



[스파르타코딩클럽] 프론트엔드의 꽃, 리액트 - 2주차



매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ab428adf-f801-41ee-9ede-61ea3ff54bbc/____2_.pdf

[수업 목표]

1. 컴포넌트의 state를 관리할 수 있다.
2. 컴포넌트의 라이프 사이클을 이해한다.
3. 오픈소스 패키지를 찾아서 설치할 수 있다.
4. event listener를 구독할 수 있다.
5. React hook 중 useState(), useEffect()를 사용할 수 있다.

[목차]

- 01. 화면을 예쁘게! - SCSS
- 02. 화면을 예쁘게! - styled-components
- 03. Quiz_버킷리스트에 styled-components 적용하기
- 04. 라이프 사이클이란?
- 05. 알아두면 좋은 라이프 사이클 함수
- 06. Ref! 리액트에서 돔요소를 가져오려면?
- 07. State 관리 (1)
- 08. State 관리 (2)
- 09. Quiz_버킷리스트에 아이템을 추가해보자!
- 10. Event Listener
- 11. 끝 & 숙제 설명
- 12. 2주차 숙제 답안 코드



모든 토글을 열고 닫는 단축키

Windows : **Ctrl** + **alt** + **t**

Mac : **⌘** + **⌘** + **t**

01. 화면을 예쁘게! - SCSS

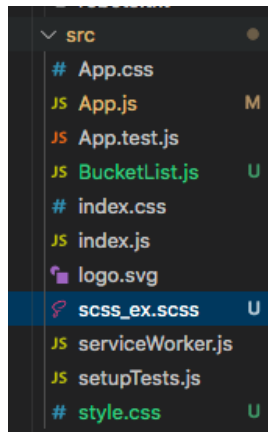
▼ 1) SASS, SCSS 개념

☞ SCSS. SASS 이름이 좀 비슷하죠? 둘 다 CSS를 편하게 쓰도록 도와주는 친구입니다! SCSS는 SASS의 3번째 버전에서 추가된 친구인데, SASS의 모든 기능을 쓸 수 있고 CSS와 호환도 잘 되는 멋진 친구죠. 😊

▼ 2) SCSS를 적용해보기

☞ 우리가 만든 버킷리스트에 SCSS를 적용해봅시다!

- 먼저 파일을 만들어요!



- style.css 내용을 scss_ex.scss 파일에 붙여 넣고,
- App.js에서 만든 파일을 불러옵니다.

```
// 일단 스타일 파일은 주석처리하고 그 아래에서 불러올거예요!  
// import "../style.css";  
import "../scss_ex.scss";
```

- 그리고 새로그침을 하면... 아앗! 에러가!

Failed to compile

```
./src/scss_ex.scss (./node_modules/css-loader/dist/cjs.js??ref--6-oneOf-5-1!./node_modules/postcss-loader/src??
postcss!./node_modules/resolve-url-loader??ref--6-oneOf-5-3!./node_modules/sass-loader/dist/cjs.js??ref--6-oneOf-5-4!./src/scss_ex.scss)
To import Sass files, you first need to install node-sass.
Run `npm install node-sass` or `yarn add node-sass` inside your workspace.
Require stack:
- /Users/nodong/sparta/sparta_/week_2/node_modules/sass-loader/dist/getDefaultSassImplementation.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/sass-loader/dist/getSassImplementation.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/sass-loader/dist/index.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/sass-loader/dist/cjs.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/loader-runner/lib/loadLoader.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/loader-runner/lib/LoaderRunner.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/webpack/lib/NormalModule.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/webpack/lib/NormalModuleFactory.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/webpack/lib/Compiler.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/webpack/lib/webpack.js
- /Users/nodong/sparta/sparta_/week_2/node_modules/react-scripts/scripts/start.js
```

→ 에러메시지를 읽어보세요! node-sass라는 친구가 없으니 설치해주라고 하네요!

- node-sass 패키지와 다른 필요 패키지 설치하기

▼ [코드스니펫] - 패키지 설치

```
# 오류 메시지를 보면서 하나씩 설치하면 너무 느리니, 이번엔 제가 필요 패키지를 썩 가져왔어요!
# 그대로 복사해주세요!
yarn add node-sass@4.14.1 open-color sass-loader classnames
```

- 패키지를 설치한 후에는 컨트롤 + C를 눌러, 종료하고 다시 yarn start로 실행해줍니다!
다시 실행하면 내 버킷리스트가 잘 나오죠! 😊

▼ 3) SCSS 써보기

- SCSS의 주요 문법

👉 사실, 전부 외울 필요는 없어요! 그냥 이렇게 쓰는구나, 맛만 봐도 좋습니다. 우리가 CSS 쓸 때 모든 속성 값을 외우진 않는 것처럼 자주 쓰는 몇 가지만 알면 나머지는 그때그때 찾아보면 됩니다.
공식문서에서요!

👉 테스트 사이트를 통해 간단히 살펴볼 수도 있습니다 😊

- 기본 문법은 CSS와 동일하다. (글씨 색 바꿀 때는? `div { color: #fff; }`)
- Nesting이 가능하다.
 - div 아래에 p, img 태그 스타일을 줄 때, 각각 다른 블록을 만들어 쓸 필요가 없어요!
 - 축약형으로 묶을 수 있어요. → xxx-yyy 식일 때, 앞에 xxx가 같은 친구끼리 묶어 쓸 수 있어요!

```
//scss
div {
  p {
    color: #888888;
    font: {
      family: sans-serif;
      size: 14px;
    }
  }
}
```

```
img {
  width: 400px;
}
```

- 상위 요소 이어쓰기는 "&"로! 클래스명 등, 글자도 이어쓸 수 있어요.

```
//scss
div {
  background-color: green
  &:hover { background-color: blue }
}

.div {
  background-color: green
  &_blue { background-color: blue }
}
```

- 문자열을 치환할 수 있습니다! (즉, 변수를 쓸 수 있어요!)

```
//scss
$defaultSize: 20px;
$className: blue;

p{
  font-size: #{$defaultSize};
  &.#{$className} {color: #{$className}}
}
```



조금 헛갈리시나요? 괜찮아요! 몇 번 하다보면 금방 익숙해집니다! 😊

02. 화면을 예쁘게! - styled-components

▼ 4) styled-components 설치하기

- 패키지 설치하기

```
yarn add styled-components
```

▼ 5) styled-components란?



컴포넌트 스타일링 기법 중, 제가 가장 좋아하는 방식입니다! 😎

왜 좋아하나구요? 많은 이유가 있지만, 두 개만 꼽아볼게요.

- class 이름 짓기에서 해방됩니다!
- 컴포넌트에 스타일을 적기 때문에, 간단하고 직관적입니다!

- CSS-in-JS 라이브러리 중 하나입니다!
컴포넌트에 스타일을 직접 입히는 방식이라고 편하게 생각하셔도 됩니다!

▼ 6) styled-components를 적용해보기

```
import React from 'react';
// 패키지에서 styled를 불러올게요!
import styled from 'styled-components';


function App() {
  return (
    <div className="App">
      { /* props로 bgColor를 줘볼까요! */ }
      <MyStyled bgColor를={"red"}>hello React!</MyStyled>
    </div>
  );
}

// scss처럼 자기 자신을 지칭할 때 &를 쓸 수 있습니다!
// props 주는 방법! 이제 알고 있죠?
const MyStyled = styled.div`
  width: 50vw;
  min-height: 150px;
  padding: 10px;
  border-radius: 15px;
  color: #fff;
  &:hover{
    background-color: #ddd;
  }
  background-color: ${props => (props.bgColor를 ? "red" : "purple")};
`;
export default App;
```




알면 덜 찜찜한 이야기



← 이렇게 생긴 따옴표( 키와 함께 있습니다!)를 **백틱**이라고 불러요.

