

1. 리액트 시작하기

소개

자판기

동전 -> 음료수

리액트는 ‘뷰’ 자판기

데이터가 들어가면 뷰가 나와요

*We built React to solve one problem:
building large applications with data
that changes over time.*

데이터의 변화


```
{  
  "title": "Hello",  
  "contents": "Hello World",  
  "author": "velopert"  
  "likes": 1  
}
```

```
<div id="post-1">  
  <div class="title">Hello</div>  
  <div class="contents">Hello World</div>  
  <div class="author">velopert</div>  
  <div class="likes">1</div>  
</div>
```

페이스북의 해결법:
다 밀어버리고 새로 만들어!

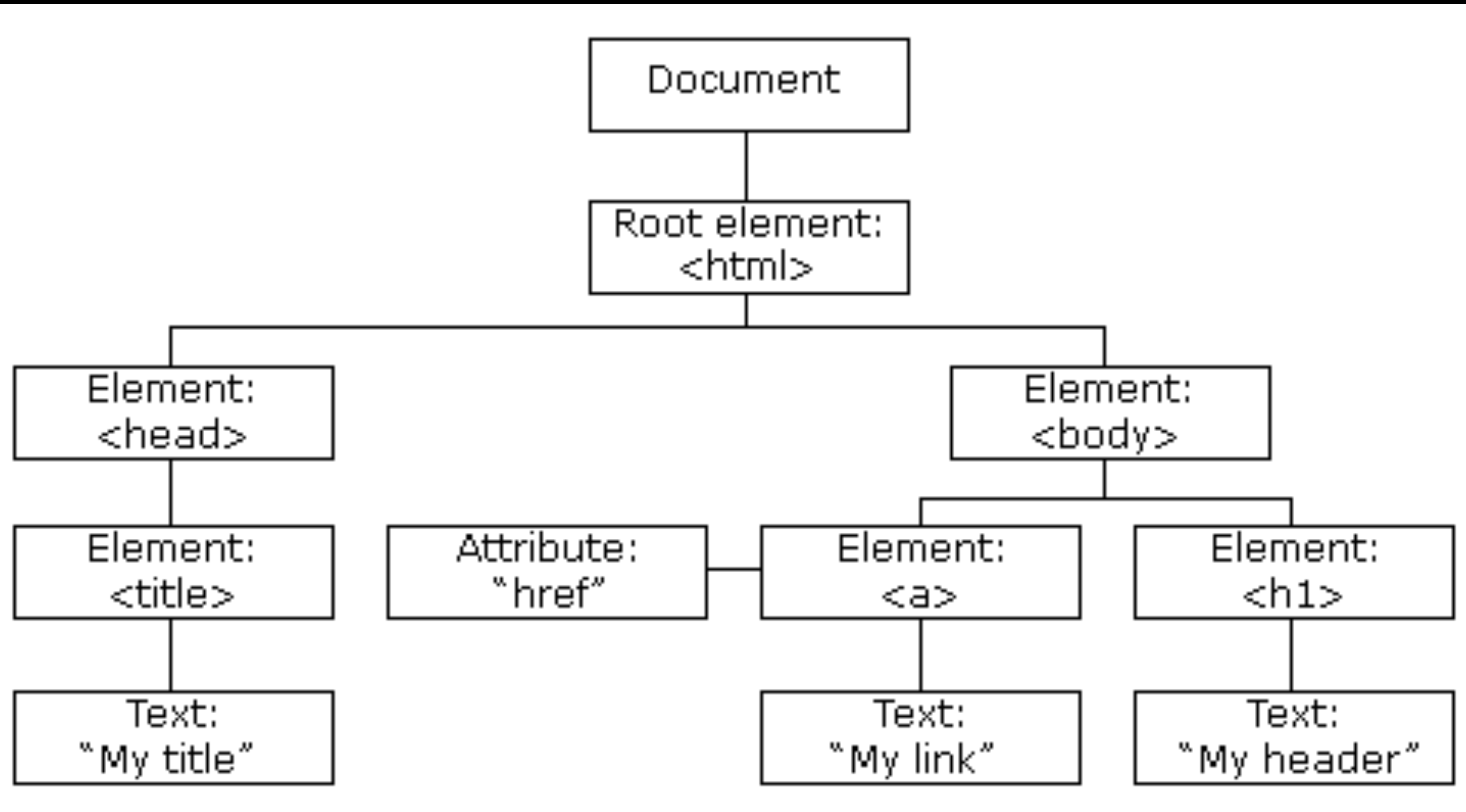
과연 이게 될까..?

Virtual DOM

<https://www.youtube.com/watch?v=muc2ZF0QIO4>

DOM?

Document Object Model



DOM 의 문제점

브라우저 레이아웃 엔진

Repaint & Reflow

```
var style = document.body.style; // 캐싱

style.padding = "20px"; // reflow, repaint
style.border = "10px solid red"; // reflow, repaint

style.color = "blue"; // repaint (레이아웃이 변경되진 않았기 때문에 reflow 안함)
style.backgroundColor = "#ffa"; // repaint

style.fontSize = "1em"; // reflow, repaint

// reflow, repaint
document.body.appendChild(document.createTextNode('hello world!'));
```

브라우저는 바보가 아니다

Batched DOM Updates

```
var style = document.body.style; // 캐싱

style.padding = "20px"; // reflow, repaint
style.border = "10px solid red"; // reflow, repaint

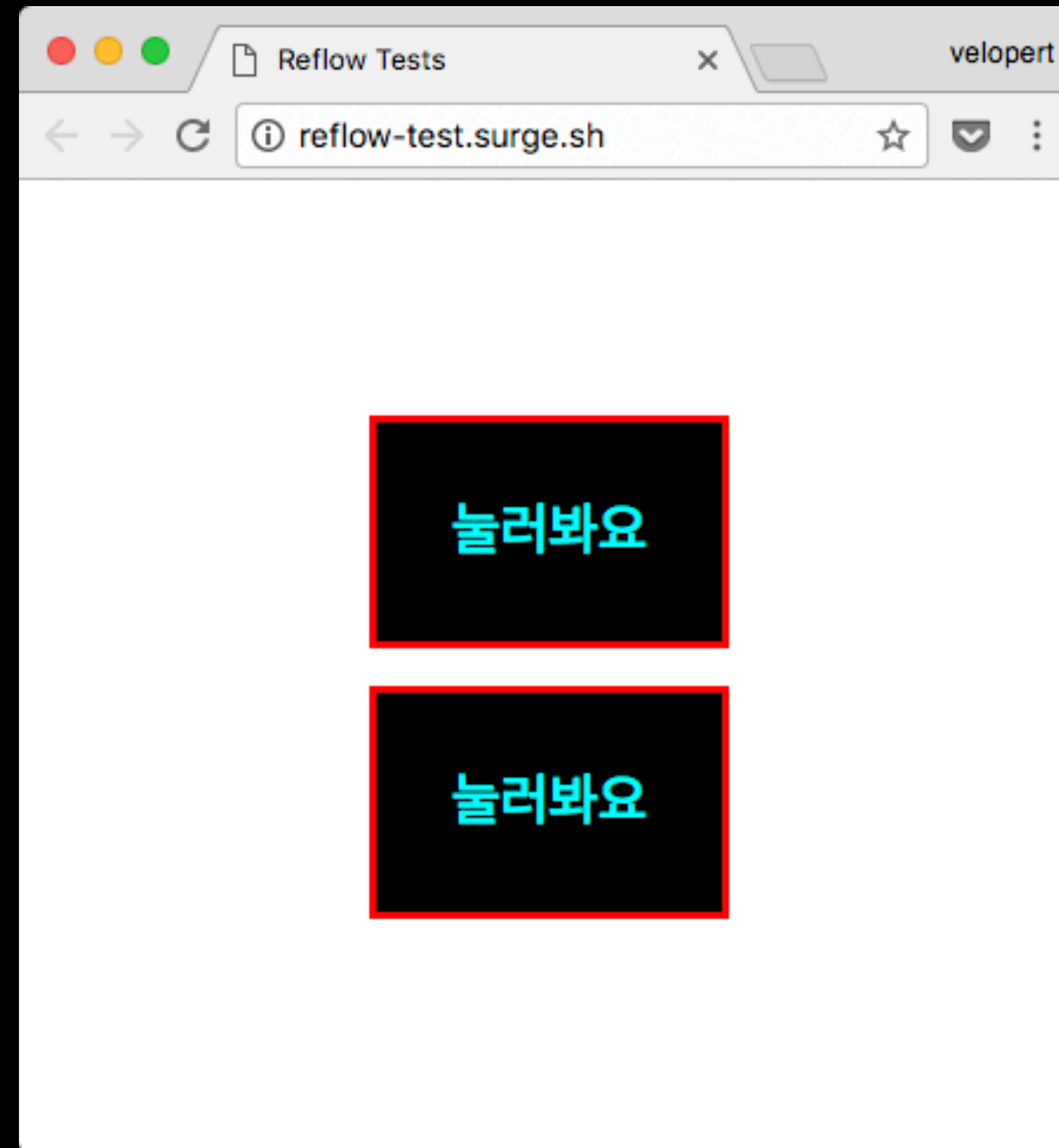
style.color = "blue"; // repaint (레이아웃이 변경되진 않았기 때문에 reflow 안함)
style.backgroundColor = "#ffa"; // repaint

style.fontSize = "1em"; // reflow, repaint

// reflow, repaint
document.body.appendChild(document.createTextNode('hello world!'));
```

Forced Reflow

<https://gist.github.com/paulirish/5d52fb081b3570c81e3a>



<http://reflow-test.surge.sh/>


```
<body>
  <div>
    <div class="block" id="one">눌러봐요</div>
    <div class="block" id="two">눌러봐요</div>
  </div>
  <script>
var one = document.getElementById('one');
var two = document.getElementById('two');
one.onclick = () => {
  one.style.background = 'red'
  one.style.padding = '0.5rem';
  one.style.margin = '100px'
  one.style.height = '100px';
  // one reflow, one repaint
}
two.onclick = () => {
  two.style.background = 'red'
  two.style.padding = '0.5rem';
  console.log(one.offsetHeight);
  two.style.margin = '100px';
  console.log(one.clientHeight);
  two.style.height = '100px';
  // three reflows, one repaint
}
</script>
```

Reflow Tests

안전함 <https://reflow-test.now.sh>

눌러봐요

눌러봐요

Elements Timeline

Screenshots Memory

500ms 1000ms 1500ms 2000ms

FPS

CPU

NET

500 ms 1000 ms 1500 ms 2000 ms 2!

Frames

Interactions

Input

Main

Summary Bottom-Up Call Tree Event Log

Layout All Loading

Scripting Rendering Painting

Start Time	Self Time	Total Time	Activity
1170.5ms	0.1ms	0.1ms	Layout reflow-tes...

Animations Console

Reflow Tests

안전함 <https://reflow-test.now.sh>

눌러봐요

눌러봐요

ElementsTimeline

ScreenshotsMemory

500ms1000ms1500msFPSCPUNET

500 ms1000 ms1500 ms

FramesInteractionsInputMain

SummaryBottom-UpCall TreeEvent Log

LayoutAllLoading

ScriptingRenderingPainting

Start Time	Self Time	Total Time	Activity
601.9ms	0.1ms	0.1ms	Layout (index):62
602.6ms	0.1ms	0.1ms	Layout (index):64
603.0ms	0.1ms	0.1ms	Layout (index):65

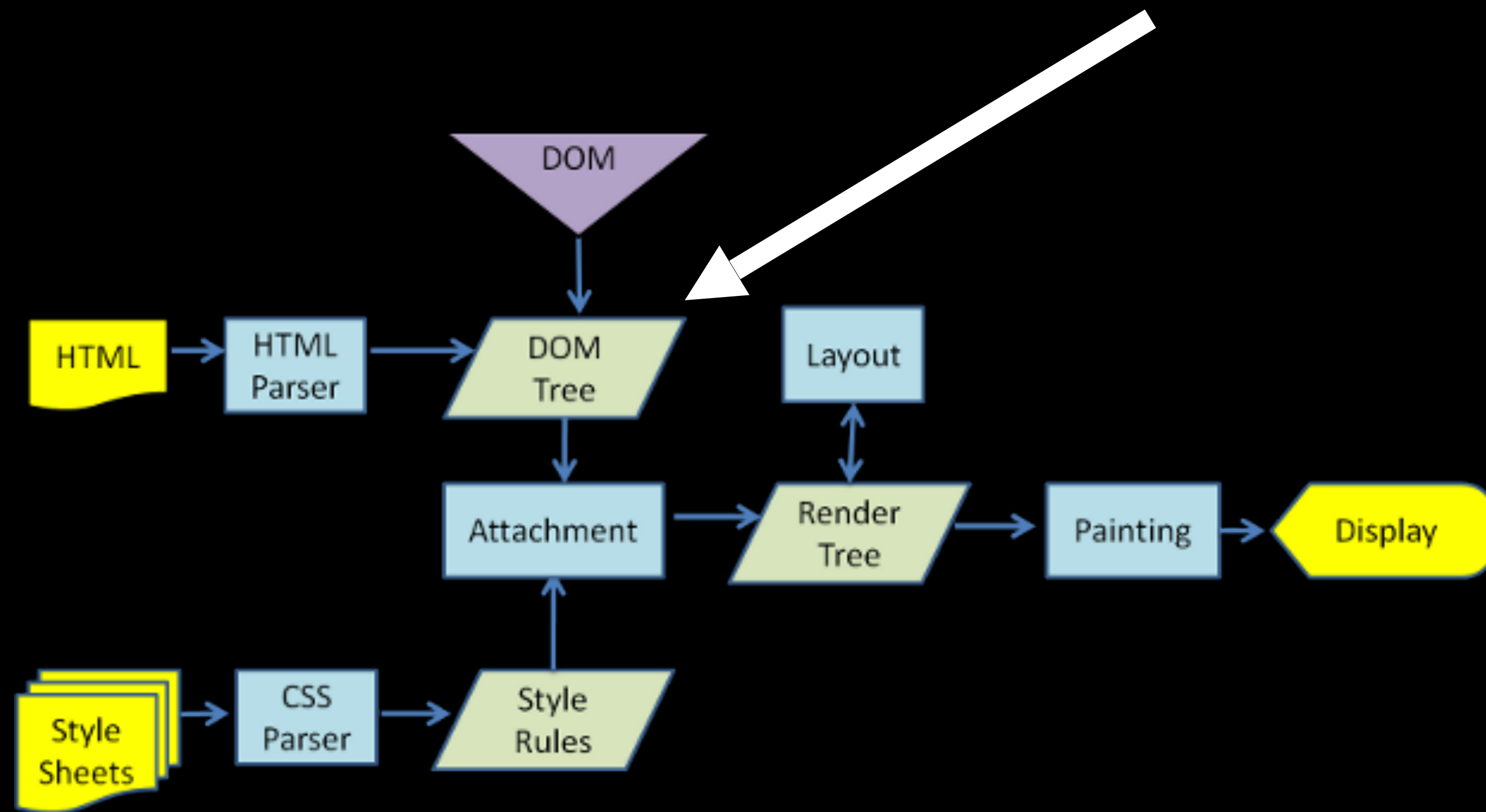
Animations x Console

Summary Bottom-Up Call Tree Event Log			
<div> <div>Paint</div> <div>All</div> <div> <input checked="" type="checkbox"/> Loading </div> <div> <input checked="" type="checkbox"/> Scripting </div> <div> <input checked="" type="checkbox"/> Rendering </div> <div> <input checked="" type="checkbox"/> Painting </div> </div>			
Start Time	Self Time	Total Time	Activity
603.3ms	0.0ms	0.0ms	<div> <div></div> <div>Paint</div> </div>

