Seoul
Software
ACademy

with

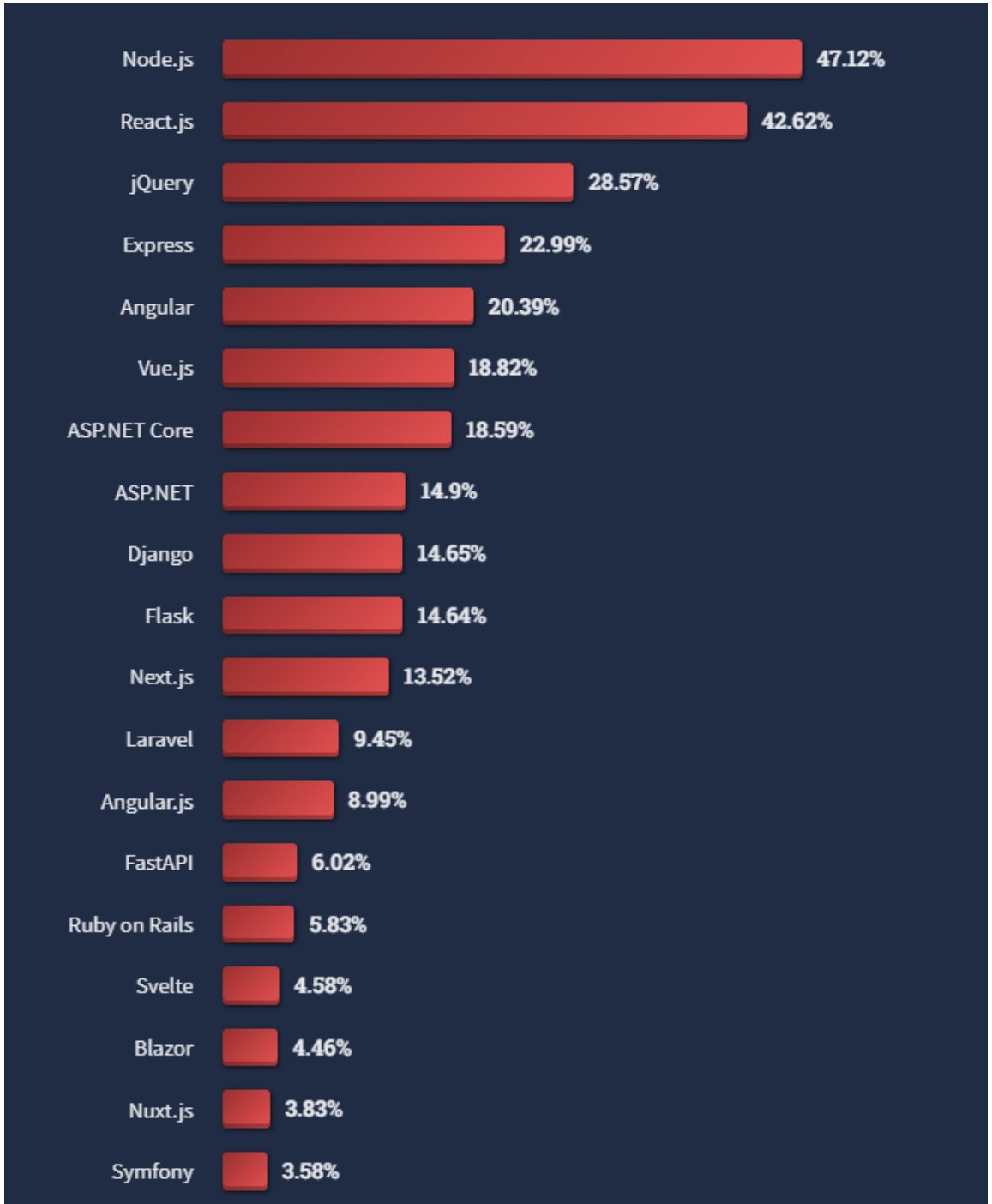

PREVIOUSLY
ON.

잘 부탁드립니다-

React JS

Why?

https://survey.stackoverflow.co/2022/#technology



site: stackoverflow.com react

전체 뉴스 동영상 이미지 쇼핑 더보기 도구

검색결과 약 31,200,000개 (0.37초)

site: stackoverflow.com angular

전체 뉴스 동영상 이미지 쇼핑 더보기 도구

검색결과 약 20,000,000개 (0.37초)

site: stackoverflow.com vue

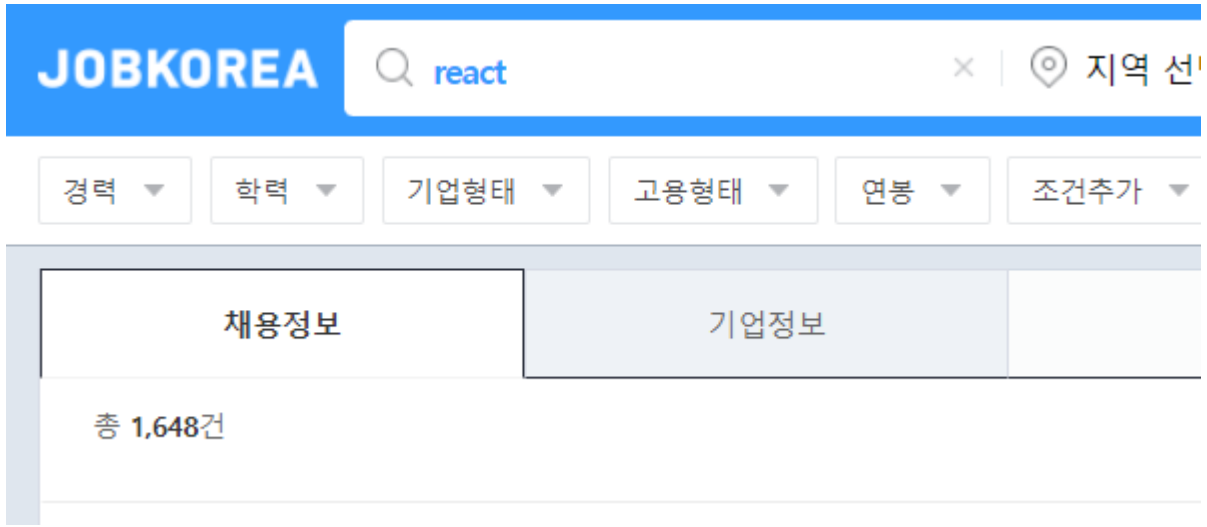
전체 뉴스 이미지 동영상 쇼핑 더보기 도구

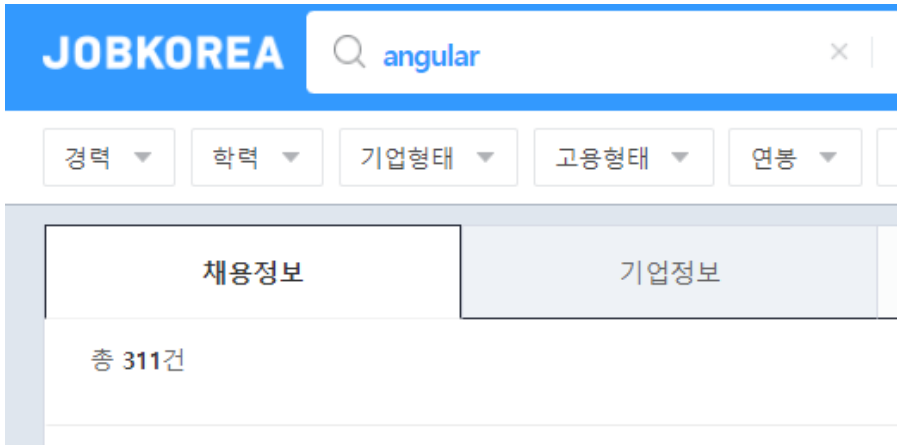
검색결과 약 1,390,000개 (0.39초)

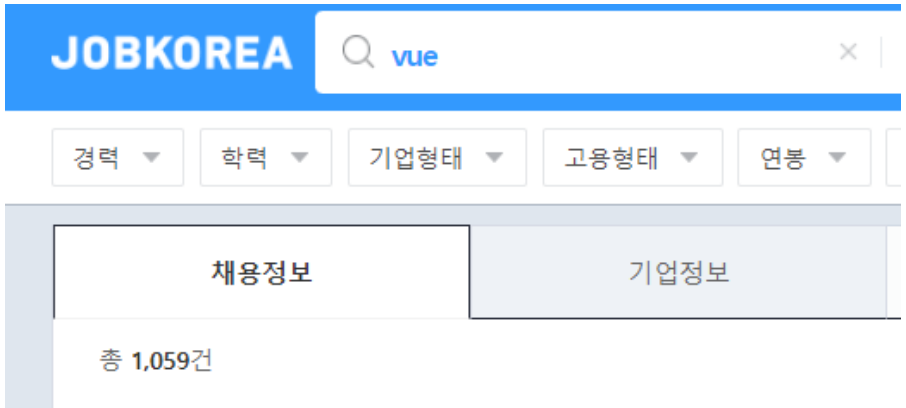
site: stackoverflow.com laravel

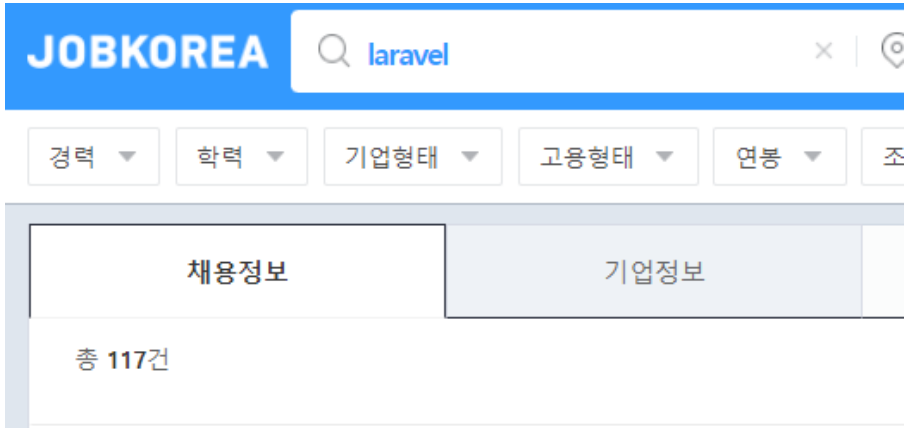
전체 동영상 뉴스 이미지 지도 더보기 도구

검색결과 약 8,220,000개 (0.39초)

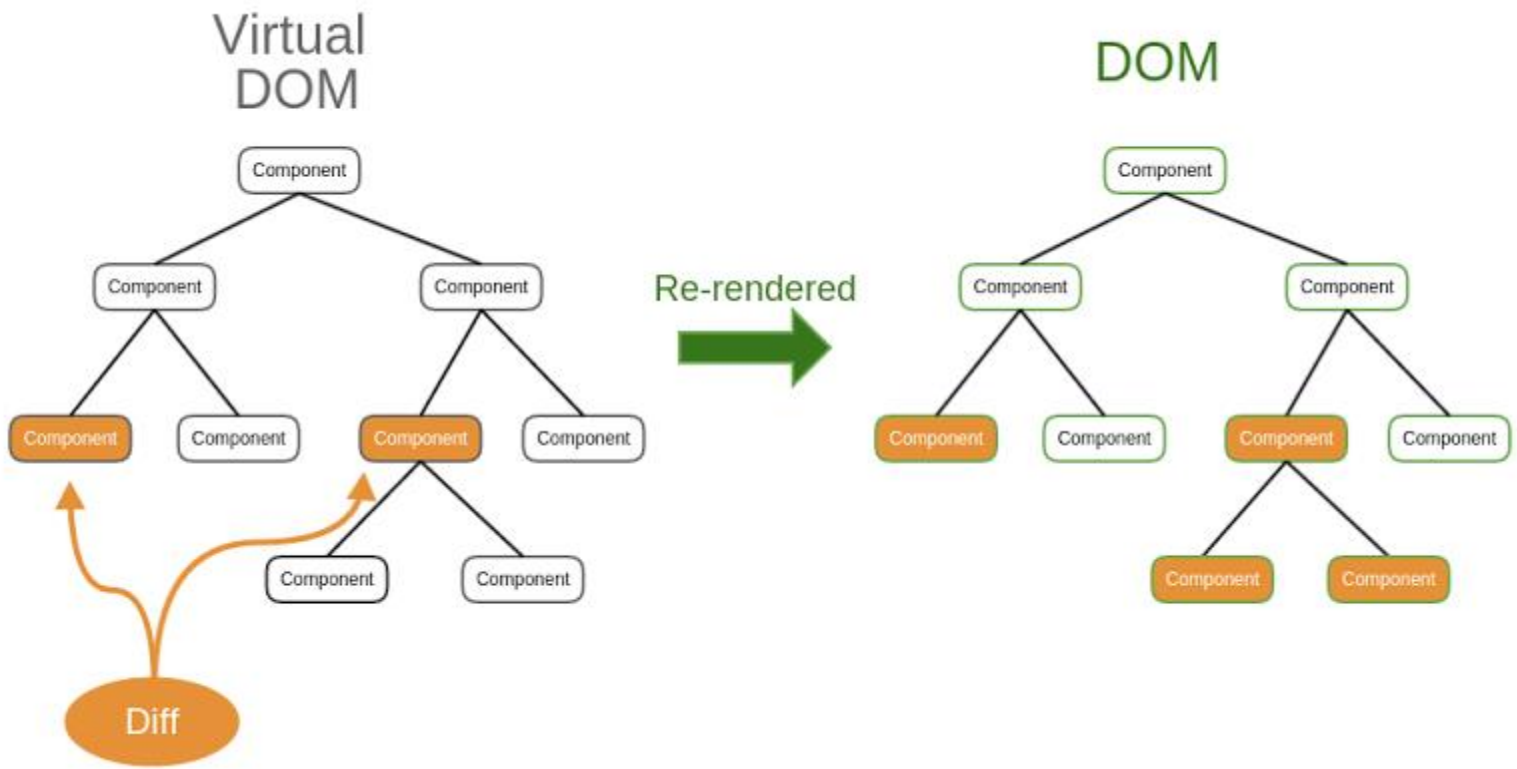

JOBKOREA
react
지역 선
경력
학력
기업형태
고용형태
연봉
조건추가
채용정보
기업정보
총 1,648건


JOBKOREA
angular
경력
학력
기업형태
고용형태
연봉
채용정보
기업정보
총 311건


JOBKOREA
vue
경력
학력
기업형태
고용형태
연봉
채용정보
기업정보
총 1,059건

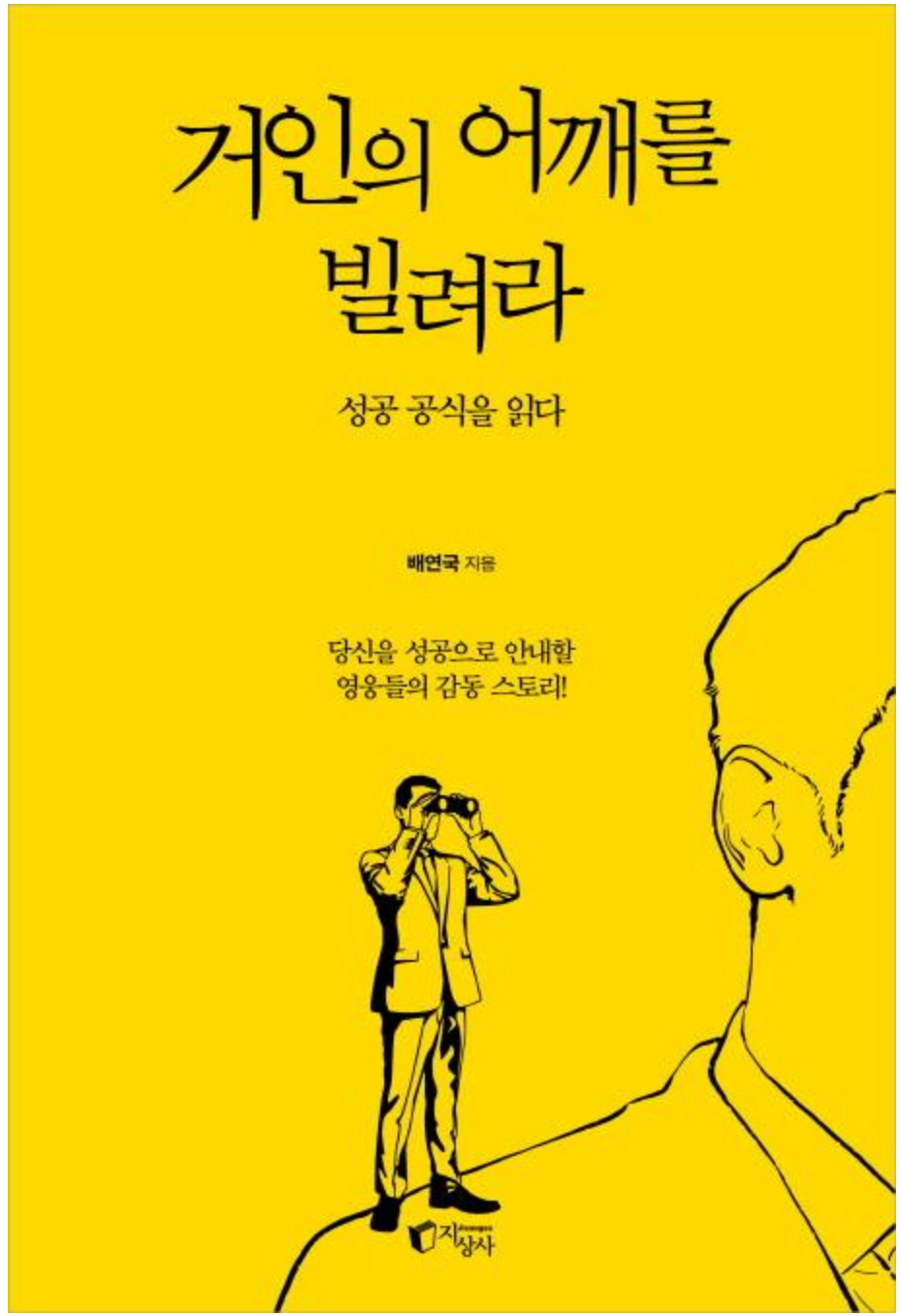

JOBKOREA
laravel
경력
학력
기업형태
고용형태
연봉
채용정보
기업정보
총 117건

Virtual DOM


Virtual DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component
Diff
Re-rendered
DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component

