state와 클래스형 컴포넌트

1. state로 숫자 증감 기능 만들어 보기

1 state로 숫자 증감 기능 만들어 보기

- state에는 어떤 특징이 있을까?
- state는 동적 데이터를 다룰 때 사용
- 동적 데이터란 말 그대로 변경될 가능성이 있는 데이터를 의미
- 객체를 예로 들면 객체의 구성 요소 중 일부가 있다가 없을 수도 있고, 구성 요소가 하나였다가 둘이 될 수도 있는데 props는 그런 데이터를 다루지 못한다.
- 이것이 state를 사용하는 이유!!!
- state는 실습하면서 천천히 알아보자.
- 그런데 state는 클래스형 컴포넌트에서 사용할 수 있는 개념이다.
- 그러니 state를 사용하려면 클래스형 컴포넌트를 알아야!!!

1 state로 숫자 증감 기능 만들어 보기

❖ 클래스형 컴포넌트 작성하기

- 슬프지만 클래스형 컴포넌트를 공부하기 위해서 지금까지 만든 컴포넌트를 모두 지워야...
- App.js에서 두 줄만 남기고 모두 지울 것!!

```
1  import React from "react";
2  export default App;
```

1 state로 숫자 증감 기능 만들어 보기

❖ 클래스형 컴포넌트 작성하기

- 다음과 같이 코드를 작성하자.
- 핵심은 App 클래스가 React.Component 클래스를 상속받도록 extends React.Component를 붙이는 것!!!
- 이게 클래스형 컴포넌트의 기본 뼈대이다.

```
1  import React from "react";
2
3  class App extends React.Component{
4
5  }
6
7  export default App;
```

- 여기서 중요한 내용은 클래스형 컴포넌트가 되려면 'App 클래스가 리액트가 제공하는 Component 클래스를 반드시 상속받아야 한다'는 것!!

1 state로 숫자 증감 기능 만들어 보기

❖ 클래스형 컴포넌트 작성하기

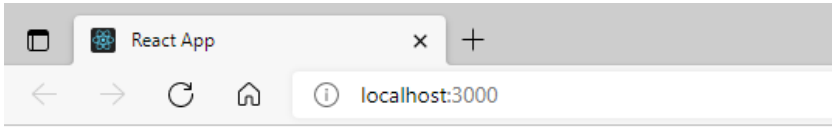
- 클래스 형태의 App 컴포넌트를 작성했다.
- 이제 App 컴포넌트가 JSX를 반환해야 한다.
- 그런데 지금의 App 컴포넌트는 클래스라서(함수가 아니라서) return 문을 사용할 수 없고, 그래서 함수 형태의 App 컴포넌트처럼 JSX를 반환할 수가 없다.
- 어떻게 해야 클래스형 컴포넌트가 JSX를 반환할 수 있을까?
- 클래스형 컴포넌트에서는 JSX를 반환하기 위해 render() 함수를 사용한다.

1 state로 숫자 증감 기능 만들어 보기

❖ 클래스형 컴포넌트 작성하기

- 그럼 **render()** 함수를 사용해 보자.
- 다음과 같이 코드를 작성하면 된다.

```
1  import React from "react";
2
3  class App extends React.Component{
4      render(){
5          return <h1>I'm a class Component</h1>;
6      }
7  }
8
9  export default App;
```



I'm a class Component

1 state로 숫자 증감 기능 만들어 보기

❖ 클래스형 컴포넌트 작성하기

- 결과를 보면 함수형 컴포넌트를 사용했을 때와 별 차이가 없다.
- 하지만 함수형 컴포넌트와 클래스형 컴포넌트는 분명히 코드에 차이가 있다.
- 함수형 컴포넌트는 **return** 문이 **JSX**를 반환한다.
- 하지만 클래스형 컴포넌트는 **render()** 함수가 **JSX**를 반환한다.
- 그리고 리액트는 클래스형 컴포넌트의 **render()** 함수를 자동으로 실행한다.
- 다시 말해 **render()** 함수는 우리가 직접 실행하지 않아도 실행되는 함수인 것!!!
- 여기까지가 클래스형 컴포넌트의 기초 개념이다.
- 우리가 클래스형 컴포넌트를 사용하는 이유는?
- 맞다. '**state**를 사용하기 위함'이다.

1 state로 숫자 증감 기능 만들어 보기

❖ state 정의하기

- state를 사용하려면 다음과 같이 state = {} ; 라고 작성하여 state를 정의하면 된다.

```
1  import React from "react";
2
3  class App extends React.Component{
4      state = {
5
6      };
7      render(){
8          return <h1>I'm a class Component</h1>;
9      }
10 }
11
12 export default App;
```

- 위에서 보듯 state는 객체 형태의 데이터이다.
- 그리고 state를 사용하려면 반드시 클래스형 컴포넌트 안에서, 소문자를 이용하여 state라고 적으면 된다.

1 state로 숫자 증감 기능 만들어 보기

❖ state에 count값 추가하고 사용하기

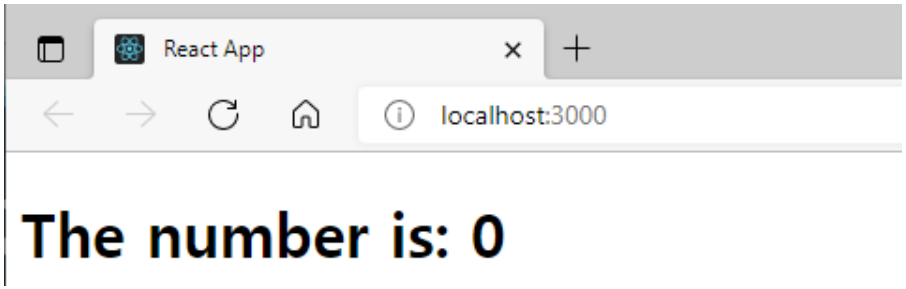
- 다음과 같이 state에 count라는 키를 추가하고 키값으로 0을 넣는다.
- 그리고 render() 함수에서 {this.state.count}를 출력한다.

```
1  import React from "react";
2
3  class App extends React.Component{
4      state = {
5          count: 0,
6      };
7      render(){
8          return <h1>The number is: {this.state.count}</h1>;
9      }
10 }
11
12 export default App;
```

1 state로 숫자 증감 기능 만들어 보기

❖ state에 count값 추가하고 사용하기

- 값이 출력되었다.
- 하지만 아직 **state**를 제대로 사용하고 있지 않다.
- 아까도 말했듯이 **state**에 동적 데이터를 저장할 수 있어야 한다.
- 바꿀 수 있는 데이터!
- 그러면 값을 바꿀 수 있도록 코드를 작성해야!!
- 이제 버튼을 클릭하는 등의 '사용자 동작'에 따라 **state**의 **count**를 바꿀 수 있도록 코드를 작성하자.