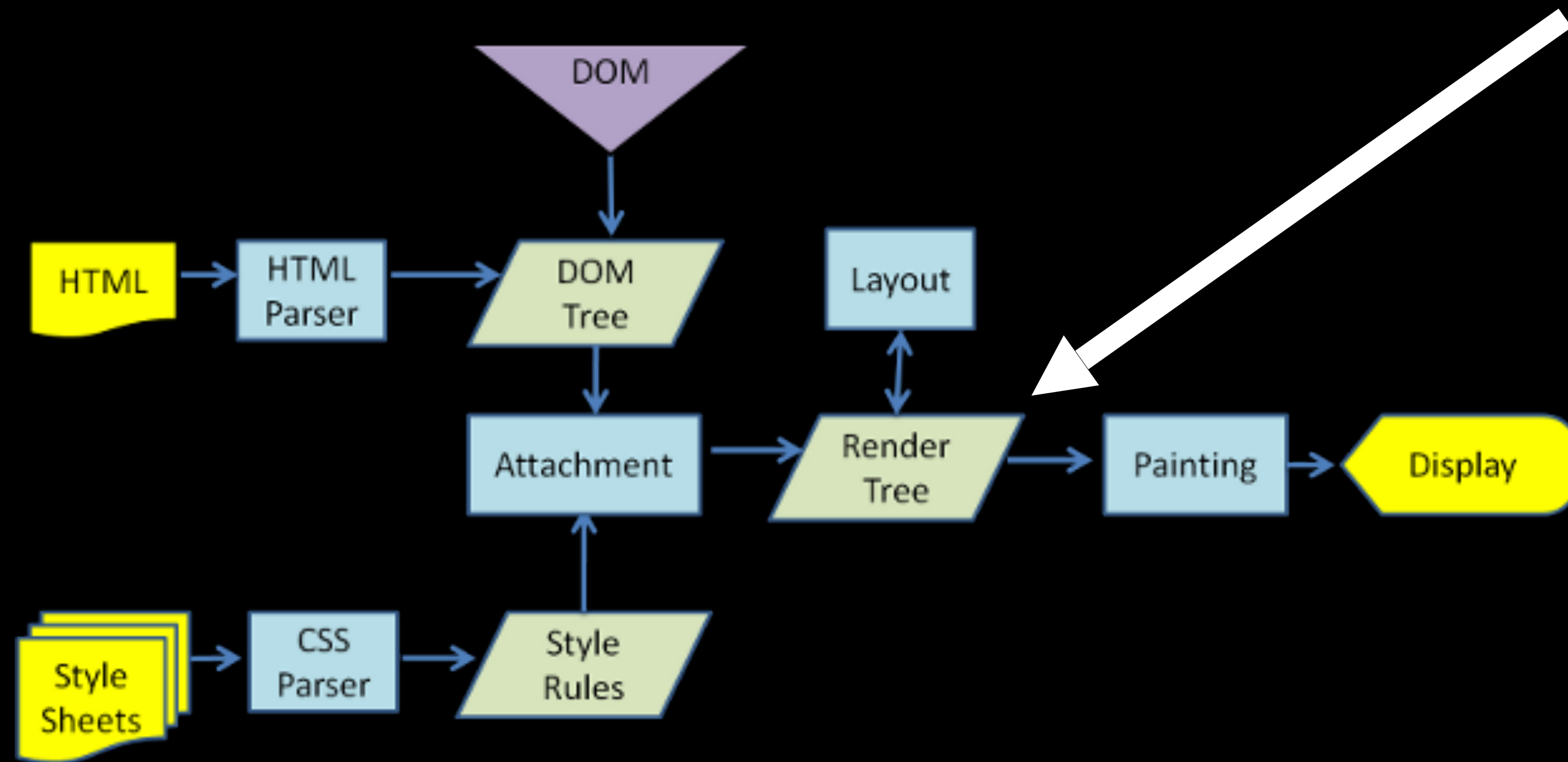
Reflow 최적화

<https://www.sitepoint.com/10-ways-minimize-reflows-improve-performance/>

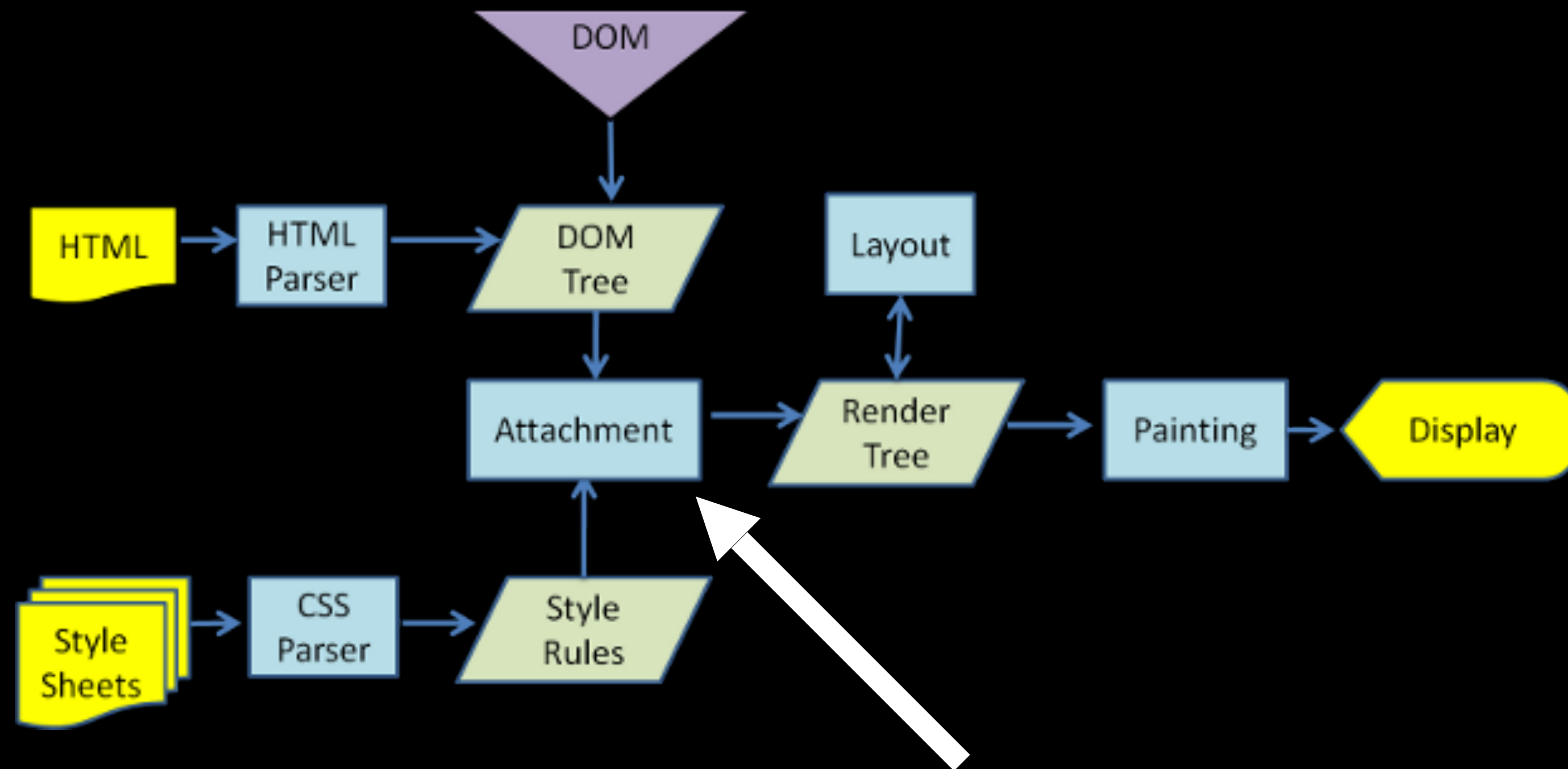
브라우저의 작동 방식



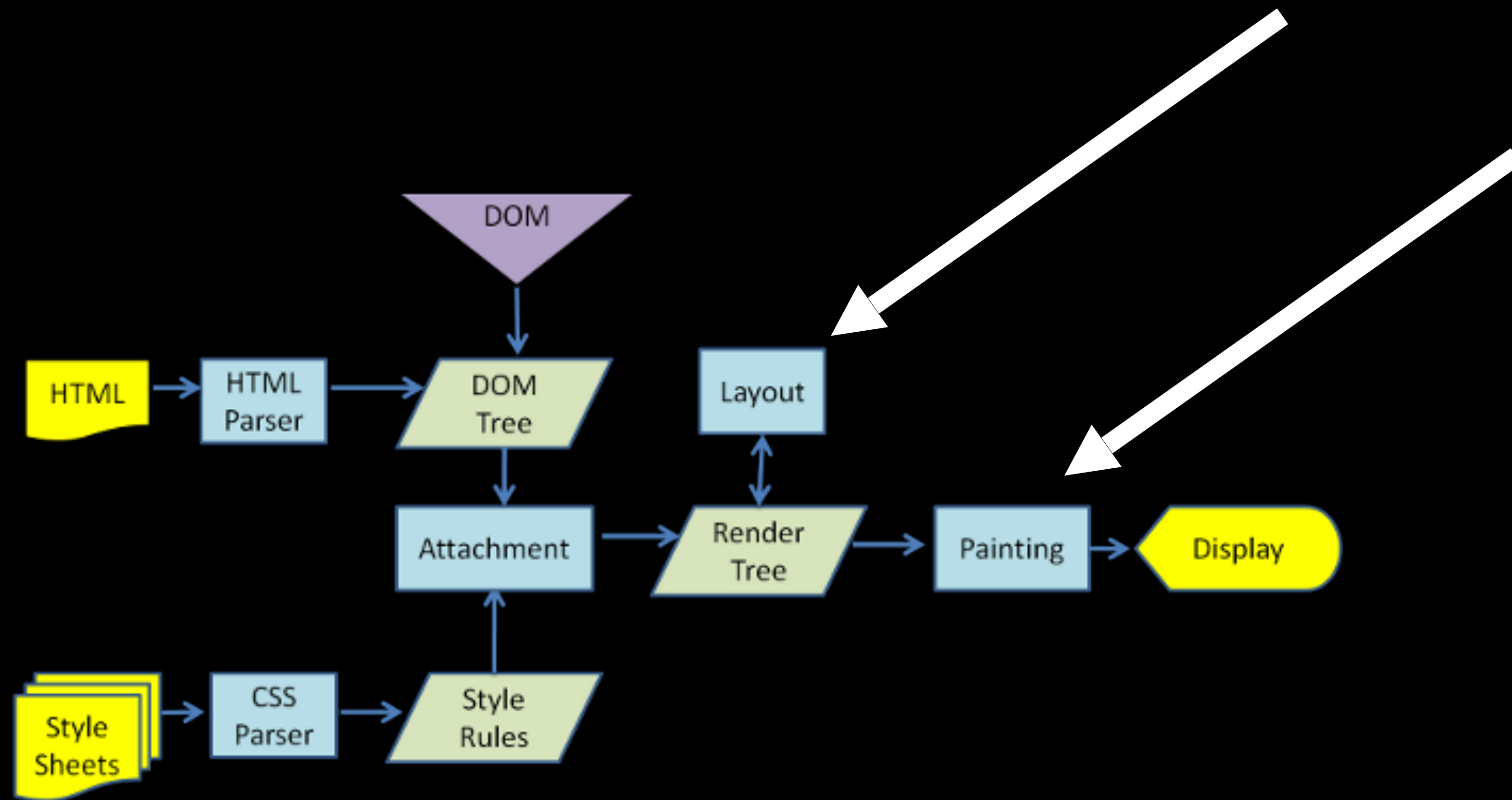
DOM Tree 생성



Render Tree 생성

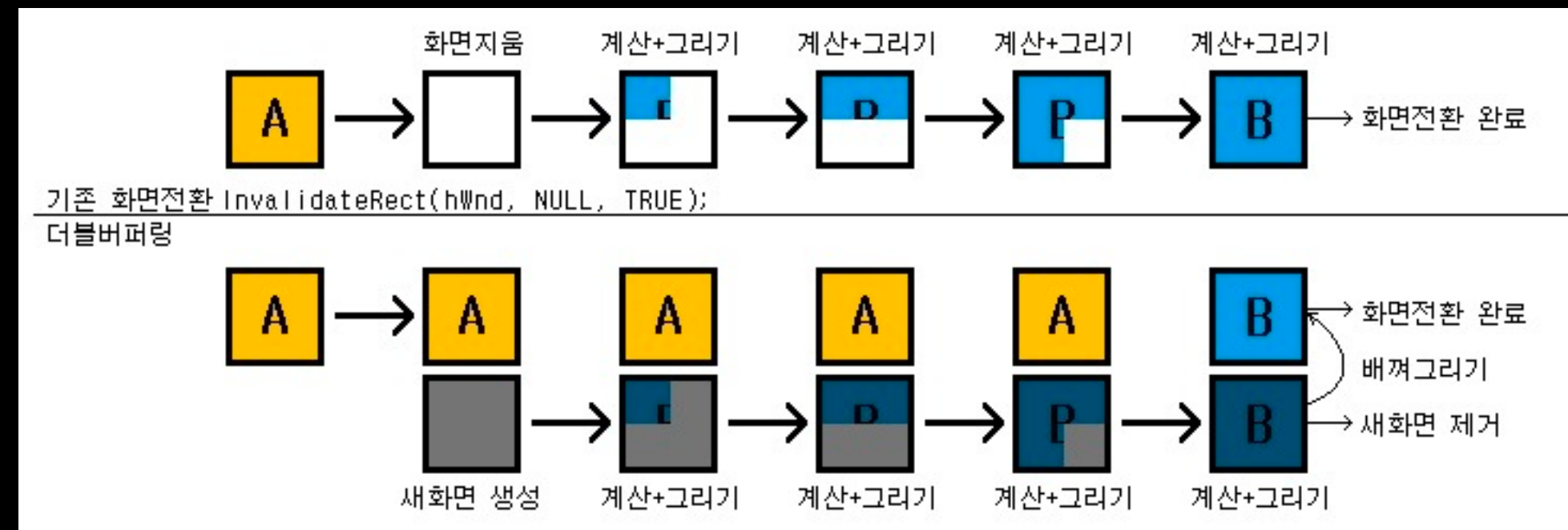


Attachment: 노드의 스타일을 처리하는 과정



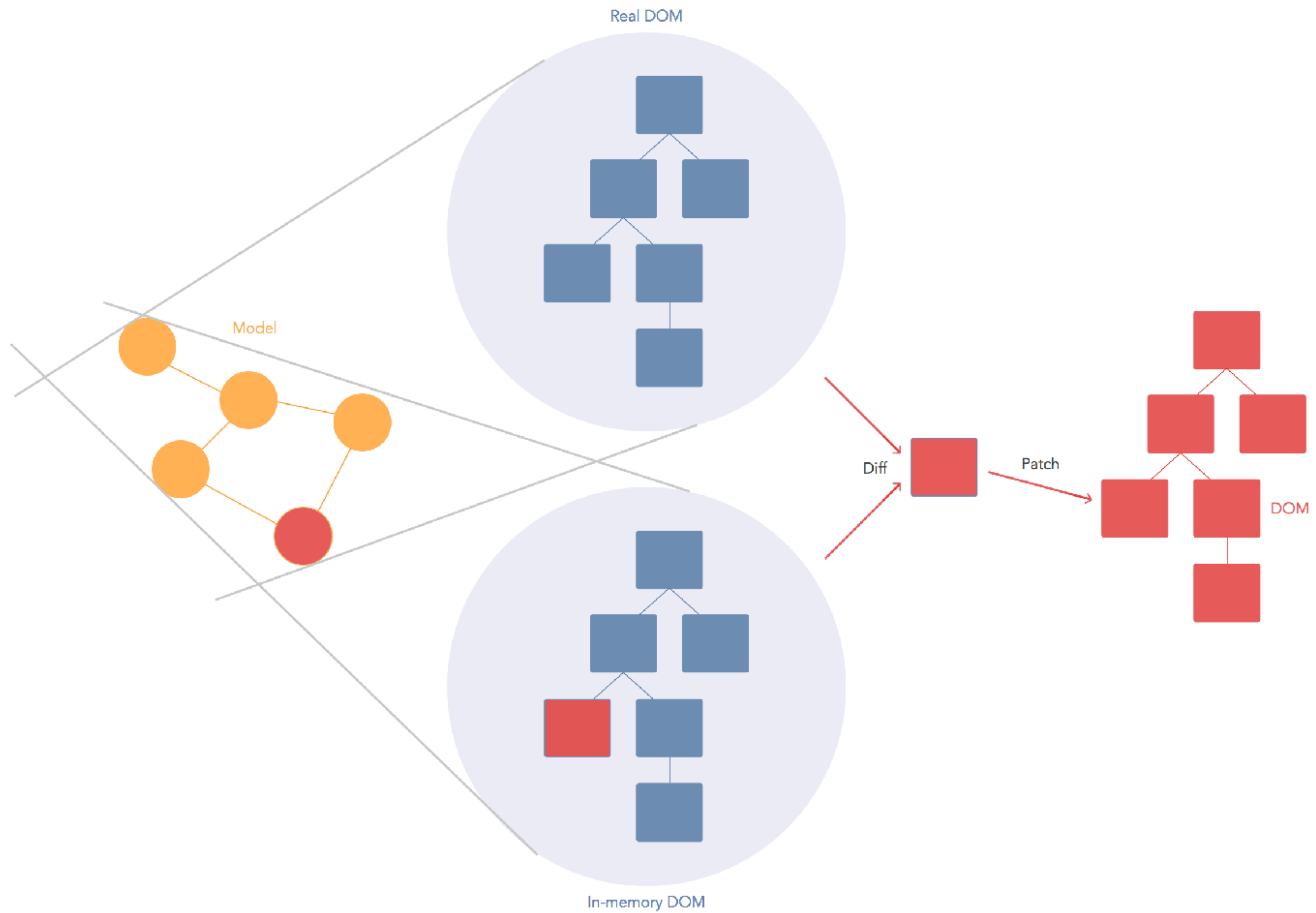
Layout (reflow) & Painting (repaint)

Virtual DOM!



(출처 : <http://cafe.naver.com/buildgame.cafe>)

더블 버퍼링



출처: <https://auth0.com/blog/face-off-virtual-dom-vs-incremental-dom-vs-glimmer/>

React 가 해주는건?

컴포넌트

Reconciliation (조화)

비교 알고리즘

비교 알고리즘

- 원래는 최소 $O(n^3)$ 의 비교를 해야합니다
- 이 과정을 최적화하여 $O(n)$ 의 비교를 합니다

두가지 전제

같은 클래스를 가진 컴포넌트들은 비슷한 DOM 트리를 가지고있고,
다른 클래스를 가진 컴포넌트는 서로 다른 DOM 트리를 가지고 있다.

(리스트를 렌더링 할 때 해당)
엘리먼트에 key 값을 설정 함으로서, 엘리먼트에 고유 값을 주고,
이를 통하여 렌더링시 새로 렌더링하지 않고 유지시킨다.

엘리먼트 타입이 다를 경우

```
renderA: <div />  
renderB: <span />
```

div 제거 span 추가

엘리먼트 타입이 다를 경우

```
renderA: <Header />  
renderB: <Content />
```

Header 제거 Content 추가

엘리먼트 타입이 같을 경우

```
renderA: <div className="hello" />  
renderB: <div className="world" />
```

className 만 변경

엘리먼트 타입이 같을 경우

```
renderA: <Counter number={0} />  
renderB: <Counter number={1} />
```

컴포넌트가 사라지지 않음

props 값이 변함

컴포넌트 라이프사이클 호출

리스트를 렌더링 할 때

```
renderA:
  <div>
    <span>Hello</span>
  </div>
renderB:
  <div>
    <span>Hello</span>
    <span>World</span>
  </div>
```

World 삽입


```
renderA:
  <div>
    <span>Hello</span>
  </div>
renderB:
  <div>
    <span>World</span>
    <span>Hello</span>
  </div>
```

Hello 의 내용을 World 로 변경
그 뒤에 Hello 삽입

해결: *key*

renderA:

```
<div>  
  <span key="hello">Hello</span>  
</div>
```

renderB:

```
<div>  
  <span key="hello">Hello</span>  
  <span key="world">World</span>  
</div>
```

World 는 그대로 두고 그 앞에

Hello 삽입

요약

- 다르게 생겼으면 그 내부 비교 하지 않음
- key 를 사용해서 리스트 렌더링 성능 최적화

LifeCycle API

shouldComponentUpdate

Browser DOM Update

오해: React는 DOM보다 빠르다



Dan Abramov
@dan_abramov

팔로잉

▼

Myth: React is “faster than DOM”. Reality: it helps create maintainable applications, and is **fast enough** for most use cases.

