



[스파르타코딩클럽] 리액트 기초반 - 2주차 (1)



매 주차 강의자료 시작에 PDF파일을 올려두었어요!

[수업 목표]

1. 컴포넌트의 라이프 사이클을 이해한다.
2. 컴포넌트의 state를 관리할 수 있다.
3. 패키지를 찾아서 설치할 수 있다.
4. React hook을 사용할 수 있다.

[목차]

- 01. 라이프 사이클이란?
- 02. 라이프 사이클 함수로 보는 라이프 사이클
- 03. Component (1)
- 04. Component (2)
- 05. Component (3)
- 06. Quiz 버킷리스트 페이지를 만들어봅시다!
- 07. 화면을 예쁘게! React에서 CSS 사용하기
- 08. 화면을 예쁘게! - styled-components
- 09. Quiz 버킷리스트에 styled-components 적용하기
- 10. Ref! 리액트에서 돔요소를 가져오려면?
- 11. State 관리 (1)
- 12. State 관리 (2)
- 13. Quiz 버킷리스트에 아이템을 추가해보자!
- 14. 끝 & 숙제 설명
- HW. 2주차 숙제 답안 코드



모든 토글을 열고 닫는 단축키

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

Mac : **⌘** + **⌥** + **t**

01. 라이프 사이클이란?

- ▼ 1) 가상돔이란?



DOM을 기억하시나요?

DOM은 html 단위 하나하나를 객체로 생각하는 모델입니다. 예를 들면, 'div'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 알아끼워줍니다!** → 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.

▼ 2) 라이프 사이클이란?



컴포넌트의 라이프 사이클(= 컴포넌트 생명주기)은 정말 중요한 개념입니다!

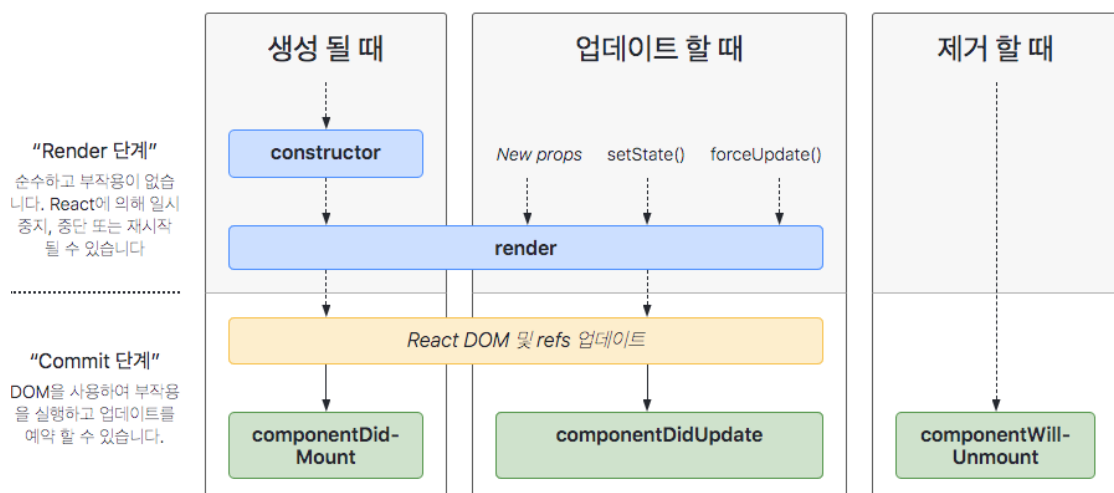
컴포넌트가 렌더링을 준비하는 순간부터, 페이지에서 사라질 때까지가 라이프 사이클이에요.

공식 문서에서도 자세히 다루고 있어요.

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

□ 덜 일반적인 라이프 사이클 표시

React 버전 ^16.4 언어 한국어



- 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 렌더링할 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.

02. 라이프 사이클 함수로 보는 라이프 사이클



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 컴포넌트를 더 권장**하기 때문입니다!

