

# Web Programming

**React programming**



# Component State

## 상태(state)

- State란 렌더링 결과물에 영향을 주는 정보
- state는 간단하게 말해서 변수이다. 하지만 `const`, `let` 등으로 선언한 변수와 다르게 값이 변하면 관련 있는 컴포넌트들이 재렌더링되어 화면이 바뀐다.
- state는 컴포넌트의 내부에서 변경 가능한 데이터를 다루기 위해 사용하는 객체이다.
- 일반적으로 컴포넌트의 내부에서 변경 가능한 데이터를 관리해야 할 때에 사용 한다.
- 프로퍼티(props)의 특징은 컴포넌트 내부에서 값을 바꿀 수 없다는 것이었다.
- 하지만 값을 바꿔야 하는 경우도 분명 존재하며, 이럴 때 state라는 것을 사용한다.
- 값을 저장하거나 변경할 수 있는 객체로 보통 이벤트와 함께 사용된다.
- State는 컴포넌트 내에서 지속적으로 변경이 일어나는 값을 관리하기 위해 사용한다.
- 개발자가 의도한 동작에 의해 변할 수도 있고 사용자의 입력에 따라 새로운 값으로 변경될 수도 있다.
- State 값이 변경되고 재 렌더링이 필요한 경우에 React가 자동으로 계산하여 변경된 부분을 렌더링 한다.
- state는 컴포넌트의 현재 상황에 대한 정보를 나타내기 위해 리액트에서 쓰는 일반 자바스크립트 객체다. 함수에 선언된 모든 변수와 마찬가지로 컴포넌트에서 관리된다.



# Component State

## 상태(state)

- state는 상태라는 뜻을 가지고 있다.
- 리액트에서의 state는 리액트 Component의 상태를 의미한다.
- 상태라는 단어가 정상, 비정상을 나타내는 것이라기보다 리액트 Component의 데이터라는 의미가 더 가깝다.
- 리액트에서의 State : React Component의 변경 가능한 데이터
- 이 state는 사전에 미리 정해진 것이 아니라 React Component를 개발하는 각 개발자가 직접 정의해서 사용한다.
- 자바스크립트에서는 변수를 사용한다.
- 리액트에서는 상태(state)를 사용한다고 생각하면 이해하기 쉽다.
- State를 정의할 때 중요한 점은 꼭 렌더링이나. 데이터 흐름에 사용되는 것만 state에 사용해야 한다.
- State가 변경될 경우 Component가 재렌더링되기 때문에 렌더링과 데이터 흐름에 관련 없는 값을 포함하면 불필요한 경우에 Component가 다시 렌더링 되어 성능을 저하시킬 수 있기 때문이다.
- 시간이 지남에 따라 변하는 데이터
- 컴포넌트의 메모리
- 모든 컴포넌트에 state를 추가하고 업데이트를 할 수 있다



# Component State

## 상태(state)

- Component 내부에서 관리하는 데이터로, 변경이 가능한 값이다.
- state는 Component 내부에서 관리하며, 상태에 따라 변하는 동적 데이터 이다.
- state는 props와 다르게 자동으로 생성되지 않아 명시적으로 state 를 기술 해야 한다.
- React.js에서 유동적인 데이터를 사용할 때 state라는 것을 사용한다.
- 컴포넌트 내에 별도의 상태가 필요할 때 사용한다.
- React 컴포넌트에서 state는 컴포넌트 내부에서 관리되는 값으로서 변경 가능하다.
- 이 값은 컴포넌트가 렌더링되는 동안 변할 수 있으며, 변경될 때마다 컴포넌트가 다시 렌더링된다.
- 즉, React에서 state는 컴포넌트 내에서 관리되는 상태 데이터이며, 컴포넌트 내부에서 변경할 수 있고, 컴포넌트의 렌더링 결과를 결정한다.
- 함수 컴포넌트에서 내부적으로 상태를 관리해야 하는 일이 필요하다. 이를 위해 필요한 것이 state다
- state는 컴포넌트의 내부에서 변경 가능한 데이터를 다루기 위해 사용하는 객체라고 할 수 있다
- const, let 등으로 선언한 변수와 다르게 state는 값이 변하면 관련 있는 컴포넌트들이 re-rendering 되어 화면이 바뀐다



# State를 사용하는 이유

## 일반 변수를 사용하는 경우

- 변수는 변경되어도 자동으로 화면이 바뀌지 않는다.
- 하지만 state는 변경되면 자동으로 화면이 바뀌기 때문에 state를 사용한다.
- 즉 유동적인 변수를 사용할 때 화면에 그려지는 변수도 정상적으로 변경되길 원한다면 사용한다.

Counter.js

```
import React from 'react';

const Counter = () => {
  let count = 0;

  const plus = () => {
    count = count + 1
    console.log(count); // 제대로 증가함
  }
  const minus = () => {
    count = count - 1
    console.log(count); // 제대로 감소함
  }
}
```

```
return (
  <div className='container' style={{margin: 15}}>
    <h2 className='int'>{ count }</h2>
    <button className='plus' onClick={plus}>+</button>
    <button className='minus' onClick={minus}>-</button>
  </div>
)
}

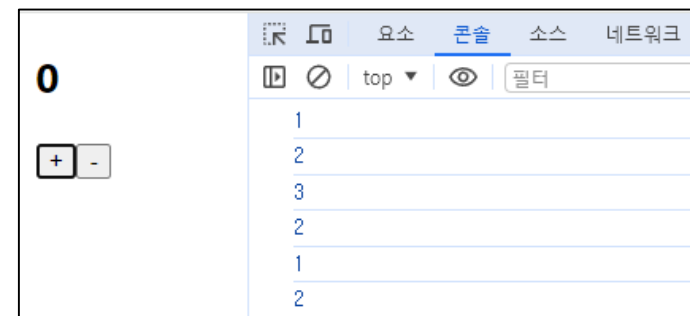
export default Counter;
```



# State를 사용하는 이유

## 일반 변수를 사용하는 경우

- 화면은 변경되지 않는다.
- 그렇다고 count가 변경되지 않은 것은 아니다. (F12)
- plus 함수와 minus 함수를 실행한 후에 콘솔로 count 값을 출력해 보면 count 값은 정상적으로 변경됨을 알 수 있다.
- 그렇다면 화면이 변경되지 않은 이유는 무엇일까?
- 화면이 변경되지 않은 이유는 일반 변수를 사용했기 때문이다.
- 일반 변수는 변경되어도 자동으로 화면이 재렌더링되지 않는다.
- 하지만 state는 다르다.
- 리액티브한 프론트엔드에서 상태는 단순한 변수가 아니라 이 값이 변했을 때 화면에 반영되도록 연결된 것을 상태라고 한다.
- 그래서 state가 아닌 일반 변수는 바뀌어도 화면이 변하지 않는 것이다.