🌐 영어 번역하기

▲ bigmanwalter 11 hours ago [-]

▼

The React documentation says it so it has to be true. DOM slow, VDOM fast. Facebook is out there buying developer mindshare.

[reply](#)

* 1 point by danabramov 1 minute ago | edit | delete [-]

We don't claim that in React documentation. React can't be faster than the same DOM mutations written by hand because by definition it has to do more work.

We do think it helps to create maintainable apps though, and it is *fast enough* for practical use cases despite its immutable design. That's why we use it at Facebook.

[reply](#)

번역: React가 DOM 보다 빠르다는건 잘못된 사실이에요. 사실은: 유지보수 가능한 어플리케이션을 만드는것을 도와주고 그리고 대부분의 경우에 '충분히 빨라요'

쉽.

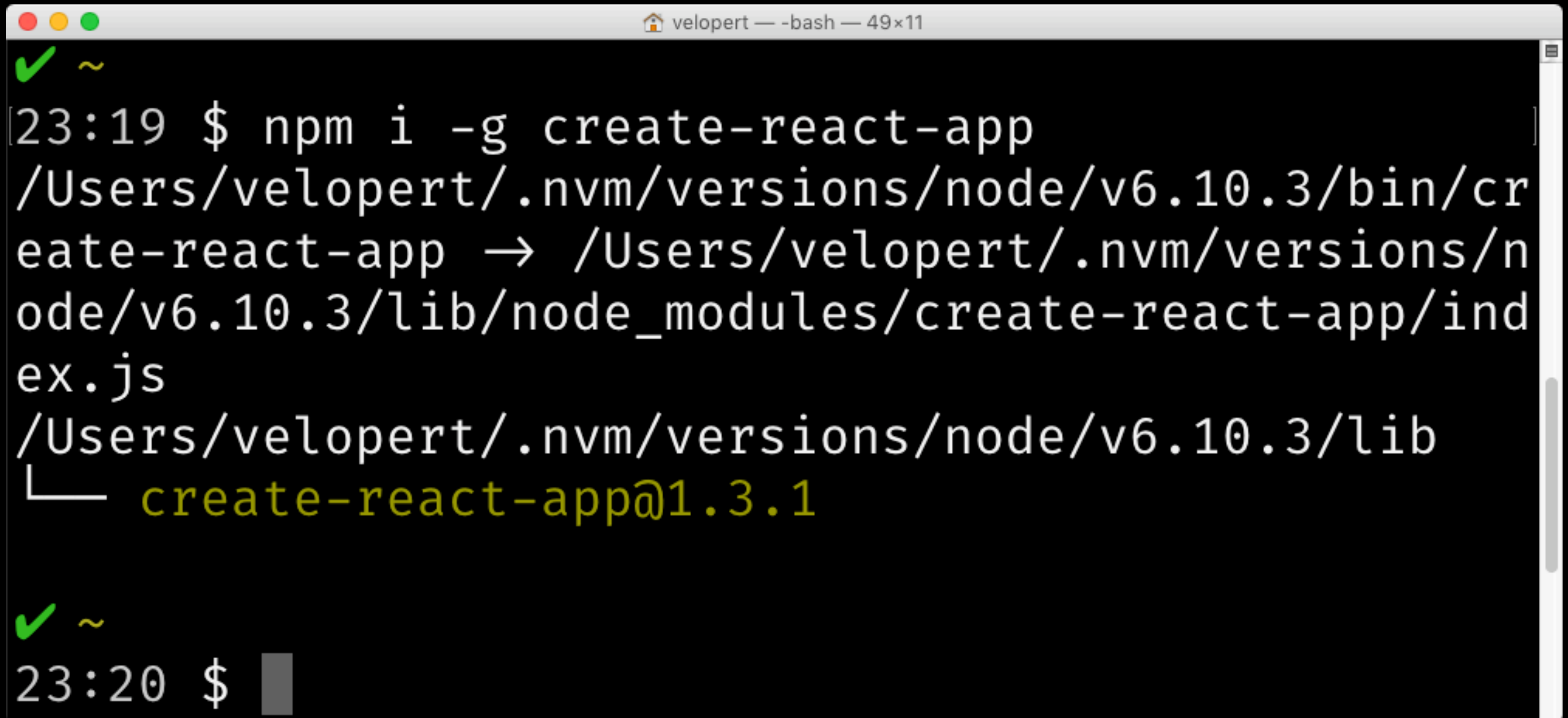
Node.js / Yarn / 에디터 사전 설치: <https://git.io/vSkPr>

작업환경 설정

```

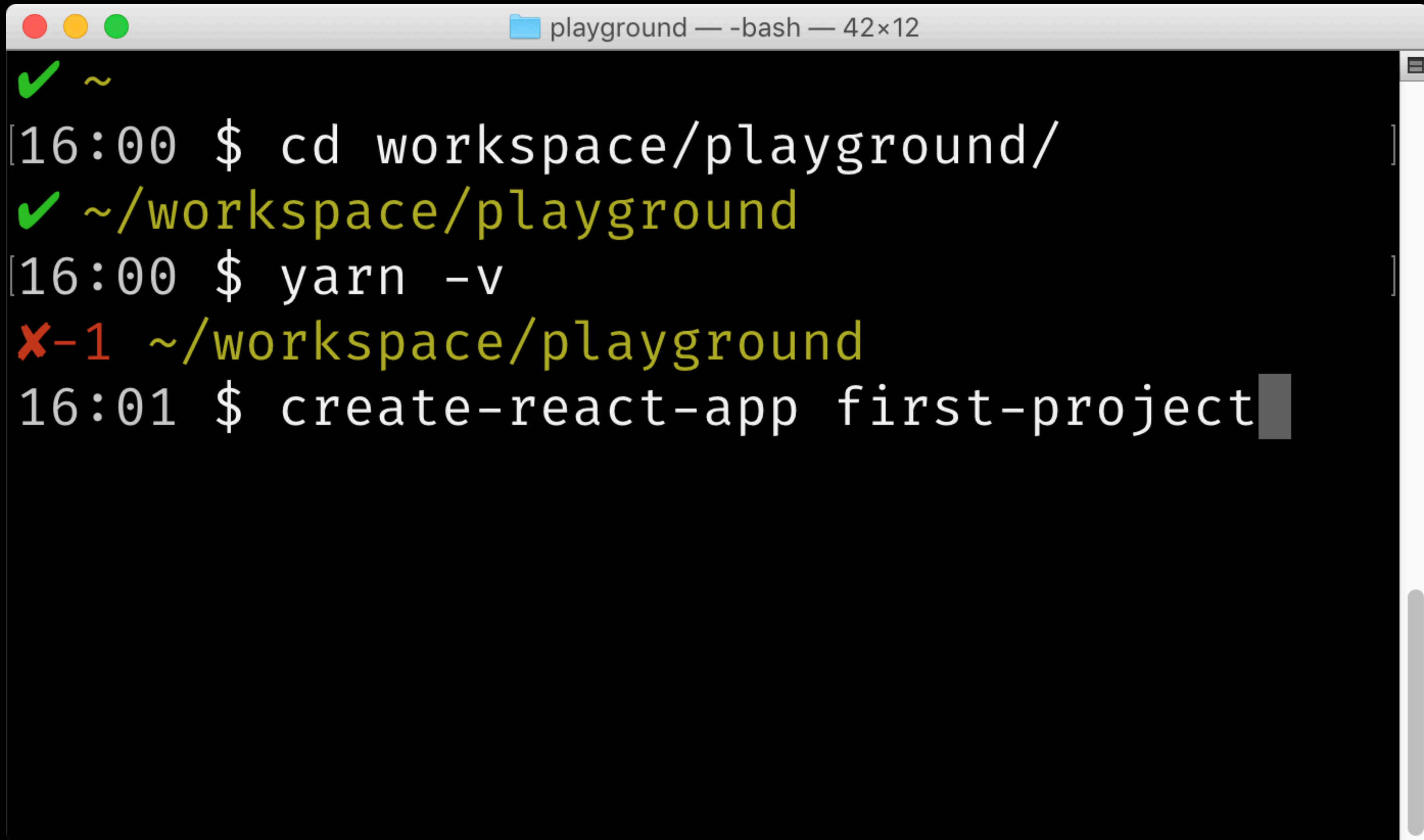
    velopert — -bash — 43x10
Last login: Sat Mar 25 15:54:35 on ttys000
✓ ~
[15:54 $ node -v
v6.10.1
✓ ~
[15:55 $ yarn --version
0.21.3
✓ ~
15:55 $ █

```



A terminal window titled "velopert — -bash — 49x11" with standard macOS window controls (red, yellow, green buttons). The terminal shows a successful installation of create-react-app. It starts with a green checkmark and a tilde symbol (~). The command "npm i -g create-react-app" is entered at 23:19. The output shows the package being installed globally to the node_modules directory of the current user's npm configuration. The version "create-react-app@1.3.1" is highlighted in yellow. The prompt returns to the shell at 23:20, indicated by another green checkmark and tilde symbol.

```
✓ ~  
[23:19 $ npm i -g create-react-app  
/Users/velopert/.npm/versions/node/v6.10.3/bin/create-react-app → /Users/velopert/.npm/versions/node/v6.10.3/lib/node_modules/create-react-app/index.js  
└─ create-react-app@1.3.1  
✓ ~  
23:20 $ █
```



```
✓ ~  
[16:00 $ cd workspace/playground/  
✓ ~/workspace/playground  
[16:00 $ yarn -v  
x-1 ~/workspace/playground  
16:01 $ create-react-app first-project
```



Success! Created first-project at /Users/velopert/workspace/playground/first-project
Inside that directory, you can run several commands:

`yarn start`

Starts the development server.

`yarn run build`

Bundles the app into static files for production.

`yarn test`

Starts the test runner.

`yarn run eject`

Removes this tool and copies build dependencies, configuration files
and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

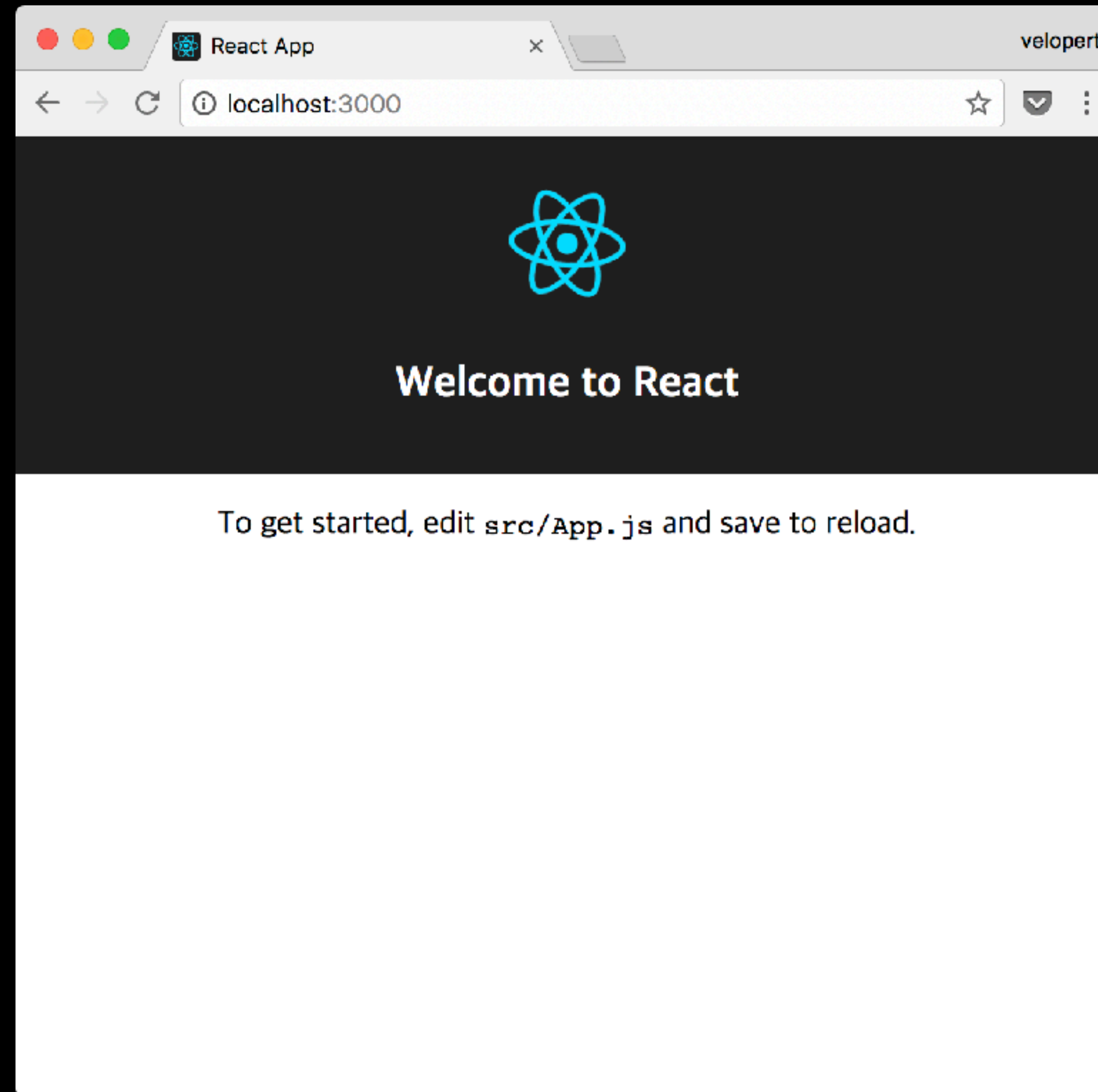
`cd first-project`

`yarn start`

Happy hacking!

✓ ~/workspace/playground

16:00 \$ █



JSX

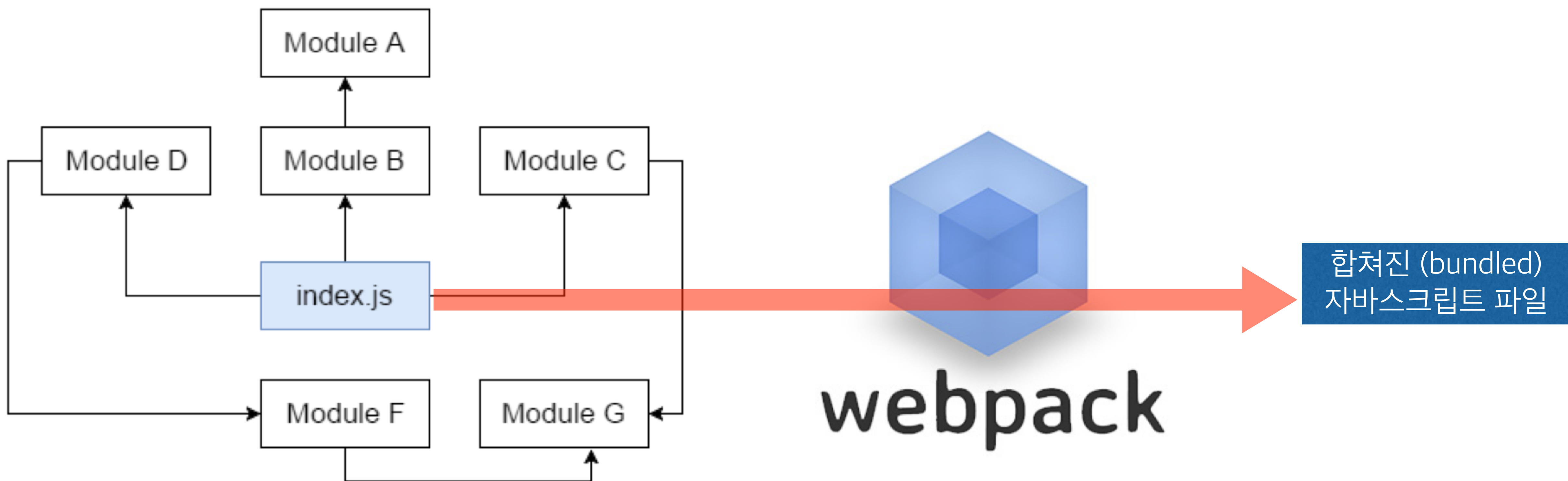
App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

```
var React = require('react');
var Component = React.Component;
```



App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome to React</h2>
        </div>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}
export default App;
```

JSX: 자바스크립트의 확장문법


```
1 var a = (  
2   <div>  
3     <h1>Awesome <b>React</b></h1>  
4   </div>  
5 )  
6
```

```
1 "use strict";  
2  
3 var a = React.createElement(  
4   "div",  
5   null,  
6   React.createElement(  
7     "h1",  
8     null,  
9     "Awesome ",  
10    React.createElement(  
11      "b",  
12      null,  
13      "React"  
14    )  
15  )  
16 );
```

<http://bit.ly/2n1PrMy>

JSX 의 장점

보기 쉽고 익숙하다

에러검사

더 넓은 활용도

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

ReactDOM.render?

JSX 문법

1) 감싸져있는 엘리먼트

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    return (
      <h1>안녕!</h1>
      <h2>즐거운 리액트</h2>
    );
  }
}
export default App;
```

Failed to compile.