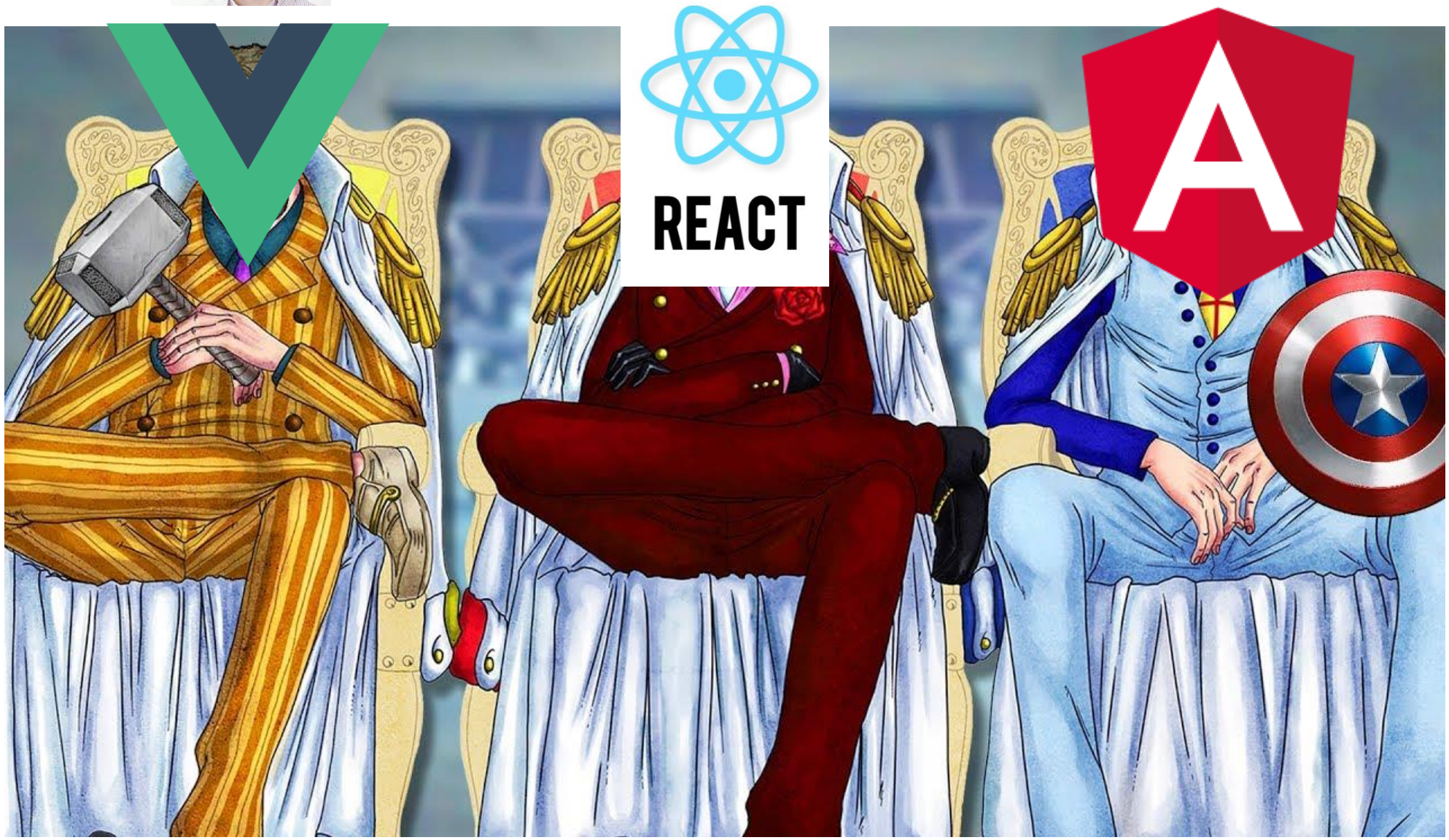
Virtual DOM

부드럽고 빠르다!


거인의 어깨를
빌려라
성공 공식을 읽다
배연국 지음
당신을 성공으로 안내할
영웅들의 감동 스토리!
지상사

facebook

Google



일단

시작하기!

CDN 링크

React와 ReactDOM 모두 CDN을 통해 사용할 수 있습니다.

```
<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
```

More REACT!?

- 클릭이 되면 버튼의 문구가 변경 되도록 수정!

```
function HelloWorldButton() {
  const [isClick, setClickState] = React.useState(false);
  const text = isClick ? "It's clicked" : "Hello, React world";

  return React.createElement(
    "button",
    { onClick: () => setClickState(!isClick) },
    text
  );
}

const e = React.createElement;
const domContainer = document.querySelector("#app");
const root = ReactDOM.createRoot(domContainer);
root.render(e(HelloWorldButton));
```

```
return React.createElement(
    "button",
    { onClick: () => setClickState(!isClick) },
    React.createElement("div", null, React.createElement("span", null, text))
  );
```


JSX
그런데, 짜잔!

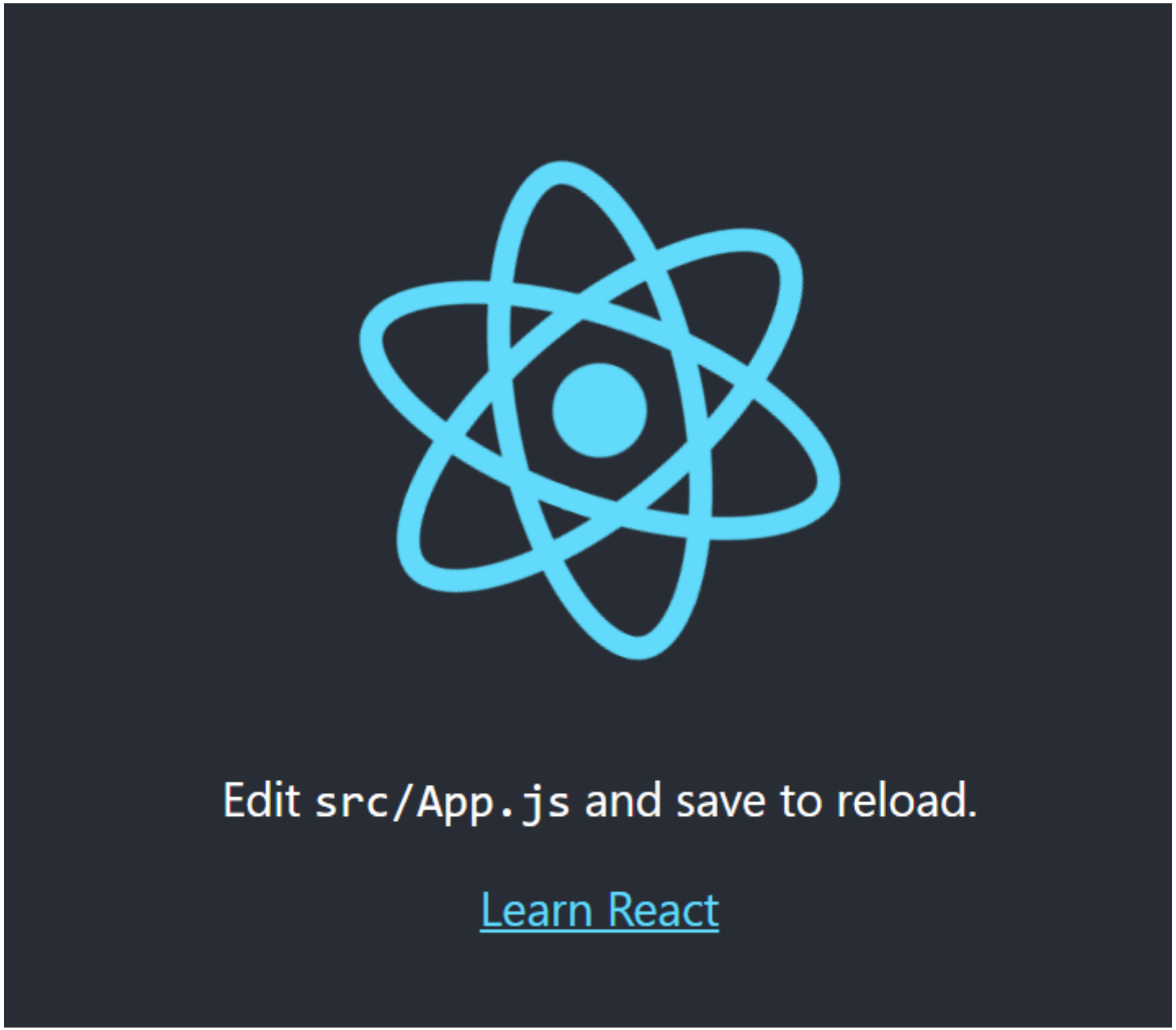
```jsx
// 함수형 컴포넌트
function HelloWorldButton() {
  const [isClick, setClickState] = React.useState(false);
  const text = isClick ? "It's clicked" : "Hello, React world";
  return (
    <button onClick={() => setClickState(!isClick)}>
      <div>
        <span>{text}</span>
      </div>
    </button>
  );
}

const domContainer = document.querySelector("#app");
const root = ReactDOM.createRoot(domContainer);
root.render(<HelloWorldButton />);
```

JSX

(JavaScript XML)

Babel

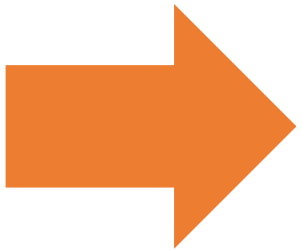
Create-react-app


Edit src/App.js and save to reload.
Learn React

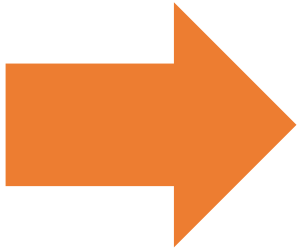
NPX?

NPM?

Welcome to JSX World!



```
return ();
```



```
{}
```

```jsx
function App() {
  const str = 'Hello, JSX World!'
  return (
    <div className="App">
      {str}
    </div>
  );
}

export default App;
```

Hello, JSX World!

JSX 문법 배우기

class → className

- 기존 DOM 요소에 class 를 부여 할 때 쓰던 class 는 className 이라고 써야만 합니다!
- JS 에서 class 는 JS 의 클래스 타입을 의미하므로 구분을 해줘야 합니다~!

```
function App() {
  const str = 'Hello, JSX World!'
  return (
    <div className="App">
      {str}
    </div>
  );
}

export default App;
```

JSX 를 쓰면 데이터 바인딩이 쉬워 집니다~!

- 하지만 JSX 가 출동 한다면!?

```
function App() {
  const str = 'Hello, JSX World!'
  return (
    <div className="App">
      {str}
    </div>
  );
}

export default App;
```

설마 인라인 스타일도?

- 맞습니다! 인라인 스타일도 {} 로 전달 해야 합니다~!
- 단, 객체 타입으로 전달을 해야 합니다! 즉, JS를 쓰겠다고 선언하는 {} 안에 객체를 의미하는 {} 를 담아서 전달해야 하죠!

```
function App() {
  return (
    <div className="App">
      <div style={{ fontSize: "32px" }}>인라인 스타일</div>
    </div>
  );
}

export default App;
```

인라인 스타일

kebob-case to camelCase

```
function App() {
  return (
    <div className="App">
      <div style={{ fontSize: "32px", backgroundColor: "orange" }}>인라인 스타일</div>
    </div>
  );
}

export default App;
```

이벤트 핸들러

적용하기!

리액트!

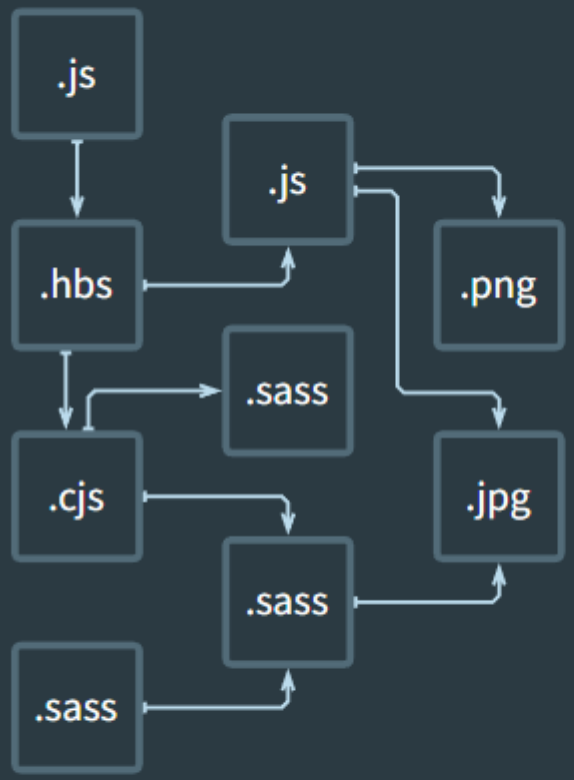
- onClick = { 클릭 되었을 때 실행할 JS 코드 }

```
function App() {
  return (
    <div className="App">
      <span onClick={() => { alert("클릭!") }}>클릭</span>
    </div>
  );
}

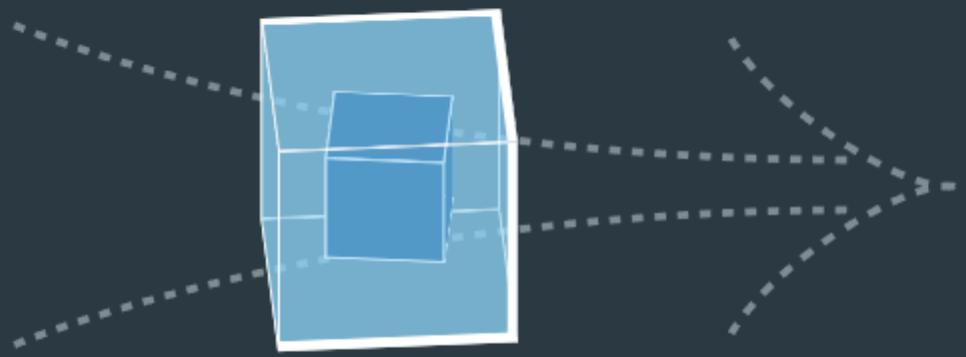
export default App;
```