# State를 사용하는 이유

## state를 사용하는 경우

- 함수가 실행될 때마다 setCount로 기존의 count 값을 count+1 혹은 count-1로 변경해 주었다.
- 일반 변수가 아닌 state를 사용하면 변수 값이 변경되었을 때 화면이 의도대로 재렌더링됨을 알 수 있다.

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

Counter.js

```
const Counter = () => {  
  const [count, setCount] = useState(0);  
  console.log(count);
```

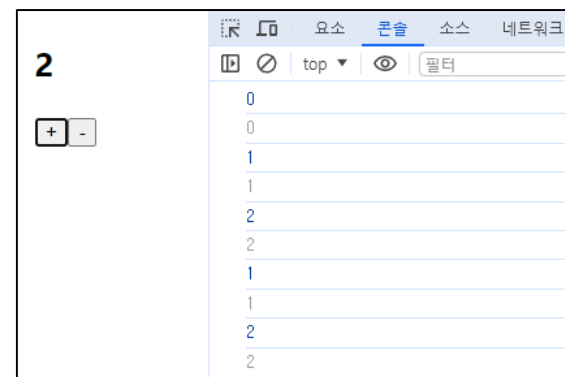
```
  const plus = () => {  
    setCount(count + 1);  
  }
```

```
  const minus = () => {  
    setCount(count - 1);  
  }
```

```
  return (  
    <div className='container'>  
      <h2 className='int'>{ count }</h2>  
      <button className='plus' onClick={plus}>+</button>  
      <button className='minus' onClick={minus}>-</button>  
    </div>  
  )  
}
```

```
export default Counter;
```

참고로 리액트 컴포넌트는 부모 컴포넌트가 리렌더링 되면 자식 컴포넌트 또한 리렌더링 된다(바뀐 내용이 없다 할지라도!).





# setState

## setState는 비동기적으로 작동한다.

- setState는 비동기적으로 동작하는데, setState 바로 아래에 console.log로 count를 출력해 봤을 때 확인할 수 있다.
- 분명히 setCount로 count를 변경했는데, 변경한 후에 console.log로 찍어보니 값이 바로 바뀌지 않는다.

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

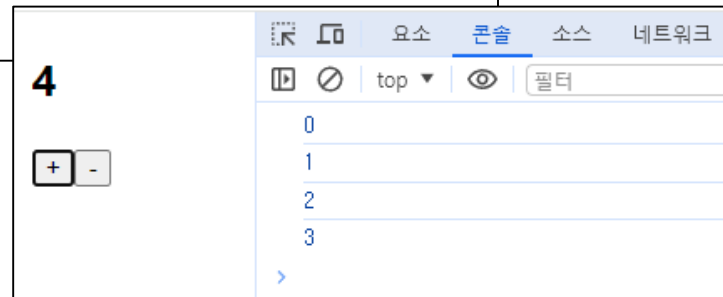
const Counter = () => {
  const [count, setCount] = useState(0);

  const plus = () => {
    setCount(count + 1);
    console.log(count);
  }
  // setCount로 count를 변경한 후 바로 콘솔에 찍었다
  }

  const minus = () => {
    setCount(count - 1);
    console.log(count);
  }
  // setCount로 count를 변경한 후 바로 콘솔에 찍었다
  }
```

```
return (
  <div className='container'>
    <h2 className='int'>{ count }</h2>
    <button className='plus' onClick={plus}>+</button>
    <button className='minus' onClick={minus}>-</button>
  </div>
)
}

export default Counter;
```







# setState

## setState는 비동기적으로 작동한다.

- 그 이유는 setState가 비동기이기 때문이다.
- 동기(Synchronous : 동시에 발생하는)
  - 요청을 보냈다면, 응답을 받아야 다음 동작이 이루어진다.
  - 순차적으로 실행되기 때문에, 어떤 작업이 수행 중이라면 다음 작업은 대기해야 한다.
  - 블로킹(작업 중단)이 발생한다.
- 비동기(Asynchronous : 동시에 발생하지 않는)
  - 작업 종료 여부에 관계없이 다음 작업을 실행한다.
  - 그러므로 동기 방식과는 달리 실행 순서를 보장하지 않는다.
  - 블로킹이 발생하지 않는다.
- setCount는 이벤트 핸들러 안에서 현재 state의 값에 대한 변화를 요청하기만 하는 것이라서 이벤트 핸들러가 끝나고 리액트가 상태를 바꾸고 화면을 다시 그리기를 기다려야 한다.



# setState

## setState는 왜 비동기적으로 동작할까

- state는 값이 변경되면 리렌더링이 발생하는데, 변경되는 state가 많을수록 리렌더링이 계속 일어나고 속도도 저하되는 등, 성능적으로 문제가 많을 것이다.
- 그래서 16ms 동안 변경된 상태 값들을 모아서 한 번에 리렌더링을 진행하는데 이를 batch(일괄) update라고 한다.