Error in ./src/App.js

Syntax error: Adjacent JSX elements must be wrapped in an enclosing tag (8:15)

```
6 |  
7 |  
> 8 |      <h1>안녕 !</h1>  
   |      <h2>즐거운 리액트</h2>  
   |      ^  
9 |  
10 |    );  
11 |  }
```

@ ./src/index.js 13:11-27

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    return (
      <div>
        <h1>안녕!</h1>
        <h2>즐거운 리액트</h2>
      </div>
    );
  }
}
export default App;
```

2) JavaScript 표현

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    const name = 'velopert';
    return (
      <div>
        <h1>안녕!</h1>
        <h2>즐거운 리액트</h2>
        <p>내 이름은 {name}</p>
      </div>
    );
  }
}
export default App;
```

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    const name = 'velopert';
    return (
      <div>
        <h1>안녕!</h1>
        <h2>즐거운 리액트</h2>
        <p>내 이름은 {name}</p>
      </div>
    );
  }
}
export default App;
```

var 은 scope 가 함수단위 입니다

```
var a = "hello";  
if(true) {  
    var a = "bye";  
    console.log(a); // bye  
}  
console.log(a); // bye
```


const와 let은 scope 가 블록단위 입니다

```
let a = "hello";  
if(true) {  
    let a = "bye";  
    console.log(a); // bye  
}  
console.log(a); // hello
```

3) if문 대신 조건부 연산자

[condition] ? [true] : [false]

1+1 === 2 ? “정답” : “오답”

```
import React, {Component} from 'react';
class App extends Component {
  render() {
    const name = 'velopert';
    return (
      <div>
        <h1>안녕!</h1>
        <h2>즐거운 리액트</h2>
        <p>내 이름은 {name}</p>
        <p>{ 1 + 1 ≡ 2 ? '정답' : '바보' }</p>
      </div>
    );
  }
}
export default App;
```

4) inline styling

```

import React, {Component} from 'react';
class App extends Component {
  render() {
    const name = 'velopert';
    const style = {
      color: 'aqua',
      backgroundColor: 'black'
    }
    return (
      <div>
        <h1>안녕!</h1>
        <h2 style={style}>즐거운 리액트</h2>
        <p>내 이름은 {name}</p>
        <p>{ 1 + 1 ≡ 2 ? '정답' : '바보' }</p>
      </div>
    );
  }
}
export default App;

```

5) class 대신 className

App.css

```
.hello {  
  color: pink;  
}
```

```

import React, {Component} from 'react';
import './App.css';
class App extends Component {
  render() {
    const name = 'velopert';
    const style = {
      color: 'aqua',
      backgroundColor: 'black'
    }
    return (
      <div>
        <h1 className="hello">안녕!</h1>
        <h2 style={style}>즐거운 리액트</h2>
        <p>내 이름은 {name}</p>
        <p>{ 1 + 1 ≡ 2 ? '정답' : '바보' }</p>
      </div>
    );
  }
}
export default App;

```

6) 꼭 달여야하는 태그

<div></div>

self-closing tag: <div/>

7) 주식

```

return (
  <div>
    { /* 주석은 이렇게 작성 */ }
    <h1
      className="hello" // 혹은 이렇게 작성
      // 여기에도 작성
      /* 이렇게도 작성 */
    >안녕!</h1>
    <h2 style={style}>즐거운 리액트</h2>
    <p>내 이름은 {name}</p>
    <p>{1 + 1 === 2
      ? '정답'
      : '바보'}</p>
    // 여기에쓰면 렌더링 되버려요
    /* 이것도 마찬가지로. */
  </div>
);

```

숨.

컴포넌트

LAFTEL

홈

취향저격

탐색

Q 검색...

tester님

▼

애니

웹툰

만화

라노벨

전체

순정

로맨스

판타지

SF

재난

스릴러

공포

액션

무협

시대물

스포츠

일상

개그

tester님에게 추천하는 [전체] 작품

애니

라퐁젤

극장판

로맨스, 판타지, 시대물

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 3.69

재밌게 본 <시간을 달리는 소녀>과 유사한 작품

★★★★★

♡

👍

🚫

애니

헌터X헌터 리메이크 (2011)

TVA

토가시 요시히로

액션, 모험, 판타지

<헌터X헌터> 작가의 또 다른 작품

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 3.92

★★★★☆

♡

👍

🚫

애니

다다미 닥 장 반 세계일주

TVA

로맨스, 개그

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 4.59

재밌게 본 <슈타인즈 게이트>과 유사한 작품

★★★★★

♡

👍

🚫

105

LAFTEL

홈

취향저격

탐색

Q 검색...

tester님

애니

웹툰

만화

라노벨

전체

순정

로맨스

판타지

SF

재난

스릴러

공포

액션

무협

시대물

스포츠

일상

개그

tester님에게 추천하는 [전체] 작품

애니

라퐁젤

극장판

로맨스, 판타지, 시대물

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 3.69

재밌게 본 <시간을 달리는 소녀>과 유사한 작품

★★★★★

♡

👍

🚫

애니

헌터X헌터 리메이크 (2011)

TVA

토가시 요시히로

액션, 모험, 판타지

<헌터X헌터> 작가의 또 다른 작품

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 3.92

★★★★☆

♡

👍

🚫

애니

다다미 닥 장 반 세계일주

TVA

로맨스, 개그

나와 비슷한 취향의 사람들이 높게 평가한 작품. 예상 별점 4.59

재밌게 본 <슈타인즈 게이트>과 유사한 작품

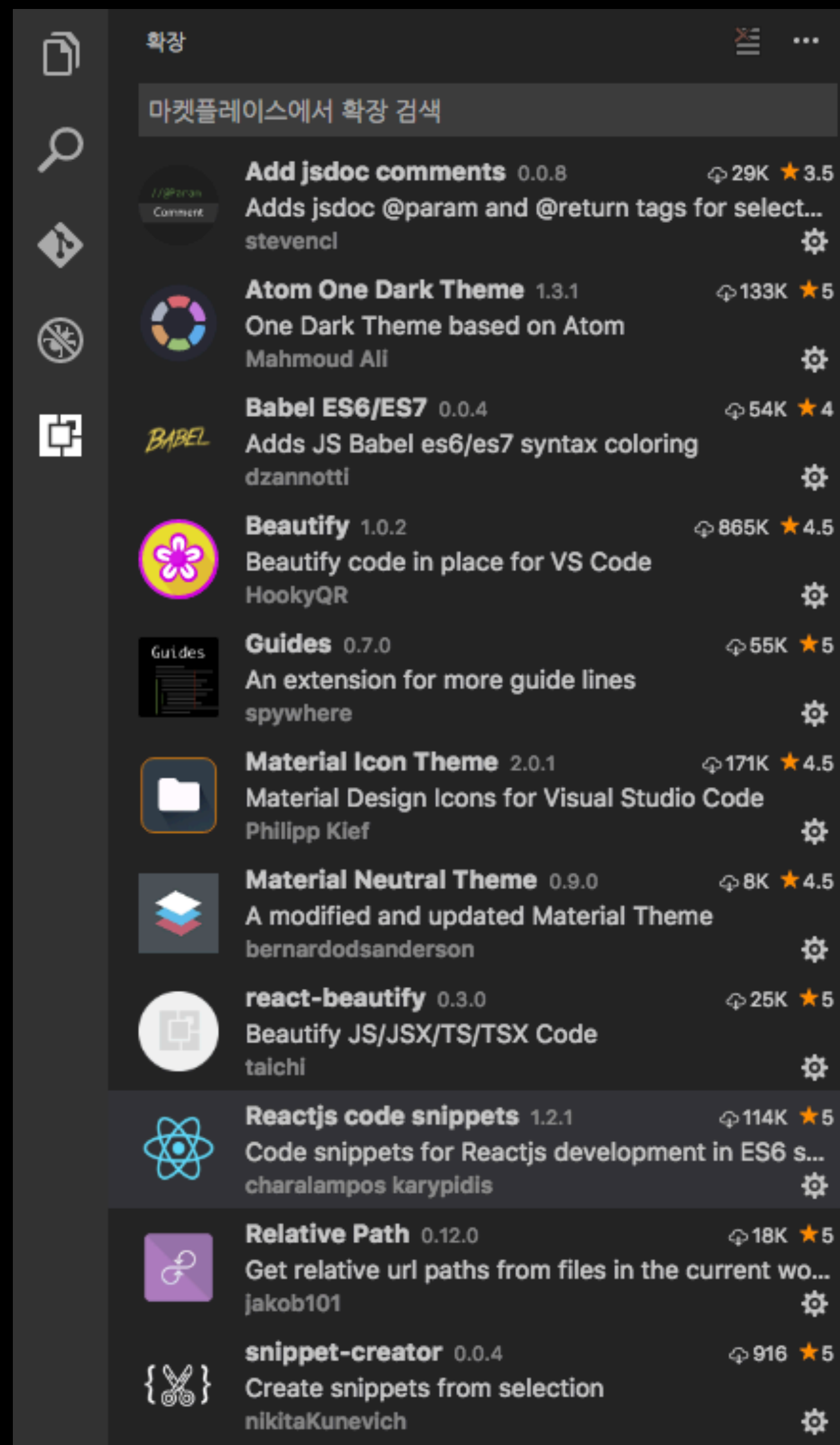
★★★★☆

♡

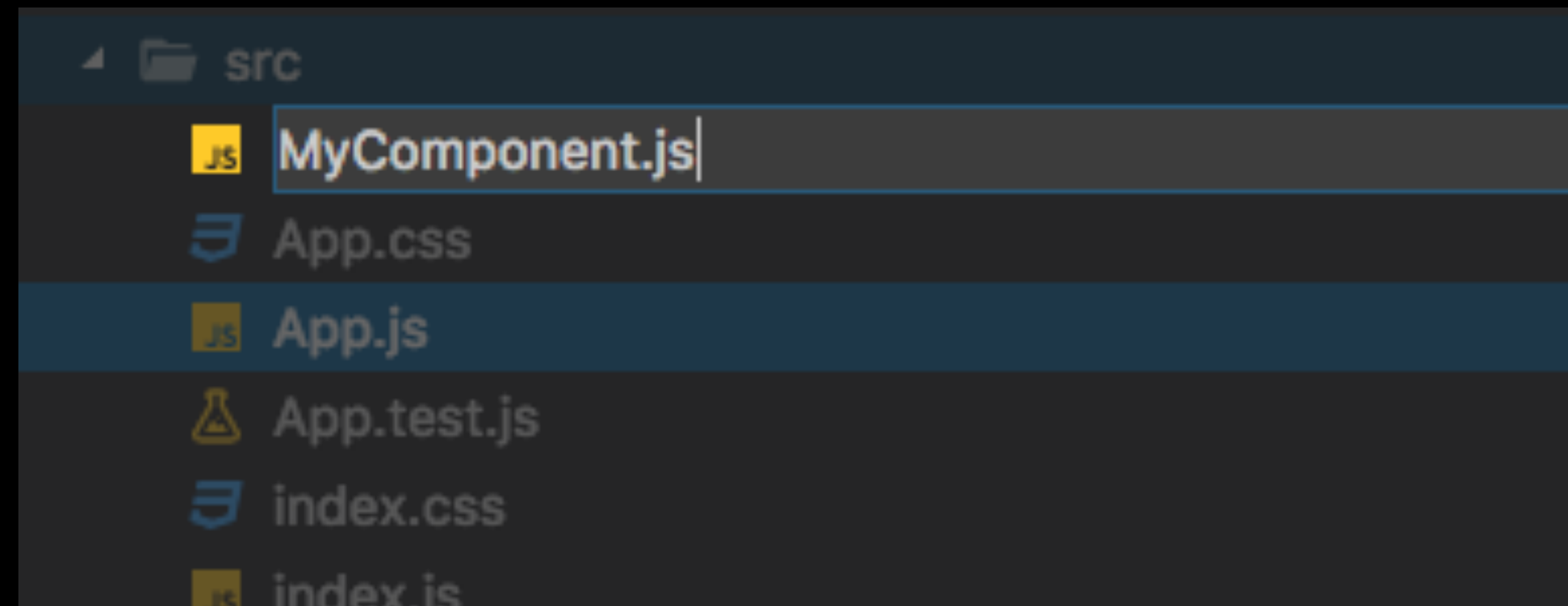
👍

🚫

106



React Code Snippets



MyComponent.js 생성

```
MyComponent.js
1 rcc
  rcc ReactSnippets
    Creates a React component class with ES...
  rccp
  RTCCConfiguration
```

```
MyComponent.js
1 import React, { Component } from 'react';
2
3 class componentName extends Component {
4   render() {
5     return (
6       <div>
7
8       </div>
9     );
10  }
11 }
12
13 export default componentName;
```

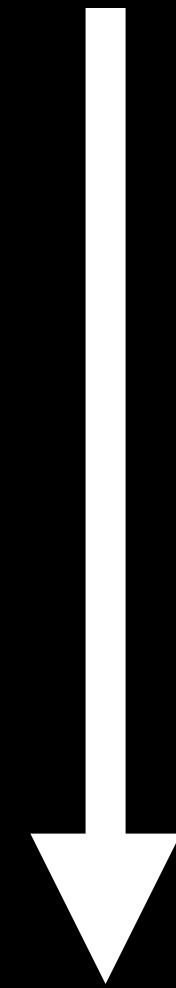
rcc 라고 적으면 자동완성이 됩니다

```
import React, { Component } from 'react';
class MyComponent extends Component {
  render() {
    return (
      <div>
        MyComponent
      </div>
    );
  }
}
export default MyComponent;
```

```
import React, {Component} from 'react';
import MyComponent from './MyComponent';
import './App.css';
class App extends Component {
  render() {
    return (
      <div>
        <MyComponent />
      </div>
    );
  }
}
export default App;
```

props

위에서 아래로



부모 - - - - 자식

부모 - - 데이터 - - 자식

부모 - - props - - 자식

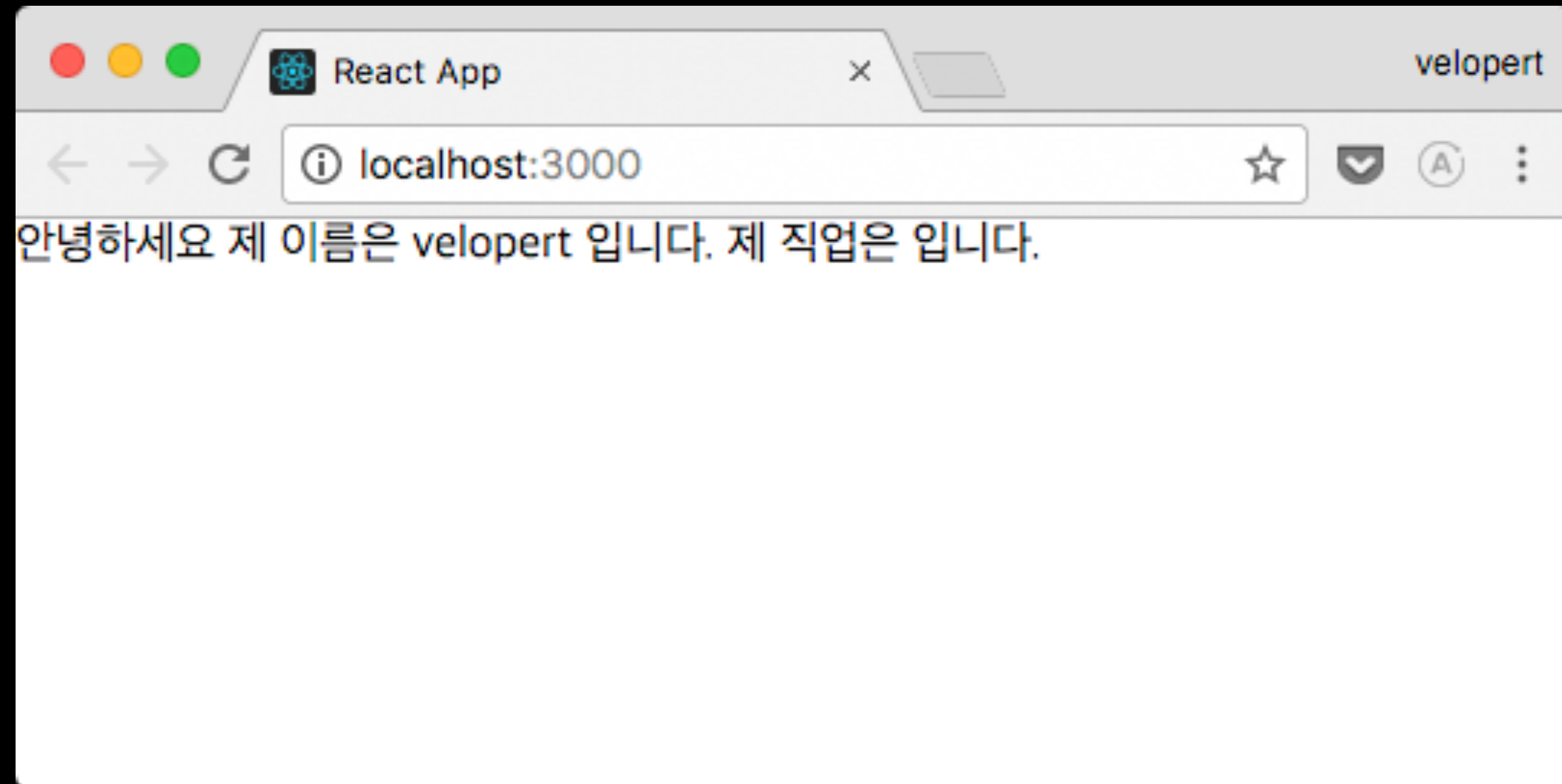
```
<MyComponent name="velopert" />
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  render() {
    return (
      <div>
        안녕하세요 제 이름은 {this.props.name} 입니다.
      </div>
    );
  }
}
export default MyComponent;
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  render() {
    const { name, job } = this.props;
    return (
      <div>
        안녕하세요 제 이름은 {name} 입니다.
        제 직업은 {job} 입니다.
      </div>
    );
  }
}
export default MyComponent;
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  render() {
    const { name, job } = this.props;
    return (
      <div>
        안녕하세요 제 이름은 {name} 입니다.
        제 직업은 {job} 입니다.
      </div>
    );
  }
}
export default MyComponent;
```