1 state로 숫자 증감 기능 만들어 보기

❖ 버튼을 눌러서 count state값 변경해 보기

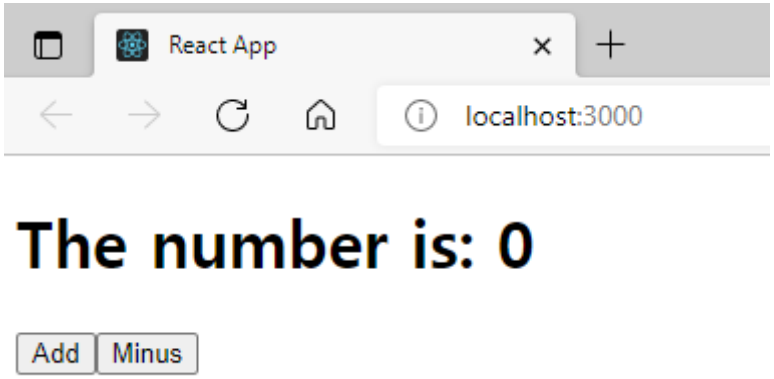
- 다음과 같이 <Add>버튼과 <Minus> 버튼을 추가하자.

```
...
    render(){
        return (
            <div>
                <h1>The number is: {this.state.count}</h1>
                <button>Add</button>
                <button>Minus</button>
            </div>
        );
    }
...
```

1 state로 숫자 증감 기능 만들어 보기

❖ 버튼을 눌러서 count state값 변경해 보기

- 버튼이 잘 추가되었다.
- 이제 버튼을 누르면 어떤 작업을 해야 할까?
- <Add> 버튼을 누르면 this.state.count의 값을 증가시키고, <Minus> 버튼을 누르면 this.state.count의 값을 감소시키면 된다.
- 그러기 위해서는 2개의 함수가 필요하다.



1 state로 숫자 증감 기능 만들어 보기

❖ add() 함수와 minus() 함수 작성하기

- 다음과 같이 add() 함수와 minus() 함수를 작성하자.

```
...
    state = {
        count: 0,
    };

    add = () => {
        console.log('add');
    }

    minus = () => {
        console.log('minus');
    }

    render(){
...
```

1 state로 숫자 증감 기능 만들어 보기

❖ add() 함수와 minus() 함수 작성하기

- 혹시 자바스크립트를 알고 있다면 add() 함수에 this.state.count++ 또는 this.state.count = this.state.count + 1 과 같은 코드를 작성하고 싶을 것이다.
- 그런데! 리액트에서는 이 방법을 허용하지 않는다.
- 그렇다면 어떻게 해야 버튼을 눌렀을 때 add(), minus() 함수가 동작할까?

1 state로 숫자 증감 기능 만들어 보기

❖ 버튼을 누르면 동작하도록 onClick 속성 추가하기

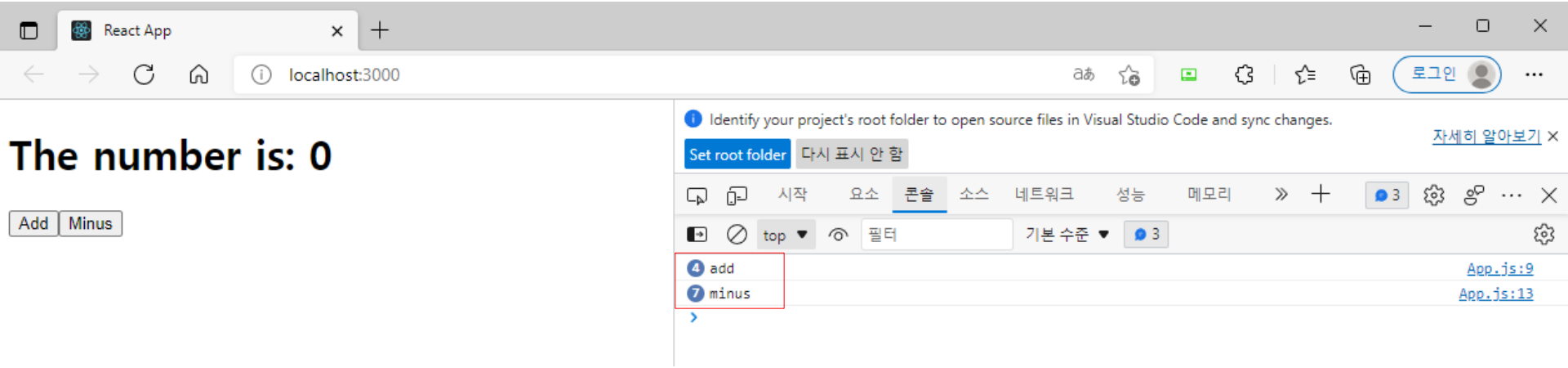
- button 엘리먼트에 onClick이라는 속성을 넣고, 속성값으로 this, add와 같이 함수를 넣어주면 된다.

```
...
    render(){
        return (
            <div>
                <h1>The number is: {this.state.count}</h1>
                <button onClick={this.add}>Add</button>
                <button onClick={this.minus}>Minus</button>
            </div>
        );
    }
...
```

1 state로 숫자 증감 기능 만들어 보기

❖ 앱 동작 확인하기

- <Add> 버튼을 4번 누르고, <Minus> 버튼을 7번 눌러보자.
- 그러면 [콘솔] 탭에 다음과 같은 문장이 출력될 것이다.



- 버튼이 잘 동작하니까, 이제 숫자가 변경될 수 있도록 코드를 작성하면 될 것같다.
- 숫자를 변경하려면 어떻게 해야 할까?

2 숫자 증감 기능을 제대로 만들어 보기

- 그렇다면 state는 그냥 객체인데 뭐가 그렇게 특별할까?
- State를 사용하려면 React.Component를 상속받은 클래스형 컴포넌트를 사용해야 하고, count state도 마음대로 변경하면 안되고...
- 맞다! state는 그냥 단순한 객체이다.
- 하지만 리액트에서는 state를 특별하게 다뤄야 한다.
- 그래서 this.state.count++와 같은 코드를 아직 작성하지 말라고 했던 것!!
- 정말 그럴까?