```
App.js
1 import { useState } from 'react';
2
3 export default function Counter() {
4   const [number, setNumber] = useState(0);
5
6   return (
7     <>
8       <h1>{number}</h1>
9       <button onClick={() => {
10         setNumber(number + 1);
11         setNumber(number + 1);
12         setNumber(number + 1);
13       }}>+3</button>
14     </>
15   )
16 }
```



Queueing a Series of State Updates



# setState

## setState는 비동기적으로 작동한다.

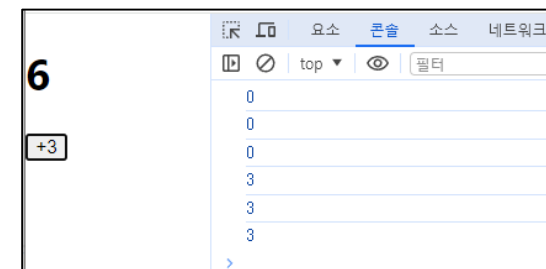
- 리액트는 이벤트 핸들러가 닫히는 시점에 setState를 종합하여 한 번에 처리한다.
- state도 결국 객체이기 때문에, 같은 키 값을 가진 경우라면 가장 마지막 실행값으로 덮어쓰워지는데 이는 객체를 합치는 함수인 Object.assign()에서 확인할 수 있다.
- 아래처럼 plus 함수 안에 setCount를 세 번 썼을 때, 1+2+3로 6씩 증가하는 것이 아니라 마지막 setCount의 결과인 3씩 증가하게 된다.

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

```
export default function Counter() {  
  const [number, setNumber] = useState(0);
```

App.js

```
  return (  
    <>  
    <h1>{number}</h1>  
    <button onClick={() => {  
      setNumber(number + 1);  
      console.log(number)  
      setNumber(number + 2);  
      console.log(number)  
      setNumber(number + 3);  
      console.log(number)  
    }}>+3</button>  
    </>  
  )  
}
```





# setState

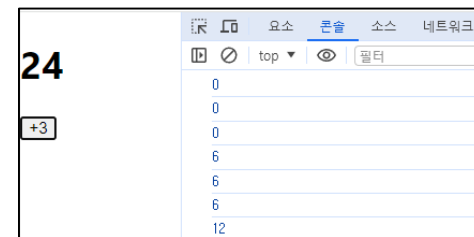
## setState는 비동기적으로 작동한다.

- 만약 6씩 더하고 싶은 거라면, 이 문제는 이렇게 하면 해결된다.
- `setCount(count+1)`에서 `count`는 렌더링 시작 시점의 `count`이기 때문에 `count`가 최근에 바뀌었어도 반영되지 않는다.
- 하지만 이렇게 콜백 함수를 사용하면 항상 최신의 값을 인자로 받아와서 처리하기 때문에 `setCount(count => count + 1)`를 쓰면 최신 값을 받아서 처리할 수 있다.

```
import { useState } from 'react';  
  
export default function Counter() {  
  const [number, setNumber] = useState(0);
```

App.js

```
  return (  
    <>  
    <h1>{number}</h1>  
    <button onClick={() => {  
      setNumber(number => number + 1);  
      console.log(number)  
      setNumber(number => number + 2);  
      console.log(number)  
      setNumber(number => number + 3);  
      console.log(number)  
    }}> +3</button>  
    </>  
  )  
}
```

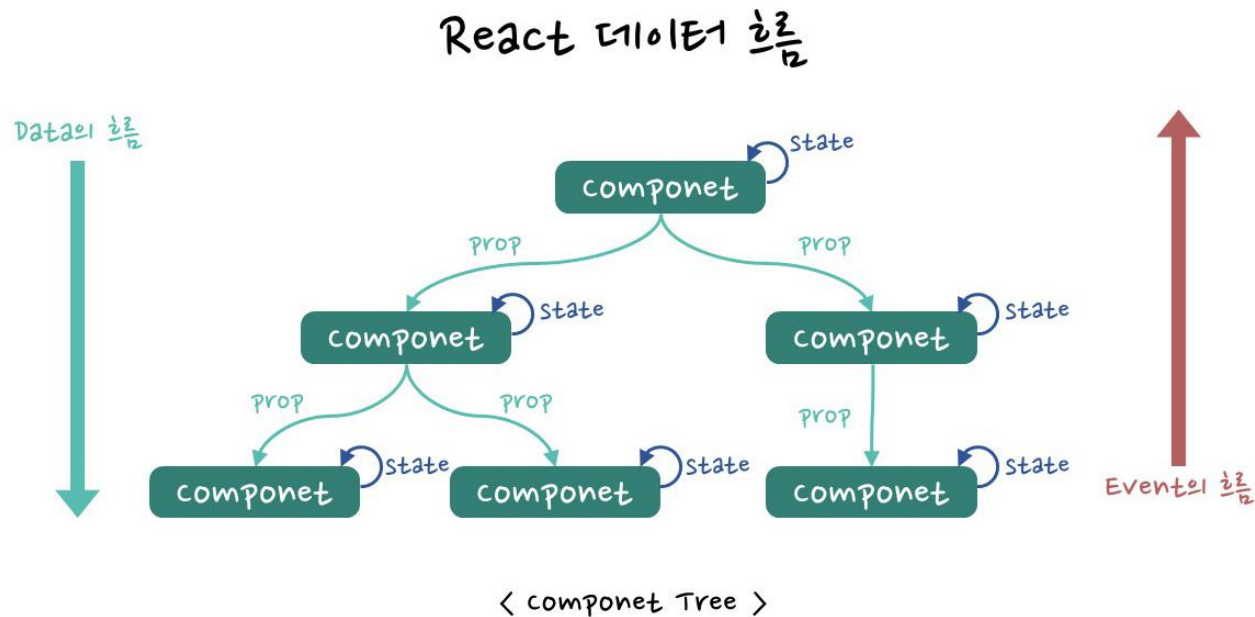
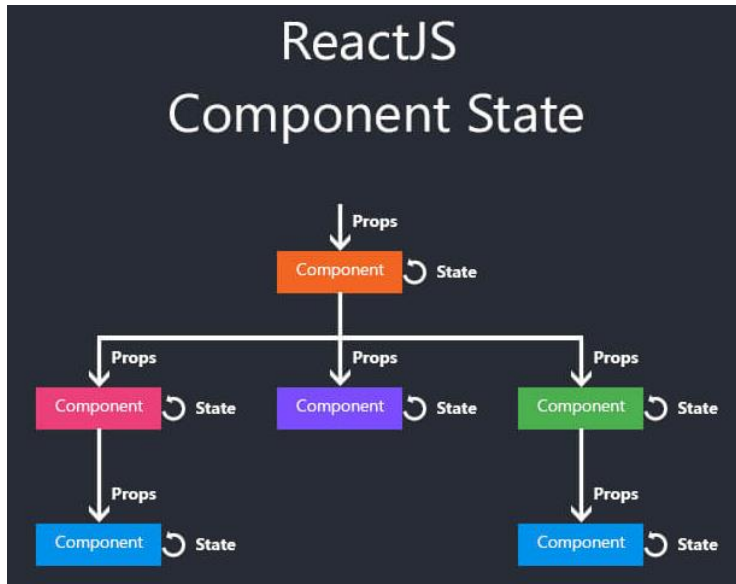
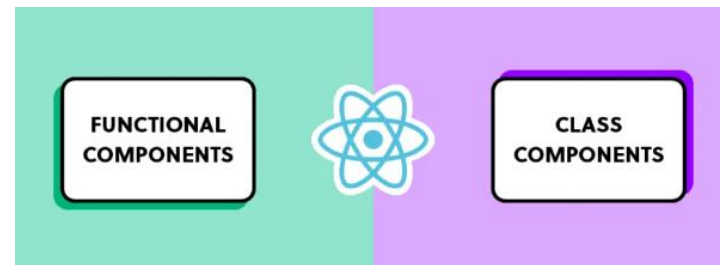




# Component State

## 상태(state)의 종류

- 클래스형 컴포넌트가 가지고 있는 state
- 함수형 컴포넌트가 useState라는 함수로 사용하는 state
- React v16.8 이전까지는 함수 컴포넌트에서는 state를 사용할 수 없었다.
- 따라서 state를 사용하기 위해서는 어쩔 수 없이 클래스 컴포넌트를 사용해야 했지만, React v16.8 부터 도입된 useState Hook을 사용하면 함수 컴포넌트에서도 state를 사용할 수 있게 되었다.





# Class Component

## State 생성

- React에서 state는 props와 다르게 자동 생성되지 않는다.
- 즉 아무 설정 없이 코드를 작성하면, state가 없는 상태가 된다.
- 따라서 props와 다르게, state을 생성하기 위해서는 생성자에서 명시적으로 state을 기술 해야 한다.
- 이때 state는 객체 형식이어야 한다.