```
const name = this.props.name;
const job = this.props.job;
```

기본값 설정하기: **defaultProps**

```
import React, { Component } from 'react';
class MyComponent extends Component {
  static defaultProps = {
    job: '개발자'
  }
  render() {
    const { name, job } = this.props;
    return (
      <div>
        안녕하세요 제 이름은 {name} 입니다.
        제 직업은 {job} 입니다.
      </div>
    );
  }
}
export default MyComponent;
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  render() {
    const { name, job } = this.props;
    return (
      <div>
        안녕하세요 제 이름은 {name} 입니다.
        제 직업은 {job} 입니다.
      </div>
    );
  }
}

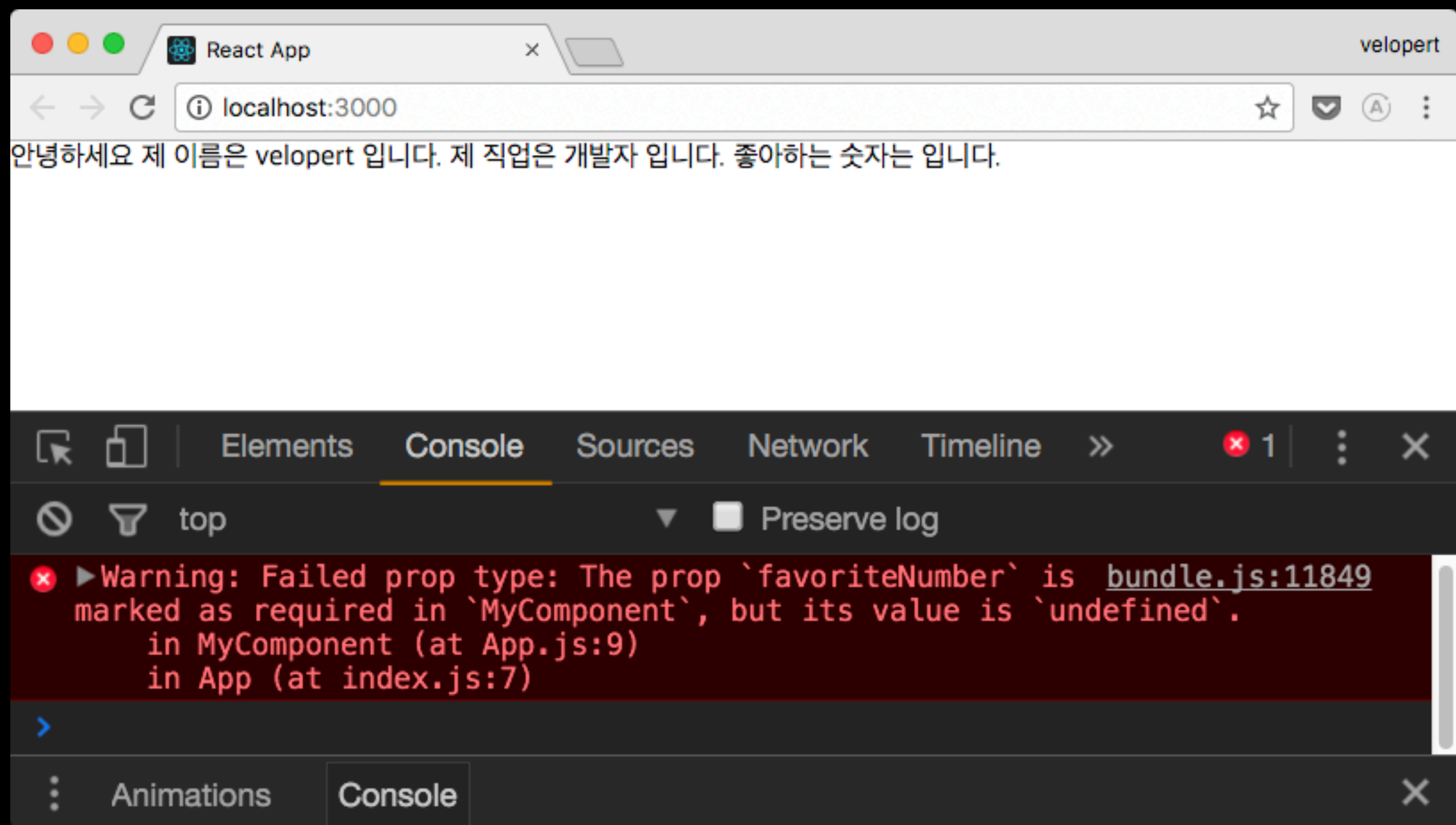
MyComponent.defaultProps = {
  job: '개발자'
}

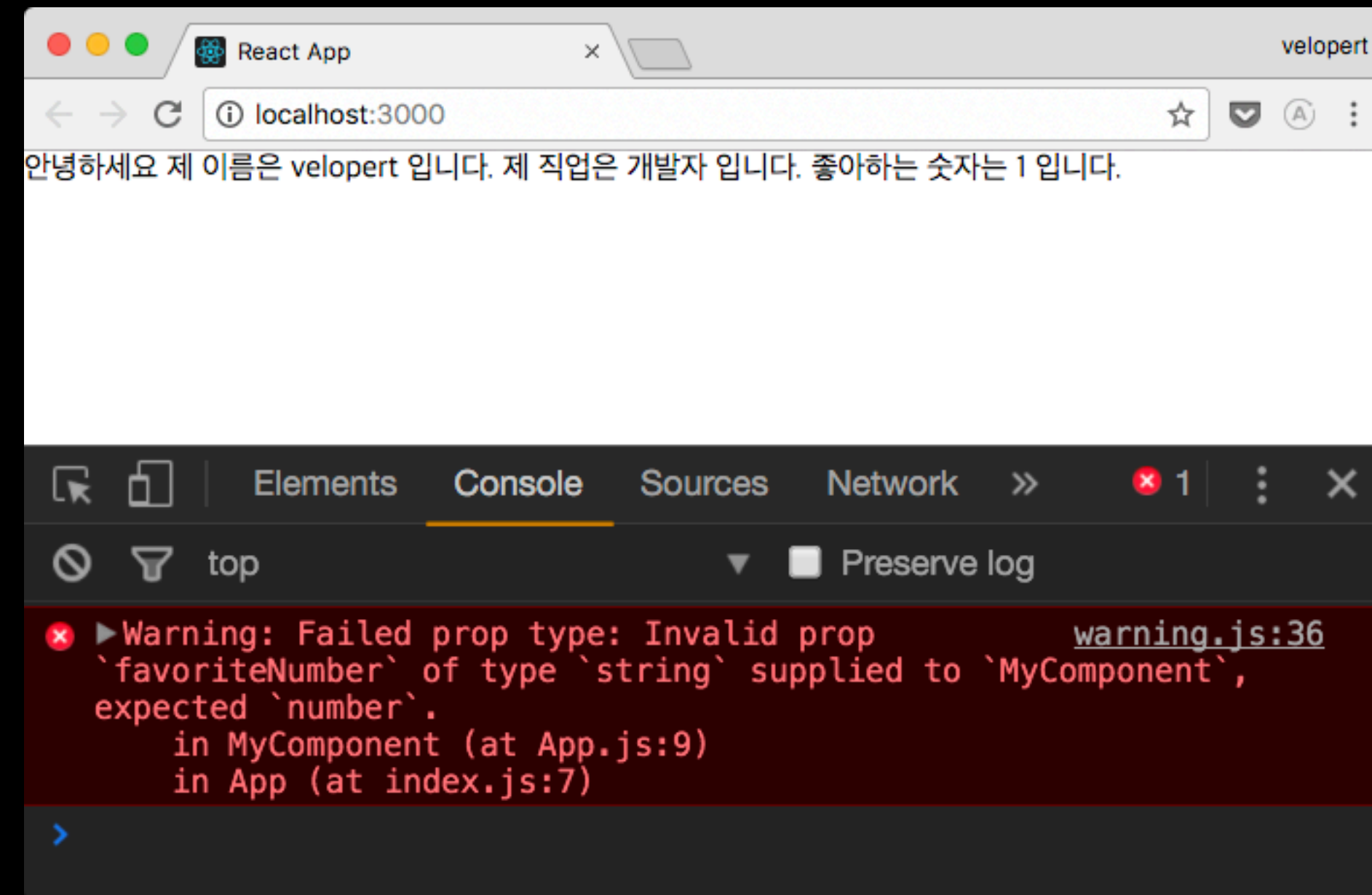
export default MyComponent;
```

props 검증하기: **propTypes**

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';

class MyComponent extends Component {
  static defaultProps = {
    job: '개발자'
  }
  static propTypes = {
    name: PropTypes.string,
    job: PropTypes.string,
    favoriteNumber: PropTypes.number.isRequired
  }
  render() {
    const { name, job, favoriteNumber } = this.props;
    return (
      <div>
        안녕하세요 제 이름은 {name} 입니다.
        제 직업은 {job} 입니다.
        좋아하는 숫자는 {favoriteNumber} 입니다.
      </div>
    );
  }
}
export default MyComponent;
```





```
<MyComponent name="velopert" favoriteNumber="1" />  
<MyComponent name="velopert" favoriteNumber={1} />
```


PropTypes 종류:

- array: 배열
- bool: true or false
- func: 함수
- number: 숫자
- object: 객체
- string: 문자열
- element: div 태그, 리액트 컴포넌트 등
- node: 렌더링 될 수 있는 모든것
- oneOf([배열]): 배열 내부의 값들
- any: 모든 것
- 더보기: <https://facebook.github.io/react/docs/typechecking-with-proptypes.html>

모든 컴포넌트마다 PropTypes 를 지정해주는것이 좋은 습관

어떤 컴포넌트가 어떤 props 를 요구하는지 명확하게 알려주어 유지보수에 도움이 됩니다
협업할때는 더욱더욱 필수입니다.

변하는 값: **state**

컴포넌트 고유의 것.

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={() => {this.setState({number: number + 1})}}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```

```

import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={() => {this.setState({number: number + 1})}}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;

```

```

constructor(props) {
  super(props);
  this.state = {
    number: 0
  }
}

```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={() => {this.setState({number: number + 1})}}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={() => {this.setState({number: number + 1})}}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```



```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={() => {this.setState({number: number + 1})}}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```

```
function() {
  this.setState({
    number: number + 1
  });
}
```

⇒ <https://github.com/tonsky/FiraCode>

```

_createClass(MyComponent, [{
  key: 'render',
  value: function render() {
    var _this2 = this;

    var number = this.state.number;

    return _react2.default.createElement(
      'div',
      null,
      _react2.default.createElement(
        'h1',
        null,
        number
      ),
      _react2.default.createElement(
        'button',
        { onClick: function onClick() {
          _this2.setState({ number: number + 1 });
        } },
        '\uB098\uB97C \uB20C\uB7EC\uBCF4uC138uC694.'
      )
    );
  }
}]);

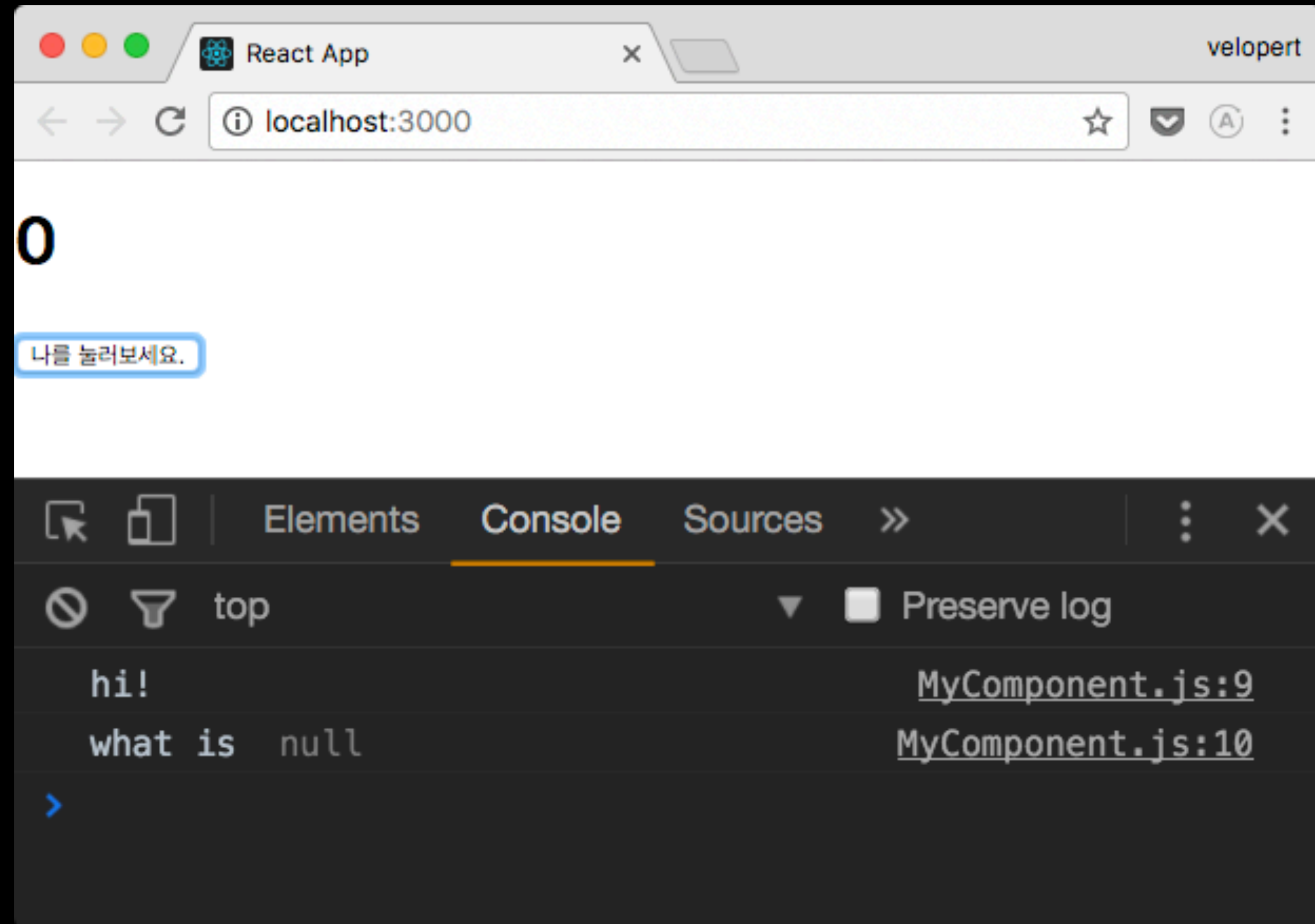
```

state 를 업데이트할 땐 언제나
this.setState({...})

함수가 실행되면, 리렌더링을 트리거합니다

임의 메소드 만들기

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  handleClick() {
    console.log('hi!');
    console.log('what is ', this);
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={this.handleClick}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```



```
var obj = {  
  prop: 'Hello',  
  sayHello: function() {  
    console.log( this.prop );  
  }  
};  
obj.sayHello(); // Logs "Hello"
```



```
var reference = obj.sayHello;  
reference(); // logs "undefined"
```

```
var obj = {  
  prop: 'Hello',  
  sayHello: function() {  
    console.log(this.prop);  
  }  
};  
var newFunction = obj.sayHello.bind(obj);  
newFunction(); // logs "Hello"
```

```
<button onClick={this.handleClick.bind(this)}>  
  나를 눌러보세요.  
</button>
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    console.log('hi!');
    console.log('what is ', this);
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={this.handleClick}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```

