점 |
| --- | --- | --- |
| blur | onBlur | 엘리먼트가 포커스가 사라진 경우 |
| mousemove | onMouseMove | 엘리먼트 위에서 마우스 커서가 움직일 때 |
| mousedown | onMouseDown | 마우스 버튼이 클릭되기 시작할 때 |
| mouseup | onMouseUp | 마우스 버튼 클릭이 끝날때 |

## 이벤트 핸들러

```
<button onClick={activateButton}>클릭하세요</button>
```

- Props : `onClick`, `on` 접두사 사용
- Function Names : `handleClick`, `handle` 접두사 사용

## 이벤트 예제

1. 클래스형 컴포넌트

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
    //콜백에서 `this`가 작동하려면 바인딩을 해줘야한다
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      //render() 함수 안에서 this 값은 render() 함수가 속한 컴포넌트를 가리킴
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

2. 함수형 컴포넌트

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

function NumberTest() {
  const [num, setNumber] = useState(0);

  const increase = () => {
    setNumber(num+1);
  }

  return (
    <div>
      <h1>Number Test</h1>
      <h2>{num}</h2>
      <button onClick={increase}>증가</button>
    </div>
  );
}

export default NumberTest;
```

```
import React, { useState } from "react";
import "../styles.css";

const Banner = () => {
  const [isVisible, setIsVisible] = useState(true);

  const getAlert = () => alert("ALERT!!");

  const closeBanner = () => setIsVisible(false);

  return (
    <div
      className="banner"
      onClick={getAlert}
      style={{ display: isVisible ? "flex" : "none" }}
    >
      이 곳을 클릭해서 alert창 띄우기
      <button className="close-btn" onClick={closeBanner}>
        닫기
      </button>
    </div>
  );
};

export default Banner;
```

```
const closeBanner = (e) => {
  e.stopPropagation(); //이벤트 전파 방지
  setIsVisible(false);
}
```