03. Quiz_버킷리스트에 styled-components 적용하기

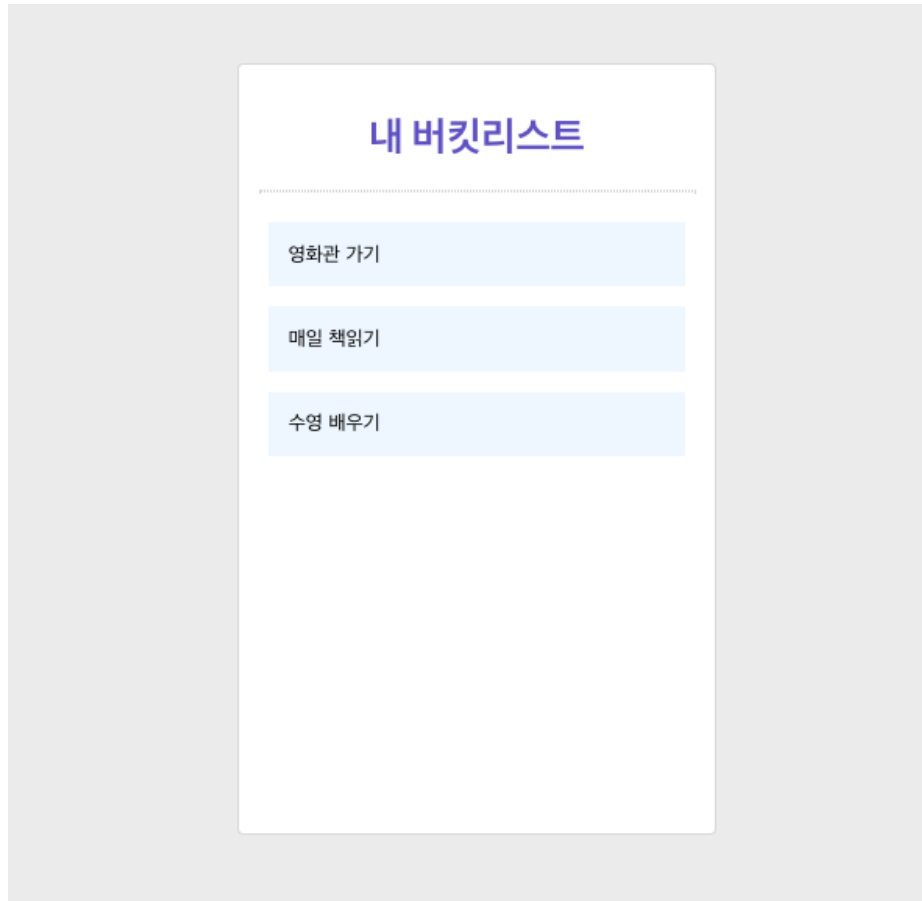
▼ 7)  버킷리스트에 styled-components를 적용해보기



버킷리스트에 입힌 스타일을 styled-components로 싹 바꿔봅시다!

▼ Q. 퀴즈설명

▼ 모습 보기



💡 힌트: SCSS, CSS와 styled-components의 백틱 내용은 동일하게 쓸 수 있다는 거! 잊지마세요.

▼ A. 함께하기(완성본)

💡 어때요, 할만했나요? 다만 조금씩 다른 방법으로 해결하셨더라도, 다음 강의 진행을 위해 아래 코드를 복사→붙여넣기 해주세요!

▼ [코드스니펫] - 02. App.js (←강의 화면 오른쪽 하단에 '복사' 클릭! 😊)

```
import React from "react";
import logo from "../logo.svg";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import styled from "styled-components";

// 클래스형 컴포넌트는 이렇게 생성합니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
  }
}
```

```

    }

    // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
    render() {
      return (
        <div className="App">
          <Container>
            <Title >내 버킷리스트</Title>
            <Line/>
            { /* 컴포넌트를 넣어줍니다. */ }
            { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
            <BucketList list={this.state.list} />
          </Container>
        </div>
      );
    }
  }
}

const Container = styled.div`
  max-width: 350px;
  min-height: 80vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;

export default App;

```

▼ [코드스니펫] - 03. BucketList.js (← 강의 화면 오른쪽 하단에 '복사' 클릭! 😊)

```

// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

const BucketList = (props) => {
  const my_lists = props.list;

  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle key={index}>
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;
  height: 100%;
  overflow-x: hidden;
  overflow-y: auto;
`;

```

```
const ItemStyle = styled.div`
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
`;

export default BucketList;
```

04. 라이프 사이클이란?

▼ 8) 가상돔이란?

👉 DOM을 아시나요?
DOM은 html 단위 하나하나를 객체로 생각하는 모델입니다. 예를 들면, 'div태그'라는 객체는 텍스트 노드, 자식 노드 등등, 하위의 어떤 값을 가지고 있겠죠? 이런 구조를 트리 구조라고 합니다. 네, 맞습니다! DOM이 트리구조란 소리입니다.

- DOM 트리 중 하나가 수정될 때마다 모든 DOM을 뒤지고, 수정할 걸 찾고, 싹 수정을 한다면?
→ 필요없는 연산이 너무 많이 일어난다!
→ 그래서 등장한 게 가상돔!
- 가상돔은 메모리 상에서 돌아가는 가짜 DOM입니다.
- 가상돔의 동작 방식: **기존 DOM과 어떤 행동 후 새로 그린 DOM(가상 돔에 올라갔다고 표현합니다)을 비교해서 정말 바뀐 부분만 갈아끼워줍니다!** → 돔 업데이트 처리가 정말 간결하죠!

👉 어렵지 않죠! 그럼 어떤 행동을 해야 DOM을 새로 그릴까요?
- 처음 페이지 진입했을 때도 그리겠죠!
- 데이터가 변했을 때도 그릴 겁니다!

💡 알면 덜 찜찜한 이야기: **DOM이 정말 그렇게 느려?**
반은 맞고 반은 틀려요. DOM은 사이트 구조에 따라 가상돔을 쓰는 것보다 훨씬 성능이 좋을 수 있고 (=빠를 수 있고), 속이 터지게 느릴 수 있습니다.

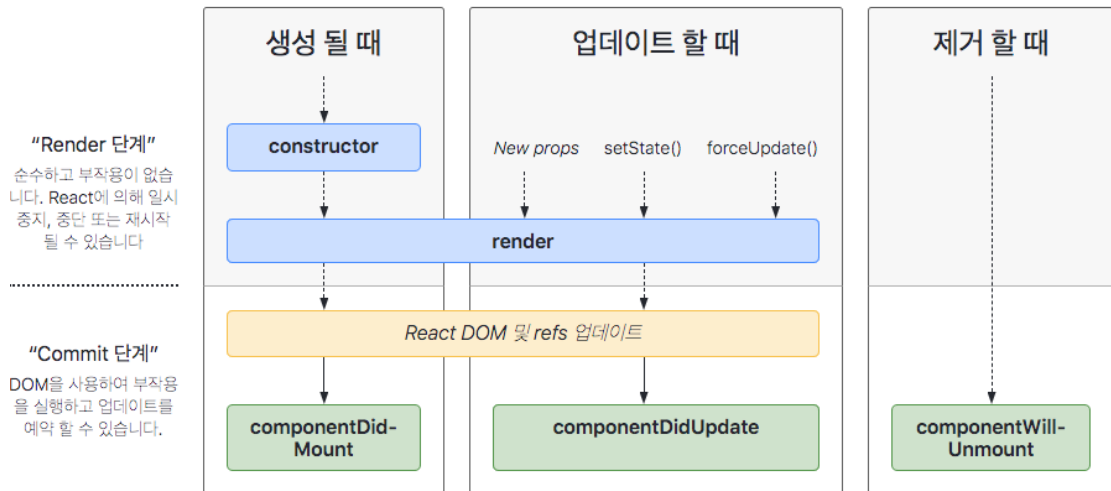
▼ 9) 라이프 사이클이란?