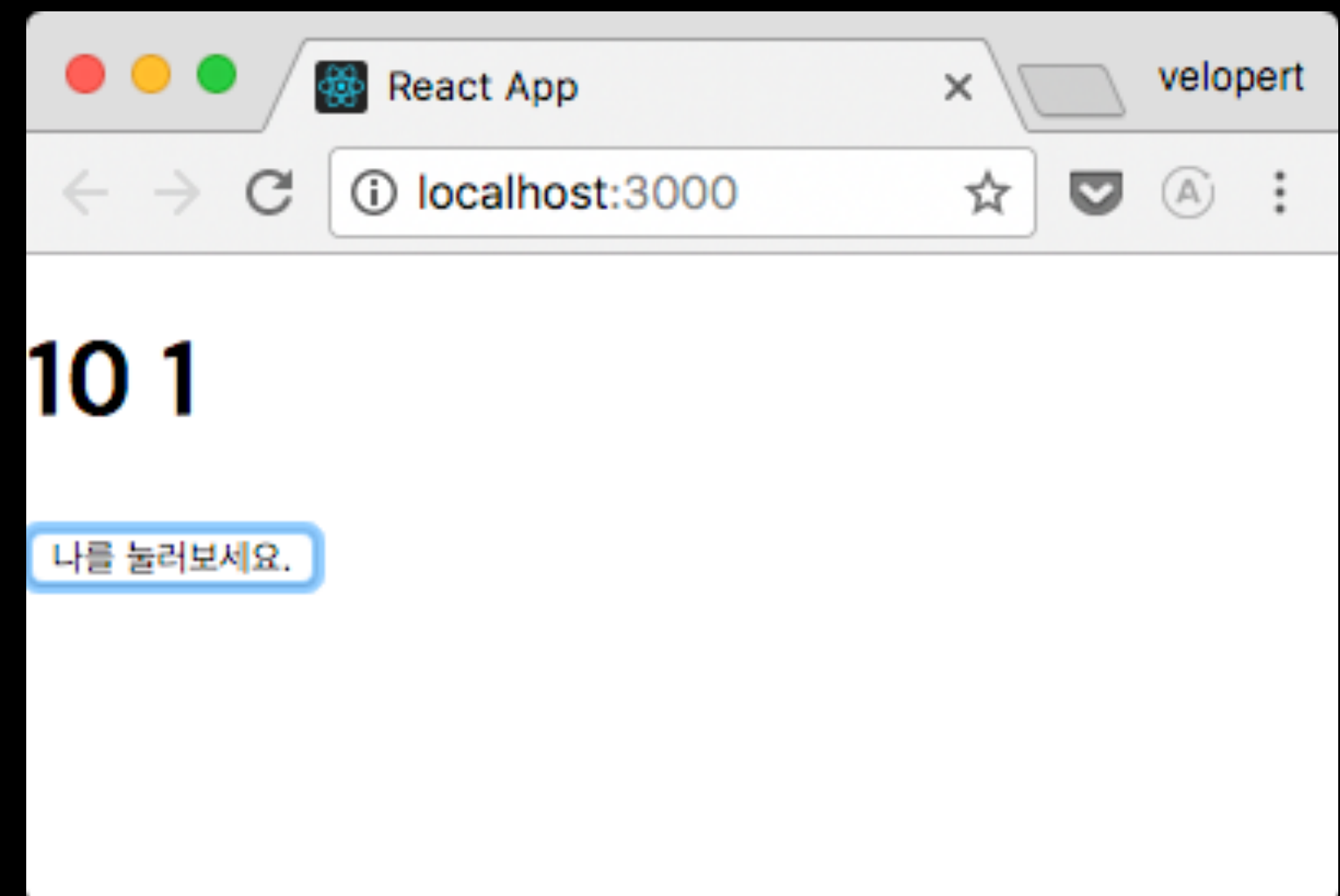
```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0
  }
  handleClick = () => {
    this.setState({
      number: this.state.number + 1
    });
  }
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>{number}</h1>
        <button onClick={this.handleClick}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;
```

setState({...})
안에 있는 값들만 처리합니다.

```

import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    number: 0,
    theNumber: 1
  }
  handleClick = () => {
    this.setState({
      number: this.state.number + 1
    });
  }
  render() {
    const {number, theNumber} = this.state;
    return (
      <div>
        <h1>{number} {theNumber}</h1>
        <button onClick={this.handleClick}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;

```

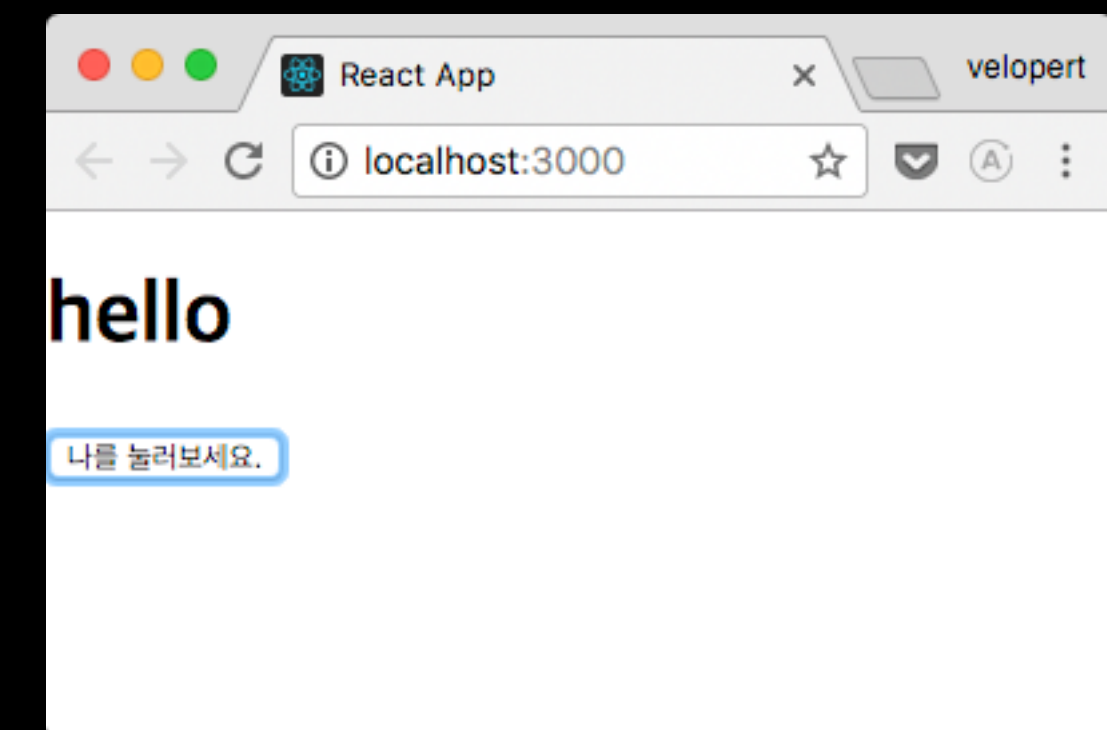
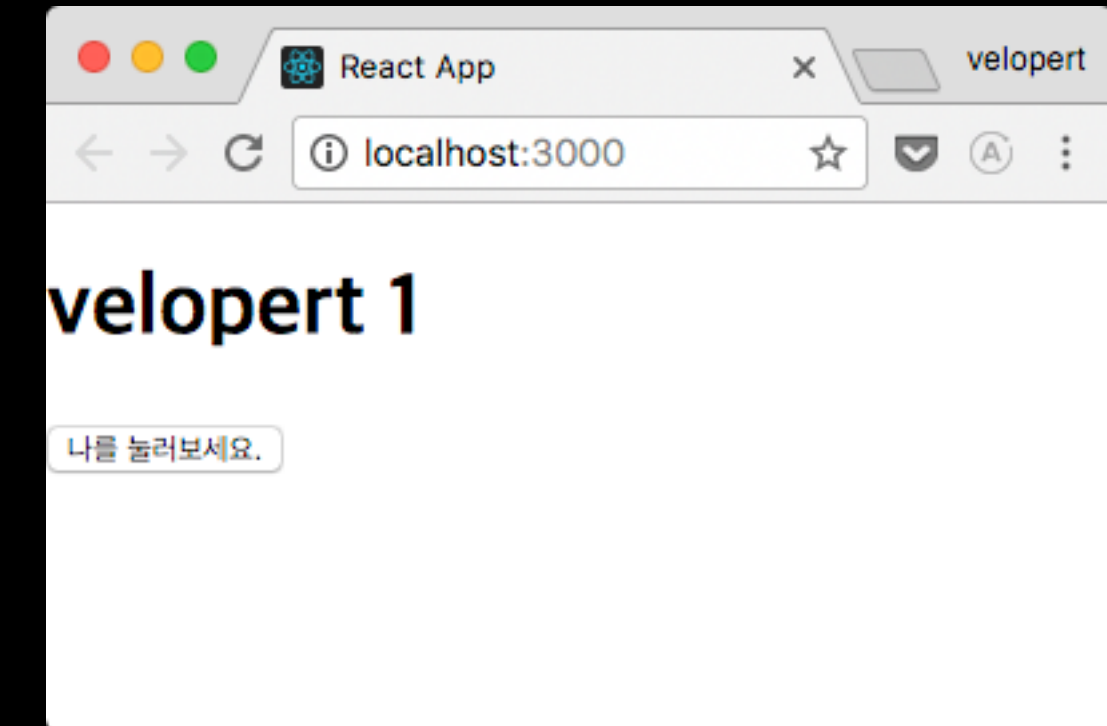


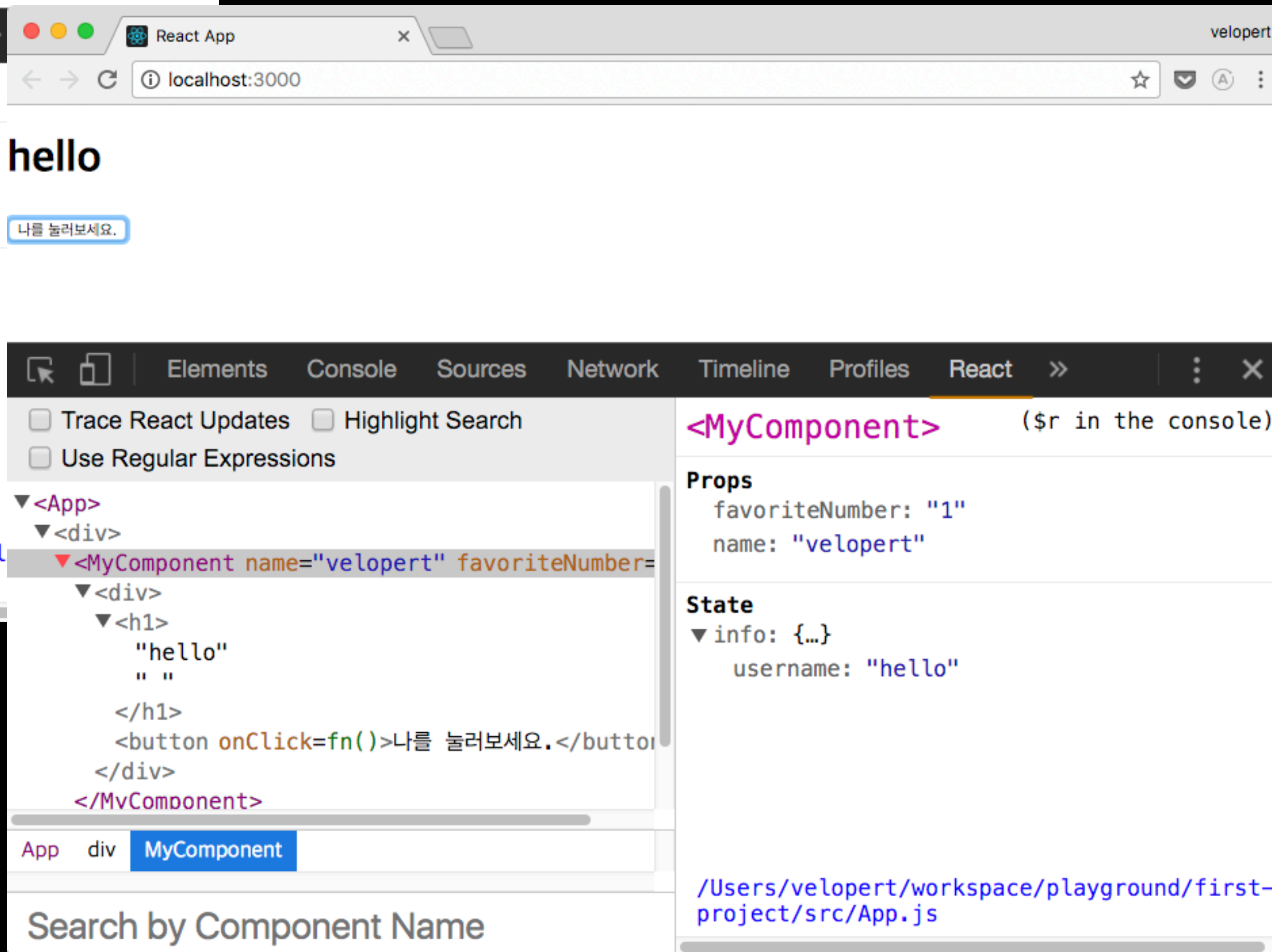
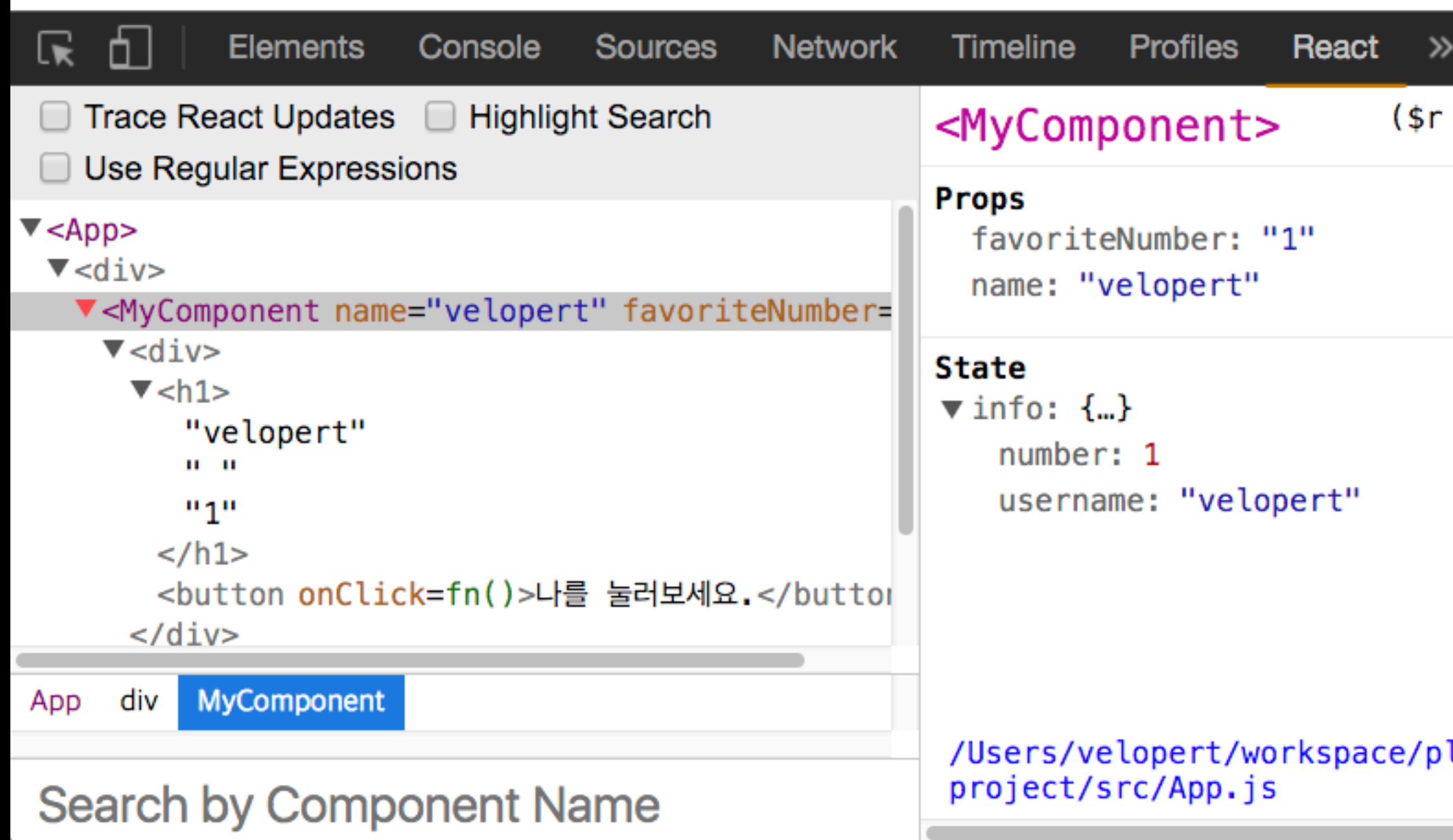
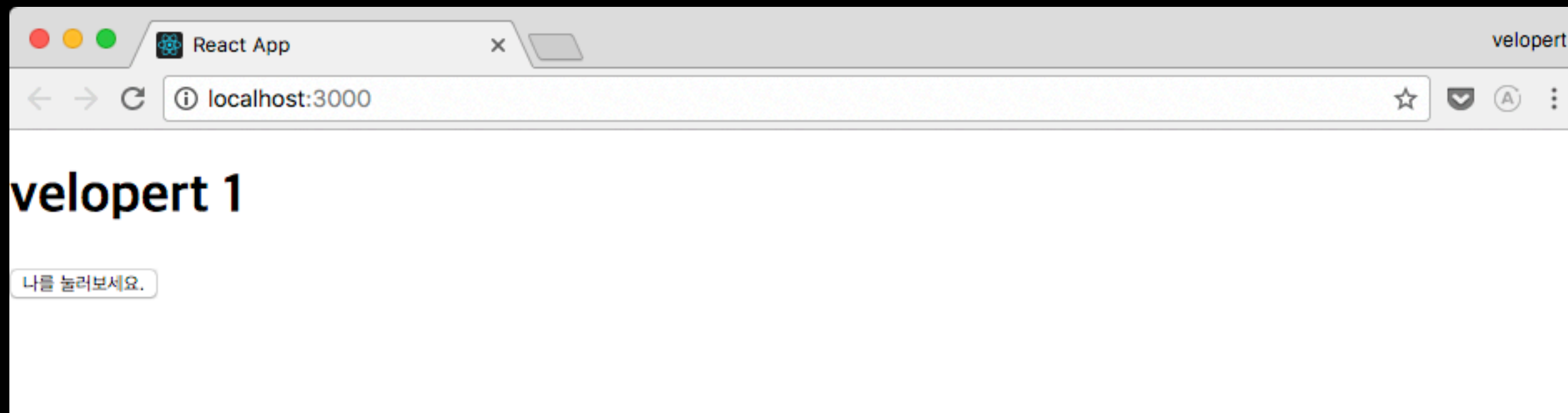
nested 된 값은 단일 처리 불가


```

import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    info: {
      username: 'velopert',
      number: 1
    }
  }
  handleClick = () => {
    this.setState({
      info: {
        username: 'hello'
      }
    });
  }
  render() {
    const {info} = this.state;
    return (
      <div>
        <h1>{info.username} {info.number}</h1>
        <button onClick={this.handleClick}>
          나를 눌러보세요.
        </button>
      </div>
    );
  }
}
export default MyComponent;

```





```
handleClick = () => {  
  this.setState({  
    info: {  
      ...this.state.info,  
      username: 'hello'  
    }  
  });  
}
```