```
constructor(props) {  
  super(props);  
  
  // state 초기값 설정  
  this.state = {  
    count: 0  
  };  
}
```

```
constructor(props) {  
  super(props);  
  this.state = {date: new Date()};  
  // 이시점에서 설정한 값이 state의 기본값이 된다.  
}
```



# Class Component

## State 생성

- constructor 함수 쓰는 경우 => 클래스 constructor 안에 this.state = {} 형태로 선언

```
class MyCompo extends Component {  
  
  constructor(props){  
    super(props);  
    this.state = {  
      state1: 초기값,  
      state2: 초기값  
    };  
  };  
  
  render(){ ... };  
}
```

- constructor 함수 안쓴다면 => 클래스 안에 state = {}; 형태로 선언

```
class MyCompo extends Component {  
  
  state = {  
    state1: 초기값,  
    state2: 초기값  
  };  
  
  render(){ ... };  
}
```



# Class Component

## State 사용

- render() 함수 안에서 `const { 스테이트1, 스테이트2, ... } = this.state`로 비구조 할당 사용 가능

```
render(){
  const { state1, state2 } = this.state;
  return <>{state1}, {state2}</>;
}
```

```
const array = [1, 2]
const [one, two] = array
```

- 값 변경 시에는 `setState`를 쓰며, `this.setState({ 스테이트명: 신규값 })` 형태로 사용 가능

```
render(){
  const { state1, state2 } = this.state;
  return(
    <>
      {state1}, {state2}
      <button onClick = { ()=>{ this.setState( {state1: '변신'} ); } }> 변경</button>
    </>
  );
}
```





# Class Component

## setState 후 특정 작업 실행

- state 값을 바꿔준 뒤 특정 작업(함수)을 실행하고 싶을 때, setState의 두 번째 파라미터에 콜백 (callback) 함수를 등록하여 작업을 처리할 수 있음

```
this.setState( { 바꿀 스테이트값 기술 }, () => { 여기 콜백 함수 } );
```

```
onClick = { () => {  
  this.setState(  
    { state1: state1 + 1 },  
    () => {  
      console.log('setState 호출');  
      console.log( this.state.state1 );  
    }  
  );  
}}
```

```
...  
<button  
  onClick={() => {  
    this.setState(  
      {  
        number : number+1  
      },  
      ()=>{  
        console.log('setState의 호출 이후');  
        console.log(this.state);  
      });  
  }  
/>...  
...
```



# Class Component

## State 설정 및 사용

- React에서 state는 가변적으로 변경 가능한 값이기 때문에, 값을 set 하는게 가능하다(변경하는게 가능).
- 하지만 state에 직접적으로 값을 설정하면 안된다!!!!!!!

// 잘못된 방법, 절대 이렇게 하지 말자

```
this.state.count = count + 1;
```

// 잘못된 방법, 절대 이렇게 하지 말자

```
this.state.comment = 'Hello';
```

- this.state에 직접적으로 값을 설정할 수 있는 건 유일하게, 처음 state을 초기화하는 생성자(constructor)에서만 가능하다.
- 그 이외에서는 직접 설정하면 안된다.



# Class Component

## State 설정 및 사용

- 올바른 state 설정은 setState 함수를 사용하는 것이다.

```
// 올바른 방법  
this.setState({ count: count+1 })
```

```
// 올바른 방법  
this.setState((prevState, props) => ({  
  counter: prevState.counter + props.increment  
}));  
  
// 아래 코드는, 위에 코드와 동일, 이전 문법으로 작성  
  
// 올바른 방법  
this.setState(function(prevState, props) {  
  return {  
    counter: prevState.counter + props.increment  
  };  
});
```

- setState의 동작
  - setState를 통해 state의 값을 변경해주고,
  - React가 state의 변경을 감지하면
  - 화면을 리렌더링해준다



# Class Component

## State 설정 및 사용

App.js

```
import React from 'react';

export default class App extends React.Component{
  constructor(props) {
    super(props);
    this.state = {
      name: '홍길동',
      age: 300,
      job: 'developer',
    };
  }

  render() {
    const {name, age, job} = this.state;
    return (
      <div style={{margin:15}}>
        <div>name: {name}</div>
        <div>age: {age}</div>
        <div>job: {job}</div>
      </div>
    );
  }
}
```

index.js