👉 컴포넌트의 라이프 사이클(= 컴포넌트 생명주기)은 정말 중요한 개념입니다!
컴포넌트가 렌더링을 준비하는 순간부터, 페이지에서 사라질 때까지가 라이프 사이클이에요.
공식 문서에서도 자세히 다루고 있어요.

- 아래 도표는 어떻게 라이프 사이클이 흘러가는 지 그린 도표입니다.
([도표 보러가기](#))

□ 덜 일반적인 라이프 사이클 표시

React 버전 언어



- 컴포넌트는 생성되고 → 수정(업데이트)되고 → 사라집니다.
- 생성은 처음으로 컴포넌트를 불러오는 단계입니다.
- 수정(업데이트)는 사용자의 행동(클릭, 데이터 입력 등)으로 데이터가 바뀌거나, 부모 컴포넌트가 렌더링 할 때 업데이트 됩니다. 아래의 경우죠!
 - props가 바뀔 때
 - state가 바뀔 때
 - 부모 컴포넌트가 업데이트 되었을 때(=리렌더링했을 때)
 - 또는, 강제로 업데이트 했을 경우! (forceUpdate()를 통해 강제로 컴포넌트를 업데이트할 수 있습니다.)
- 제거는 페이지를 이동하거나, 사용자의 행동(삭제 버튼 클릭 등)으로 인해 컴포넌트가 화면에서 사라지는 단계입니다.

05. 알아두면 좋은 라이프 사이클 함수



라이프 사이클 함수는 클래스형 컴포넌트에서만 사용할 수 있습니다.

라이프 사이클을 아는 건 중요한데 왜 우리는 클래스형 컴포넌트보다 함수형 컴포넌트를 많이 쓰냐구요?

리액트 공식 매뉴얼에서 **함수형 컴포넌트**를 더 권장하기 때문입니다!

(리액트 16.8버전부터 등장한 **React Hooks**으로 라이프 사이클 함수를 대체할 수 있거든요.)

더 많은 라이프 사이클 함수는 [공식 문서](#)에서 확인할 수 있어요 😊

▼ [코드스니펫] - 라이프 사이클 예제

- 이 코드를 붙여넣고 콘솔 창에 어떤 순서로 찍히는 지 확인해봅시다!
라이프 사이클 함수가 어떤 순서로 움직이는 지 보면 이해하기 쉬울 거예요.
LifecycleEx.js 파일을 만들고 → 이 코드를 붙여넣기!
그리고 **App.js**에서 **LifecycleEx** 파일을 불러오는 거 잊지마세요! 저는 새 프로젝트를 만들어서 해볼게요.

```

import React from "react";

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class LifecycleEx extends React.Component {

  // 생성자 함수
  constructor(props) {
    super(props);

    this.state = {
      cat_name: '나비',
    };

    console.log('in constructor!');
  }

  changeCatName = () => {
    // 다음 강의에서 배울, state 업데이트 하는 방법입니다!
    // 지금은 componentDidMount()를 보기 위해 쓰는 거니까, 처음보는 거라고 당황하지 말기!
    this.setState({cat_name: '바둑이'});

    console.log('고양이 이름을 바꾼다!');
  }

  componentDidMount(){
    console.log('in componentDidMount!');
  }

  componentDidUpdate(prevProps, prevState){
    console.log(prevProps, prevState);
    console.log('in componentDidUpdate!');
  }

  componentWillUnmount(){
    console.log('in componentWillUnmount!');
  }

  // 렌더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {

    console.log('in render!');

    return (
      <div>
        {/* render 안에서 컴포넌트의 데이터 state를 참조할 수 있습니다. */}
        <h1>제 고양이 이름은 {this.state.cat_name}입니다.</h1>
        <button onClick={this.changeCatName}>고양이 이름 바꾸기</button>
      </div>
    );
  }
}

export default LifecycleEx;

```

▼ 10) constructor()



생성자 함수라고도 부릅니다. 컴포넌트가 생성되면 가장 처음 호출되는 친구죠!

▼ 11) render()



컴포넌트의 모양을 정의하는 친구입니다!
여기에서도 state, props에 접근해서 데이터를 보여줄 수 있어요.

리액트 요소를 return에 넣어 반환해줬던 거 기억하시죠?
render() 안에 들어갈 내용은 컴포넌트의 모양에만 관여하는 것이 가장 좋습니다.
즉, state나, props를 건드려 데이터를 수정하려고 하면 안됩니다!

▼ 12) componentDidMount()



컴포넌트가 화면에 나타나는 것을 **마운트(Mount)**한다고 표현합니다. componentDidMount()는 마운트가 완료 되었다는 소리겠죠?
이 함수는 첫번째 렌더링을 마친 후에만 딱 한 번 실행됩니다. 컴포넌트가 리렌더링할 때는 실행되지 않아요.
보통은 이 안에서 ajax 요청, 이벤트 등록, 함수 호출 등 작업을 처리합니다.
또, 이미 가상돔이 실제돔으로 올라간 후니까 DOM 관련 처리를 해도 됩니다!

▼ 13) componentDidUpdate(prevProps, prevState, snapshot)



DidMount()가 첫 렌더링 후에 호출 되는 함수라면, DidUpdate()는 리렌더링을 완료한 후 실행되는 함수입니다.

이 함수에 중요한 파라미터가 2개 있는데, prevProps와 prevState입니다. 각각 업데이트 되기 전 props, state예요. 이전 데이터와 비교할 일이 있다면 가져다 쓰도록 합시다.
DidUpdate()가 실행될 때도 가상돔이 실제돔으로 올라간 후니까 DOM 관련 처리를 해도 됩니다!

▼ 14) componentWillUnmount()



컴포넌트가 DOM에서 제거 될 때 실행하는 함수입니다.
만약 우리가 스크롤 위치를 추적 중이거나, 어떤 이벤트 리스너를 등록했다면 여기에서 꼭꼭 해제를 해 줘야 합니다.
컴포넌트 없이 이벤트만 남겨둘 순 없잖아요!



알면 덜 스트레스 받는 리액트 상식:

어?! 왜 render도 constructor도 두번씩 찍히는거죠?!
src/index.js 파일을 열어보세요! 아마 <React.StrictMode>라는 녀석이 <App/>을 감싸고 있을거예요.
strictmode는 우리가 만든 애플리케이션이 문제는 없는지, 좀 더 간간히 검사해주는 모드예요. 우리 로컬 환경에서만 활성화되니, 걱정말기!
(보고 싶지 않다면 지우고 <App/>만 남겨두셔도 좋습니다.)



componentWillUnmount()가 호출되는 걸 보려면, app.js에서 <LifecycleEx/>를 없애봐야겠죠?
삼함연산자를 사용해서 컴포넌트를 보여주거나, 없애는 걸 **조건부 렌더링**이라고 불러요.
이건 저만 진행할게요! 눈으로만 따라와주گی!