2 숫자 증감 기능을 제대로 만들어 보기

❖ this.state.count 마음대로 바꿔 보기

- 여러분의 생각대로 'state는 말 그대로 그냥 객체'니까 this.state.count = 1 또는 this.state.count = -1과 같이 값을 바꿀 수 있다고 생각할 수 있다.
- 그 생각대로 App 컴포넌트의 add(), minus() 함수를 수정해 보자.

```
...
    state = {
        count: 0,
    };

    add = () => {
        this.state.count = 1;
    }

    minus = () => {
        this.state.count = -1;
    }

    render(){
...
```

2 숫자 증감 기능을 제대로 만들어 보기

❖ this.state.count 마음대로 바꿔 보기

- 버튼을 눌러서 add(), minus() 함수가 동작하는지 확인해 보자.
- 동작하지 않을 것이다.
- 왜냐하면 리액트는 state를 직접 변경하지 못하게 하니까!
- 원래 리액트는 state가 변경되면 render() 함수를 다시 실행하여 변경된 state를 화면에 출력한다.
- 그런데 state를 직접 변경하는 경우에는 render() 함수를 다시 실행하지 않는다.
- 리액트는 이런 방식으로 state를 직접 변경할 수 없도록 제한한다.
- 그러면 state를 간접적으로 변경하는 방법을 알아야 하겠네?

2 숫자 증감 기능을 제대로 만들어 보기

❖ setState() 함수로 count state 변경하기

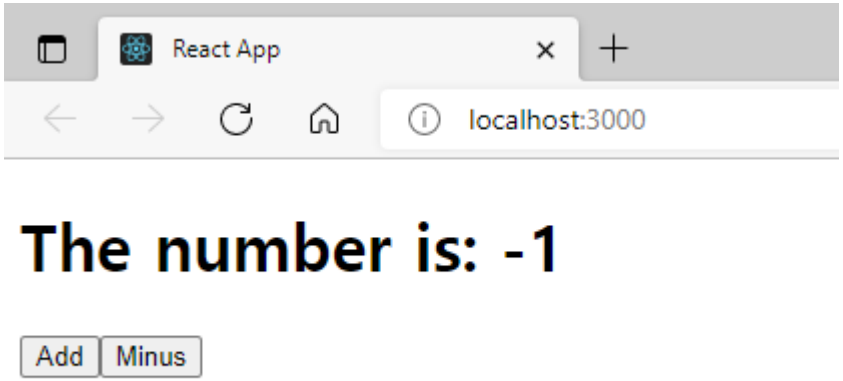
- 다음과 같이 setState() 함수의 첫 번째 인자로 count 키와 키 값을 넣은 객체를 전달해 보자.

```
...
    state = {
        count: 0,
    };

    add = () => {
        this.setState({count: 1});
    }

    minus = () => {
        this.setState({count: -1});
    }

    render(){
...
```

2 숫자 증감 기능을 제대로 만들어 보기

❖ setState() 함수로 count state 변경하기

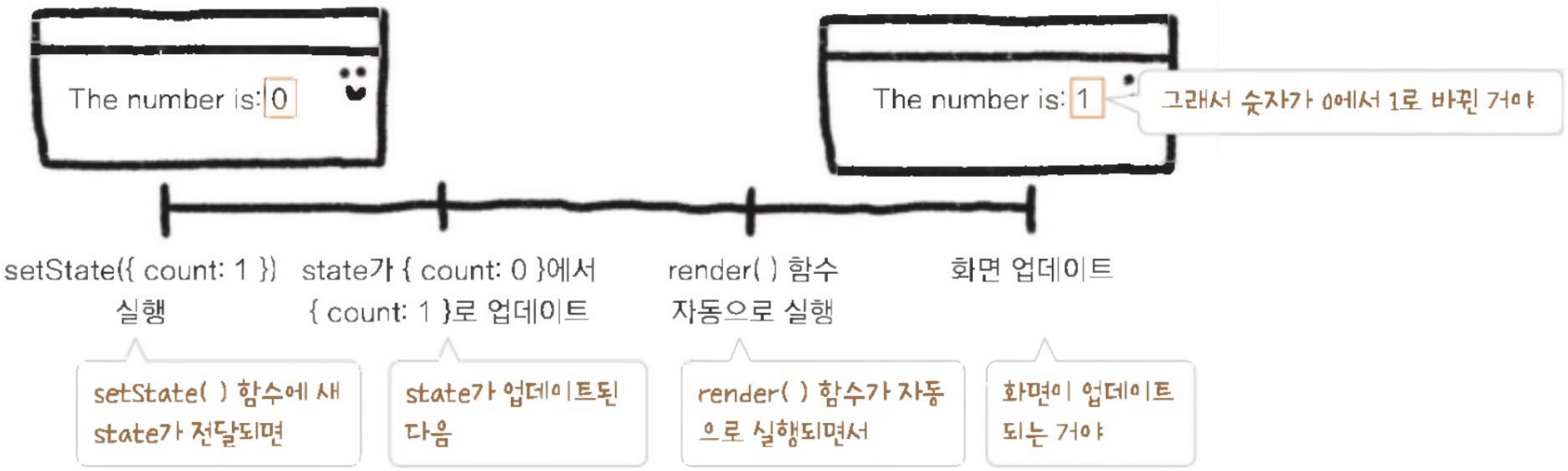
- 코드에서 보듯 **setState()** 함수를 사용하는 방법은 아주 간단하다.
- 내가 원하는 새로운 **state**를 인자로 넘겨주면 된다.
- 결과를 보니 **count state**가 **1** 또는 **-1**로 변하고 있다.



2 숫자 증감 기능을 제대로 만들어 보기

❖ setState() 함수로 count state 변경하기

- 어떻게 이런 일이 가능할까?
- 리액트가 **setState()** 함수의 호출을 감시하고 있기 때문이다.
- **setState()** 함수가 동작하면 **state**가 새로운 값으로 바뀌고, 이어서 **render()** 함수를 동작시켜 화면을 업데이트시키는 것이다.