```
...
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
...
```

```
name: 홍길동
age: 300
job: developer
```



# Class Component

## State 설정 및 사용

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

class Counter extends Component {
  constructor(props) {
    super(props);

    // state 초기값 설정
    this.state = {
      count: 0
    };
  }
}
```

Counter.js

```
render() {
  const { count } = this.state;
  return (
    <div>
      <h2>현재 Count : {count}</h2>
      <button onClick={() => {
        this.setState({ count: count+1 })
      }}>카운트 +1</button>
    </div>
  );
}

export default Counter;
```

index.js

```
...
root.render(
  <React.StrictMode>
    <Counter />
  </React.StrictMode>,
);
...
```

현재 Count : 2

카운트 +1

현재 Count : 3

카운트 +1   카운트 -1



# Event handling

## 컴포넌트에서 DOM 이벤트 사용

- 컴포넌트에서 출력된 특정 DOM 객체에 이벤트 컴포넌트가 동작하기 위해선 DOM이벤트 프로퍼티를 사용해야 한다.
- 우리가 흔히 쓰고 있는 HTML 엘리먼트의 이벤트들은 JSX내에서 'on + 이벤트명' 형태의 프로퍼티로 제공된다.

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 눌렀다 떼는 경우 ( 값 입력후에 발생하며, onKeyDown 과 동일하게 동작한다. )
keypress	onKeyPress	키보드 버튼이 눌러져 있는 경우 ( 실제 글자가 작성될때 이벤트이며, ASCII 값으로 사용되어 특수키를 인식 못한다.)
focus	onFocus	엘리먼트가 포커스 된 경우
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



# Event handling

## 리액트의 이벤트 문법

- 소문자 대신 카멜 케이스(camelCase)를 사용한다.  
onClick={changenamex} (x)  
onClick={changeName} (o)
- JSX를 사용하여 문자열이 아닌 함수로 이벤트 핸들러를 전달한다.  
onClick="changeName()" (x)  
onClick={changeName} (o)

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

const Main = () => {
  const [ myName, setMyName ] = useState("홍길동")
  function changeName() {
    setMyName(myName === "홍길동" ? "김길동" : "홍길동");
  }

  return (
    <div>
      <h1>안녕하세요. {myName} 입니다.</h1>
      <button onClick={changeName}>Change</button>
    </div>
  );
};

export default Main;
```

안녕하세요. 홍길동 입니다.

Change



# Event handling

## 리액트의 이벤트 문법

- 함수를 직접 선언하여 사용할 수도 있다.

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

const Main = () => {
  const [ myName, setMyName ] = useState("홍길동")

  return (
    <div>
      <h1>안녕하세요. {myName} 입니다.</h1>
      <button onClick={() => {
        setMyName(myName === "홍길동" ? "김길동" : "홍길동");
      }}>
        Change</button>
    </div>
  );
};

export default Main;
```





# Event handling

## 너비와 높이 증가시키기

```
Area.js
import { useState } from "react";

const Area = () => {
  const [size, setSize] = useState({ width: 200, height: 100 });

  return (
    <div>
      <h1>
        너비 : {size.width}, 높이 : {size.height}
      </h1>
      <button
        onClick={() => {
          const copy = { ...size };
          copy.width += 20;
          setSize(copy);
        }}
      >
```

```
너비 증가
    </button>
    <button
      onClick={() => {
        const copy = { ...size };
        copy.height += 10;
        setSize(copy);
      }}
    >
      높이 증가
    </button>
  </div>
); };

export default Area;
```

const copy = { ...size }; //spread 연산자  
다음 코드는 size 객체의 모든 값을 복사하여 새로운 copy 객체를 생성하는 코드이다.



# Event handling

## 경고창 띄우기

- 일반적인 클릭 이벤트 지정 방법

```
<div onclick="clickHandler(e)">Click Me</div>
```

- 리액트에서 클릭 이벤트를 지정하는 방법

```
<div onClick={clickHandler}>Click Me</div>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="https://unpkg.com/react@15/dist/react.min.js"> </script>
    <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"> </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.38/browser.min.js"> </script>
  </head>
  <body>
    <div id="root"> </div>
    <script type="text/babel">
```



# Event handling

## 경고창 띄우기

```
// 컴포넌트를 정의합니다.  
class Hello extends React.Component {  
  render () {  
    // 이벤트를 정의합니다.  
    const clickHandler = (e) => {  
      window.alert('안녕하세요.')  
    }  
    // 클릭 이벤트를 지정합니다.  
    return (  
      <div onClick={clickHandler}>Click Me</div>  
    )  
  }  
}  
// 컴포넌트를 사용합니다.  
ReactDOM.render(  
  <Hello />,  
  document.getElementById('root'))  
</script>  
</body>  
</html>
```

Click Me

127.0.0.1:5500 내용:  
안녕하세요.

확인



# Event handling

## 클릭했을 때 클래스의 메소드 호출

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <script src="https://unpkg.com/react@15/dist/react.min.js"> </script>
    <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"> </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.38/browser.min.js"> </script>
  </head>
  <body>
    <div id="root"> </div>
    <script type="text/babel">
      // 컴포넌트를 정의합니다.
      class Hello extends React.Component {
        constructor(props) {
          super(props);
          // 이벤트 핸들러의 this를 설정합니다. --- (※1)
          this.clickHandler = this.clickHandler.bind(this);
        }
      }
```



# Event handling

## 클릭했을 때 클래스의 메소드 호출

```
clickHandler(e) {  
  const name = this.props.name;  
  window.alert(`${name}님 안녕하세요.`);  
}  
render() {  
  // 클릭 이벤트를 지정합니다. --- (※2)  
  return <div onClick={this.clickHandler}>Click Me</div>;  
}  
}  
// 컴포넌트를 사용합니다.  
ReactDOM.render(<Hello name="홍길동" />, document.getElementById("root"));  
</script>  
</body>  
</html>
```

Click Me

127.0.0.1:5500 내용:  
홍길동님 안녕하세요.

확인



# Event handling

## 간단한 체크박스 구현하기

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <script src="https://unpkg.com/react@15/dist/react.min.js"> </script>
    <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"> </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.38/browser.min.js"> </script>
  </head>
  <body>
    <div id="root"> </div>
    <script type="text/babel">
      // 상태를 가진 컴포넌트를 정의합니다.
      class CBox extends React.Component {
        // 생성자
        constructor(props) {
          super(props);
          // 상태를 초기화합니다.
          this.state = { checked: false };
        }
      }
```