Webpack

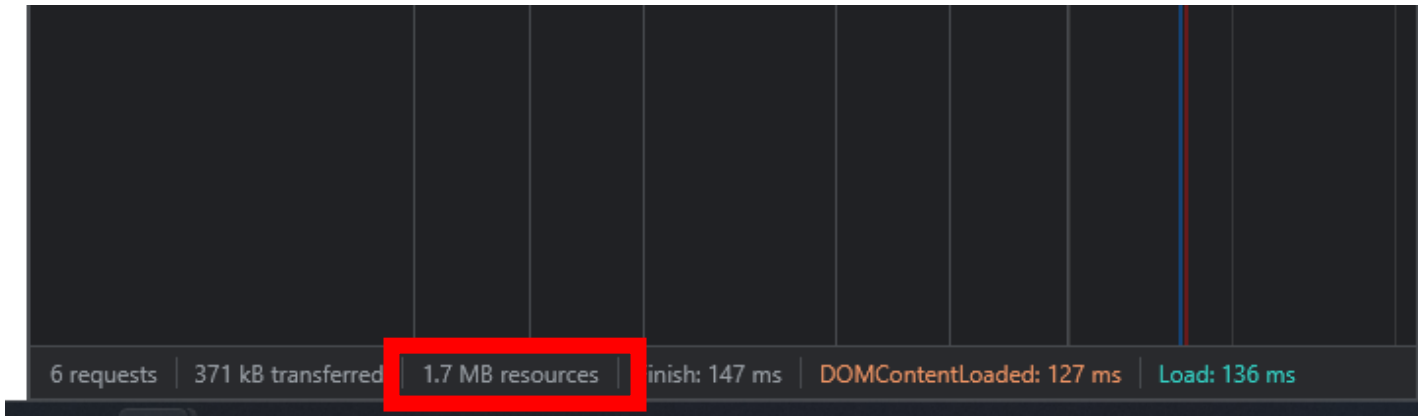
bundle your images

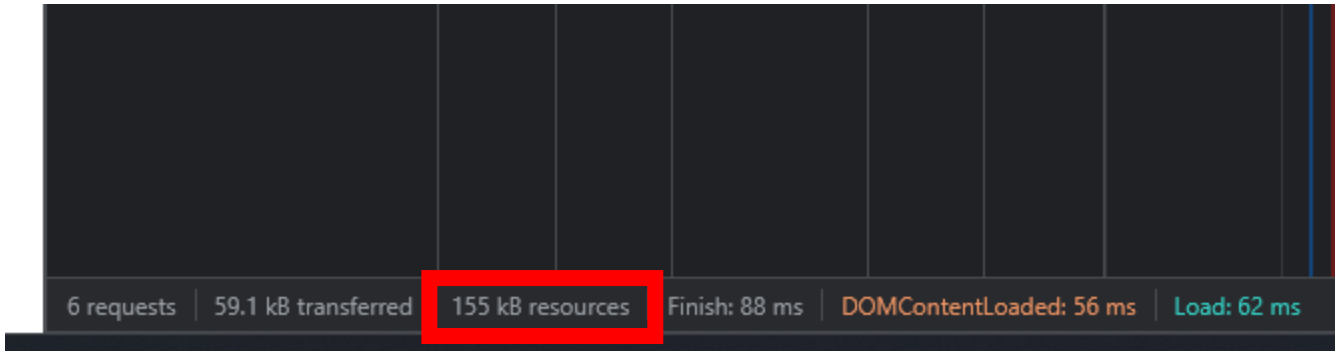


MODULES WITH DEPENDENCIES



STATIC ASSETS


6 requests
371 kB transferred
1.7 MB resources
inish: 147 ms
DOMContentLoaded: 127 ms
Load: 136 ms


6 requests
59.1 kB transferred
155 kB resources
Finish: 88 ms
DOMContentLoaded: 56 ms
Load: 62 ms

Component

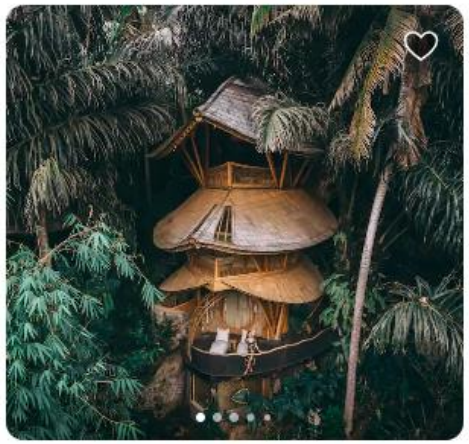
airbnb

어디든지 | 언제든 일주일 | 게스트 추가

호스트 되기

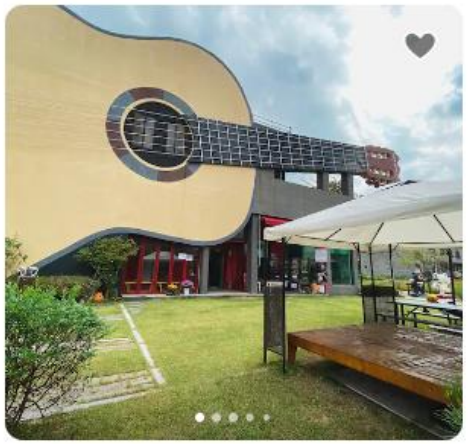
기상천외한 숙소 | 국립공원 | 통나무집 | 풍차 | 상징적 도시 | 섬 | 해변 근처 | 초소형 주택 | 디자인 | 캠핑카 | A자형 주택 | 호숫가 | 북극 | 멋진 수영장 | 동굴 | 서핑

필터

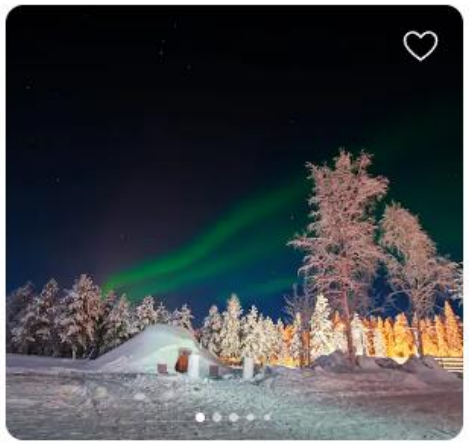


Abiansemal, 인도네시아 ★ 4.88
5,275km
4월 13일~18일
₩483,768 /박

Abiansemal, 인도네시아 ★ 5.0
5,275km
12월 13일~19일
₩1,550,904 /박



Sindun-myeon, Icheon-si, 한국 ★ 4.8
46km
10월 3일~8일
₩107,673 /박



Pelkosenniemi, 핀란드 ★ 4.81
6,604km
12월 28일 ~ 1월 2일
₩202,046 /박

Tambon Nong Kae, 태국 ★ 4.95
3,866km
10월 4일~9일
₩153,815 /박

Component 의 종류

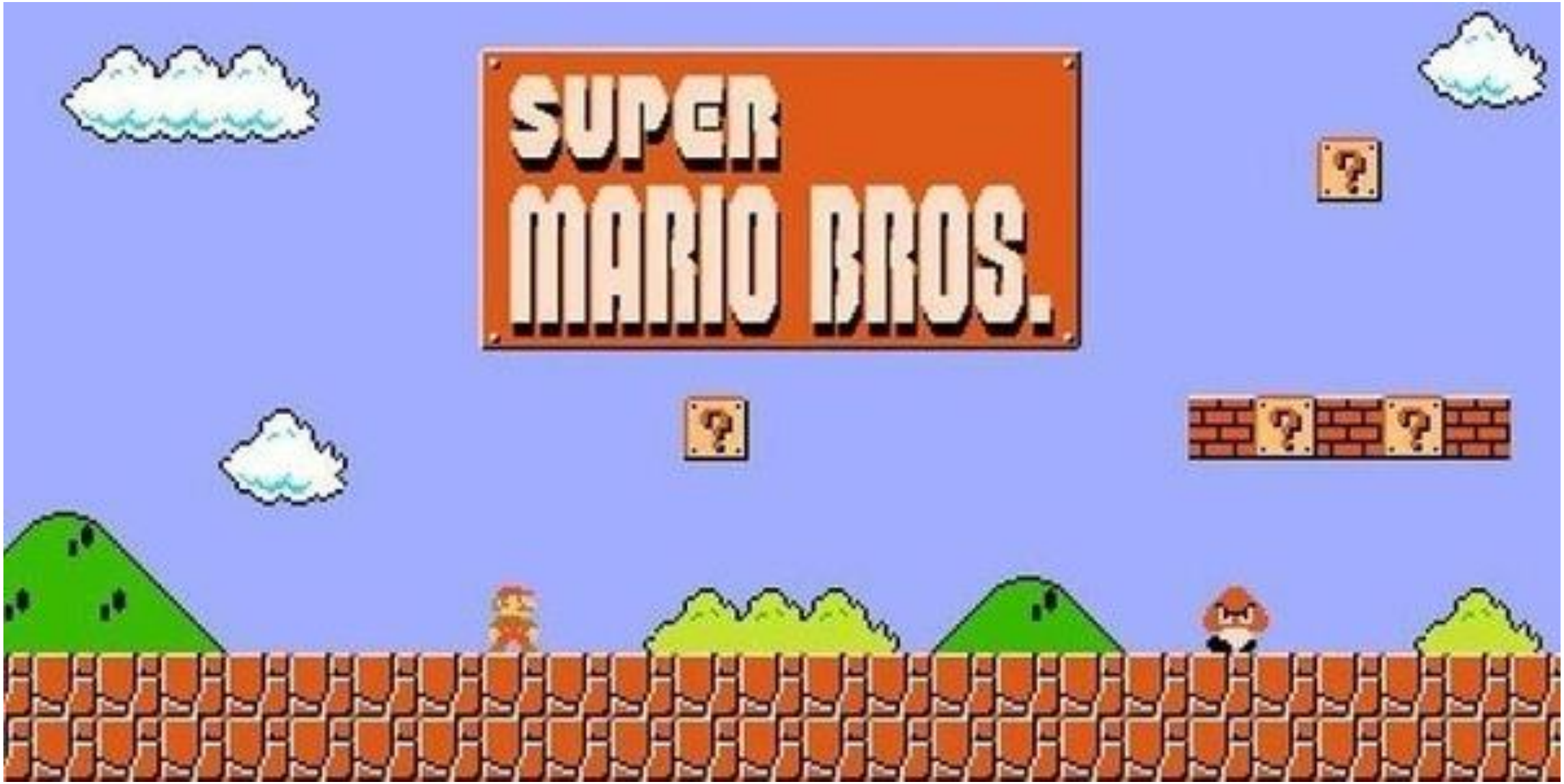
```
import React, { Component } from "react";

class MainHeader extends Component {
    render() {
        return (
            <h1>Hello, Component world!</h1>
        )
    }
}

export default MainHeader;
```

```
function MainHeader() {
  return <h1>Hello, Component world!</h1>;
}

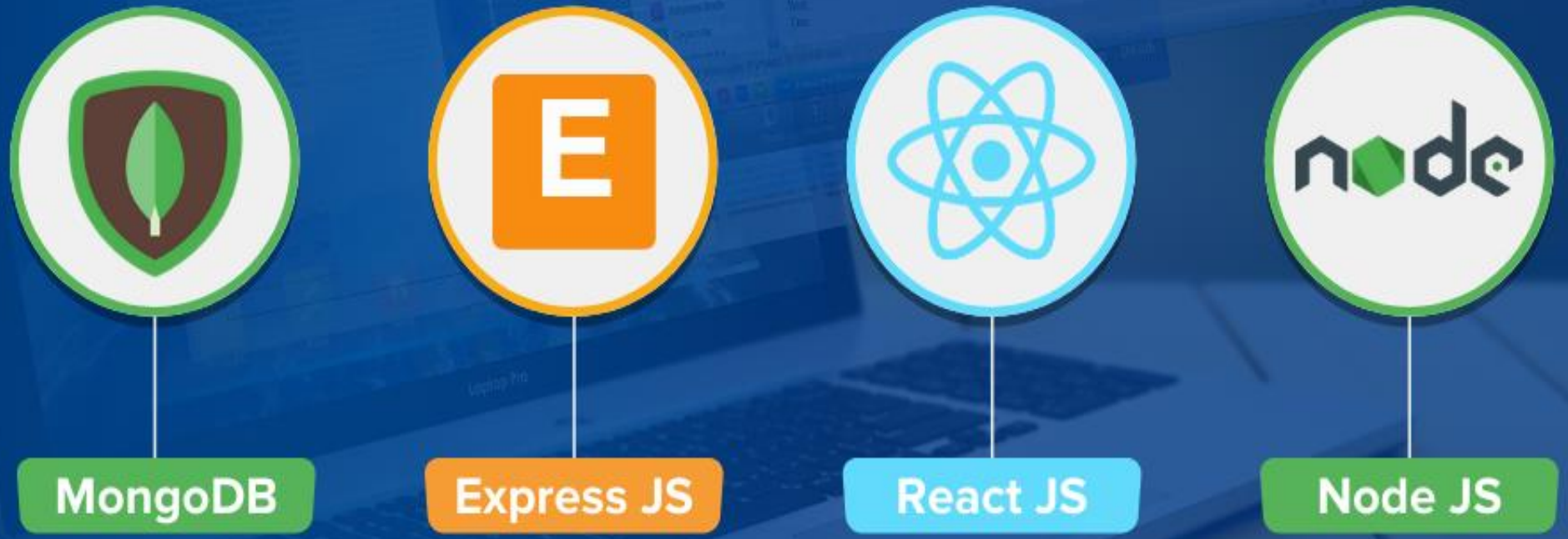
export default MainHeader;
```


SUPER
MARIO BROS.

rlogical techsoft pvt. ltd

MERN Stack

Best for Developing Web Apps




가 즈 아
https://coinpan.com 코인판 on 2018-01-09
K316
디지털
미디어시티
Digital Media City
K315
가 좌
Gajwa 加佐
P314
신 촌
Sinchon 新村
KORAIL
동서가구
민 성 한 의 원
373-1651

Formatting,

Linting,

Typescript 세팅!

Formatting

Formatting?

README.md — prettier-config-example

README.md

```
# Prettier support for other languages

This folder has JavaScript, CSS, HTML, JSON and even this Markdown file all formatted using
Prettier

Note that while formatting [index.js](index.js) Prettier uses 2 spaces per tab, but in the
Markdown code blocks it uses 4 spaces.

```js
const code = true;if (code) {console.log('code is on')}
```

This is because we override options inside the .prettierrc.json file.
```

Formatting

- Code의 스타일을 통일 시켜 줍니다!
- 함수의 소괄호와 중괄호는 띄울 것인지? 세미 콜론은 찍을 것인지? 탭을 누르면 몇 칸을 띄울 것인지? 등등등
- 문법이 아닌 코드의 스타일을 통일 시켜줘서 가독성을 높이고 버그를 예방합니다!
- Prettier 를 사용!

프로젝트에 Prettier 설정하기!

- npm install --save-dev prettier

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev prettier
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 1 package, and audited 2 packages in 660ms

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- --save 는 패키지 모듈에 추가한다는 의미입니다!
- --save-dev 는 개발할 때에만 사용하겠다는 의미 입니다!
  - 실제로 프로젝트를 빌드 & 배포하면 해당 패키지는 포함 X

프로젝트에 Prettier 설정하기!

```
{
  "scripts": {
    "test": "echo \"Hello, Node.js\""
  },
  "devDependencies": {
    "prettier": "^2.7.1"
  }
}
```

- Package.json 파일에 변화가 생겼죠?
- 방금 설치한 Prettier가 Package.json에 추가 되었습니다!
- 즉, 이렇게 npm 은 프로젝트의 패키지를 관리합니다!

프로젝트에 Prettier 설정하기!

- .prettierrc 파일로 prettier 세부 설정하기!

```
{
    "semi": true,
    "singleQuote": true
}
```

- Vs-code 에게 prettier 사용하라고 알려주기!
  - .vscode 폴더 만들고 settings.json 파일 만들기

```
{
    "[javascript]": {
        "editor.formatOnSave": true,
        "editor.defaultFormatter": "esbenp.prettier-vscode"
    }
}
```

프로젝트에 Prettier 설정하기!

- Prettier 확장 설치
- Main.js 파일의 코드를

```
console.log("Hello, Node.js")
```

- 로 작성하고 ctrl + s 로 저장해보기!!
- Prettier 가 작동합니다! :)

Linting

Linting?

Linting

- Formatting 에 가깝지만 더 많은 규약과 규율을 검사해주는 방법입니다!
- 웹 개발에서는 Airbnb 에서 사용하는 Linting 규율이 유명합니다!

ESLint 설치하기!

- npm install --save-dev eslint

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev eslint
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 111 packages, and audited 113 packages in 3s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- package-lock.json 파일을 보면 prettier에 비해 훨씬 많은 package 가 설치 되었음을 확인이 가능합니다 → 그만큼 많은 규약을 가지고 있다는 것!

ESLint 설정하기!