06. Ref! 리액트에서 돔요소를 가져오려면?

▼ 15) React.createRef()



이제 가상돔이 뭔지, 돔이 뭔지 알겠죠? 라이프 사이클도 알구요.
그런데 만약에, 내가 어떤 인풋박스에서 텍스트를 가져오고 싶으면 어떻게 접근해야할까요?
(render()가 끝나고 가져오면 될까요? 아니면 mount가 끝나고?
아니, 그 전에 가상돔에서 가져오나? 아니면 DOM에서? 🤔)
→ 답은, **리액트 요소에서 가져온다!**

- React 요소를 가지고 오는 방법: React.createRef()

```
import React from "react";
import logo from "../logo.svg";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import styled from "styled-components";

// 클래스형 컴포넌트는 이렇게 생겼습니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount(){
    // 콘솔에서 확인해보자!
    console.log(this.text);
    console.log(this.text.current);
  }

  // 렌더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {

    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          { /* 컴포넌트를 넣어줍니다. */ }
          { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
          <BucketList list={this.state.list} />
        </Container>

        <div>
          <input type="text" ref={this.text}/>
        </div>
      </div>
    );
  }
}

const Container = styled.div`
  max-width: 350px;
  min-height: 80vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
```

```
border-radius: 5px;
border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;

export default App;
```

07. State 관리 (1)

▼ 16) 단방향 데이터 흐름이란?

👉 이름에서부터 감이 오지 않나요?
데이터는 위에서 아래로, 부모에서 자식으로 넘겨줘야 한다는 소리입니다.

(1) 왜 단방향으로 써야하지?

- 라이프 사이클과 함께 생각해보기
부모 컴포넌트의 state가 업데이트 되면 자식 컴포넌트도 리렌더링이 일어납니다.
만약 자식 컴포넌트의 state가 바뀐 걸 부모 컴포넌트가 props로 받는다고 생각해봅시다.
그러면 자식 컴포넌트가 업데이트 될 때 부모 컴포넌트도 업데이트 되겠죠?
앗..., 그럼 또 자식 컴포넌트가 리렌더링될거구요. 😞

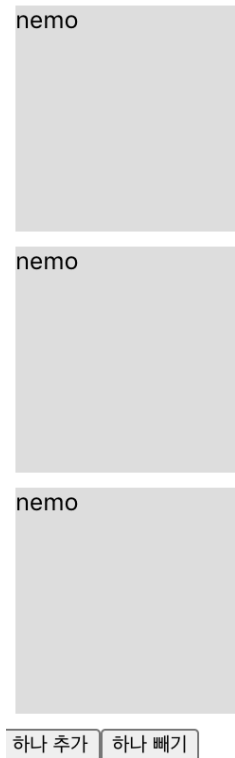
▼ 17) 클래스형 컴포넌트에서 state 관리 - setState()

👉 라이프 사이클을 볼 때 잠깐 봤던 setState()!
클래스형 컴포넌트의 state를 업데이트할 때 사용하는 함수입니다.

- 새프로젝트를 만들고 state를 가지고 놀아볼까요?

▼ 🐡 네모칸 만들기

👉 이런걸 만들거예요! state에 넣은 숫자대로 네모칸이 나오고, 추가/빼기 버튼으로 네모를 늘리거나 줄일 수 있도록요.
(이번 예제는 state에 대해서만 다룰거니까 style은 inline으로 작성합니다.)



⚠ 아래의 (1)~(3)은 이후 새 프로젝트를 만들 때마다 하실 부분입니다!
이후에 [새 프로젝트를 만들어주세요]라고 말씀드리면 이 과정을 잊지말고 사색 해치워주세요.
😎

▼ (1) 새 CRA 만들기

```
yarn create react-app nemo
```

▼ (2) index.js에서 <React.StrictMode> 부분을 지우기

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render([
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
]);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
  <App />,
  document.getElementById('root')
);

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

▼ (3) App.js를 class형 컴포넌트로 바꾸고 시작!

```
// App component를 class형으로!
import React from 'react';

class App extends React.Component {

  constructor(props){
    super(props);

    this.state = {}
  }

  componentDidMount(){

  }

  render(){

    return (
      <div className="App">

        </div>
      );
    )
  }
}

export default App;
```

▼ (4) state에 count라는 변수를 추가하고, count 숫자만큼 네모칸을 화면에 띄우기

- [Array.from\(\) 자세히 알아보기](#)→

```
import React from "react";

class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 3, // 숫자넣기!
    };
  }

  componentDidMount() {}

  render() {
    // 배열을 만듭니다.
    // Array.from()은 배열을 만들고 초기화까지 해주는 내장 함수입니다.
    // Array.from()의 첫번째 파라미터로 {length: 원하는 길이} 객체를,
```

```
// 두번째 파라미터로 원하는 값을 반환하는 콜백함수를 넘겨주면 끝!
// array의 내장함수 대부분은 콜백 함수에서 (현재값, index넘버)를 인자로 씁니다.
const nemo_count = Array.from({ length: this.state.count }, (v, i) => i);

// 콘솔로 만들어진 배열을 확인해봅니다. 숫자가 0부터 순서대로 잘 들어갔나요?
console.log(nemo_count);

return (
  <div className="App">
    {nemo_count.map((num, idx) => {
      return (
        <div key={idx}
          style={{
            width: "150px",
            height: "150px",
            backgroundColor: "#ddd",
            margin: "10px",
          }}
        >
          nemo
        </div>
      );
    })}
  </div>
);
}
}

export default App;
```

▼ (5) 더하기, 빼기 버튼을 만들고,

```
return (
  <div className="App">
    {nemo_count.map((num, idx) => {
      return (
        <div key={idx}
          style={{
            width: "150px",
            height: "150px",
            backgroundColor: "#ddd",
            margin: "10px",
          }}
        >
          nemo
        </div>
      );
    })}

    <div>
      <button>하나 추가</button>
      <button>하나 빼기</button>
    </div>
  </div>
);
```

▼ (6) 함수를 만들어서

```
addNemo = () => {
  // this.setState로 count를 하나 더해줍니다!
  this.setState({ count: this.state.count + 1 });
};

removeNemo = () => {
  // 네모 갯수가 0보다 작을 순 없겠죠! if문으로 조건을 걸어줍니다.
  if (this.state.count > 0) {
    // this.setState로 count를 하나 빼줍니다!
  }
};
```



```

        this.setState({ count: this.state.count - 1 });
      }else{
        window.alert('네모가 없어요!');
      }
    }
  };

```

▼ (7) 연결하자!

```

return (
  <div className="App">
    {nemo_count.map((num, idx) => {
      return (
        <div
          key={idx}
          style={{
            width: "150px",
            height: "150px",
            backgroundColor: "#ddd",
            margin: "10px",
          }}
        >
          nemo
        </div>
      );
    })}

    <div>
      { /* 함수를 호출합니다. 이 클래스 안의 addNemo 함수를 불러오기 때문에 this.addNemo로 표기해요. */ }
      <button onClick={this.addNemo}>하나 추가</button>
      <button onClick={this.removeNemo}>하나 빼기</button>
    </div>
  </div>
);

```

▼ (8) 완성본 코드

▼ [코드스니펫] - 05. 네모칸 만들기 완성본

```

import React from "react";

class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 3, // 숫자넣기!
    };
  }

  componentDidMount() {}

  addNemo = () => {
    // this.setState로 count를 하나 더해줍니다!
    this.setState({ count: this.state.count + 1 });
  };

  removeNemo = () => {
    // 네모 갯수가 0보다 작을 순 없겠죠! if문으로 조건을 걸어줍니다.
    if (this.state.count > 0) {
      // this.setState로 count를 하나 빼줍니다!
      this.setState({ count: this.state.count - 1 });
    }else{
      window.alert('네모가 없어요!');
    }
  };

  render() {
    // 배열을 만듭니다.
    // Array.from()은 배열을 만들고 초기화까지 해주는 내장 함수입니다.