# Event handling

## 간단한 체크박스 구현하기

```
render() {  
  // 체크되지 않았을 때의 상태  
  let mark = "□";  
  let bstyle = { fontWeight: "normal" };  
  // 체크돼 있는지 확인합니다.  
  if (this.state.checked) {  
    mark = "■";  
    bstyle = { fontWeight: "bold" };  
  }  
  // 클릭했을 때의 이벤트를 지정합니다.  
  const clickHandler = (e) => {  
    const newValue = !this.state.checked;  
    this.setState({ checked: newValue });  
  };  
  // 렌더링할 내용을 반환합니다.  
  return (  
    <div onClick={clickHandler} style={bstyle}>  
      {mark} {this.props.label}  
    </div>  
  );  
}
```

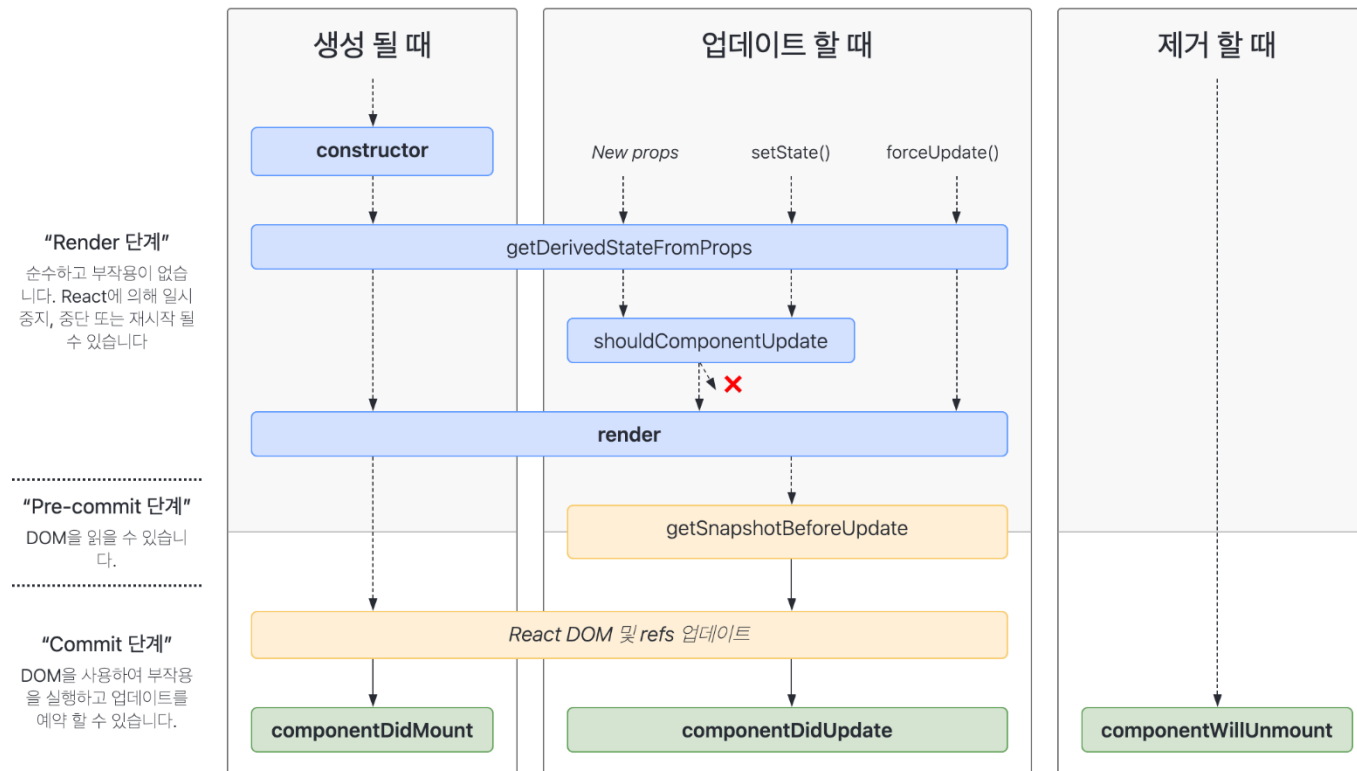
```
// 리액트로 DOM의 내용을 변경합니다.  
const dom = (  
  <div>  
    <CBox label="Apple" />  
    <CBox label="Banana" />  
    <CBox label="Orange" />  
    <CBox label="Mango" />  
  </div>  
>;  
ReactDOM.render(dom, document.getElementById("root"));  
</script>  
</body>  
</html>
```

■ Apple  
□ Banana  
■ Orange  
□ Mango

# Component Life Cycle

## 생명 주기(Life cycle)

- Lifecycle은 생명주기를 말한다.
- 사람은 태어나서 인생을 살다가 나이가 들어 죽는 생명주기를 가지고 있다.
- React Component도 마찬가지로 생명주기가 있다.
- Component가 생성되는 시점과 사라지는 시점이 정해져 있다.







# Component Life Cycle

## <Mounting>

- 사람이 태어나는 시기로 리액트에서는 Component가 생성되는 시점이다.
- Class Copmponent를 사용할 경우 Component의 constructor 즉 생성자가 실행된다.
- 생성자에서는 Component의 state를 정의하게 된다.
- Component가 렌더링 되며 이후에 componentDidMount함수가 호출된다.



## <Updating>

- 태어난 사람은 각자의 인생을 살아간다.
- 사람은 인생을 살아가는 동안 신체적, 정신적 변화를 겪는다.
- React Component도 생애 동안 변화를 겪으면서 여러 번 렌더링 된다.
- 이를 React Component가 업데이트(Update)되는 과정이라고 할 수 있다.
- Update과정에서는 Component의 props가 변경되거나 setState함수 호출에 의해 state가 변경되거나 falseUpdate 라는 강제 업데이트 함수 호출로 인해 Copmponent가 다시 렌더링 된다.