- .eslintrc.js 파일 생성

```
module.exports = {};
```

- 모든 Lint 관련 룰을 우리가 전부 지정 할 수는 없겠죠?
- Airbnb의 Linting Rule를 가져 옵시다!!
- npm install --save-dev eslint-config-airbnb-base eslint-plugin-import

ESLint 설정하기!

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev eslint-config-airbnb eslint-plugin-import
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 88 packages, and audited 201 packages in 6s

68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
{
  "scripts": {
    "test": "echo \"Hello, Node.js\""
  },
  "devDependencies": {
    "eslint": "^8.22.0",
    "eslint-config-airbnb": "^19.0.4",
    "eslint-plugin-import": "^2.26.0",
    "prettier": "^2.7.1"
  }
}
```

ESLint 설정하기!

- .eslintrc.js 파일을 생성 후, Airbnb 모듈 추가!

```
module.exports = {
  extends: ['airbnb-base'],
};
```

- Windows 사용자라면 LF, CRLF 문제 해결을 위해 아래의 코드도 추가!

```
module.exports = {
  extends: ['airbnb-base'],
  rules: {
    'linebreak-style': 0,
    'no-console': 'off',
  },
};
```

ESLint 체험하기!

- main.js 파일 만들어 보기

```
var x = 1;
console.log(x);
```

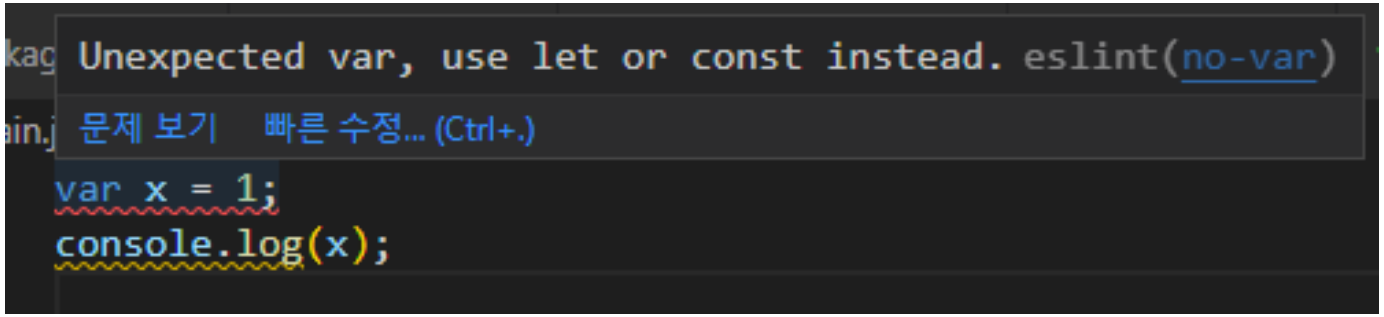
- 이 코드에 문제가 있을까요?

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
1
```

- 실행을 해도 문제는 없습니다!

ESLint 체험하기!

- Javascript 상으로는 문제가 없지만 ESLint 의 규약에는 문제가 생깁니다!



- Var 는 위험하지 쓰지 말고 let 이나 const 를 쓰라고 하네요?

ESLint 체험하기!

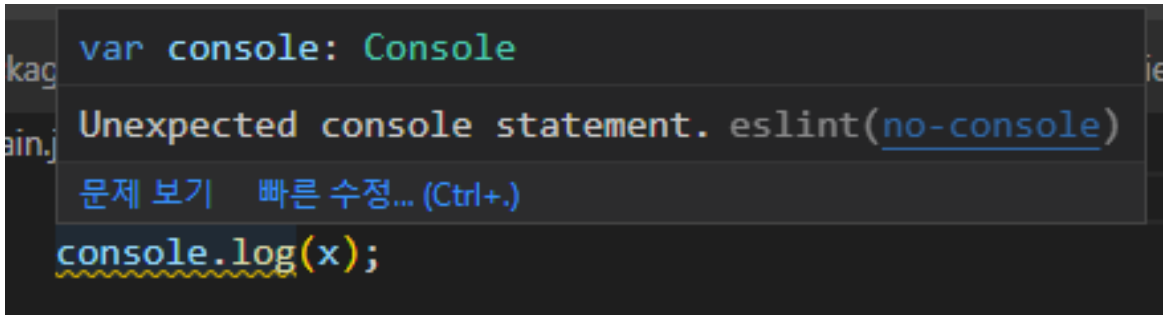
- Var 를 let 으로 바꾸면?

```
let x = 1;
cons
    let x: number
    'x' is never reassigned. Use 'const' instead. eslint(prefer-const)
    문제 보기   빠른 수정... (Ctrl+.)
```

- Let 은 변화하는 변수일 경우에만 할당하는 편이 좋으니 const 를 쓰라고 합니다!
- 즉, 이렇게 개발자가 간과할 수 있는 Rule 들을 바로바로 알려주기 때문에 나중에 발생할 예상외의 버그 또는 문제를 많이 해결해 줍니다!

ESLint 체험하기!

- Console.log 도 노란줄이 있네요?



- Console.log 는 보통 디버깅용으로 쓰다보니 사용자에게 출시 할 때는 대부분 제거를 해야만 합니다! 그렇다 보니 사용자에게 특정 목적을 알리기 위해 따로 만든 console statement 가 아닌 것은 쓰지 말라는 것이죠!
- 하지만 이건 너무 가혹하니까!! Rule 을 수정 합시다!

ESLint 체험하기!

- .eslintrc.js 로 고고고

```
module.exports = {
  extends: ['airbnb-base'],
  rules: {
    'linebreak-style': 0,
    'no-console': 'off',
  },
};
```

- No-console 옵션을 꺼줍니다!

```
JS main.js > ...
1   let x = 1;
2   console.log(x);
3
```


그분들
adidas
CHEVROLET
해축드립사전

Guide

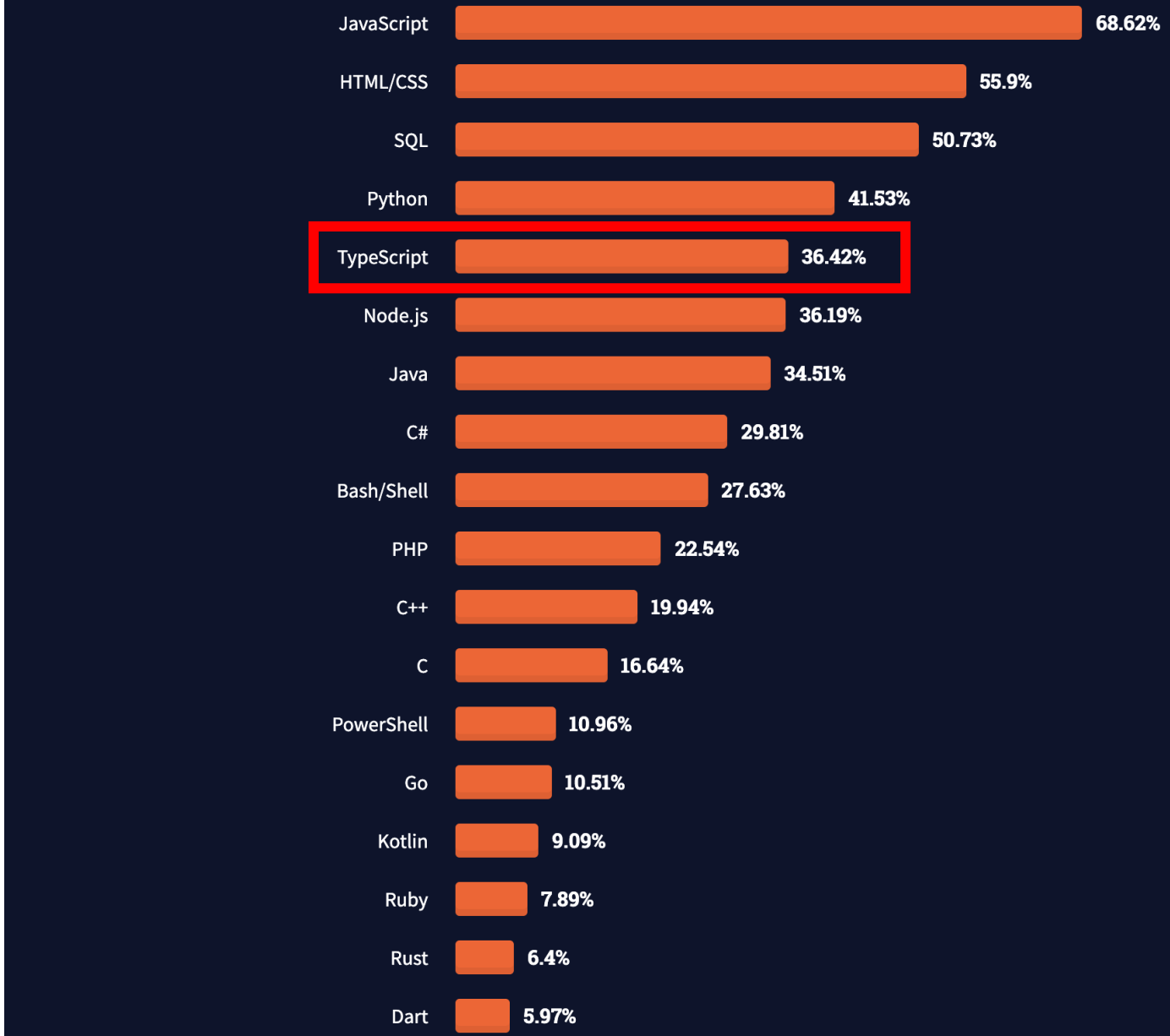
협업왕이 되어보자!

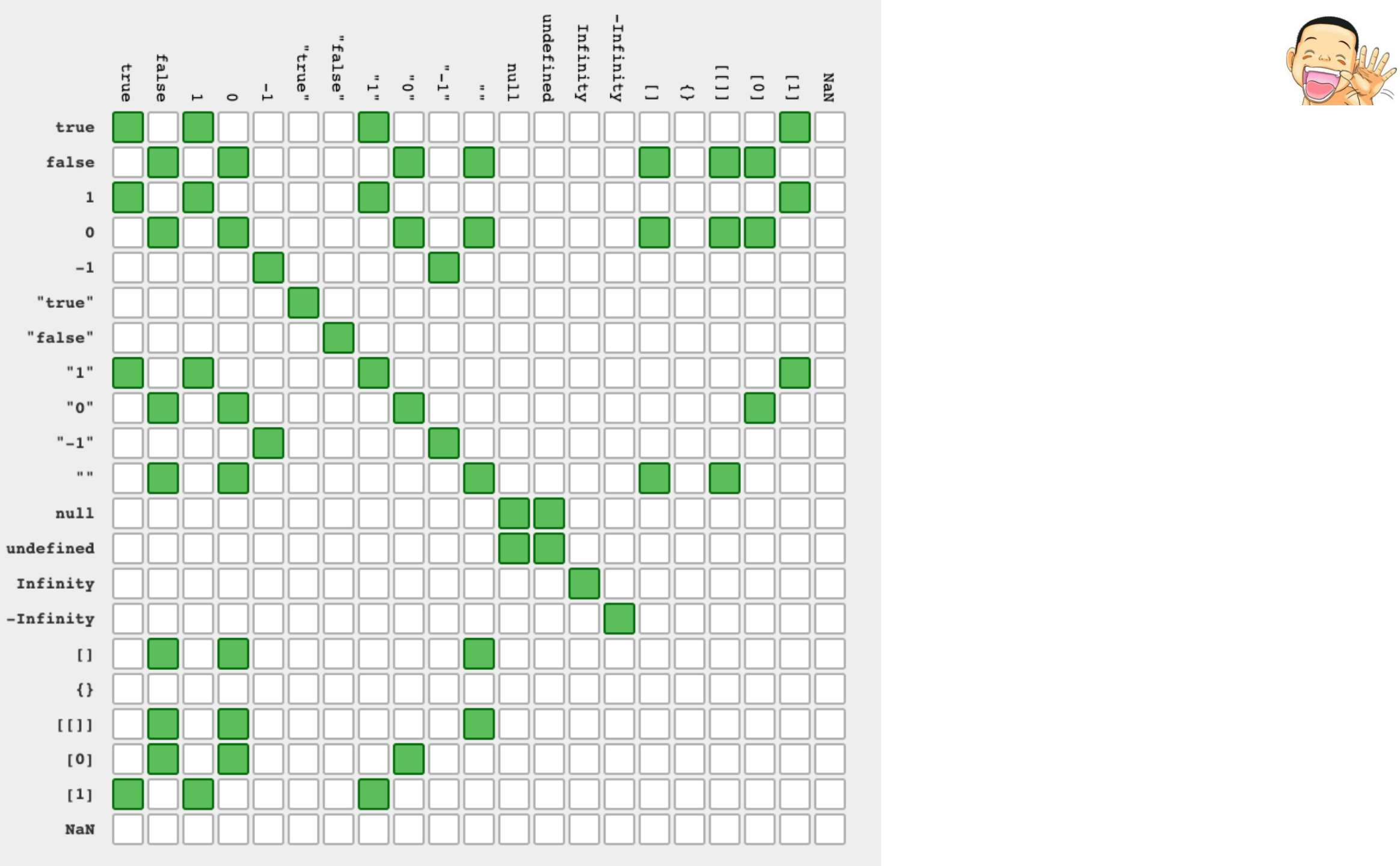
#개발자 #디자이너
#후배 #외부업체

Typescript

All Respondents | Professional Developers

58,031 responses




true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
NaN

자, 코드를 봅시다!

- ES Lint 상으로는 문제가 없는 코드입니다!

```
const str = 'Hello';
const num = Math.log(str);
console.log(num);
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
NaN
```

- 그런데 사실 프로그래밍 적으로 NaN 은 쓸 이유가 없는 타입입니다!

Typescript 설치

- 이러한 문제를 막아 주는 것이 Typescript!
- npm install --save-dev typescript

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev typescript
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

changed 1 package, and audited 202 packages in 2s

68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Typescript 적용

- Main.js 파일에 // @ts-check 주석 추가

```
// @ts-check
```

```
JS main.js > ...
1   // @ts-check
2
3   const str = 'Hello';
4   const num = Math.log(str);
5   console.log(num);
6
```

```
const str: "Hello"
'string' 형식의 인수는 'number' 형식의 매개 변수에 할당될 수 없습니다. ts(2345)
문제 보기   빠른 수정... (Ctrl+.)
```

- 이제 Type 관련 문제는 Typescript 가 알려 줍니다!

Type 관련 문제는
TS가 처리했으니 안심하너굴!

State

State

- 매우 중요한 개념입니다!!
- 간단하게 표현하면 중요한 변수 정도로 생각을 하셔도 됩니다
- 그런데 State 의 뜻이 뭐죠? → 상태를 의미 합니다
- 즉, 리액트에서 컴포넌트에 대한 상태를 의미 합니다
- 사용하는 이유는? → State 가 변경되면 해당 컴포넌트는 바로 다시 렌더링 이 되기 때문에 컴포넌트의 유동성 관리가 쉽습니다!

State

체험하기

State 체험하기!

- 이런 코드가 있다고 가정해 봅시다!

```
function App() {
  const teacher = "이효석";

  return (
    <div className="App">
      <button>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

State 체험하기!

- 영어로! 버튼을 누르면 teacher 를 이효석 → tetz 로 변경해 봅시다!

```
function App() {
  let teacher = "이효석";

  function inEnglish() {
    teacher = "tetz";
    console.log(teacher);
  }

  return (
    <div className="App">
      <button onClick={() => inEnglish()}>영어로!</button> <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

State 체험하기!

- 버튼을 누르면 어떻게 될까요? 글자가 변경 될까요?

- 애석하게도 글자는 변경되지 않습니다! 페이지 새로 고침이 안되었으니까요!
- 그런데 console.log 를 보면 분명 teacher 변수는 tetz 로 변경이 되어 있네요!

State 체험하기!