```

```
// Array.from()의 첫번째 파라미터로 {length: 원하는 길이} 객체를,
// 두번째 파라미터로 원하는 값을 반환하는 콜백함수를 넘겨주면 끝!
// array의 내장함수 대부분은 콜백 함수에서 (현재값, index번호)를 인자로 씁니다.
const nemo_count = Array.from({ length: this.state.count }, (v, i) => i);

// 콘솔로 만들어진 배열을 확인해봅니다. 숫자가 0부터 순서대로 잘 들어갔나요?
console.log(nemo_count);

return (
  <div className="App">
    {nemo_count.map((num, idx) => {
      return (
        <div
          key={idx}
          style={{
            width: "150px",
            height: "150px",
            backgroundColor: "#ddd",
            margin: "10px",
          }}
        >
          nemo
        </div>
      );
    })}

    <div>
      { /* 함수를 호출합니다. 이 클래스 안의 addNemo 함수를 불러오기 때문에 this.addNemo로 표기해요. */ }
      <button onClick={this.addNemo}>하나 추가</button>
      <button onClick={this.removeNemo}>하나 빼기</button>
    </div>
  </div>
);
}
}
export default App;
```

08. State 관리 (2)

▼ 18) 함수형 컴포넌트에서 state 관리 - useState()

👉 이번에는 함수형 컴포넌트에서 어떻게 state를 쓸 수 있는 지 봅시다.
1주차 강의에서 함수형 컴포넌트는 state가 없다고 말씀드렸었죠?
맞아요. 함수형 컴포넌트는 state가 없지만 react hooks를 사용하면 state를 가질 수 있습니다!

- 새 컴포넌트를 만들어서 해볼까요?
Nemo.js 파일을 만들고 시작합니다!

▼ (1) Nemo 컴포넌트 만들기

⚠ 앞으로 새 함수형 컴포넌트를 만들 때는 파일을 만들고, 함수형 컴포넌트 껍데기까지 만들어주세요! export 잊지 말기! 😊

```
import React from "react";

const Nemo = (props) => {
```

```

    // 반환할 리액트 요소가 없을 때는 null을 넘겨주세요! 처음 컴데기 잡으실때도 null을 넘겨주면 굳!
    return null;
  }

  export default Nemo;

```

▼ (2) App에서 <Nemo/> 불러오기

```

// import 먼저 하고
import Nemo from "./Nemo";
...
  <div className="App">
    /*컴포넌트 불러다 쓰기*/
    <Nemo/>
    ...
  ...

```

▼ (3) useState()로 count를 state로 등록하자

```

// count에는 state 값이, setCount는 count라는 state 값을 수정하는 함수가 될거예요.
// useState(초기값): () 안에 초기값을 넣어줍니다.
const [count, setCount] = React.useState(3);

```

▼ (4) 뷰를 만들고(=반환할 리액트 요소를 만들고),

```

const nemo_count = Array.from({ length: count }, (v, i) => i);
// 반환할 리액트 요소가 없을 때는 null을 넘겨주세요!
return (
  <div className="App">
    {nemo_count.map((num, idx) => {
      return (
        <div
          key={idx}
          style={{
            width: "150px",
            height: "150px",
            backgroundColor: "#ddd",
            margin: "10px",
          }}
        >
          nemo
        </div>
      );
    })}

    <div>
      <button>하나 추가</button>
      <button>하나 빼기</button>
    </div>
  </div>
);

```

▼ (5) 함수를 만들어서,

```

const addNemo = () => {
  // setCount를 통해 count에 저장된 값을 + 1 해줍니다.
  setCount(count + 1);
};

const removeNemo = () => {
  // setCount를 통해 count에 저장된 값을 - 1 해줍니다.

```

```
// 이번엔 if문 대신 삼항 연산자로 해볼거예요!
setCount(count > 0 ? count - 1 : 0);
};
```

▼ (6) 연결하자!

```
<div>
  {/* 함수를 호출합니다. */}
  <button onClick={addNemo}>하나 추가</button>
  <button onClick={removeNemo}>하나 빼기</button>
</div>
```

▼ (7) 완성본 코드

```
import React from "react";

const Nemo = (props) => {
  // count에는 state 값이, setCount는 count라는 state 값을 수정하는 함수가 될거예요.
  // useState(초기값): () 안에 초기값을 넣어줍니다.
  const [count, setCount] = React.useState(3);

  const addNemo = () => {
    // setCount를 통해 count에 저장된 값을 + 1 해줍니다.
    setCount(count + 1);
  };

  const removeNemo = () => {
    // setCount를 통해 count에 저장된 값을 - 1 해줍니다.
    // 이번엔 if문 대신 삼항 연산자로 해볼거예요!
    setCount(count > 0 ? count - 1 : 0);
  };

  const nemo_count = Array.from({ length: count }, (v, i) => i);
  // 반환할 리액트 요소가 없을 때는 null을 넘겨주세요!
  return (
    <div className="App">
      {nemo_count.map((num, idx) => {
        return (
          <div
            key={idx}
            style={{
              width: "150px",
              height: "150px",
              backgroundColor: "#ddd",
              margin: "10px",
            }}
          >
            nemo
          </div>
        );
      })}

      <div>
        {/* 함수를 호출합니다. */}
        <button onClick={addNemo}>하나 추가</button>
        <button onClick={removeNemo}>하나 빼기</button>
      </div>
    </div>
  );
};

export default Nemo;
```



이제 state 관리하는 법까지 모두 배웠어요. 😊

정말 중요한 부분입니다. 헛갈리면 다시 보기!

state를 관리하는 것도, 함수형 컴포넌트와 클래스형 컴포넌트를 만드는 순서도 중요해요.

09. Quiz_버킷리스트에 아이템을 추가해보자!

▼ 19) 🛠 버킷리스트에 아이템을 추가해보기



자바스크립트 배열을 다룰 줄 모른다면? MDN 문서를 한 번 읽고 해봅시다! ([링크](#))

▼ Q. 퀴즈설명: 버킷리스트에 새로운 항목을 추가해봅시다.