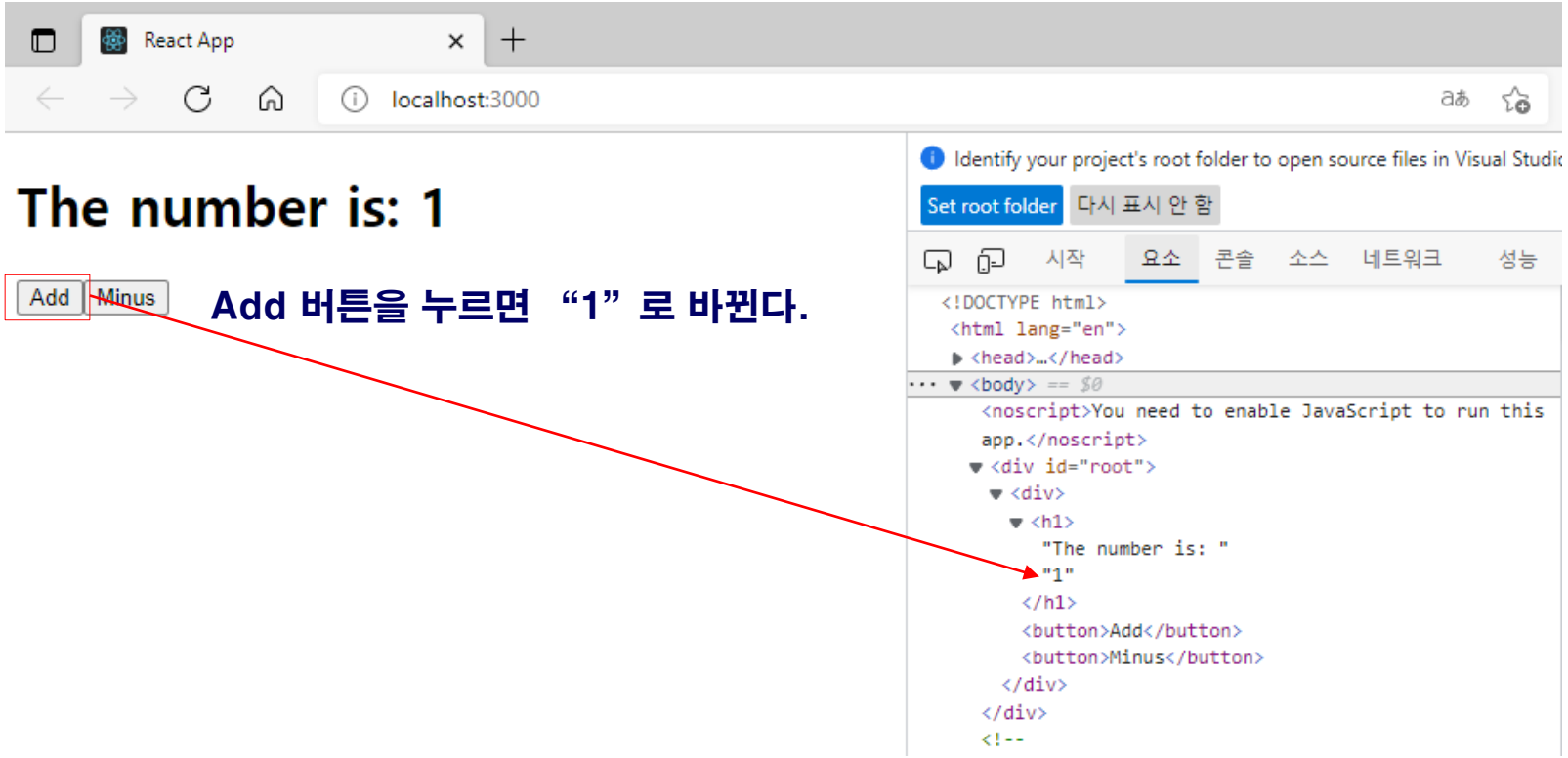


- 그러면 실제로 리액트 앱의 **HTML**은 어떻게 변할까?
- 확인해 보자.

2 숫자 증감 기능을 제대로 만들어 보기

❖ state의 변화에 따라 바뀌는 HTML 살펴보기

- [요소] 탭을 연 다음, div 엘리먼트 옆에 있는 ► 표시를 눌러서 펼쳐 보자.
- 그런 다음 <Add> 버튼과 <Minus>버튼을 눌러 보자.



2 숫자 증감 기능을 제대로 만들어 보기

❖ state의 변화에 따라 바뀌는 HTML 살펴보기

- 버튼을 번갈아 누르면 변경된 state의 값을 반영하려고 HTML만("1" 또는 "-1") 바뀐다.
- 이게 바로 '리액트가 화면 구성이 빠르다'고 말했던 이유이다(필요한 부분만 바뀌니까).
- 게다가 화면도 깜빡거리지 않는다.
- 화면은 바뀌었지만 새로 고침이 일어나지 않는 것이다.
- 리액트의 장점이 느껴지지 않는가?

2 숫자 증감 기능을 제대로 만들어 보기

❖ 버튼을 누르면 count state의 값을 증가 또는 감소시키기

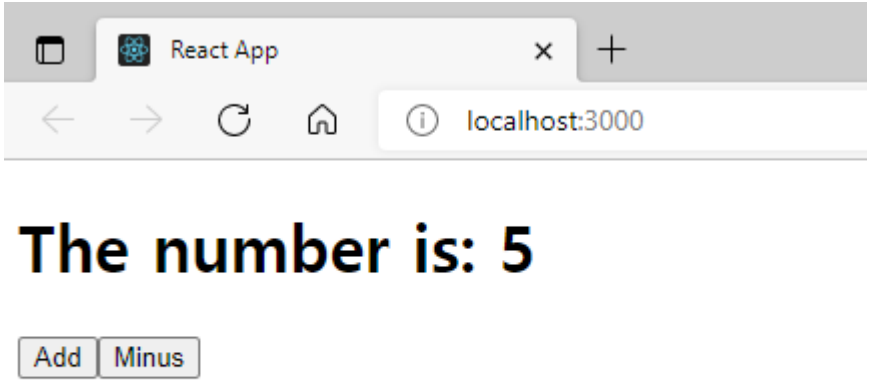
- 이제 다시 <Add>, <Minus> 버튼을 누르면 add, minus() 함수에서 숫자를 증가시키거나 감소시키기로 했던 목적으로 돌아가자.
- 다음과 같이 코드를 작성하자.

```
...
    state = {
        count: 0,
    };

    add = () => {
        this.setState({count: this.state.count + 1});
    }

    minus = () => {
        this.setState({count: this.state.count - 1});
    }

    render(){
...
```

2 숫자 증감 기능을 제대로 만들어 보기

❖ 버튼을 누르면 count state의 값을 증가 또는 감소시키기

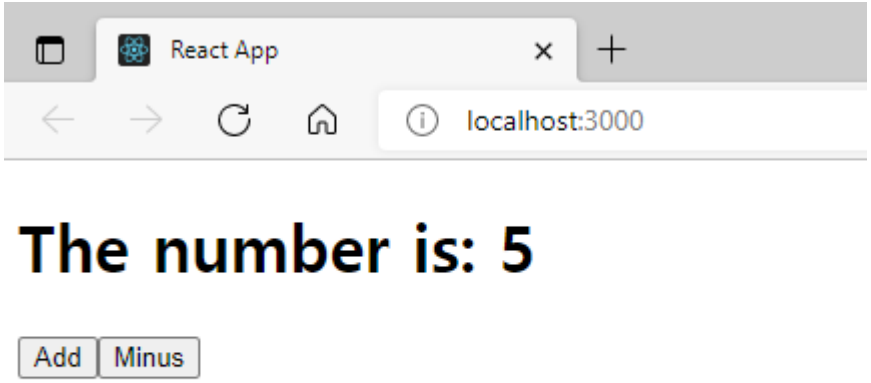
- 앱을 실행하면 잘 동작한다!
- 다만 {count : this.state.count + 1} 와 같이 코드를 작성하여 state를 업데이트하는 방법은 별로 좋지 않다.
- 성능 문제가 생길 수 있기 때문이다.
- 그 대신 setState() 함수의 인자로 함수를 전달하면 성능 문제 없이 state를 업데이트할 수 있다.