... : ES6 spread 문법, 객체를 풀어서 그 자리에 대입한다

컴포넌트를 만드는 또 다른 방법

함수형 컴포넌트

src/Container.js

```
import React from 'react';
const Container = (props) => {
  return (
    <div>
      <h1>{props.title}</h1>
      <div>
        {props.children}
      </div>
    </div>
  );
};
export default Container;
```

src/App.js

```
import React, {Component} from 'react';
import MyComponent from './MyComponent';
import Container from './Container';
class App extends Component {
  render() {
    return (
      <Container title="Welcome">
        <MyComponent />
      </Container>
    );
  }
}
export default App;
```

src/Container.js

```
import React from 'react';
const Container = ({children, title}) => {
  return (
    <div>
      <h1>{title}</h1>
      <div>
        {children}
      </div>
    </div>
  );
};
export default Container;
```

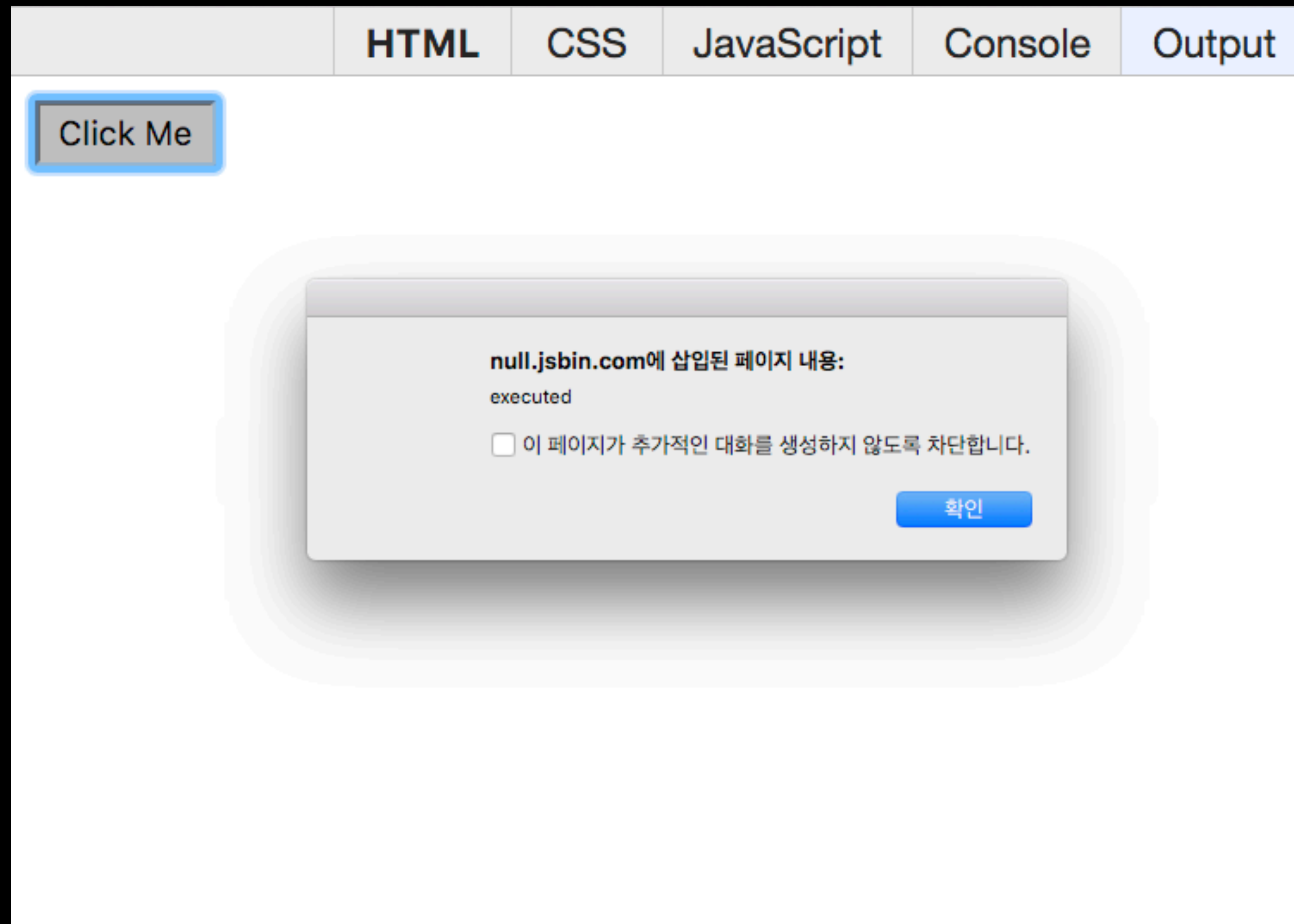
숨.

이벤트 핸들링

이벤트:

유저와 DOM 의 상호작용

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Events in HTML</title>
</head>
<body>
  <button onclick="alert('executed')">
    Click Me
  </button>
</body>
</html>
```



```
<button onClick={()=>{this.setState({number: number + 1})}}>  
  나를 눌러보세요.  
</button>
```

주의사항

1) 이벤트 이름은 camelCase

onclick → onClick

onkeyup → onKeyUp

...

주의사항

2) 함수형태의 객체를 전달

```
onClick={handleClick()}  
onClick={handleClick}
```

주의사항

3) DOM 요소에만 이벤트 설정 가능


```
<MyComponent onClick={doSomething} />
```

```
import React from 'react';  
const MyComponent = ({onClick}) => {  
  return (  
    <button onClick={onClick}>  
      클릭  
    </button>  
  );  
};  
export default MyComponent;
```

이벤트 종류

- Clipboard
- Composition
- Keyboard
- Focus
- Form
- Mouse
- Selection
- Touch
- UI
- Wheel
- Media
- Image
- Animation
- Transition

```
import React, { Component } from 'react';
class MyComponent extends Component {
  handleChange = (e) => {
    e.persist();
    console.log(e);
  }
  handleClick = (e) => {
    e.persist();
    console.log(e);
  }
  render() {
    const { handleChange, handleClick } = this;
    return (
      <div>
        <input onChange={handleChange} type="text" name="last-name" placeholder="성" />
        <input onChange={handleChange} type="text" name="first-name" placeholder="이름" />
        <button onClick={handleClick}>등록</button>
      </div>
    );
  }
}
export default MyComponent;
```

```
Proxy {dispatchConfig: Object, _targetInst:
▼ ReactDOMComponent, nativeEvent: Event, type:
  "change", target: input...} ⓘ
  ► [[Handler]]: Object
  ▼ [[Target]]: SyntheticEvent
    bubbles: true
    cancelable: false
    currentTarget: null
    defaultPrevented: false
    ► dispatchConfig: Object
    eventPhase: 3
    ► isDefaultPrevented: function ()
    ► isPersistent: function ()
    ► isPropagationStopped: function ()
    isTrusted: true
    ► nativeEvent: Event
    ▼ target: input
      accept: ""
      accessKey: ""
      align: ""
      alt: ""
      assignedSlot: null
      ► attributes: NamedNodeMap
      autocapitalize: "sentences"
      autocomplete: ""
      autofocus: false
```

```
name: "lastname"
value: "hello"
```

```
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    lastname: '',
    firstname: ''
  }
  handleChange = (e) => {
    this.setState({
      [e.target.name]: e.target.value
    });
  }
  ( ... )
```

```
render() {
  const { handleChange, handleClick } = this;
  const { lastname, firstname } = this.state;
  return (
    <div>
      <input
        onChange={handleChange}
        value={lastname}
        type="text"
        name="lastname"
        placeholder="성" />
      <input
        onChange={handleChange}
        value={firstname}
        type="text"
        name="firstname"
        placeholder="이름" />
      <button onClick={handleClick}>등록</button>
      <h2>{lastname} {firstname}</h2>
    </div>
  );
}
```

```

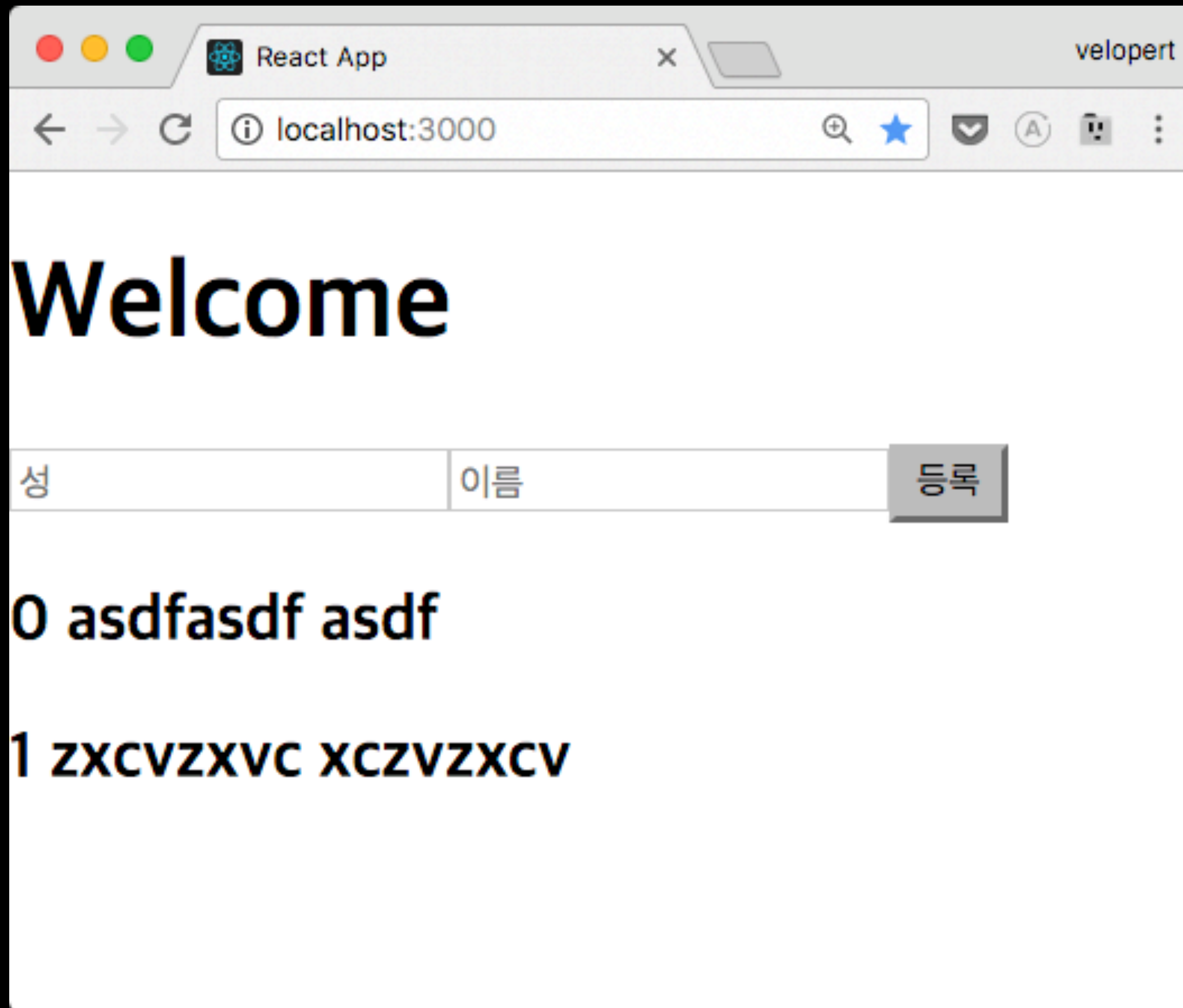
import React, { Component } from 'react';
class MyComponent extends Component {
  state = {
    lastname: '',
    firstname: '',
    names: []
  }

  ( ... )

  handleClick = (e) => {
    const { lastname, firstname, names } = this.state;
    this.setState({
      lastname: '',
      firstname: '',
      names: [ ...names, `${lastname} ${firstname}` ]
    });
  }

  render() {
    const { handleChange, handleClick } = this;
    const { lastname, firstname, names } = this.state;
    return (
      <div>
        ( ... )
        <button onClick={handleClick}>등록</button>
        {names.map((name, index) => <h3 key={name}>{index} {name}</h3>)}
      </div>
    );
  }
}
export default MyComponent;

```

// 메소드 추가

```
handleKeyPress = (e) => {  
  console.log(e.key);  
}
```

// 렌더함수 변경

```
const { handleChange, handleClick, handleKeyPress } = this;  
( ... )
```

```
<input
```

```
  onChange={handleChange}
```

```
  value={firstname}
```

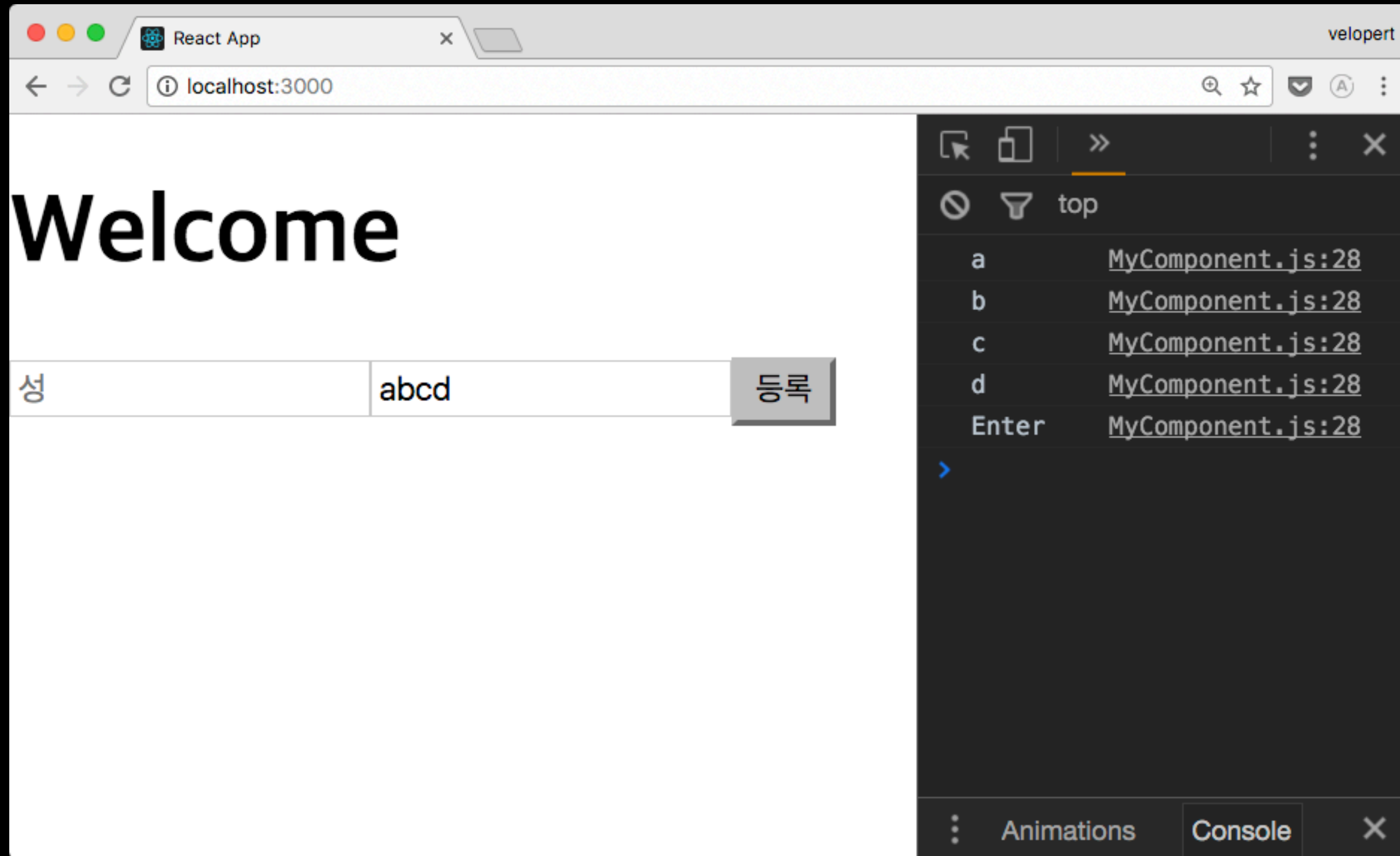
```
  type="text"
```