화면

내 버킷리스트

영화관 가기

매일 책임기

수영 배우기

추가하기

▼ [코드스니펫] - 06. App.js (← 강의 화면 오른쪽 하단에 '복사' 클릭! 😊)

```
import React from "react";
import logo from "../logo.svg";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import styled from "styled-components";

// 클래스형 컴포넌트는 이렇게 생성합니다!
class App extends React.Component {
  constructor(props) {
    super(props);
  }
}
```

```

    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
  }

  // 렌더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <Container>
          <Title >내 버킷리스트</Title>
          <Line/>
          { /* 컴포넌트를 넣어줍니다. */ }
          { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
          <BucketList list={this.state.list} />
        </Container>
      </div>
    );
  }
}

const Container = styled.div`
  max-width: 350px;
  min-height: 80vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;

export default App;

```

▼ [코드스니펫] - 07. BucketList.js (← 강의 화면 오른쪽 하단에 '복사' 클릭! 😊)

```

// 리액트 패키지를 불러옵니다.
import React from "react";
import styled from "styled-components";

const BucketList = (props) => {
  const my_lists = props.list;

  return (
    <ListStyle>
      {my_lists.map((list, index) => {
        return (
          <ItemStyle key={index}>
            {list}
          </ItemStyle>
        );
      })}
    </ListStyle>
  );
};

const ListStyle = styled.div`
  display: flex;
  flex-direction: column;

```

```

height: 100%;
overflow-x: hidden;
overflow-y: auto;
`;

const ItemStyle = styled.div`
padding: 16px;
margin: 8px;
background-color: aliceblue;
`;

export default BucketList;

```



힌트:

1. 뷰를 먼저 만들어주세요! 텍스트를 입력할 공간과 [추가하기] 버튼을 잊지말고 추가하기!
2. ref! 잊지 않았죠? text를 가져올 때 써줍니다.
3. 인풋박스에 입력한 값은 [인풋박스 ref].current.value로 가져올 수 있습니다.
4. 버킷리스트 컴포넌트는 만질 필요가 없어요. (소근)

▼ A. 함께하기(완성본)

▼ [코드스니펫] - 08. 버킷리스트 추가하기 App.js 완성본

```

import React from "react";
import logo from "../logo.svg";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from "../BucketList";
import styled from "styled-components";

// 클래스형 컴포넌트는 이렇게 생성합니다!
class App extends React.Component {
  constructor(props) {
    super(props);
    // App 컴포넌트의 state를 정의해줍니다.
    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
    // ref는 이렇게 선언합니다!
    this.text = React.createRef();
  }

  componentDidMount(){
    console.log(this.text);
  }

  addBucketList = () => {
    let list = this.state.list;
    const new_item = this.text.current.value;

    // 리액트에서는 concat으로 배열항목을 합쳐주는 게 좋아요. 아래처럼요!
    // list = list.concat(new_item);
    // this.setState({list: list});

    // 이걸 가장 편한 배열 합치기 방법입니다.
    // ...은 배열 안에 있는 항목을 전부 꺼내서 넣어놓는다는 뜻입니다. (스프레드 문법이라고 불러요.)
    this.setState({list: [...list, new_item]});
  }

  // 렌더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {

    return (
      <div className="App">

```

```

    <Container>
      <Title>내 버킷리스트</Title>
      <Line />
      { /* 컴포넌트를 넣어줍니다. */ }
      { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
      <BucketList list={this.state.list} />
    </Container>
    { /* 인풋박스와 추가하기 버튼을 넣어줬어요. */ }
    <Input>
      <input type="text" ref={this.text} />
      <button onClick={this.addBucketList}>추가하기</button>
    </Input>
  </div>
);
}
}

const Input = styled.div`
  max-width: 350px;
  min-height: 10vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Container = styled.div`
  max-width: 350px;
  min-height: 60vh;
  background-color: #fff;
  padding: 16px;
  margin: 20px auto;
  border-radius: 5px;
  border: 1px solid #ddd;
`;

const Title = styled.h1`
  color: slateblue;
  text-align: center;
`;

const Line = styled.hr`
  margin: 16px 0px;
  border: 1px dotted #ddd;
`;

export default App;

```

10. Event Listener

▼ 21) 이벤트 리스너란?

A 이벤트 리스너는 **사용자가 어떤 행동(=이벤트)을 하는 지** 아닌 **지 지켜보다가 알려주는 것**입니다.
대표적으로는 마우스 클릭, 터치, 마우스 오버, 키보드 누름 등이 자주 쓰여요!
더 많은 이벤트가 궁금하다면? →



이벤트 리스너는 <div onClick={} >에서처럼 엘리먼트에 직접 넣어줄 수도 있지만, 이번 강의에서는 addEventListener를 통해 추가해볼거예요.

눈으로만 보고 어떻게 쓰는구나 감만 잡아봅시다. 😊
(저는 네모 추가하기 프로젝트에 진행합니다!)

▼ 22) 클래스형 컴포넌트에서 event listener 구독하기



이벤트 리스너는 어디에 위치해야할까요?

클릭을 하건, 마우스를 올리건 DOM 요소가 있어야 이벤트가 발생하는 지 지켜볼 수 있겠죠?
→ 네! componentDidMount()에 넣어주면 됩니다.

- (1) 일단, ref부터 잡아볼까요!



퀴즈 화면과 점수 화면을 만들어주세요!

App.js의 state를 이용해서 조건부 렌더링을 하며 점수화면과 퀴즈화면, 시작화면을 오가봅시다.
이미지를 하나 넣고, 이벤트 리스너로 좌/우 스와이프를 만들어봐요.
(react-tinder-card(스와이프 이벤트를 미리 구현해준 패키지)를 이용해서 훨씬 간단하게 만들어 볼 수 있어요!)

```
constructor(props) {
  super(props);

  this.state = {
    count: 3, // 숫자넣기!
  };

  // div에 ref를 먼저 잡아줍시다.
  this.div = React.createRef();
}

...

return (
  <div className="App" ref={this.div}>
    <Nemo/>
  </div>
);

...
```

- (2) 이제 addEventListener()를 이용해서 이벤트를 등록합니다.

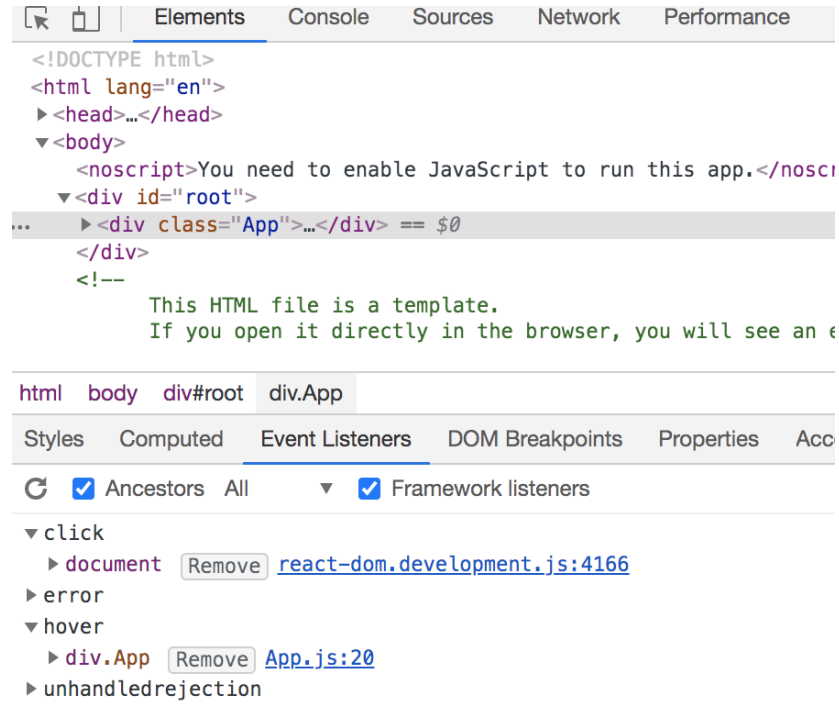
```
hoverEvent = (e) => {
  // 콘솔로 이 이벤트가 누구에게서 일어났는 지 확인할 수 있습니다.
  console.log(e.target);

  if(e.target.className === 'app'){
    // 이벤트의 장본인의 배경 색을 바꿔볼까요?
    e.target.style.background = "#eee";
  }
}

componentDidMount() {
  // 리액트 요소가 잘 잡혔나 확인해봅시다!
  console.log(this.div);
```

```
// 마우스를 올렸을 때, 이벤트가 일어나는 지 확인해봅시다.
this.div.current.addEventListener("mouseover", this.hoverEvent);
}
```