2 숫자 증감 기능을 제대로 만들어 보기

❖ 버튼을 누르면 count state의 값을 증가 또는 감소시키기

- 앱을 실행하면 잘 동작한다!
- 다만 {count : this.state.count + 1} 와 같이 코드를 작성하여 state를 업데이트하는 방법은 별로 좋지 않다.
- 성능 문제가 생길 수 있기 때문이다.
- 그 대신 setState() 함수의 인자로 함수를 전달하면 성능 문제 없이 state를 업데이트할 수 있다.



2 숫자 증감 기능을 제대로 만들어 보기

❖ add, minus() 함수 개선하기

- 다음과 같이 current 인자를 받아 객체({count : current.count + 1})를 반환하는 함수를 작성하여 setState() 함수에 전달해 보고, 결과도 확인해 보자.

	...
4	state = {
5	count: 0,
6	};
7	
8	add = () => {
9	this.setState(current => ({
10	count: current.count + 1,
11	}));
12	}
13	

2 숫자 증감 기능을 제대로 만들어 보기

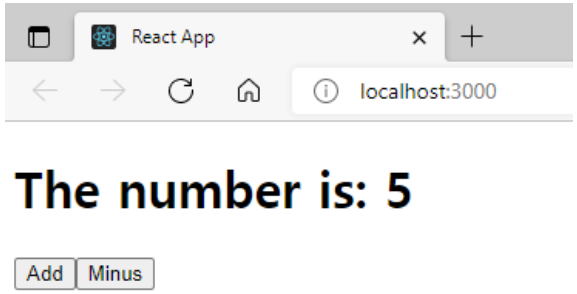
❖ add, minus() 함수 개선하기

- 다음과 같이 current 인자를 받아 객체({count : current.count + 1})를 반환하는 함수를 작성하여 setState() 함수에 전달해 보고, 결과도 확인해 보자.

current에는 현재 state가 넘어온다.

```
14      minus = () => {
15          this.setState(current => ({
16              count: current.count - 1,
17          }));
18      }
19      render(){
20  ...
```

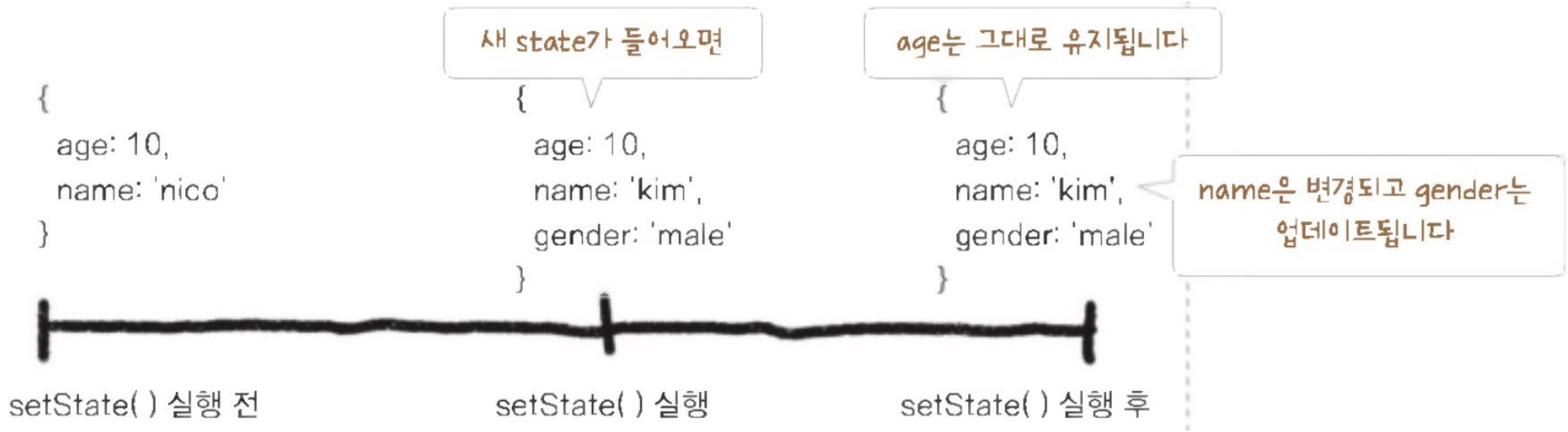
그 state의 count에 1을 뺀다.



2 숫자 증감 기능을 제대로 만들어 보기

❖ **setState() 함수는 바뀐 state의 데이터만 업데이트합니다**

- **setState( )의 인자로 state를 전달하면 구체적으로 어떤 일이 일어날까?**
- **이전 state가 완전히 새로운 state로 교체될까?**
- **아니다.**
- **리액트는 이전 state와 새로운 state를 비교하여 바뀐 데이터만 업데이트한다.**
- **그래서 변경 대상이 아닌 키와 키값은 그대로 유지된다.**



3 클래스형 컴포넌트의 일생 알아보기

- 숫자 증감 기능을 만들면서 state와 클래스형 컴포넌트를 공부했다.
- 클래스형 컴포넌트를 쓰면 state와 render() 함수와 같은 우리가 구현하지 않았거나 리액트가 미리 구현해 놓은 함수를 쓸 수 있었다.
- 클래스형 컴포넌트에는 render()함수 외에도 여러 함수가 있다.
- 그 중에서도 클래스형 컴포넌트의 일생을 만들어 주는 생명주기 함수를 순서대로 알아보자!
- 생명 주기 함수를 왜 공부하냐고?
- 생명주기 함수를 이용해서 영화 데이터를 가져와야 하니까 그렇다.
- 무슨 말인지는 차차 알게 될 것이다.