- 그럼 JS 적으로 버튼을 클릭 했을 때, 이효석을 tetz 로 변경 하려면 어찌하면 될까요?

```
function App() {
  const teacher = "이효석";

  function inEnglish() {
    const spanEl = document.querySelector(".App > span");
    spanEl.innerHTML = "tetz";
  }

  return (
    <div className="App">
      <button onClick={() => inEnglish()}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

State 체험하기!

- React 는 컴포넌트 적으로 변화가 자주 일어나는 곳에서 사용하면 좋다고 했었습니다! (빠르고 부드럽기 때문에!)
- 그런데 이렇게 쓴다면……

useState!

useState

- 앞서 state 가 변경 되면, 리액트에서는 해당 부분을 바로 리렌더링(다시 그려주기) 한다고 말씀을 드렸었는데요!
- 그럼 한번 useState 를 사용해 보시죠!

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

function App() {
  const [teacher, setTeacher] = useState("이효석");

  return (
    <div className="App">
      <button onClick={() => setTeacher("tetz")}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

useState

- state 를 활용하면 훨씬 쉽게 변경 사항을 HTML 반영할 수 있습니다!
- 그럼 useState 문법에 대해서 차근차근 알아 보시죠!

useState 문법!

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

const [스테이트이름, 스테이트변경함수] = useState(초기값);
```

- 먼저 상태를 관리하는 state 를 배열의 첫번째로 정해주고, 해당 state 를 변경 할 수 있는 함수를 두번째로 지정해 주면 됩니다!
- 그리고 useState 의 () 안에는 state 의 초기 값을 넣어 줍니다!

const [state, setState] = useState(initialState)

The name of your state

The function you'll eventually use to change the value of this state

The initial value of your state

useState 동작 원리

- useState 는 단순하게 동작 합니다!
- state 가 이전의 값과 달라지면 해당 컴포넌트를 다시 렌더링 합니다!
- 즉, “이효석”으로 지정 되어있던 초기 값이 “tetz”로 변경 되었기 때문에 해당 HTML을 다시 렌더링 한 것이죠!

useState 동작 원리

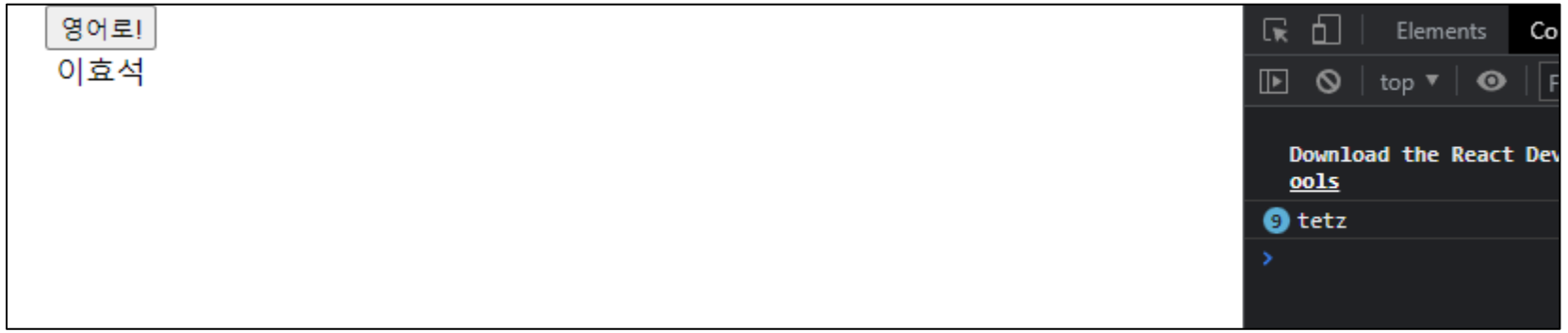
- 이런 코드는 무한히 새롭게 렌더링이 됩니다!

```
function App() {
  const [strState, setStrState] = useState("init");

  return (
    <div className="App">
      <button onClick={() => setStrState(strState + "+")}>반복!</button>
      <br />
      <span>{strState}</span>
    </div>
  );
}
export default App;
```

그럼 이런 코드도 한번 봅시다~!

```
function App() {
  let [teacher, setTeacher] = useState("이효석");

  function customSetTeacher(name) {
    teacher = name;
    console.log(teacher);
  }

  return (
    <div className="App">
      <button onClick={() => customSetTeacher("tetz")}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

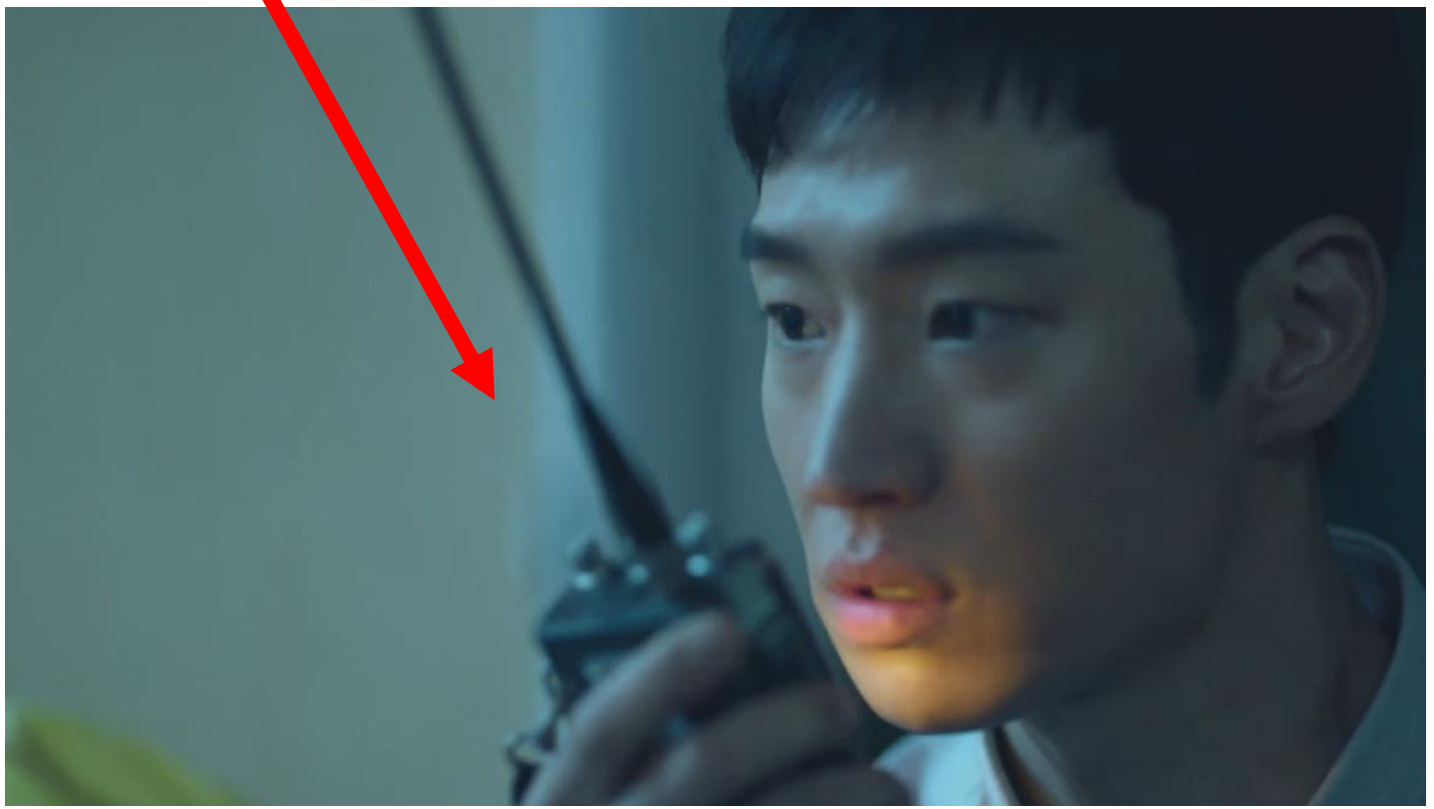
그럼 이런 코드도 한번 봅시다~!



- 보시는 것처럼! state 가 변경이 되네요?
- 그런데 리렌더링은 발생하지 않습니다!

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

const [스테이트이름, 스테이트변경함수] = useState(초기값);
```

- 상태 변경 함수로 변경을 해야만 React 가 state 변경을 알아차리고 html 을 변경해 줍니다!



주의 사항!

상태 변경 함수는 꼭 함수 정의를 하고 사용!

- Return 내부에서 함수 정의를 하지 않고 바로 상태 변경 함수를 호출하면 클릭이 안되어도 실행이 되기 때문에 무한 루프에 빠지게 됩니다!
- State 가 변경되면 당연히 return 내부에 있는 요소를 다시 렌더링 해야 하니 return 을 호출 → 다시 상태 변경 함수 호출 → return 호출의 무한 루프가 되죠!

```jsx
function App() {
  let [teacher, setTeacher] = useState("이효석");

  return (
    <div className="App">
      <button onClick={setTeacher("tetz")}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

```
▶Uncaught Error: Too many re-renders. React limits the number of     react-dom.develop
renders to prevent an infinite loop.
    at renderWithHooks (react-dom.development.js:16317:1)
    at mountIndeterminateComponent (react-dom.development.js:20074:1)
    at beginWork (react-dom.development.js:21587:1)
    at HTMLUnknownElement.callCallback (react-dom.development.js:4164:1)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:4213:1)
    at invokeGuardedCallback (react-dom.development.js:4277:1)
    at beginWork$1 (react-dom.development.js:27451:1)
    at performUnitOfWork (react-dom.development.js:26557:1)
    at workLoopSync (react-dom.development.js:26466:1)
    at renderRootSync (react-dom.development.js:26434:1)
```

요렇게 쓰세요!

- return 에서 익명함수로 정의해서 사용

```
function App() {
  let [teacher, setTeacher] = useState("이효석");

  return (
    <div className="App">
      <button onClick={() => setTeacher("tetz")}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

요렇게 쓰세요!

- 아예 함수로 만들어서 호출하기!

```
function App() {
  let [teacher, setTeacher] = useState("이효석");

  function customSetTeacher() {
    setTeacher("tetz");
  }

  return (
    <div className="App">
      <button onClick={customSetTeacher}>영어로!</button>
      <br />
      <span>{teacher}</span>
    </div>
  );
}
export default App;
```

실습, 카운터 만들기!

- useState Hook 을 이용해서 간단한 카운터를 만들어 봅시다~!

1

- +

- + 버튼을 누르면 출력 되는 숫자에 +1 이 – 버튼을 누르면 -1 이 되도록 만들어 주시면 됩니다!
- 해당 카운터는 컴포넌트로 파일로 만드셔서 임포트 하시면 됩니다!

그럼 약간 더 'REACT'스럽게 만들어 볼까요?

3항 연산자 사용하기!

- state 를 변경하면? → 컴포넌트가 리렌더링이 됩니다!
- 그럼, state 가 변경 될 때 다른 걸 변경하면!? → 리렌더링이 일어나면서 스무스하게 변경이 되겠군요!?
- 그럼 3항 연산자를 사용해서 리액트 스러운 뭔가를 만들어 봅시다~!

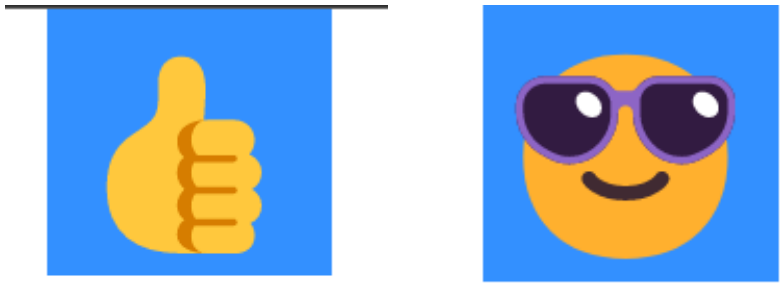
우리가 만들 것은~

- 컨디션 변경이라는 버튼을 하나 만들고, 컨디션 변경을 클릭 하면 이모지가 변경되는 컴포넌트를 만들어 볼게요!
- 컨디션을 state 로 반들고 컨디션의 상태에 따라서 다른 결과물을 출력하도록 3항 연산자를 사용해서 만들 겁니다~!

```jsx
function App() {
  let [condition, setCondition] = useState(true);

  return (
    <div className="App">
      <button onClick={() => setCondition(!condition)}>컨디션 변경!!</button>
      <br />
      <span style={{ fontSize: "100px" }}>{condition ? "👍" : "😂"}</span>
    </div>
  );
}
export default App;
```

실습, 좋아요 버튼 만들기!

- 👍 이모지를 클릭하면 아래의 숫자가 +1 이 되는 컴포넌트를 만들어 주세요!
- 아래의 숫자가 10이 넘으면 이모지를 😎 로 변경해 주세요!



7

15

클래스형 컴포넌트의 State

클래스형 컴포넌트의 State

- 클래스에서 사용하는 생성자에 state 값을 지정합니다
- this.state 라는 객체에 변경하고자 하는 값을 저장합니다
  - 단, state 는 반드시 객체로 지정해서 사용해야 합니다!
- 그리고 this.setState 메소드를 이용하여 this.state 라는 객체에 저장 된 값을 변경 합니다
- State 변경이 일어나면 컴포넌트는 알아서 다시 렌더링 됩니다!

```jsx
import React, { Component } from "react";

class ClassState extends Component {
  constructor(props) {
    super(props);
    this.state = {
      teacher: "이효석",
    };
  }
  render() {
    const { teacher } = this.state;

    return (
      <div>
        <button onClick={() => this.setState({ teacher: "tetz" })}>
          영어로!
        </button>
        <br />
        <span style={{ fontSize: "100px" }}>{teacher}</span>
      </div>
    );
  }
}

export default ClassState;
```

```jsx
import React, { Component } from "react";

class ClassState extends Component {
  // 현재 버전
  state = {
    teacher: "이효석",
  };
  render() {
    const { teacher } = this.state;

    return (
      <div>
        <button onClick={() => this.setState({ teacher: "tetz" })}>
          영어로!
        </button>
        <br />
        <span style={{ fontSize: "100px" }}>{teacher}</span>
      </div>
    );
  }
}

export default ClassState;
```

함수형 컴포넌트의 State

- 함수형 컴포넌트의 초창기에는 리액트의 핵심 기능(State, Lifecycle)들을 기능을 쓸 수 없었습니다!
- 하지만 16.8 버전 이후 부터는 hooks 라는 메소드를 제공하여 함수형 컴포넌트에서도 핵심 기능(State, Lifecycle)들을 사용이 가능해 졌습니다
- 그래서 더 간단하고 쉬운 함수형 컴포넌트만을 사용하는 시대가 열렸죠!
- 저렇게 썼다는 것만 기억해 주세요!

State 와 변수!

State 와 변수!

- 그럼 state 와 변수를 한번 비교해 볼까요?

```jsx
function App() {
  let [state, setState] = useState(0);
  let variable = 0;
  const setVariable = () => {
    variable += 1;
    console.log(`state: ${state} / variable: ${variable}`);
  };

  return (
    <div className="App">
      {state} / {variable}
      <br />
      <button
        onClick={() => {
          setState((state += 1));
          setVariable();
        }}
      >
        +1
      </button>
    </div>
  );
}
```

```
state: 1 / variable: 1
state: 2 / variable: 1
state: 3 / variable: 1
state: 4 / variable: 1
state: 5 / variable: 1
state: 6 / variable: 1
state: 7 / variable: 1
state: 8 / variable: 1
state: 9 / variable: 1
state: 10 / variable: 1
state: 11 / variable: 1
state: 12 / variable: 1
state: 13 / variable: 1
```

State!?

- 변수는 왜 계속 1이죠?
- State 가 변경되면 해당 state 를 정의한 컴포넌트(함수)가 다시 랜더링 됩니다! → 함수가 다시 읽혀지면? 변수는 그 때 다시 정의가 되겠죠? → 그래서 계속 1로 머물게 되는 겁니다!
- 이런 부분은 유의를 하고 사용하셔야 합니다~!