- 개발자 도구를 통해서도 확인할 수 있습니다 😊



- (3) 이벤트는 꼭 컴포넌트가 사라지면 지워주세요!

```
componentWillUnmount() {
  // 컴포넌트가 사라질 때 이벤트를 지워줍니다.
  this.div.current.removeEventListener("mouseover", this.hoverEvent);
}
```

11. 끝 & 숙제 설명

- ▼ 기획서(레이아웃) 보기

점수 보기

이범규 퀴즈에 대한 내 점수는

58 점

우와! 이 정도면 가깝다고도 멀다고도 할 수
없는 그런 애매한 사이군요! 조금 더 관심을
갖고 친해지면 어떨까요?

정답 보기

이범규에게 한 마디

문제풀기



1번째 문제

이범규가 가장 좋아하는 과일은
사과이다.

▼ 예시 화면

스파르타 코딩 클럽 퀴즈에
대한 내 점수는?

100 점

이 정도면 아주 친한 친구 사이! 앞으로도 더 친
하게 지내요! :)

점수보기

랭킹보기

5번 문제

르탄이는 2살이다.



12. 2주차 숙제 답안 코드

▼ [코드스니펫] - 2주차 숙제 답안 코드

전체 코드

▼ App.js

```
import logo from './logo.svg';
import './App.css';
import React from "react";

import Start from "./Start";
import Quiz from "./Quiz";
import Score from "./Score";

class App extends React.Component{
  constructor(props){
    super(props);
    // state에 필요한 데이터를 넣어줘요!
    this.state = {
      name: "스파르타 코딩 클럽",
      page: "score",
      scoreMsg: "이 정도면 아주 친한 친구 사이! 앞으로도 더 친하게 지내요! :)",
      list: [
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
        { question: "르탄이는 2살이다.", answer: "0" },
      ],
      ranking: [
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
        { rank: 1, name: "임민영", message: "안녕 르탄아!" },
      ],
    };
  }

  render () {
    return (
      <div className="App">
        { /* 조건부 렌더링을 합니다 / state의 page를 바꿔가면서 확인해보요! */ }
        {this.state.page === "quiz" && (<Quiz list={this.state.list} />)}
        {this.state.page === "start" && (<Start name={this.state.name} />)}
        {this.state.page === "score" && (<Score name={this.state.name} scoreMsg={this.state.scoreMsg}/>)}
      </div>
    );
  }
}

export default App;
```

▼ Score.js

```
import React from "react";
import styled from "styled-components";

const Score = (props) => {
  // 컬러셋 참고: https://www.shutterstock.com/ko/blog/pastel-color-palettes-rococo-trend/
  return (
    <ScoreContainer>
      <Text>
```

```

        <span>{props.name}</span>
        퀴즈에 <br />
        대한 내 점수는?
    </Text>
    <MyScore>
        <span>100</span>점
        <p>{props.scoreMsg}</p>
    </MyScore>

    { /* <Button>다시 하기</Button> */ }
    <Button outlined>랭킹보기</Button>

    </ScoreContainer>
  );
};

const ScoreContainer = styled.div`
display: flex;
width: 100vw;
height: 100vh;
overflow: hidden;
padding: 16px;
box-sizing: border-box;
flex-direction: column;
justify-content: center;
align-items: center;
`;

const Text = styled.h1`
font-size: 1.5em;
margin: 0px;
line-height: 1.4;
& span {
  background-color: #fef5d4;
  padding: 5px 10px;
  border-radius: 30px;
}
`;

const MyScore = styled.div`
  & span {
    border-radius: 30px;
    padding: 5px 10px;
    background-color: #fef5d4;
  }
  font-weight: 600;
  font-size: 2em;
  margin: 24px;

  & > p {
    margin: 24px 0px;
    font-size: 16px;
    font-weight: 400;
  }
`;

const Button = styled.button`
padding: 8px 24px;
background-color: ${props => (props.outlined ? "#ffffff" : "#dadafc")};
border-radius: 30px;
margin: 8px;
border: 1px solid #dadafc;
width: 80vw;
`;

export default Score;

```

▼ Quiz.js

```

import React from "react";
import styled from "styled-components";
import img from "../scc_img01.png";
import TinderCard from "react-tinder-card";

```

```

//
const Quiz = (props) => {
  console.log(props);
  // state로 관리하자!
  const [num, setNum] = React.useState(0);

  const onSwipe = (direction) => {
    console.log("You swiped: " + direction);
    setNum(num + 1);
  };

  if (num > 10) {
    return <div>퀴즈 끝!</div>;
  }

  return (
    <QuizContainer>
      <p>
        <span>{num + 1}번 문제</span>
      </p>
      {props.list.map((l, idx) => {
        if (num === idx) {
          return <Question key={idx}>{l.question}</Question>;
        }
      })}

      <AnswerZone>
        <Answer>{"0"}</Answer>
        <Answer>{"X"}</Answer>
      </AnswerZone>

      {props.list.map((l, idx) => {
        if (idx === num) {
          return (
            <DragItem key={idx}>
              <TinderCard
                onSwipe={onSwipe}
                onCardLeftScreen={onSwipe}
                onCardRightScreen={onSwipe}
                preventSwipe={["up", "down"]}
              >
                <img src={img} />
              </TinderCard>
            </DragItem>
          );
        }
      })}
    </QuizContainer>
  );
};

const QuizContainer = styled.div`
  & > p > span {
    padding: 8px 16px;
    background-color: #fef5d4;
    // border-bottom: 3px solid #ffd6aa;
    border-radius: 30px;
  }
`;

const Question = styled.h1`
  font-size: 1.5em;
`;

const AnswerZone = styled.div`
  width: 100%;
  display: flex;
  flex-direction: row;
  min-height: 70vh;
`;

const Answer = styled.div`
  width: 50%;

```

```

display: flex;
justify-content: center;
align-items: center;
font-size: 100px;
font-weight: 600;
color: #dadafc77;
`;

const DragItem = styled.div`
display: flex;
align-items: center;
justify-content: center;
position: fixed;
top: 0;
left: 0;
width: 100vw;
height: 100vh;

& > div {
border-radius: 500px;
background-color: #ffd6aa;
}
& img {
max-width: 150px;
}
`;
export default Quiz;

```

+) 직접 이벤트 리스너로 스와이프를 만들어보자!