3 클래스형 컴포넌트의 일생 알아보기

❖ constructor() 함수 알아보기

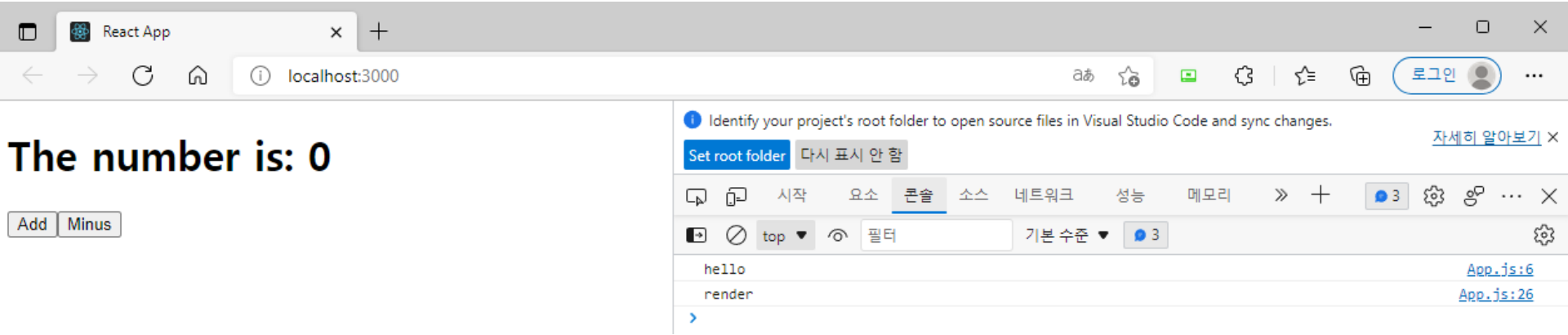
- 파일은 App.js를 그대로 사용하면 된다.
- 파일을 열고 constructor() 함수를 클래스형 컴포넌트 안에 작성하고 console.log()로 아무 문장이나 출력해 보자.
- 그리고 render() 함수에도 console.log()로 아무 문장이나 출력해 보자.
- 그런 다음 어떤 함수가 먼저 실행되는지 비교해 보자.

```
1   import React from "react";
2
3   class App extends React.Component{
4       constructor(props){
5           super(props);
6           console.log('hello');
7       }
    ...
25      render(){
26          console.log('render');
27          return (
    ...
```

3 클래스형 컴포넌트의 일생 알아보기

❖ constructor() 함수 알아보기

- [콘솔] 탭의 결과를 보면 constructor() 함수에 있는 console.log() 함수가 먼저 실행될 것이다.
- render() 함수보다 constructor() 함수가 먼저 실행된다!!



- 다음으로 알아볼 함수는 componentDidMount( )이다.
- 함수 이름에서 짐작할 수 있듯이 컴포넌트가 처음 화면에 그려지면 실행되는 함수이다.

3 클래스형 컴포넌트의 일생 알아보기

❖ componentDidMount() 함수 알아보기

- componentDidMount() 함수를 작성한 다음, 그 안에 console.log() 함수를 작성하자.
- 함수를 작성하는 위치는 App 클래스 안이면 어디든 상관없다.
- render() 함수의 console.log() 함수의 인자도 조금 수정하자.

```
...
25      componentDidMount(){
26          console.log('component rendered');
27      }
28
29      render(){
30          console.log("I'm rendering");
31          return (
32              <div>
...
```

3 클래스형 컴포넌트의 일생 알아보기

❖ componentDidMount() 함수 알아보기

- [콘솔] 탭을 보면 render() 함수가 실행된 다음 componentDidMount() 함수가 실행된 것을 알 수 있다.
- 자! 여기까지 알아본 3가지 함수가 바로 리액트에서 마운트(Mount)로 분류하는 생명주기 함수이다.
- render() 함수, constructor() 함수, componentDidMount() 함수를 잊지 말자!!

3 클래스형 컴포넌트의 일생 알아보기

❖ componentDidUpdate() 함수 알아보기

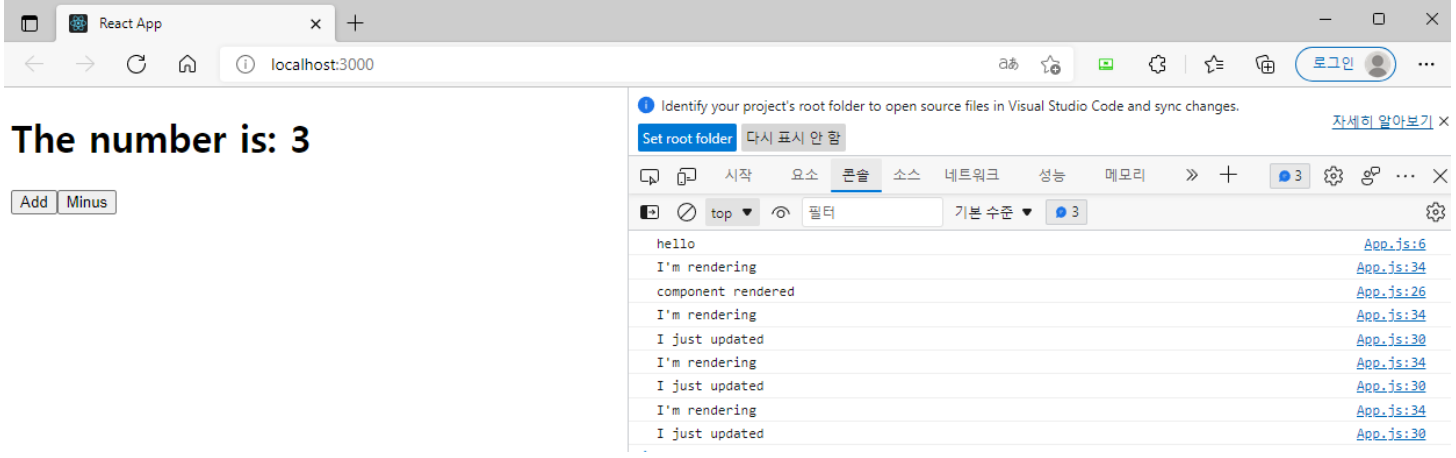
- componentDidUpdate() 함수를 작성한 다음, 그 안에 console.log() 를 작성하자.
- 이 함수의 위치도 App 컴포넌트 안이라면 어디든 괜찮아.
- 여기에서는 componentDidMount() 함수 아래에 작성했어.

```
    ...
25      componentDidMount(){
26          console.log('component rendered');
27      }
28
29      componentDidUpdate(){
30          console.log('I just updated');
31      }
32
33      render(){
34          console.log("I'm rendering");
35          return (
36              <div>
    ...
```