```
  name="firstname"
```

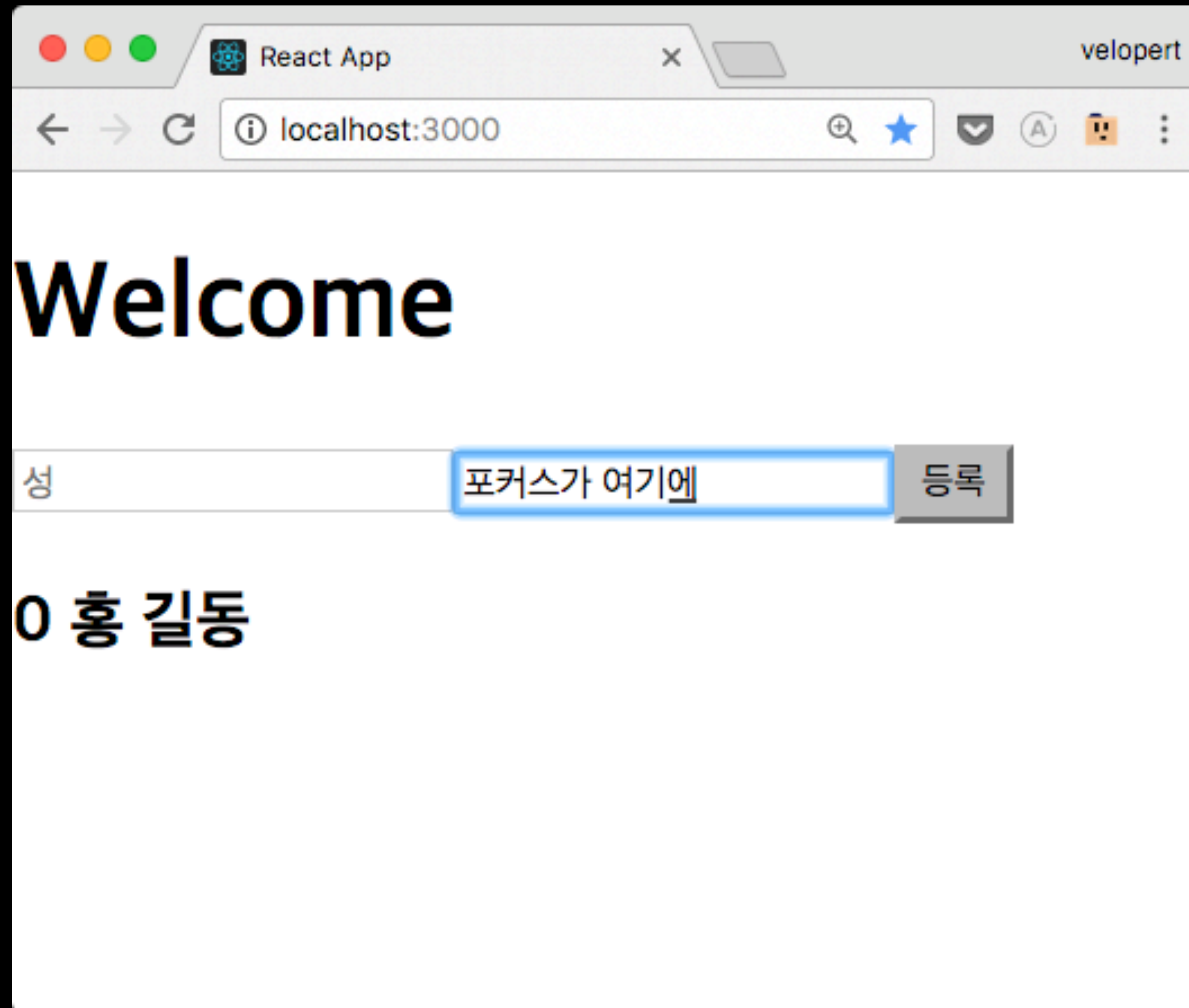
```
  placeholder="이름"
```

```
  onKeyPress={handleKeyPress}
```

```
>
```



```
handleKeyPress = (e) => {  
    if(e.key === 'Enter') this.handleClick();  
}
```



ref: DOM에 직접 접근

```
<div ref={ref=>{this.element = ref}}></div>
```

```
<input
  onChange={handleChange}
  value={lastname}
  type="text"
  name="lastname"
  placeholder="성"
  ref={ref⇒this.lastnameInput=ref}
/>
```

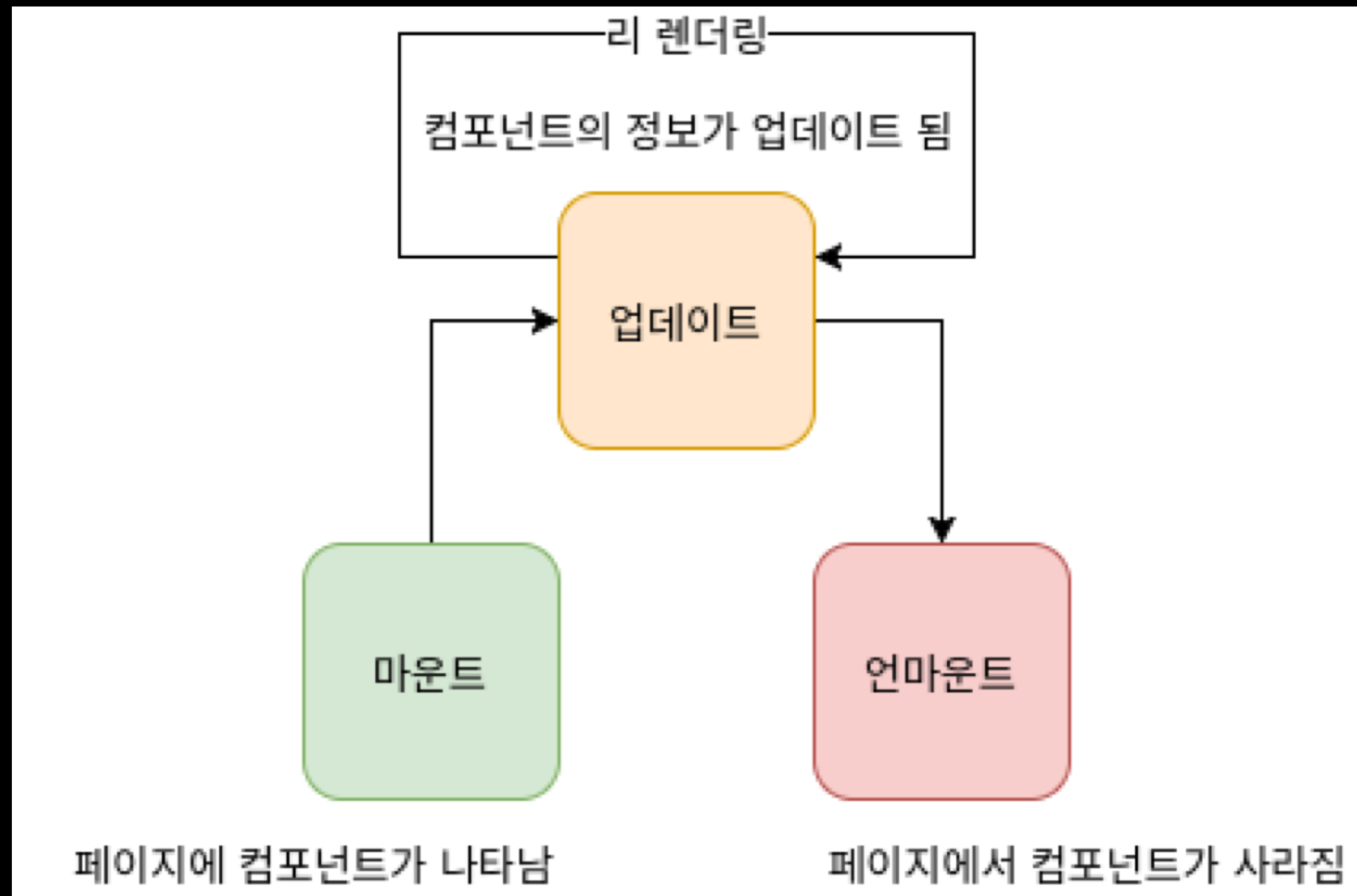


```
handleClick = (e) => {  
  const { lastname, firstname, names } = this.state;  
  this.setState({  
    lastname: '',  
    firstname: '',  
    names: [ ...names, `${lastname} ${firstname}` ]  
  });  
  this.lastnameInput.focus();  
}
```

전체코드

<https://gist.github.com/velopert/e41829893446f110adb3a015dd67d83e>

컴포넌트의 LifeCycle 메소드



마운트

constructor()

component**WillMount**()

render()

component**DidMount**()

업데이트

component**Will**ReceiveProps()
shouldComponentUpdate()
component**Will**Update()
render()
component**Did**Update()

언마운트

componentWillUnmount()

render() { ... }

컴포넌트의 모양새를 정의해줌

컴포넌트 생성

constructor → componentWillMount → **render** → componentDidMount

컴포넌트 업데이트

componentWillReceiveProps → shouldComponentUpdate
→ componentWillUpdate → **render** → componentDidUpdate

constructor(props) { ... }

컴포넌트의 생성자 메소드

초기 멤버변수 및 state 정의 - class property 로도 할 수 있음
내부에서 super(props) 실행해 주어야함

컴포넌트 생성

constructor → componentWillMount → render → componentDidMount

`componentWillMount()` { ... }

DOM에 나타나기 전에 실행되는 메소드
한번만 실행되며, `this.props` / `this.state` 사용 가능
`setState` 사용 가능, DOM 에 접근 불가능
서버사이드에서도 실행됨

컴포넌트 생성

`constructor` → **`componentWillMount`** → `render` → `componentDidMount`

`componentDidMount()` { ... }

첫 렌더링 후 실행되는 메소드

다른 자바스크립트 라이브러리 / 프레임워크 함수 호출

이벤트 등록 / `setTimeout` 및 네트워크 요청 / 비동기작업은 여기서 합니다

DOM 에 접근 가능

컴포넌트 생성

`constructor` → `componentWillMount` → `render` → **`componentDidMount`**

componentWillReceiveProps(nextProps) {...}

부모로부터 새 props를 전달 받게될때 실행됩니다

props 값에 따라 state 에 변화를 주어야 할때 여기서 작업을 하면 됩니다

setState 메소드를 실행 가능하며, 새 props 는 nextProps 키워드로 접근가능합니다

컴포넌트 업데이트

componentWillReceiveProps → shouldComponentUpdate
→ componentWillUpdate → render → componentDidUpdate

`shouldComponentUpdate(nextProps, nextState) {...}`

props 나 state 가 변경되면 이 메소드가 실행됩니다

true / false 값을 반환해야 합니다.

false 를 반환하면 업데이트가 중지됩니다.

컴포넌트를 최적화할 때 중요한 역할을 합니다.

기본값으로 true를 반환합니다

컴포넌트 업데이트

`componentWillReceiveProps` → **`shouldComponentUpdate`**
→ `componentWillUpdate` → `render` → `componentDidUpdate`

`componentWillUpdate(nextProps, nextState) {...}`

shouldComponentUpdate 가 true 를 반환했을때 실행됩니다
업데이트를 실행하기 전에 실행됩니다
DOM 조작 및 setState 를 하면 정상적으로 작동하지 않습니다

컴포넌트 업데이트

componentWillReceiveProps → shouldComponentUpdate
→ **componentWillUpdate** → render → componentDidUpdate

`componentDidUpdate(prevProps, prevState) {...}`

리렌더링이 마친 후 실행됩니다
DOM 관련 처리를 해도 됩니다
prevProps 혹은 prevState 를 사용하여
이전에 지니고있던 데이터에 접근할수있습니다

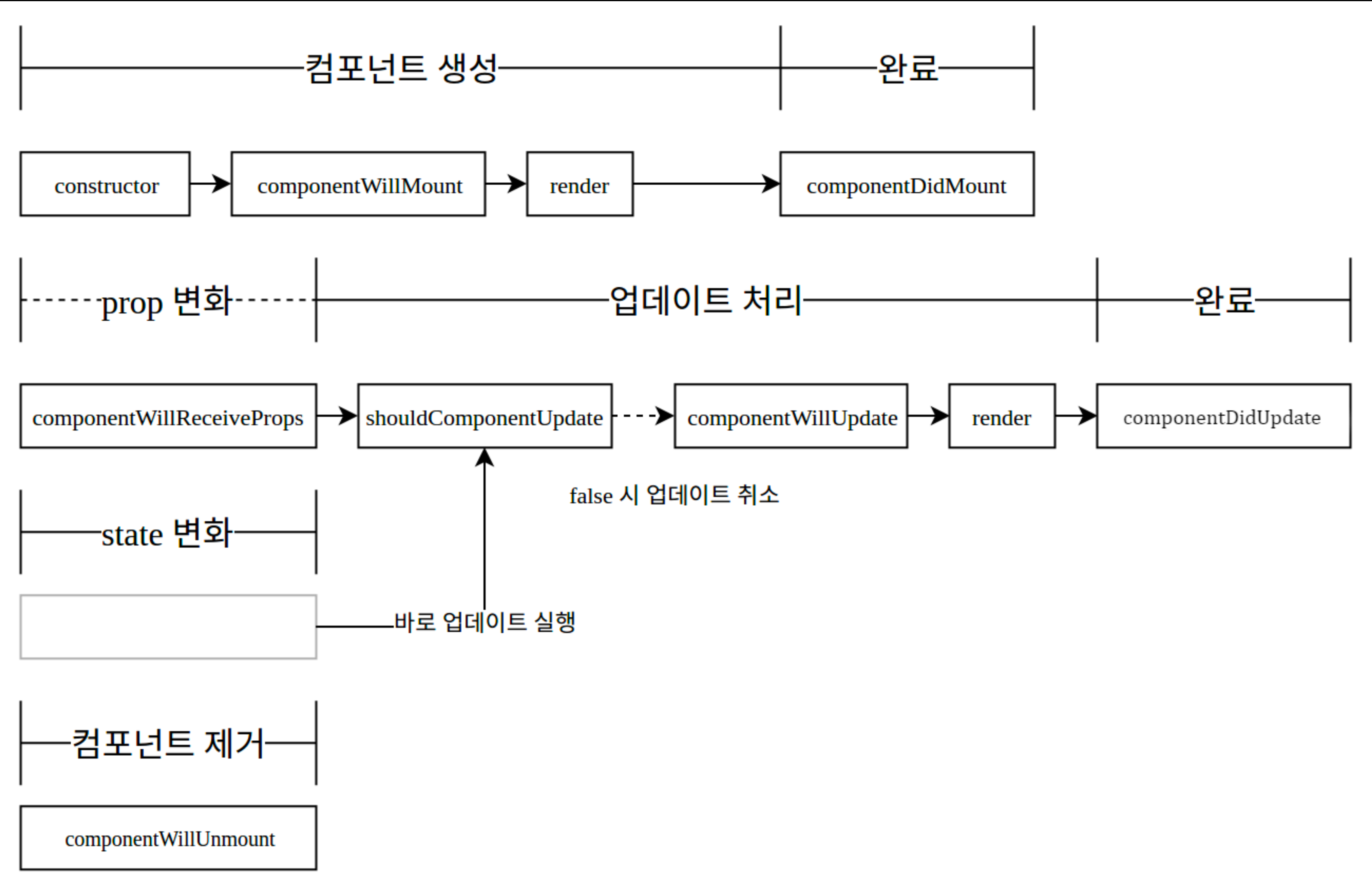
컴포넌트 업데이트

`componentWillReceiveProps` → `shouldComponentUpdate`
→ `componentWillUpdate` → `render` → **`componentDidUpdate`**

componentWillUnmount

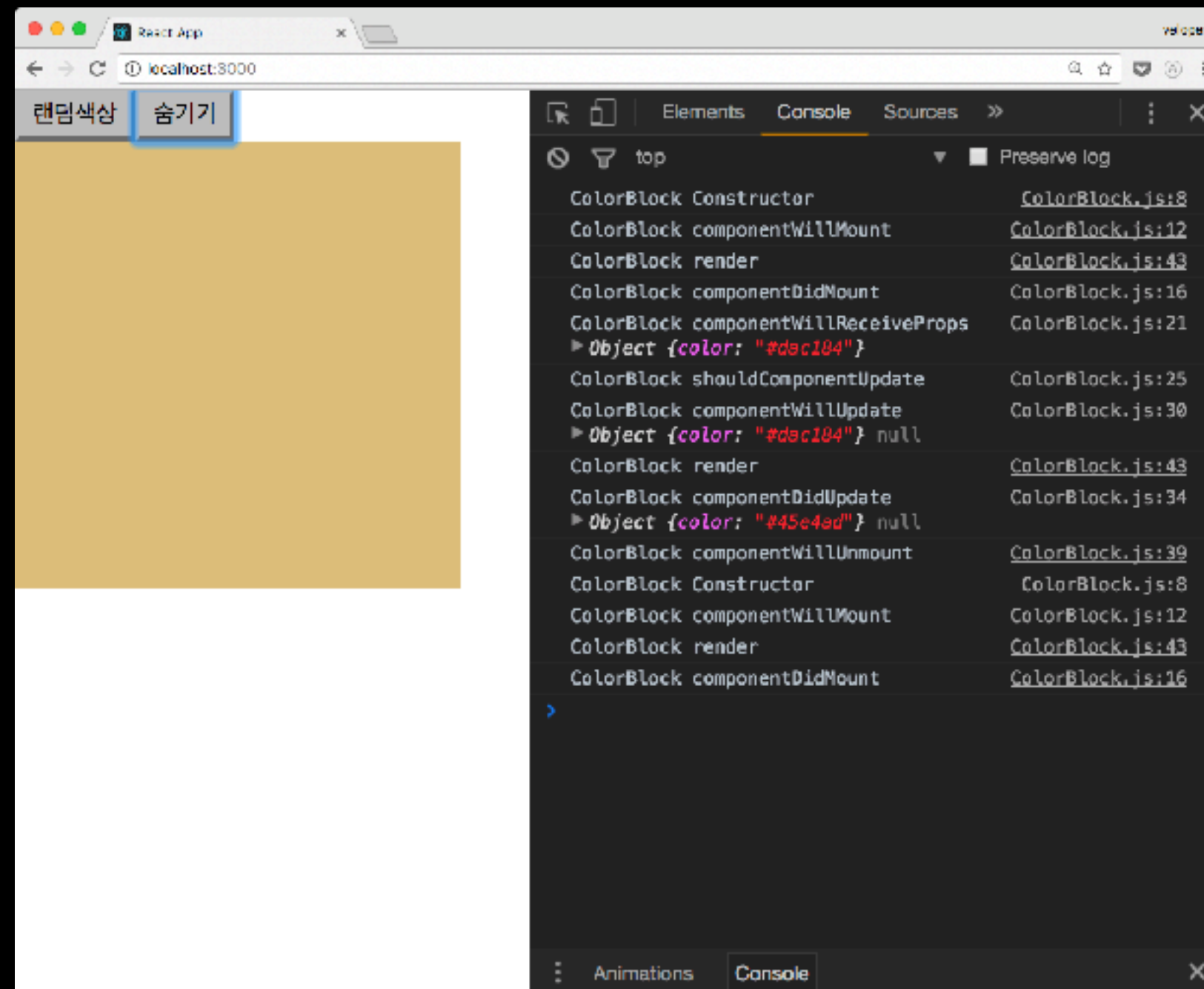
컴포넌트가 DOM에서 제거 될 때 실행됩니다.
여기서 타이머 / 이벤트 등을 제거합니다

컴포넌트 언마운트
componentWillUnmount



LifeCycle API 실습

<https://gist.github.com/velopert/3126fa38067183b7e03ce7f64ba056a6>



숨.