▼ Swipeltem.js

```

import React from "react";
import styled from "styled-components";
import img from "./scc_img01.png";

const SwipeItem = React.memo(({ onSwipe }) => {
  const swipe_div = React.useRef(null);
  let swipe_status = "ready";
  let target_classname = "";
  let coordinate = {
    start_x: 0,
    start_y: 0,
    end_x: 0,
    end_y: 0,
  };
};

React.useEffect(() => {
  const reset = () => {
    // console.log("in reset");
    swipe_status = "ready";

    coordinate = {
      start_x: 0,
      start_y: 0,
      end_x: 0,
      end_y: 0,
    };

    swipe_div.current.className = target_classname;

    swipe_div.current.style.left = 0 + "px";
    swipe_div.current.style.top = 0 + "px";
  };

  const touchStart = (e) => {
    // console.log("start");

    // 터치 시작 시, swipe_status를 touchstart로 변경해줍니다.

```



```

// 그리고 터치 시작한 좌표를 기록합니다!
// (중요! 그래야 터치 종료할 때 위치를 보고 왼쪽인지, 오른쪽인지 판별할 수 있겠죠!)
swipe_status = "touchstart";
target_classname = swipe_div.current.className;
// console로 터치 이벤트가 시작될 때 좌표를 확인해볼까요?
// console.log(e.touches[0]);
// console.log(e.touches[0].clientX);
// console.log(e.touches[0].clientY);

// 좌표도 기록해줍니다 :)
coordinate = {
  ...coordinate,
  start_x: e.touches[0].clientX,
  start_y: e.touches[0].clientY,
};
};

const touchEnd = (e) => {
  swipe_status = "touchend";
  // touchEnd이벤트는 touches 대신, changedTouches가 있어요.
  // console.log(e.changedTouches[0]);
  coordinate = {
    ...coordinate,
    end_x: e.changedTouches[0].clientX,
    end_y: e.changedTouches[0].clientY,
  };

  // x좌표 이동 거리를 구해줍니다.
  let diff_x = coordinate.end_x - coordinate.start_x;
  // 스와이프 방향 / 기본은 left로 뒀습니다!
  let direct = "left";

  // Math.abs() : 절대값을 구해주는 친구입니다.
  if (Math.abs(diff_x) > 50) {
    swipe_div.current.className = target_classname + " swipe";

    // 움직인 방향에 따라 더 움직이고 투명도를 0으로 (점점 사라지게) 줘봐요!
    if (diff_x > 0) {
      // console.log('move right');
      direct = "right";
      swipe_div.current.style.left = diff_x + 150 + "px";
      swipe_div.current.style.opacity = 0;
    } else {
      direct = "left";
      // console.log('move left');
      // console.log(diff_x - 150);
      swipe_div.current.style.left = diff_x - 150 + "px";
      swipe_div.current.style.opacity = 0;
    }

    // 300 ms후 reset 해줍니다!
    // 이 300ms는 props로 받아서 처리해줘도 좋겠네요!
    // props로 받아온, 콜백 함수도 여기서 처리해줄게요!
    window.setTimeout(() => {
      reset();
      onSwipe(direct);
    }, 300);
    return;
  }

  // reset 해줍니다.
  reset();
};

const touchMove = (e) => {
  // 스와이프 중 다른 이벤트가 발생하는 것을 막아줍니다
  e.preventDefault();

  // console.log("in touch move!");
  // 현재 좌표(이동 중인 좌표)를 기록해줍니다.
  let current_coordinate = {
    x: e.touches[0].clientX,
    y: e.touches[0].clientY,
  };
};

```

```

// 콘솔로 이동한 값이 어떻게 나오는 지 한번 확인해볼까요?
// console.log(
//   current_coordinate.x - coordinate.start_x,
//   current_coordinate.y - coordinate.start_y
// );

// 터치 중일 때 div가 따라 움직이도록 해줍니다!
swipe_div.current.style.left =
  current_coordinate.x - coordinate.start_x + "px";
swipe_div.current.style.top =
  current_coordinate.y - coordinate.start_y + "px";
};

// 터치 이벤트가 취소될 경우 원래 상태로 돌려줍니다!
const touchCancel = (e) => {
  swipe_status = "cancel";
  reset();
};

swipe_div.current.addEventListener("touchstart", touchStart);
swipe_div.current.addEventListener("touchend", touchEnd);
swipe_div.current.addEventListener("touchmove", touchMove);
swipe_div.current.addEventListener("touchcancel", touchCancel);

// 이부분은 이벤트 해제 부분이에요!
return () => {
  // 만약 이벤트 걸었던 엘리먼트가 없으면 해제하지 않습니다!
  if (!swipe_div.current) {
    return;
  }
  swipe_div.current.removeEventListener("touchstart", touchStart);
  swipe_div.current.removeEventListener("touchend", touchEnd);
  swipe_div.current.removeEventListener("touchmove", touchMove);
  swipe_div.current.removeEventListener("touchcancel", touchCancel);
};
}, []);

return (
  <DragItem ref={swipe_div}>
    <img src={img} />
  </DragItem>
);
});

const DragItem = styled.div`
  display: flex;
  align-items: center;
  justify-content: center;
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;

  &.swipe {
    transition: 300ms;
  }

  & > div {
    border-radius: 500px;
    background-color: #ffd6aa;
  }
  & img {
    max-width: 150px;
  }
`;

SwipeItem.defaultProps = {
  onSwipe: (direction) => {},
};

export default SwipeItem;

```

▼ Quiz.js

```
import React from "react";
import styled from "styled-components";
import img from "./scc_img01.png";
import TinderCard from "react-tinder-card";
import SwipeItem from "./SwipeItem";

//
const Quiz = (props) => {
  console.log(props);
  // state로 관리하자!
  const [num, setNum] = React.useState(0);

  const onSwipe = (direction) => {
    console.log("You swiped: " + direction);
    setNum(num + 1);
  };

  if (num > 10) {
    return <div>퀴즈 끝!</div>;
  }

  return (
    <QuizContainer>
      <p>
        <span>{num + 1}번 문제</span>
      </p>
      {props.list.map((l, idx) => {
        if (num === idx) {
          return <Question key={idx}>{l.question}</Question>;
        }
      })}

      <AnswerZone>
        <Answer>{"0"}</Answer>
        <Answer>{"X"}</Answer>
      </AnswerZone>

      {props.list.map((l, idx) => {
        if (idx === num) {
          return <SwipeItem key={idx} onSwipe={onSwipe} />;
          // return (
          //   <DragItem key={idx}>
          //     <TinderCard
          //       onSwipe={onSwipe}
          //       onCardLeftScreen={onSwipe}
          //       onCardRightScreen={onSwipe}
          //       preventSwipe={["up", "down"]}
          //     >
          //       <img src={img} />
          //     </TinderCard>
          //   </DragItem>
          // );
        }
      })}
    </QuizContainer>
  );
};

const QuizContainer = styled.div`
  & > p > span {
    padding: 8px 16px;
    background-color: #fef5d4;
    // border-bottom: 3px solid #ffd6aa;
    border-radius: 30px;
  }
`;

const Question = styled.h1`
  font-size: 1.5em;
`;
```

```

const AnswerZone = styled.div`
  width: 100%;
  display: flex;
  flex-direction: row;
  min-height: 70vh;
`;

const Answer = styled.div`
  width: 50%;
  display: flex;
  justify-content: center;
  align-items: center;
  font-size: 100px;
  font-weight: 600;
  color: #dadafc77;
`;

const DragItem = styled.div`
  display: flex;
  align-items: center;
  justify-content: center;
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;

  & > div {
    border-radius: 500px;
    background-color: #ffd6aa;
  }
  & img {
    max-width: 150px;
  }
`;

export default Quiz;

```

Copyright © TeamSparta All rights reserved.