3 클래스형 컴포넌트의 일생 알아보기

❖ componentDidUpdate() 함수 알아보기

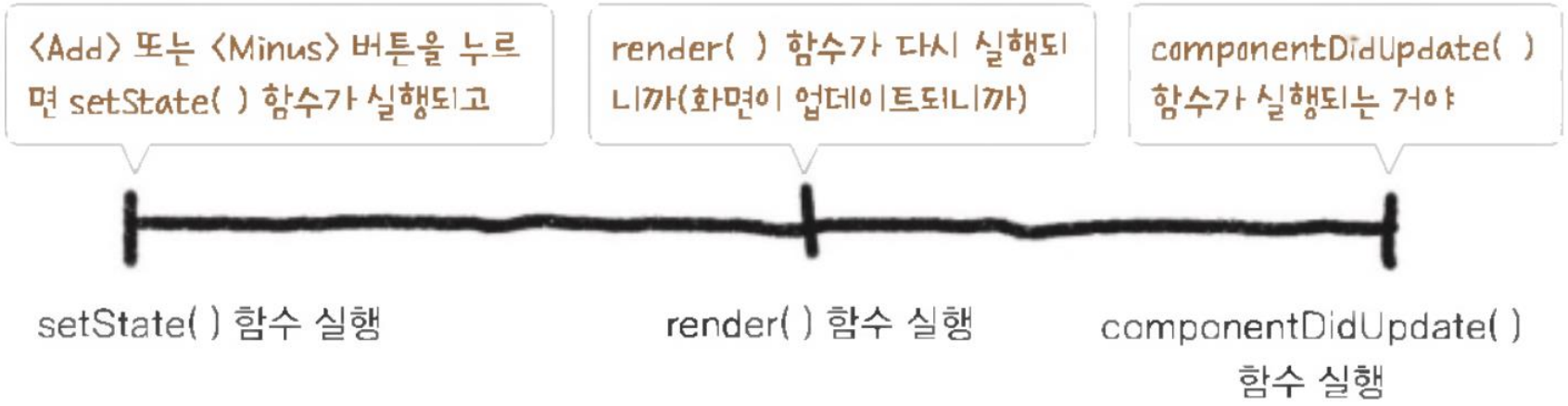
- 함수 이름에서 짐작할 수 있듯이 이 함수는 화면이 업데이트되면(새로 그려지면) 실행된다.
- 앞에서 만든 숫자 증감 앱에서 화면은 언제 업데이트되는가?
- 맞다.
- **<Add>** 또는 **<Minus>** 버튼을 눌러서 **setState()** 함수를 실행시키는 경우였다.
- **setState()** 함수가 실행되면 자동으로 **render()** 함수가 다시 실행되면서 화면이 업데이트된다.
- 그래서 **<Add>** 또는 **<Minus>**버튼을 누르면 **I'm rendering**과 **I just updated**라는 문장이 [콘솔] 탭에 출력된다.



3 클래스형 컴포넌트의 일생 알아보기

❖ componentDidUpdate() 함수 알아보기

- 자! 리액트에서 업데이트로 분류한 생명주기 함수 중 **componentDidUpdate()** 함수를 알아봤다.
- 이 함수가 실행되는 시점은 화면이 업데이트되는 경우이다.
- 꼭 기억할 것!!



- 자! 이제 마지막 단계이다.
- 리액트에서는 컴포넌트가 죽을 때를 언마운트(Unmount)라고 분류한다.
- 언마운트로 분류한 생명 주기에서 꼭 알아야 할 함수는 **componentwill Unmount()**이다.

3 클래스형 컴포넌트의 일생 알아보기

❖ componentWillUnmount() 함수 알아보기

- componentWillUnmount() 함수를 작성한 다음, 그 안에 console.log()를 작성하자.
- 이 함수 역시 App 컴포넌트 안이면 어디에 작성해도 괜찮다.

```
    ...
29      componentDidUpdate(){
30          console.log('I just updated');
31      }
32
33      componentWillUnmount(){
34          console.log('Goodbye, cruel world');
35      }
36
37      render(){
38          console.log("I'm rendering");
39          return (
40              <div>
    ...
```

3 클래스형 컴포넌트의 일생 알아보기

❖ componentWillUnmount() 함수 알아보기

- 아쉽게도 이 함수는 실행되지 않는다.
- 왜냐하면 아직 우리는 컴포넌트가 화면에서 떠나게 만드는 코드를 작성한 적이 없기 때문이다.
- componentWillUnmount() 함수는 컴포넌트가 화면에서 떠날 때 실행된다.
- 자! 이제 우리는 클래스형 컴포넌트에서 실행되는 생명주기 함수를 다 배웠다.

3 클래스형 컴포넌트의 일생 알아보기

❖ componentWillUnmount() 함수 알아보기

- 아쉽게도 이 함수는 실행되지 않는다.
- 왜냐하면 아직 우리는 컴포넌트가 화면에서 떠나게 만드는 코드를 작성한 적이 없기 때문이다.
- componentWillUnmount() 함수는 컴포넌트가 화면에서 떠날 때 실행된다.
- 자! 이제 우리는 클래스형 컴포넌트에서 실행되는 생명주기 함수를 다 배웠다.

4 영화 앱 만들기 워밍업

❖ App 컴포넌트 비우기

- 다음과 같이 App 컴포넌트를 깨끗하게 정리하자.

```
1  import React from "react";
2
3  class App extends React.Component{
4      render(){
5          return <div />;
6      }
7  }
8
9  export default App;
```

- 그런 다음 영화 앱 데이터를 로딩하는 모습을 상상해 보자.
- 처음에는 영화 앱 데이터가 없지만, 영화 앱 데이터를 로딩하면 그때는 영화 앱 데이터가 있을 것이다.
- 그런 상태를 구분해 줄 변수가 필요하다.
- 그게 바로 isLoading state이다.

4 영화 앱 만들기 워밍업

❖ 영화 데이터 로딩 상태 표시해 주기

- isLoading state를 추가해 보자.
- isLoading state는 컴포넌트가 마운트되면 true여야 하기 때문에 (처음에는 로딩 상태니까) 다음과 같이 코드를 작성하면 된다.

```
1  import React from "react";
2
3  class App extends React.Component{
4      state = {
5          isLoading: true,
6      };
7
8      render(){
9          return <div />;
10     }
11 }
12
13 export default App;
```

4 영화 앱 만들기 워밍업

❖ 영화 데이터 로딩 상태 표시해 주기

- isLoading state에 따라 '로딩 중이다', '로딩이다 됐다'와 같은 문장을 화면에 출력하면 좋을것 같다?
- 구조 분해 할당과 삼항 연산자를 활용해서 로딩 상태를 알려주는 문장을 출력하도록 만들자.