State

사용시 주의 사항

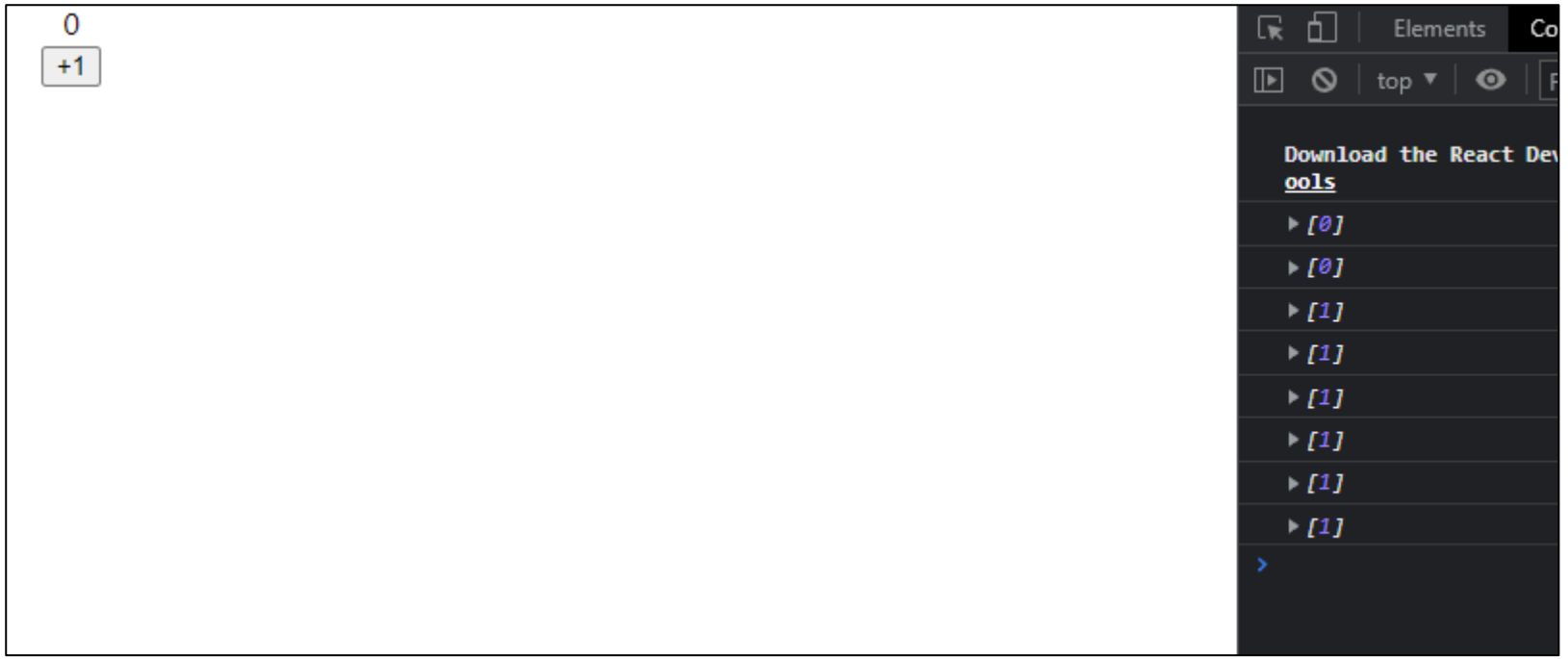
객체 또는 배열을 State 로 사용!

```
function App() {
  let [state, setState] = useState([0]);
  console.log(state);
  return (
    <div className="App">
      {state}
      <br />
      <button
        onClick={() => {
          state[0] = 1;
          setState(state);
          console.log(state);
        }}
      >
        +1
      </button>
    </div>
  );
}
```

- State 의 초기 값을 배열로 세팅해 보겠습니다!

객체 또는 배열을 State 로 사용!

- 아무리 눌러도 변화가 없네요!?
- 분명 값이 0 에서 1 로 변화 했는데 왜? 변경이 안되죠?
- 아시는 분???



원시 (Primitive) 타입

String
Number
Boolean
Null
Undefined
BigInt
Symbol

객체 (Object) 타입

원시 타입을 제외한 모든 것

Object
Array
...

원시 (Primitive) 타입

```
const location = "korea"
```

location

"korea"

객체 (Object) 타입

```
const location = {
    country: "korea"

}
```

location

#12345

#12345

{ country: "korea" }

원시 (Primitive) 타입

```
const locationOne = "korea"
const locationTwo = "korea"



locationOne === locationTwo
> true
```

객체 (Object) 타입

```
const locationOne = {
    country: "korea"
}

const locationTwo = {
    country: "korea"
}


locationOne === locationTwo
> false
```

객체 또는 배열을 State 로 사용!

- 그렇다면!? 새롭게 배열을 만들어서 메모리 주소를 바꿉시다!

```
function App() {
  let [state, setState] = useState([0]);

  return (
    <div className="App">
      {state}
      <br />
      <button
        onClick={() => { setState([1]);}}
      >
        +1
      </button>
    </div>
  );
}
```

객체 또는 배열을 State 로 사용!

- 또는?

```
function App() {
  let [state, setState] = useState([0]);

  return (
    <div className="App">
      {state}
      <br />
      <button
        onClick={() => {
          state[0] = 1;
          const copyArr = [...state];
          setState(copyArr);
        }}
      >
        +1
      </button>
    </div>
  );
}
```

객체 또는 배열을 State 로 사용!

- 객체를 쓰면!?

```
function App() {
  let [state, setState] = useState({ teacher: "이효석" });
  console.log(state);
  return (
    <div className="App">
      <button
        onClick={() => {
          state.teacher = "tetz";
          setState(state);
          console.log(state);
        }}
      >
        영어로!
      </button>
      <br />
      <span>{state.teacher}</span>
    </div>
  );
}
```

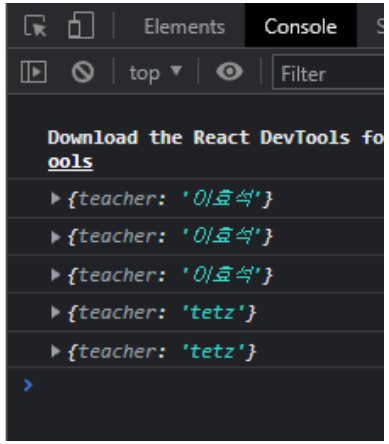
객체 또는 배열을 State 로 사용!

- 역시 변화가 없습니다!

```jsx
function App() {
  let [state, setState] = useState({ teacher: "이효석" });
  console.log(state);
  return (
    <div className="App">
      <button
        onClick={() => {
          setState({ teacher: "tetz" });
          console.log(state);
        }}
      >
        영어로!
      </button>
      <br />
      <span>{state.teacher}</span>
    </div>
  );
}
```

영어로!

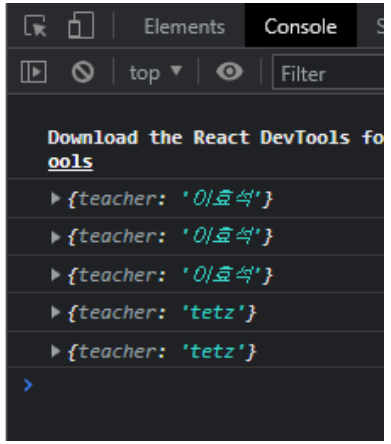
tetz



```jsx
function App() {
  let [state, setState] = useState({ teacher: "이효석" });
  console.log(state);
  return (
    <div className="App">
      <button
        onClick={() => {
          state.teacher = "tetz";
          const copyObj = { ...state };
          setState(copyObj);
          console.log(state);
        }}
      >
        영어로!
      </button>
      <br />
      <span>{state.teacher}</span>
    </div>
  );
}
```

영어로!

tetz



결론!

- 자주 값이 변경이 되고, 변경 사항을 바로바로 보여줘야 하면? → State 로 선언해서 쓰자!
- 배열 또는 객체로 쓰기 보다는 하나의 원시 값으로 쓰는 것이 좋다!
- 배열 또는 객체로 사용을 해야 한다면, JS의 객체 타입 데이터의 특성을 잘 생각해서 사용하자!

컴포넌트로

코드를 줄이자~!

컴포넌트 활용하기 – 코드 줄이기!

- 기존 HTML 에서는 기존의 것과 동일한 것을 만들어도 코드를 동일하게 써야만 했습니다~!
- 즉, 템플릿 기능이 없었죠!
- 리액트에서는 이러한 HTML 의 문제를 컴포넌트를 활용하여 손쉽게 해결이 가능합니다~! 느낌이 오시죠!?

```jsx
function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
      <div className="modal">
        <h2>오늘 해야할 일 2개</h2>
        <h2>오늘 완료한 일 0개</h2>
      </div>
    </div>
  );
}

export default List();
```

갑자기 요 modal DIV를

최상단에도 넣고 싶다면!?

```jsx
export default function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <div className="modal">
        <h2>오늘 해야할 일 2개</h2>
        <h2>오늘 완료한 일 0개</h2>
      </div>
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
      <div className="modal">
        <h2>오늘 해야할 일 2개</h2>
        <h2>오늘 완료한 일 0개</h2>
      </div>
    </div>
  );
}
export default List();
```

HTML 에서는 어쩔 수 없이

HTML 코드를 그대로 복사해서

써야만 했었죠!

컴포넌트 활용하기 – 코드 줄이기!

- 그런데 저런 Modal 을 약 1000 개의 페이지 마다 계속 써줘야 한다면!?
- 그리고 Modal 에 갑자기 내용이 추가 되어야 한다면?
- 매우 골치가 아플 것입니다~!

컴포넌트 활용하기 – 코드 줄이기!

- 이럴 땐 해당 Modal 을 컴포넌트로 만들고 불러서 쓰면 쉽겠죠?
- 그럼 한번 해보시죠!

```jsx
function Modal() {
  return (
    <div className="modal">
      <h2>오늘 해야할 일 2개</h2>
      <h2>오늘 완료한 일 0개</h2>
    </div>
  );
}
export default Modal;
```

```jsx
import Modal from "./Modal";

function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <Modal />
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
      <Modal />
    </div>
  );
}
export default List;
```

오늘 해야할일

오늘 해야할 일 2개

오늘 완료한 일 0개

리액트 공부하기

state 사용법 익히기

코테 문제 풀기

Lv 0 정복 가즈아

오늘 해야할 일 2개

오늘 완료한 일 0개

컴포넌트 활용하기 – 코드 줄이기!

- 이렇게 컴포넌트를 쓰면 코드를 크게 줄일 수 있으며, 재사용도 편리 합니다!
- 그리고, 모달에 내용이 추가 되어도 원본 파일의 코드만 바꾸면 모든 모달이 변하므로 이전 처럼 1000 번이나 각각 코드에 가서 수정을 할 필요도 없죠~!

엇, 그렇다면!?

```jsx
import Modal from "./Modal";

function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <Modal />
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
      <Modal />
    </div>
  );
}
export default List;
```

이 친구들도 뭔가 반복적으로

사용되는데……

컴포넌트로 처리할 수 있지 않을까요?


108
녹 양
Nogyang
109
가 능
Ganeung
佳陵
110
의정부
Uijeongbu
KORAIL

그런데…

```
import Modal from "./Modal";

function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <Modal />
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
      <Modal />
    </div>
  );
}
export default List;
```

- 얘들은 내용이 서로 좀 다르네요?
- 그럼 컴포넌트로 못하는 거 아닌가요?


108
녹 양
Nogyang
109
가 능
Ganeung
佳陵
110
의정부
Uijeongbu
KORAIL

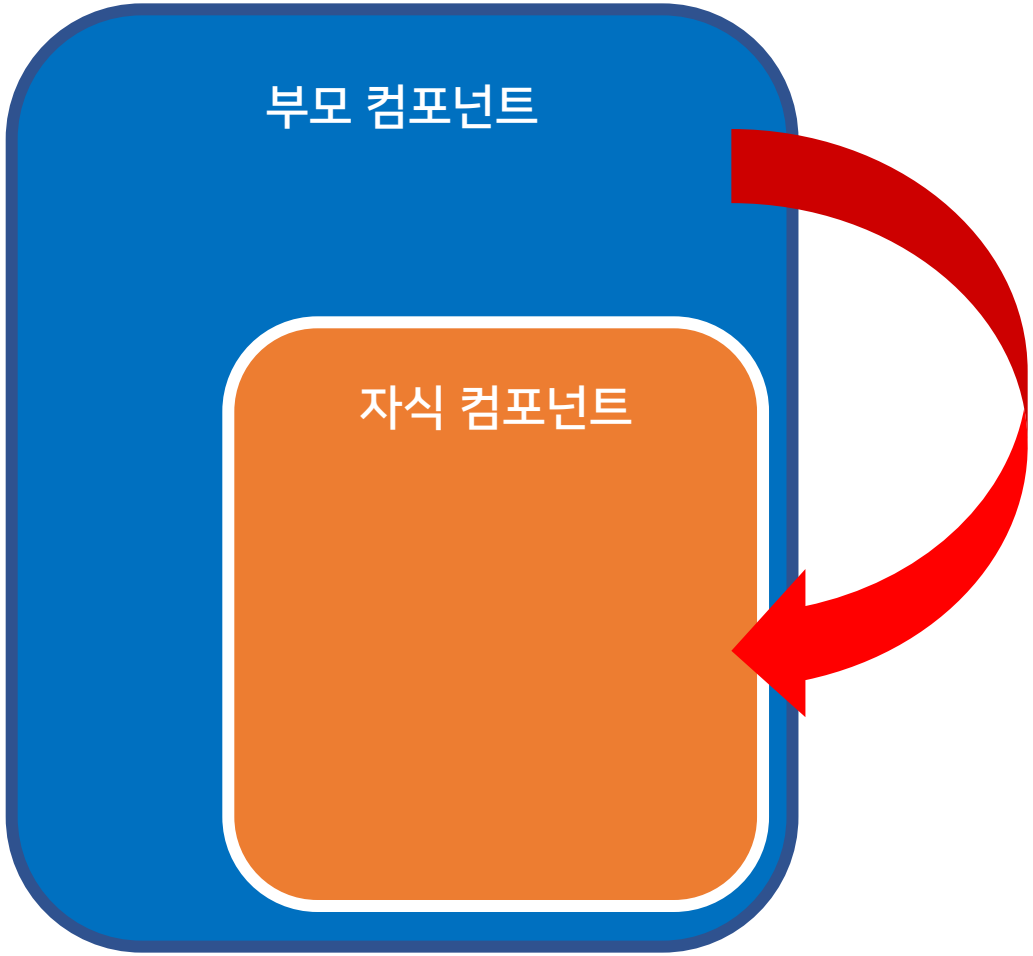
Props

리액트는 이런 걸 다 생각했죠!

- 그래서 Props 라는 것을 준비 했습니다~!
- EJS 파일에서 백엔드 데이터를 받아서, 그것을 통해 HTML 을 그려주던 방법이 생각나시나요!?
- 리액트에서는 props(properties) 라는 것으로 손쉽게 전달이 가능합니다!

Props?

- 부모 컴포넌트에서 자식 컴포넌트에 원하는 데이터를 보내는 방법입니다!



- props 라는 이름의 객체로 전달

간단한 MainHeader 라는 컴포넌트 만들기

- MainHeader 의 매개변수 전달 부분에 {} 를 추가하고 부모로 부터 받아올 props 명을 적어 주시면 됩니다!

```
import React from "react";

function MainHeader({ text }) {
    return (
        <h1>{text}</h1>
    )
}

export default MainHeader;
```

App.js 에서 props 사용하기

- App.js 의 컴포넌트 옆에 html 태그에 속성을 부여하는 것 처럼 props 명과 전달 하고자하는 데이터를 적어서 전달 하면 됩니다!

```
function App() {
  return (
    <div className="App">
      <MainHeader text="Hello, props world!" />
    </div>
  );
}

export default App;
```

src/App.js

```jsx
function App() {
  return (
    <div className="App">
      <MainHeader text="Hello, props world!" />
      <MainHeader text="Bye Bye, props world!" />
      <MainHeader text="Well come back, props world!" />
    </div>
  );
}

export default App;
```

src/App.js

Hello, props world!

Bye Bye, props world!

Well come back, props world!

다양한 데이터 받아오기!

- Props 는 다양한 데이터를 한꺼번에 받아 올 수 있습니다!
- 그리고 다양한 데이터는 props 라는 객체 하나로 받아서 사용이 가능합니다!

```
function App() {
  return (
    <div className="App">
      <MainHeader text="Go naver" href="https://naver.com" userID="tetz!" />
    </div>
  );
}

export default App;
```

src/App.js

```jsx
function MainHeader(props) {
    return (
        <div>
            <h1>{props.userID} 님 반갑습니다.</h1>
            <a href={props.href}>{props.text}</a>
        </div>

    )
}
```

src/components/MainHeader.js

- 물론 구조 분해 할당 문법도 사용이 가능합니다!

```
function MainHeader({ userID, href, text }) {
  return (
    <div>
      <h1>{userID} 님 반값습니다.</h1>
      <a href={href}>{text}</a>
    </div>
  );
}
```

src/components/MainHeader.js

```
function MainHeader(props) {
    const { userID, text, href } = props;
    return (
        <div>
            <h1>{userID} 님 반값습니다.</h1>
            <a href={href}>{text}</a>
        </div>

    )
}

export default MainHeader;
```

src/components/MainHeader.js

클래스형 컴포넌트

Props

클래스형 컴포넌트에서의 props

- 클래스형 컴포넌트인 만큼 기존의 props 로 접근하던 것을 this.props 로 접근 하시면 됩니다!
- 클래스이기 때문에 매개변수를 따로 받지 않습니다!
- This.props 를 쓰면 코드가 길어져서 보통 구조 분해 할당으로 변수로 만들어 사용합니다

```jsx
import React, { Component } from "react";

class ClassProps extends Component {
    render() {
        const { name, age, nickName } = this.props;
        return (
            <div>
                <h1>이름: {name}</h1>
                <h2>나이: {age}</h2>
                <h2>별명: {nickName}</h2>
            </div>
        )
    }
}

export default ClassProps;
```

src/components/ClassProps.js

```jsx
function App() {
  return (
    <div className="App">
      <ClassProps name="뽀로로" age="5" nickName="사고뭉치" />
    </div>
  );
}

export default App;
```

src/App.js

이름: 뽀로로

나이: 5

별명: 사고뭉치

실습, 컴포넌트와 props 활용하기!

```
import Modal from "./Modal";

function List() {
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <hr />
      <h2>리액트 공부하기</h2>
      <p>state 사용법 익히기</p>
      <hr />
      <h2>코테 문제 풀기</h2>
      <p>Lv 0 정복 가즈아</p>
      <hr />
    </div>
  );
}
export default List;
```

- 해당 코드를 컴포넌트와 Props 를 사용해서 수정하기!
- <h2>, <p>, <hr> 태그로 구성 된 오늘 할일 목록을 ListChild 라는 컴포넌트로 만들어 주세요~!
- 내용은 Props 로 전달하여 오른쪽 코드의 결과물과 같은 결과물이 나올 수 있도록 만들어 주세요~!