# Component Life Cycle

## <Unmounting>

- 인생을 살아가던 사람은 나이가 들어 죽는다.
- React Component도 언젠가는 사라지는 과정을 겪게 되는데 이 과정을 unmount라고 부른다.
- Component의 unmount는 상위컴포넌트를 더 이상 화면에 표시하지 않게 될 때 unmount 된다.





# Component Life Cycle

## 라이프 사이클 확인하기

```
import React from "react";
const styles = {
  wrapper : {
    margin:8,
    padding:8,
    display : "flex",
    flexDirection:"row",
    border:"1px solid grey",
    borderRadius:16,
  },
  messageText:{
    color:"black",
    fontSize:16,
  },
};
```

```
class Notification extends React.Component{
  constructor(props){
    super(props);
    this.state={};
  }
  componentDidMount(){
    console.log(`${this.props.id} componentDidMount() called.`)
  }
  componentDidUpdate(){
    console.log(`${this.props.id} componentDidUpdate() called.`)
  }
  componentWillUnmount(){
    console.log(`${this.props.id} componentWillUnmount() called.`)
  }
  render(){
    return(
      <div style={styles.wrapper}>
        <span style={styles.messageText}>{this.props.message}</span>
      </div>
    );
  }
}
export default Notification;
```



# Component Life Cycle

## 라이프 사이클 확인하기

```
import React from "react";
import Notification from "../Notification";
const reservedNotifications=[
  {
    id:1,
    message:"안녕하세요, 오늘 일정을 알려드립니다.",
  },
  {
    id:2,
    message:"점심식사 시간입니다.",
  },
  {
    id:3,
    message:"이제 곧 미팅이 시작됩니다.",
  },
];
var timer;
```

```
class NotificationList extends React.Component{
  constructor(props){
    super(props);
    this.state={
      notifications:[],
    };
  }
  componentDidMount(){
    const {notifications}=this.state;
    timer=setInterval(()=>{
      if(notifications.length < reservedNotifications.length){
        const index=notifications.length;
        notifications.push(reservedNotifications[index]);
        this.setState({
          notifications: notifications,
        });
      }else{
        this.setState({notifications:[],});
        clearInterval(timer);
      }
    },1000);
  }
}
```



# Component Life Cycle

## 라이프 사이클 확인하기

```
componentWillUnmount() {  
  if (timer) {  
    clearInterval(timer);  
  }  
}  
  
render(){  
  return(  
    <div>  
      {this.state.notifications.map((notification)=>{  
        return <Notification  
          key={notification.id}  
          id={notification.id}  
          message={notification.message}/>;  
      })}  
    </div>  
  );  
}  
}  
  
export default NotificationList;
```

React Quiz



# Component Life Cycle

## 라이프 사이클 확인하기

```
import React from "react";
import ReactDOM from "react-dom/client";
class Test extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hello: "World!" };
  }
  componentDidMount() {
    console.log("componentDidMount()");
  }
  changeState() {
    this.setState({ hello: "Geek!" });
  }
}
```

```
render() {
  return (
    <div>
      <h1>
        GeeksForGeeks.org, Hello
        {this.state.hello}
      </h1>
      <h2>
        <a
          onClick={this.changeState.bind(
            this
          )}
        >
          Press Here!
        </a>
      </h2>
    </div>
  );
}
```

**GeeksForGeeks.org,  
HelloWorld!**



# Component Life Cycle

## 라이프 사이클 확인하기

```
shouldComponentUpdate(nextProps, nextState) {  
  console.log("shouldComponentUpdate()");  
  return true;  
}  
componentDidUpdate() {  
  console.log("componentDidUpdate()");  
}}  
const root = ReactDOM.createRoot(  
  document.getElementById("root"));  
root.render(<Test />);
```



# Component Life Cycle

## 라이프 사이클 확인하기

```
import React, { Component } from 'react'
class App extends Component {
  // 마운트
  constructor (props) {
    super(props)
    console.log('constructor')
  }
  componentWillMount () {
    console.log('componentWillMount')
  }
  componentDidMount () {
    console.log('componentDidMount')
  }
  // 변경
  componentWillReceiveProps (nextProps) {
    console.log('componentWillReceiveProps')
  }
  shouldComponentUpdate (nextProps, nextState) {
    console.log('shouldComponentUpdate')
    return true
  }
  componentWillUpdate () {
    console.log('componentWillUpdate')
  }
}
```

```
  componentDidUpdate () {
    console.log('componentDidUpdate')
  }
  // 언마운트
  componentWillUnmount () {
    console.log('componentWillUnmount')
  }
  render () {
    console.log('render')
    const setStateHandler = (e) => {
      console.log('* call setState()')
      this.setState({r: Math.random()})
    }
    return (
      <div>
        <button onClick={setStateHandler}>
          setState</button>
        </div>
      )
    )
  }
}
export default App
```

**DEPRECATED**





# Component Life Cycle

## 스톱워치 만들기

```
.Stopwatch {
  text-align: center;
}
.disp {
  font-size: 70px;
}
button {
  font-size: 24px;
  width: 280px;
}
```

```
import React, { Component } from 'react'
import './Stopwatch.css';
// Stopwatch 컴포넌트를 정의합니다.
class Stopwatch extends Component {
  constructor (props) {
    super(props)
    this.state = { // 초깃값을 설정합니다.
      isLive: false,
      curTime: 0,
      startTime: 0
    }
    this.timerId = 0
  }
  // 마운트했을 때
  componentWillMount () {
    this.timerId = setInterval(e => {
      this.tick()
    }, 1000)
  }
  // 언마운트했을 때
  componentWillUnmount () {
    clearInterval(this.timerId)
  }
}
```

```
// 매 초 실행됩니다.
tick () {
  if (this.state.isLive) {
    const v = new Date().getTime()
    this.setState({curTime: v})
  }
}
// 시작/중지 버튼을 클릭했을 때
clickHandler (e) {
  // 중지할 때
  if (this.state.isLive) {
    this.setState({isLive: false})
    return
  }
  // 시작할 때
  const v = new Date().getTime()
  this.setState({
    curTime: v,
    startTime: v,
    isLive: true})
};
```