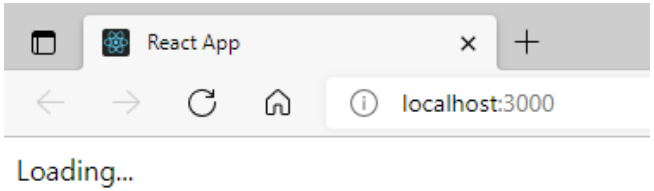
```
    ...
3   class App extends React.Component{
    ...
8       render(){
9           const {isLoading} = this.state;
10          return <div>{isLoading ? 'Loading...' : 'We are ready'}</div>;
11      }
12  }
13
14  export default App;
```

구조 분해 할당으로 this.state에 있는 isLoading을 우선 얻으면 항상 this.state를 입력하지 않아도 된다.

4 영화 앱 만들기 워밍업

❖ 영화 데이터 로딩 상태 표시해 주기

- 영화 앱을 실행해 보면 영화 데이터를 로딩하고 있음을 알려주고 있다.
- 별 내용이 없긴 하지만 영화 앱의 구성 요소를 개발했다.



- 자, 여기서 질문 하나!
- App 컴포넌트가 그려지면(render() 함수가 실행되면) 호출되는 생명 주기 함수는 무엇일까?
- 바로 componentDidMount() 함수이다.
- 이 함수에 setTimeout() 함수를 적용해서 영화 데이터가 로딩되는 현상을 구현해보자.

4 영화 앱 만들기 워밍업

❖ 로딩 현상 구현하기

- setTimeout() 함수는 첫 번째 인자로 전달한 함수를 두 번째 인자로 전달한 값(밀리초) 후에 실행한다.
- 6초 후에 isLoading state를 false로 바꿔보자.

```
...
3   class App extends React.Component{
    ...
8       componentDidMount(){
9           setTimeout(() => {
10              this.setState({isLoading: false});
11          }, 6000);
12      }
13
14      render(){
    ...
```

첫 번째 인자로 setTimeout(...)을 전달했고

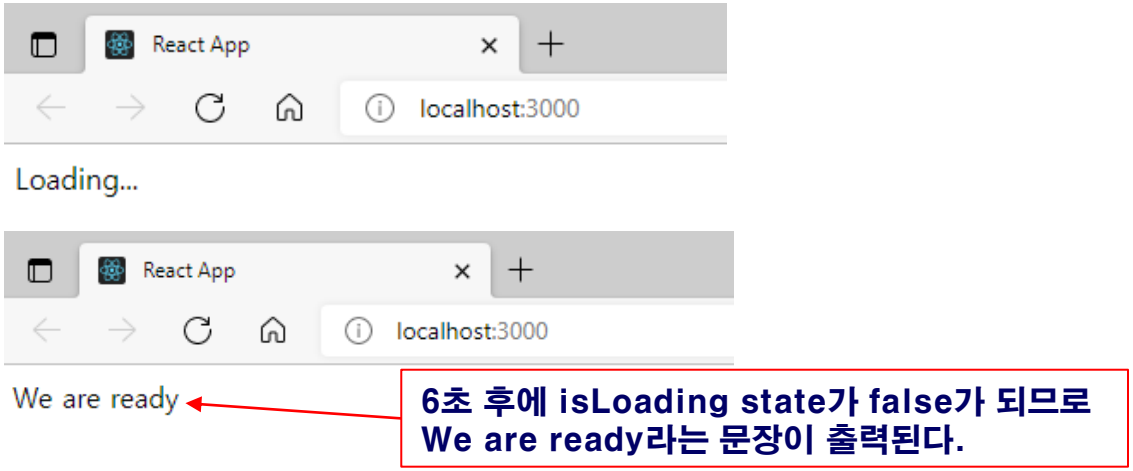
state를 바꾸려면 setState() 함수

두 번째 인자로 6000 밀리초를 전달

4 영화 앱 만들기 워밍업

❖ 로딩 현상 구현하기

- 영화 앱을 실행한 상태라면 componentDidMount() 함수가 실행될 수 있도록 새로 고침을 한다.
- 그러면 6초 후에 Loading...이라는 문장이 We are ready라는 문장으로 바뀔 것이다.



4 영화 앱 만들기 워밍업

❖ 로딩 현상 구현하기

- componentDidMount() 함수에서 무엇을 해야 할지 감이 잡히는가?
- 바로 영화 앱을 로딩하는 것이다.
- 그러려면 자바스크립트의 fetch() 함수를 알아야 하는데, fetch()함수 역시 이 책의 설명 범위를 넘어가므로 생략하겠다.
- 그리고 fetch() 함수는 리액트 초보자가 사용하기에는 난이도가 조금 있는 편이라서 fetch() 함수 대신 Axios라는 도구를 사용하겠다.
- 아직 Axios를 사용할 단계는 아니니까 Axios라는 단어는 잠시 잊자.

4 영화 앱 만들기 워밍업

❖ 영화 데이터를 어디에 저장할까?

- componentDidMount() 함수의 주석으로 된 부분을 똑같이 수정하자.
- 바로 거기에 영화데이터를 로딩할 것이다.

```
    ...
3   class App extends React.Component{
    ...
8       componentDidMount(){
9           // 영화 데이터 로딩!
10          setTimeout(() => {
11              this.setState({isLoading: false});
12          }, 6000);
13      }
14
15      render(){
16          const {isLoading} = this.state;
17          return <div>{isLoading ? 'Loading...' : 'We are ready'}</div>;
    ...
```

영화 데이터 로딩이 완료되면

여기에 영화 데이터를 출력

- 그러면 로딩된 영화 데이터는 어디에 저장해야 할까?
- state에 저장하면 된다.

4 영화 앱 만들기 워밍업

❖ 영화 데이터를 어디에 저장할까?

- 영화 데이터를 로딩한 다음 movies state에 저장하려면 어떻게 해야 할까?
- 로딩된 영화 데이터를 저장할 수 있도록 movies state를 만들자.
- 자료형은 당연히 배열이고, 여기에 객체 원소가 들어올 것이다.

```
...
3  class App extends React.Component{
4      state = {
5          isLoading: true,
6          movies: [],
7      };
8
9      componentDidMount(){
10         // 영화 데이터 로딩!
...
```

- 앱을 실행해 보면 여전히 변화가 없다.(6초 후에 We are ready라는 문장 출력).
- 이제 여기에 영화 데이터를 로딩해 보자.

Thank You !