리액트에서

배열 활용하기!

리액트에서 배열 활용하기!

- 이제 컴포넌트와 Props 를 사용하는 방법을 배웠습니다!
- 그런데 props 로 전달하는 부분을 매번 저희가 작성을 해줘야 할까요?
- 아닙니다. 보통은 백엔드에서 데이터를 받아오게 되고 받은 데이터를 props 에 전달하는 형태를 많이 사용합니다~!

이런 데이터를 받았다고 가정을 해봅시다~!

```
const dataArr = [
  {
    title: "리액트 공부하기",
    detail: "state 사용법 익히기",
  },
  {
    title: "코테 문제 풀기",
    detail: "Lv0 정복 가즈아",
  },
];
```

- 데이터를 보니 뭔가 딱, 아까 ListChild 에 props로 전달하면 딱이다 싶으시죠?

```jsx
import ListChild from "./ListChild";

function List() {
  const dataArr = [
    {
      title: "리액트 공부하기",
      detail: "state 사용법 익히기",
    },
    {
      title: "코테 문제 풀기",
      detail: "Lv 0 정복 가즈아",
    },
  ];

  return (
    <div>
      <h1>오늘 해야할일</h1>
      <hr />
      <ListChild title={dataArr[0].title} detail={dataArr[0].detail} />
      <ListChild title={dataArr[1].title} detail={dataArr[1].detail} />
    </div>
  );
}
export default List;
```

크흠...

- 저렇게 코드를 쓰면 편할까요?? 당연히 불편하겠죠!?
- 그리고 데이터 배열의 길이가 변화한다면?
- 이래저래 안 좋은 코드가 될 겁니다!
- 그런데 지금 우리가 쓰고 있는건!? R.E.A.C.T 입니다!
- 즉, JSX 를 통해 HTML 과 JS 를 혼용해서 사용이 가능하죠!

배열 데이터를 리액트로 그리는 방법!

- React 에서는 배열 데이터를 그려 줄 때 배열 함수인 map 을 사용합니다!

```
return (
  <div>
    <h1>오늘 해야할일</h1>
    <hr />
    {dataArr.map((el) => {
      return <ListChild title={el.title} detail={el.detail} />;
    })}
  </div>
);
```

오늘 해야할일

리액트 공부하기

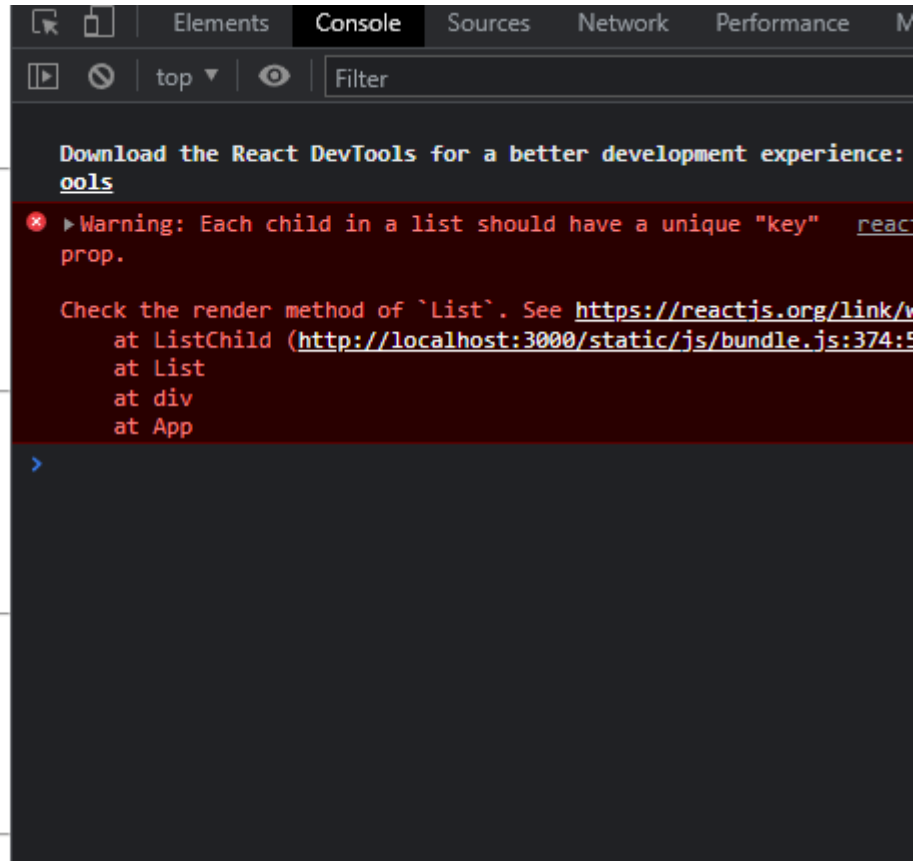
state 사용법 익히기

코테 문제 풀기

Lv 0 정복 가즈아

리액트 정복하기

리액트로 취뽀 가즈아



그런데 Warning 이!?

- React 에서 map 으로 배열 데이터를 그려줄 때에는 해당 요소에 Unique 한 Key 값 props 로 부여해 줘야 합니다~!
- 그래야만 나중에 리액트에서 map 으로 그려진 엘리먼트 중에서 어떤 것을 컨트롤(변경, 추가, 삭제 등)을 할 지 구분 할 수 있기 때문입니다!
- 구분을 해야하기 때문에 반드시 unique 한 값으로 넣어줘야 합니다!

```jsx
import ListChild from "./ListChild";
function List() {
  const dataArr = [
    {
      title: "리액트 공부하기",
      detail: "state 사용법 익히기",
    },
    {
      title: "코테 문제 풀기",
      detail: "Lv 0 정복 가즈아",
    },
  ];
  return (
    <div>
      <h1>오늘 해야할일</h1>
      <hr />
      {dataArr.map((el, index) => {
        return <ListChild title={el.title} detail={el.detail} key={index} />;
      })}
    </div>
  );
}
export default List;
```

구별이 가능한 unique 한 key 값을

props 로 부여하기!

단, index 는 최후의 수단으로 사용!

Map 을 쓰면~!

- 귀찮게 하나하나 배열의 데이터를 넣어줄 필요도 없습니다!
- 그리고 배열 데이터가 변경이 되어도 자연스럽게 적용이 됩니다!
- 아 그리고, 당연히 컴포넌트에만 적용할 필요도 없습니다!
- Map 의 리턴 값으로 원하는 것을 넣어줘도 됩니다!
    - 단, 리턴되는 요소는 반드시 하나의 부모 요소를 가져야만 합니다!

```jsx
return (
    <div>
      <h1>오늘 해야할일</h1>
      <hr />
      {dataArr.map((el, index) => {
        return (
          <div key={index}>
            <h2>{el.title}</h2>
            <p>{el.detail}</p>
            <hr />
          </div>
        );
      })}
    </div>
  );
```

실습, 배열을 map 으로 그리기!

- 왼쪽과 같은 데이터를 map 을 이용해서 다음과 같은 화면으로 출력하기!
- 컴포넌트와 Props 사용 / HTML 코드를 리턴 둘 다 만들어 주세요~!

```
const items = [
    {
      item: "PS5",
      price: "685,000원",
    },
    {
      item: "에어 프라이어",
      price: "50,000원",
    },
    {
      item: "사세 치킨윙",
      price: "11,500원",
    },
  ];
```

품목명 : PS5

가격 : 685,000원

품목명 : 에어 프라이어

가격 : 50,000원

품목명 : 사세 치킨윙

가격 : 11,500원

Props

활용하기

배열을 전달하고 props 로 받아서 처리!

- Props 로는 배열 같은 다양한 자료형도 전달이 가능합니다!
- 배열을 받아서 처리하는 CustomList.js 컴포넌트를 만들어 봅시다!

```
function CustomList(props) {
    return (
        <ul>
            {props.arr.map((el) => {
                return <li>{el}</li>
            })}
        </ul>
    )
}

export default CustomList;
```

src/components/CustomList.js

App.js 에서 배열을 전달

- App 에서 임의의 배열을 만들어서 전달하기!

```
function App() {
  const nameArr = ['뽀로로', '루피', '크롱이'];
  return (
    <div className="App">
      <CustomList arr={nameArr} />
    </div>
  );
}

export default App;
```

src/App.js

뽀로로
루피
크롱이

객체를 전달하고 props 로 받아서 처리!

- 객체을 받아서 처리하는 CustomObj.js 컴포넌트를 만들어 봅시다!

```
function CustomObj(props) {
    const { name, age, nickName } = props.obj;
    return (
        <div>
            <h1>이름 : {name}</h1>
            <h2>나이 : {age}</h2>
            <h3>별명 : {nickName}</h3>
        </div>
    )
}

export default CustomObj;
```

src/components/CustomObj.js

App.js 에서 객체를 전달

- App 에서 임의의 객체를 만들어서 전달하기!

```
function App() {
  const pororoObj = {
    name: "뽀로로",
    age: "5",
    nickName: "사고뭉치"
  }
  return (
    <div className="App">
      <CustomObj obj={pororoObj} />
    </div>
  );
}
```

src/App.js

이름 : 뽀로로

나이 : 5

별명 : 사고뭉치

실습, props 와 state 활용하기!

- 프로필 변경하기 버튼을 클릭하면 props 로 전달 된, 오브젝트 배열의 값이 순서대로 변하는 ChangeObj.js 컴포넌트를 만들어 주세요!
- 전달되는 오브젝트의 배열은 다음 장의 코드를 참고해 주세요!
- 함수형으로 구현해 주세요!

```jsx
import ChangeObj from "./ChangeObj";

function Ex5() {
  const pororoObjArr = [
    {
      name: "뽀로로",
      age: "5",
      nickName: "사고뭉치",
    },
    {
      name: "루피",
      age: "4",
      nickName: "공주님",
    },
    {
      name: "크롱이",
      age: "5",
      nickName: "장난꾸러기",
    },
  ];
  return (
    <div className="App">
      <ChangeObj objArr={pororoObjArr} />
    </div>
  );
}
export default Ex5;
```

useRef

useRef? 이 친구는 또 뭐죠?

- 무언가 컴포넌트가 리렌더링 되어도 값을 유지하고 싶을 때가 있습니다!
- 어? 그럼 state 쓰면 되는거 아닌가요?
- 그런데 값이 변경 되어도 컴포넌트 리렌더링을 하고 싶지는 않아요!
- 변수를 쓰면? → 리렌더링 시 변수 값은 초기화가 됩니다!
- 이럴 때 쓰는 것이 useRef 입니다!

useState

useRef

Variable

3가지 타입에 대해서 정리해 봅시다!

- 지금까지 배운 useState 의 state 그리고 useRef 의 ref 와 리액트 내부의 변수가 렌더링에 따라 어떤 식으로 변화하는지 알아 봅시다!
- 그리고 state 에 변화에 따른 컴포넌트 리렌더링의 개념도 다시 한번 잡아 봅시다!

State

변경

Ref

변경

Variable

변경

코드로 봅시다!

- 세 가지 버튼으로 각각 state / ref / variable 값을 올리는 컴포넌트를 구성해 봅시다!
- 그리고 값의 변화 없이 컴포넌트 리렌더링을 위해서 하나의 버튼을 더 만들어 봅시다
- 그리고 각각의 버튼을 클릭하면서 해당 값의 변화에 대해 관찰해 봅시다!

```jsx
import { useRef, useState } from "react";

const Comparing = () => {
    const [countState, setState] = useState(0);
    const [render, setRender] = useState(0);
    const countRef = useRef(0);
    let countVar = 0;

    const countUpState = () => {
        setState(countState + 1);
        console.log('State: ', countState);
    }

    const countUpRef = () => {
        countRef.current = countRef.current + 1;
        console.log('Ref: ', countRef.current);
    }

    const countUpVar = () => {
        countVar = countVar + 1;
        console.log('Variable: ', countVar);
    }

    const reRender = () => {
        setRender(render + 1);
    }
```

```jsx
return (
  <>
    <h1>State: {countState}</h1>
    <h1>Ref: {countRef.current}</h1>
    <h1>Variable: {countVar}</h1>
    <button onClick={countUpState}>State UP!</button>
    <button onClick={countUpRef}>Ref UP!</button>
    <button onClick={countUpVar}>Variable UP!</button>
    <button onClick={reRender}>렌더링!</button>
  </>
 )
}

export default Comparing;
```

src/components/Comparing.js

```jsx
import { useRef, useState } from "react";

const Comparing = () => {
    const [countState, setState] = useState(0);
    const [render, setRender] = useState(0);
    const countRef = useRef(0);
    let countVar = 0;

    const countUpState = () => {
        setState(countState + 1);
        console.log('State: ', countState);
    }

    const countUpRef = () => {
        countRef.current = countRef.current + 1;
        console.log('Ref: ', countRef.current);
    }

    const countUpVar = () => {
        countVar = countVar + 1;
        console.log('Variable: ', countVar);
    }

    const reRender = () => {
        setRender(render + 1);
    }

    return (
        <>
            <h1>State: {countState}</h1>
            <h1>Ref: {countRef.current}</h1>
            <h1>Variable: {countVar}</h1>
            <button onClick={countUpState}>State UP!</button>
            <button onClick={countUpRef}>Ref UP!</button>
            <button onClick={countUpVar}>Variable UP!</button>
            <button onClick={reRender}>렌더링!</button>
        </>
    )
}

export default Comparing;
```

State: 4

Ref: 14

Variable: 0

State UP! | Ref UP! | Variable UP! | 렌더링!

src/components/Comparing.js

useRef

useRef? 이 친구는 또 뭐죠?

- 자 Input 태그를 사용해서 무언가를 입력 받는 상황을 그려 봅시다!
- 만약 이럴 때, useState 를 사용한다면?
- 인풋 태그 자체가 계속 리렌더링이 되어서 문제 + 낭비가 생길 겁니다!
- 이럴 때 쓰라고 있는 것이 useRef 되겠습니다!

useRef

- useRef 를 사용하면 참조하고자 하는 DOM 요소에 ref 속성을 주고 해당 태그의 변화를 감지하거나 DOM 요소를 컨트롤 할 수 있습니다
- 보통은 컴포넌트에 존재하는 인풋 태그의 값을 받거나, JS에서 DOM 요소를 관리하던 역할을 해줍니다!