# Component Life Cycle

## 스톱워치 만들기

```
// 출력할 시계를 생성합니다.
getDisp () {
  const s = this.state
  const delta = s.curTime - s.startTime
  const t = Math.floor(delta / 1000)
  const ss = t % 60
  const m = Math.floor(t / 60)
  const mm = m % 60
  const hh = Math.floor(mm / 60)
  const z = (num) => {
    const s = '00' + String(num)
    return s.substr(s.length - 2, 2)
  }
  return <span className='disp'>
    {z(hh)}:{z(mm)}:{z(ss)}
  </span>
}
```

```
// 화면 렌더링
render () {
  let label = 'START'
  if (this.state.isLive) {
    label = 'STOP'
  }
  const disp = this.getDisp()
  const fclick = (e) => this.clickHandler(e)
  return (<div className='Stopwatch'>
    <div>{disp}</div>
    <button onClick={fclick}>{label}</button>
  </div>)
}
export default Stopwatch
```

00:00:06

STOP



# Input Form

## 간단한 입력 양식 만들기

```
import React from 'react'
// 입력 양식 컴포넌트입니다.
export class SimpleForm extends React.Component {
  constructor (props) {
    super(props)
    // 상태를 초기화합니다.
    this.state = { value: '' }
  }
  // 값이 변경됐을 때
  doChange (e) {
    const newValue = e.target.value
    this.setState({value: newValue})
  }
  // 전송 버튼을 눌렀을 때
  doSubmit (e) {
    window.alert('전송: ' + this.state.value)
    e.preventDefault()
  }
}
```

```
// 화면 렌더링
render () {
  // 이벤트를 메서드에 바인딩합니다.
  const doSubmit = (e) => this.doSubmit(e)
  const doChange = (e) => this.doChange(e)
  return (
    <form onSubmit={doSubmit}>
      <input type='text'
        value={this.state.value}
        onChange={doChange} />
      <input type='submit' value='전송' />
    </form>
  )
}
```

Form input: Hello 전송

localhost:3000 내용:  
전송: Hello

확인



# Input Form

## 숫자만 입력받는 텍스트박스 만들기

```
import React, {Component} from 'react'
// 숫자 입력 컴포넌트
export default class NumberForm extends Component {
  constructor (props) {
    super(props)
    this.state = { value: '' }
  }
  // 값이 변경됐을 때
  doChange (e) {
    const curValue = e.target.value
    // 숫자 이외의 값을 제거합니다.
    const newValue = curValue.replace(/[^0-9]/g, '')
    this.setState({value: newValue})
  }
  // 전송 버튼을 눌렀을 때
  doSubmit (e) {
    window.alert('전송: ' + this.state.value)
    e.preventDefault()
  }
}
```

```
// 화면 렌더링
render () {
  // 이벤트를 메서드에 바인딩합니다.
  const doSubmit = (e) => this.doSubmit(e)
  const doChange = (e) => this.doChange(e)
  return (
    <form onSubmit={doSubmit}>
      <input type='text'
        value={this.state.value}
        onChange={doChange} />
      <input type='submit' value='전송' />
    </form>
  )
}
```

123456

localhost:3000 내용:  
전송: 123456

확인



# Input Form

## 여러 개의 입력 항목을 가진 입력 양식 만들기

```
.multiForm {  
  width: 400px;  
  text-align: left;  
  margin-left: auto;  
}  
.multiForm p {  
  margin: 4px  
; padding: 4px;  
}
```

```
import React, {Component} from 'react'  
// 여러 개의 입력 항목을 가진 컴포넌트  
export default class MultiForm extends Component {  
  constructor (props) {  
    super(props)  
    // 입력 양식의 초깃값을 설정합니다.  
    this.state = {  
      name: '홍길동',  
      age: 300,  
      hobby: '개발'  
    }  
  }  
  // 값이 변경됐을 때  
  doChange (e) {  
    const userValue = e.target.value  
    const key = e.target.name  
    this.setState({[key]: userValue})  
  }  
  // 전송 버튼을 눌렀을 때  
  doSubmit (e) {  
    e.preventDefault()  
    const j = JSON.stringify(this.state)  
    window.alert(j)  
  }  
}
```



# Input Form

## 여러 개의 입력 항목을 가진 입력 양식 만들기

```
// 화면 렌더링 render () {  
  // 이벤트를 메서드에 바인딩합니다.  
  const doSubmit = (e) => this.doSubmit(e)  
  const doChange = (e) => this.doChange(e)  
  return (  
    <form onSubmit={doSubmit}>  
      <div> <label>  
        이름: <br />  
        <input name='name'  
          type='text'  
          value={this.state.name}  
          onChange={doChange} />  
      </div> </div>  
      <div> <label>  
        나이: <br />  
        <input name='age'  
          type='number'  
          value={this.state.age}  
          onChange={doChange} />  
      </div> </div>  
    )  
  )  
}
```

```
</label> </div>  
<div> <label>  
  취미: <br />  
  <input name='hobby'  
    type='text'  
    value={this.state.hobby}  
    onChange={doChange} />  
</label> </div>  
<input type='submit' value='전송' />  
</form>  
)  
}  
}
```

이름:  
홍길동  
나이:  
300  
취미:  
개발  
전송

localhost:3000 내용:  
{"name": "홍길동", "age": 300, "hobby": "개발"}  
확인





## Report

웹사이트를 5개 이상의 컴포넌트로 구성하세요.

2개 이상의 props를 전달하는 컴포넌트를 하나 이상 포함하세요.

2개 이상의 state를 가진 컴포넌트를 하나 이상 포함하세요.

1개 이상의 props와 1개 이상의 state를 가진 컴포넌트를 하나 이상 포함하세요

회원가입 페이지 만들기

## 발표주제

구조 분해 할당(Destructuring Assignment)

콜백 함수

버블링과 캡처링

React Developer Tools





# Reference

- <https://goddaehee.tistory.com/299>
- <https://wikidocs.net/197615>
- 모던 자바스크립트 개발자를 위한 리액트 프로그래밍, 쿠지라히코우즈키에, 윤인성, 위키북스
- 명품 웹 프로그래밍, 황기태, 생능출판사
- 리액트 & 리액트 네이티브 통합 교과서, 아담 보두치, 강경일, 신희철, 에이콘
- Do it! 리액트 모던 웹 개발 with 타입스크립트, 전예홍, 이지스퍼블리싱