JS 방식으로 구현하기

- Components 폴더에 TestRef.js 컴포넌트를 작성해 주세요!
- 인풋에 입력 된 값이, 상단에 있는 h1 태그의 컨텐츠가 되도록 구성하여 봅시다!
- 일단은, 기존의 JS 처럼 onChange 속성을 사용해서 이벤트 객체를 이용하여 구현하여 봅시다!

```jsx
import TestRef from './components/TestRef';

function App() {
  return (
    <div className="App">
      <TestRef />
    </div>
  );
}

export default App;
```

src/App.js

```jsx
import { useState, useRef } from "react";

export default function TestRef() {
    const [text, setText] = useState("안녕하세요!");

    const onChangeText = (e) => {
        const inputText = e.target.value;
        setText(inputText);
    }

    return (
        <div>
            <h1>{text}</h1>
            <input onChange={(e) => { onChangeText(e) }}></input>
        </div>
    )
}
```

dasdadadada

dasdadadada

src/components/TestRef.js

useRef 로 구현하기

- useRef 를 사용하여 변화를 감지하거나 DOM 요소를 컨트롤 하고 싶은 태그에 ref 속성을 부여하기!

```jsx
import { useState, useRef } from "react";

export default function TestRef() {
    const [text, setText] = useState("안녕하세요!");

    const inputValue = useRef();

    const onChangeText = () => {
        setText(inputValue.current.value);
    }

    return (
        <div>
            <h1>{text}</h1>
            <input ref={inputValue} onChange={onChangeText}></input>
        </div>
    )
}
```

src/components/TestRef.js

useRef 값 확인하기

- useRef 로 설정한 값을 console.log 에 찍어 봅시다!

```
const onChangeText = () => {
        console.log(inputValue);
        setText(inputValue.current.value);
    }
```

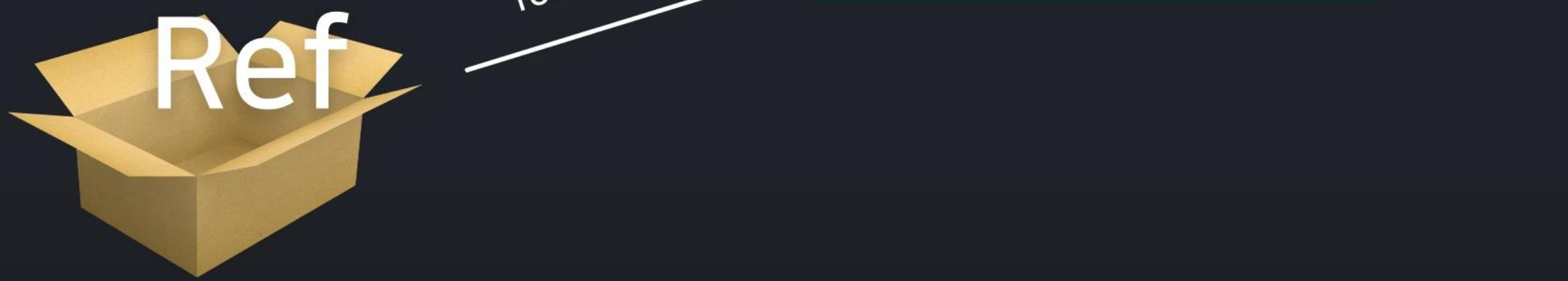
```
▼ {current: input} i
  ▼ current: input
      value: "dsadad"
    ▶ __reactEvents$3ew9d0gbjvx: Set(1) {'invalid__bubble'}
    ▶ __reactFiber$3ew9d0gbjvx: FiberNode {tag: 5, key: null, el
    ▶ __reactProps$3ew9d0gbjvx: {onChange: f}
    ▶ _valueTracker: {getValue: f, setValue: f, stopTracking: f}
    ▶ _wrapperState: {initialChecked: undefined, initialValue: '
```

useRef 로

포커스 이동 시키기!

useRef 로 포커스 이동!

- useRef 가 자주 사용되는 포커스 사용 방법을 알아 봅시다!
- 컴포넌트 폴더에 ChangeFocus.js 컴포넌트를 만들어 봅시다!
- 2개의 인풋 태그를 만들고 해당 인풋에 ref 속성을 부여!
- useRef 로 각각 인풋의 속성 값을 변수에 담고 해당 변수를 통해 input 태그에 포커스를 부여해 봅시다!
- 해당 값에 대한 접근은 current 객체를 통해 해야합니다!


Ref
focus ()
hello123
로그인

```jsx
import { useState, useRef } from "react";

export default function TestRef() {
    const input1 = useRef();
    const input2 = useRef();

    const changeFocusOne = () => {
        input1.current.focus();
    }

    const changeFocusTwo = () => {
        input2.current.focus();
    }

    return (
        <div>
            <input ref={input1}></input>
            <input ref={input2}></input>
            <br></br>
            <button onClick={changeFocusOne}>1</button>
            <button onClick={changeFocusTwo}>2</button>
        </div>
    )
}
```

src/components/ChangeFocus.js

useRef 로

DOM 컨트롤

```
const ref = useRef(value)
```

```
<input ref={ ref } />
```

useRef 를 사용하여 DOM 컨틀롤!

- useRef 로 선언한 변수를 DOM 요소에 ref 속성으로 부여하면 해당 요소에 접근할 수 있습니다
- 마치 이전 VanilaJS 의 querySelector 또는 getElementById 같은 역할을 손쉽게 구현할 수 있죠! → 2개를 대체해서 사용이 가능합니다!
- 그럼 ref 로 접근한 DOM 요소를 컨트롤 해봅시다!

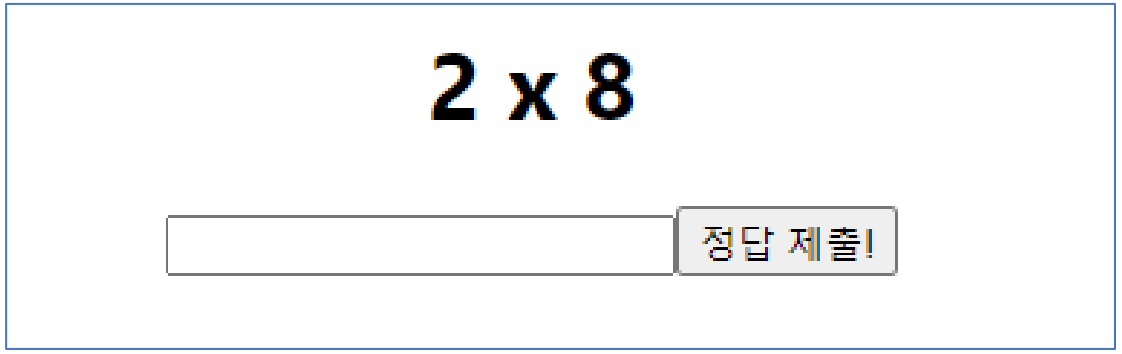
useRef 를 사용하여 DOM 컨틀롤!

- H1 태그 2개를 사용하고 useRef 로 DOM 요소에 접근하여 해당 DOM 요소의 스타일을 변경해 보겠습니다!
- 인풋 태그를 초기화도 해보죠!

```jsx
import React, { useRef } from "react";

export default function RefDOM() {
  const orangeEl = useRef();
  const skyblueEl = useRef();
  const inputEl = useRef();

  const adjustCSS = () => {
    orangeEl.current.style.backgroundColor = "orange";
    skyblueEl.current.style.backgroundColor = "skyblue";
  };

  const clearInput = () => {
    inputEl.current.value = "";
  };

  return (
    <div>
      <h1 ref={orangeEl}>Orange</h1>
      <h1 ref={skyblueEl}>Skyblue</h1>
      <input type="text" ref={inputEl} />
      <br />
      <button onClick={adjustCSS}>CSS 적용</button>
      <button onClick={clearInput}>인풋 초기화</button>
    </div>
  );
}
```

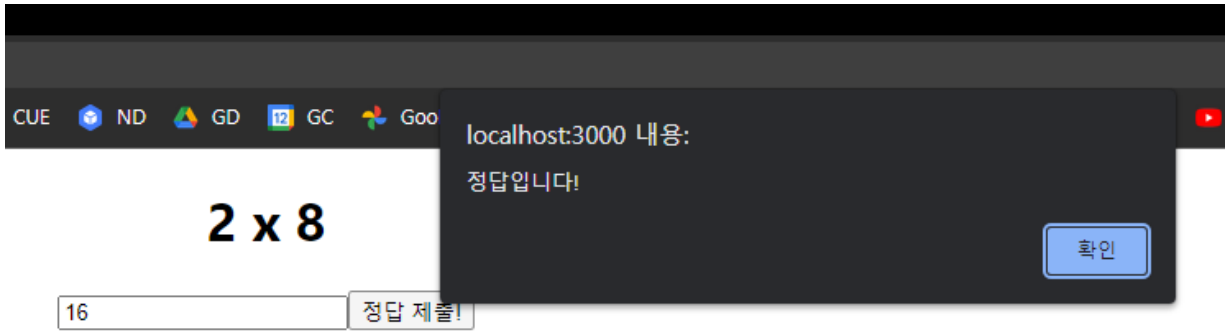
실습, useRef 활용하기!

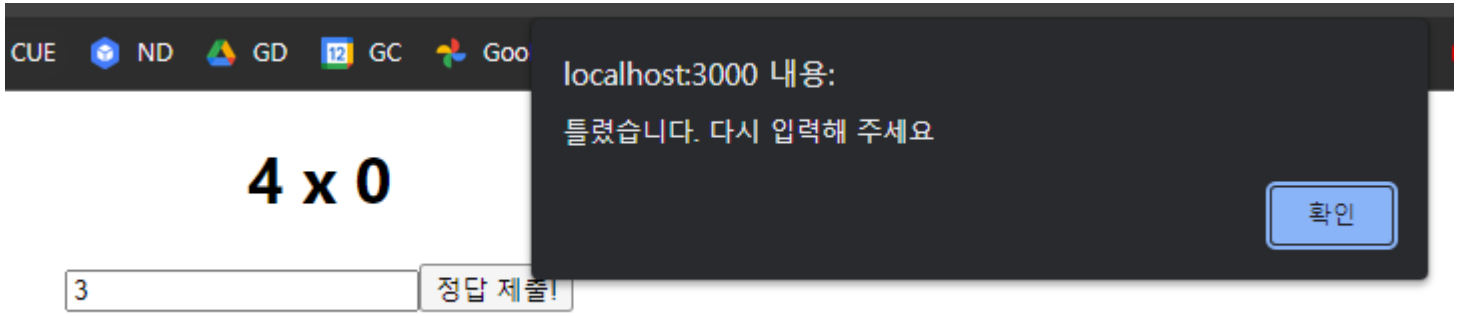
- 숫자 퀴즈 프로그램을 만들어 봅시다!
- 컴포넌트가 랜더링이 되면 0 ~ 9 사이의 수를 랜덤으로 2개 생성합니다.
- 그리고 랜덤으로 더하기, 빼기, 곱하기 중에서 어떤 연산을 할지 정하여 퀴즈를 출제합니다.



실습, useRef 활용하기!

- useRef 를 활용하여 input 태그의 값을 입력 받아서 퀴즈의 정답이 맞을 경우, 틀릴 경우 각각 alert 을 띄워 줍니다





실습, useRef 활용하기!

- 정답을 입력한 경우 input 태그의 내용을 비워주고, input 태그로 포커스를 이동 시킨 뒤 컴포넌트를 리랜더링 하여 새로운 퀴즈를 출제합니다
- 오답을 입력한 경우 input 태그의 내용을 비워주고, input 태그로 포커스를 이동 시켜 줍니다
- 나누기는 소수점 계산이 필요하여 제외 합니다
- 답은 음수가 될 수 있습니다!

React.Fragment

React.Fragment?

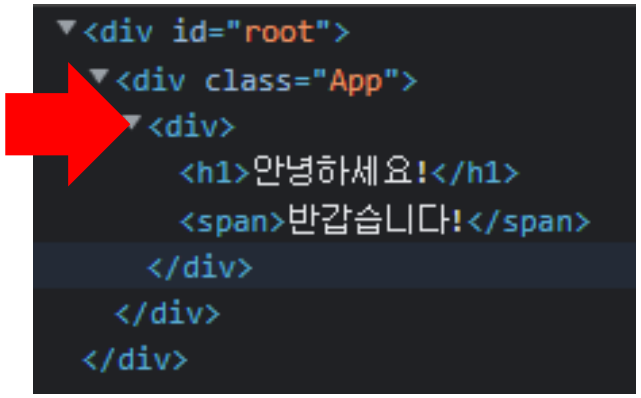
- 자, 정말 간단한 컴포넌트를 만들어 볼게요!

```
export default function ReactFragment() {
  return (
    <div>
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </div>
  );
}
```

src/components/ReactFragment.js

React.Fragment?

- 그리고 페이지에서 개발자 도구를 열어 보면!



- 간단한 컴포넌트임에도 div 요소가 하나 추가 되었네요!
- 그럼, 저 div 를 없앨수 없을까요?

React.Fragment?

- 리턴 값에서 최상위 태그 역할을 하는 DIV를 빼보시죠!

```jsx
export default function ReactFragment() {
  return (
    <h1>안녕하세요!</h1>
    <span>반갑습니다!</span>
  );
}
```

```
ERROR in [eslint]
src\components\ReactFragment.js
  Line 4:6:  Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (4:6)
```

- 바로 에러가 뜹니다! 그런데 React 가 JSX fragment 를 추천하네요!?

React.Fragment!

- 실제 리액트에서 컴포넌트를 조합할 때, 최상위에 div 를 사용하지 않고 반환해야만 하는 경우가 생기게 됩니다!
- CSS 가 깨진다거나, 테이블 요소 사이에 div 요소가 들어가면 에러가 뜨기 때문이죠!
- 그럴 때 쓰는 것이 바로 React.Fragment 입니다!

React.Fragment!

- 이제 컴포넌트를 React.Fragment 로 감싸 봅시다!
- 이건 React 라이브러리의 기능이므로 React 라이브러리 추가 필요

```jsx
import React from "react";

export default function ReactFragment() {
  return (
    <React.Fragment>
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </React.Fragment>
  );
}
```

src/components/ReactFragment.js

React.Fragment!

- 이제는 div 가 사라졌네요!?

```
▼<div id="root">
  ▼<div class="App">
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </div>
  </div>
```

- 바로 이러한 역할을 해주는 것이 React.Fragment 입니다!

<> </>

- 개발자들은 축약의 민족이기 때문이 이렇게 긴 코드를 용납 못합니다!
- <React.Fragment> 는 <> 로 대체가 가능합니다! :)

```jsx
import React from "react";

export default function ReactFragment() {
  return (
    <>
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </>
  );
}
```

src/components/ReactFragment.js

React.Fragment

가 필요할 때!

CSS 님이 화가 나셨어!

- 먼저 App.js 에 가서 ReactFragment 컴포넌트와 동일한 코드를 작성해 봅시다

```
function App() {
  return (
    <div className="App">
      <h1>안녕하세요</h1>
      <span>반갑습니다</span>
    </div>
  );
}

export default App;
```

src/App.js

CSS 님이 화가 나셨어!

- 그리고 App.css 에 아래의 코드도 추가해 봅시다!

```css
.App {
  display: flex;
  justify-content: space-between;
}

.App h1 {
  background-color: skyblue;
}

.App span {
  background-color: orange;
}
```

src/App.css

결과물 화면~!

안녕하세요

반갑습니다

- 이런 결과물 화면이 나와야 정상입니다!

ReactFragment 컴포넌트를 적용!

- ReactFragment 컴포넌트의 최상위 요소를 div 로 변경해서 적용시켜 봅시다!

```
import React from "react";

export default function ReactFragment() {
  return (
    <div>
      <h1>안녕하세요!</h1>
      <span>반갑습니다!</span>
    </div>
  );
}
```

src/components/ReactFragment.js

ReactFragment 컴포넌트를 적용!

안녕하세요!

반갑습니다!

- Div 가 생기게 되어서 이전 결과물과는 완전히 다른 결과물이 나오게 됩니다!
- 따라서, 이런 일을 피하려면 React.Fragment 를 쓰셔야 합니다!

나는 너에게 속할 수 없어!

- 테이블 요소에 테이블 내용을 컴포넌트로 삽입하는 경우를 생각해 봅시다
  (사실 이게 쓸 일이 거의 없긴 합니다 ㅎㅎㅎ;;)
- 그런데 테이블 요소 안에는 div 태그가 들어가지 못합니다!
- 이럴때 문제가 생기는데 이것을 React.Fragment 로 해결이 가능합니다!

```jsx
import TableColumn from
"./components/TableColumn";

function App() {
  return (
    <div className="App">
      <table border="1">
        <tbody>
          <tr border="1">
            <td>1</td>
            <td>2</td>
            <td>3</td>
          </tr>
          <tr>
            <TableColumn />
          </tr>
        </tbody>
      </table>
    </div>
  );
}

export default App;
```

src/App.css

```jsx
import React from "react";

export default function ReactFragment() {
  return (
    <div>
      <td>a</td>
      <td>b</td>
      <td>c</td>
    </div>
  );
}
```

src/components/TableColumn.js

```
Download the React DevTools for a better development experience: https://reactjs
▸Warning: validateDOMNesting(...): <td> cannot appear as a child of <div>.
    at td
    at div
    at TableColumn
    at tr
    at tbody
    at table
    at div
    at App
▸Warning: validateDOMNesting(...): <div> cannot appear as a child of <tr>.
    at div
    at TableColumn
    at tr
    at tbody
    at table
    at div
    at App
>
```


수고하셨습니다!