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

더 많은 라이프 사이클 함수는 공식 문서에서 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.js에서 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;

```

▼ 3) constructor()



생성자 함수라고도 부릅니다. 컴포넌트가 생성되면 가장 처음 호출되는 친구죠!

▼ 4) 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!

▼ 5) componentDidMount()



컴포넌트가 화면에 나타나는 것을 **마운트(Mount)**한다고 표현합니다. didMount()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!

▼ 6) 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 실제로 올랐을 후니까 DOM 관련 처리를 해도 됩니다!

▼ 7) componentWillUnmount()



컴포넌트가 DOM에서 제거 될 때 실행하는 함수입니다.

만약 우리가 스크롤 위치를 추적 중이거나, 어떤 이벤트 리스너를 등록했다면 여기에서 꼭꼭 해제를 해줘야 합니다.

컴포넌트 없이 이벤트만 남겨둘 순 없잖아요!



componentWillUnmount()가 호출되는 걸 보려면, app.js에서 <LifecycleEx/>를 없애봐야겠죠?

삼항연산자를 사용해서 컴포넌트를 보여주거나, 없애는 걸 **조건부 렌더링**이라고 불러요.

이건 저만 진행할게요! 눈으로만 따라와주기!

03. Component (1)



Component는 클래스형과 함수형이 있습니다. 직전 강의에서 본 컴포넌트가 바로 클래스형 컴포넌트였죠!

이제 클래스형 컴포넌트는 잘 쓰지 않지만, 우리는 두 가지를 모두 살펴볼거예요.

→ 왜냐하면 이미 공개된 프로젝트들(아마 여러분이 가야할 회사에서도...!)은 클래스형 컴포넌트를 사용 중일 수도 있거든요. 최소한 코드를 알아보고 고칠 수 있을 정도는 알아두는 편이 좋습니다.

▼ 8) Component란?

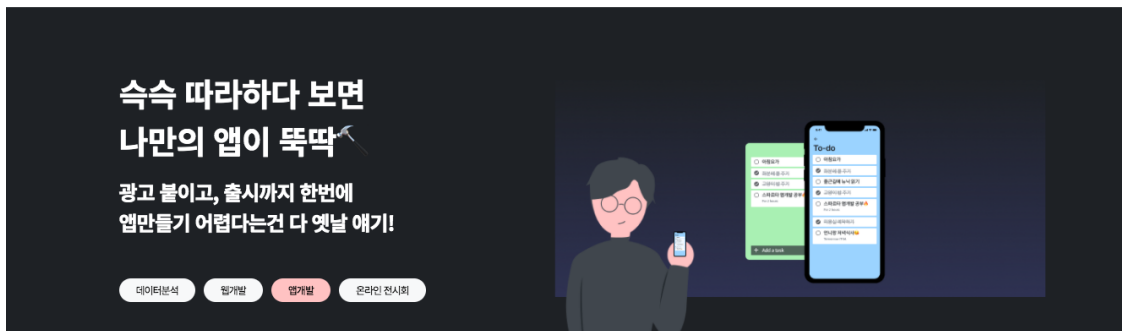


위에서 **리엑트는 레고**라고 말씀 드렸죠? **컴포넌트는 블록**입니다!

- 웹 사이트를 조각내자!

컴포넌트를 이해하고 잘 써먹으려면 웹 사이트를 잘 조각낼 줄 알아야 합니다.

가령 우리가 아래와 같은 웹 사이트를 만든다고 생각해봅시다!



스파르타 웹/앱개발

왕초보 코딩교육의 명가

스파르타코딩클럽은 사전 경험이 전혀 없는 입문자를 위해 고안된 수업입니다. 필요한 것만 짧고 굵게 배우고, 내 것을 만들면서 기본기를 다집니다. 웹/앱 서비스의 작동 방식을 이해하고, 코딩 하는 사람으로서의 삶을 누리세요.

코딩 공부 어디서 뭐부터 시작해야하지?
더 이상 고민하지 마세요!

문의/상담은 여기로



이 웹사이트를 HTML로 간단히 표현해보면 아래와 같을 겁니다.

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <header>
    ...
  </header>
  <div class="container">
    <div id="image-banner">
      ...
    </div>
    <div id="contents-1">
      ...
    </div>
  </div>
</body>
</html>
```

이 코드를 조각조각 내보면 아래와 같이 나눌 수 있을 거예요.
나눈 조각 하나하나를 컴포넌트라고 불러요!

1. <header/>
2. <container/>
 - a. <imagebanner/>
 - b. <contents1/>
3. <footer/>



즉, 이 웹 사이트는,
크게 <header/>, <container/>, <footer/> 세 개의 컴포넌트가 있고,
<container/> 컴포넌트는,
<imagebanner/>, <contents1/> 컴포넌트로 이루어져 있는 거죠! (쉽죠!)

- Component는 웹 사이트의 조각이고, 우리는 이 조각을 모아서 웹사이트에 뿌려준다.



웹 사이트를 잘 조각낼 줄 아는 사람 === 리엑트를 잘 쓰는 사람

▼ 9) State와 Props



Component에서는 그럼 데이터 관리를 어떻게 할까요?

데이터 관리를 아~주 살짝 맛만 볼게요. 곧 배울 **라이프 사이클**을 위한 선행이라고 생각하시고 가벼운 마음으로 들어보세요!

- State



state는 Component가 가지고 있는 데이터입니다.

아까 스파르타코딩클럽 사이트 조각 중 <header/> 컴포넌트를 예로 들어볼게요!



온라인 ▾

오프라인 ▾

기업교육

내 강의실 ▾

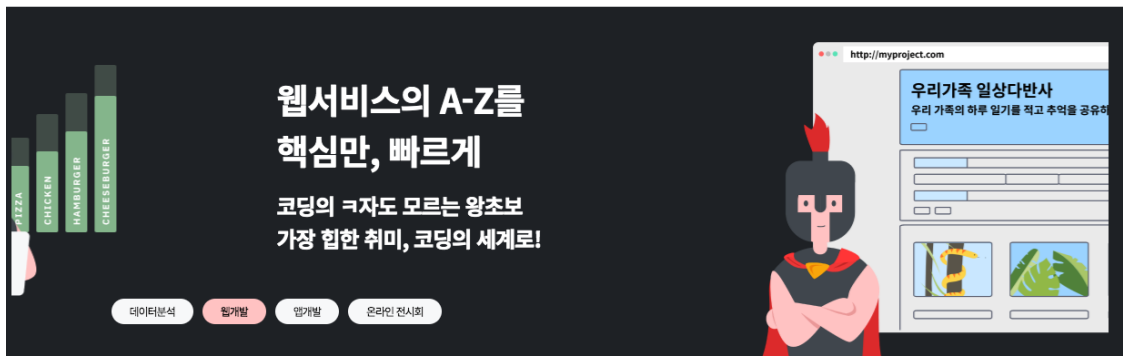
- 이 헤더에 들어갈 데이터는 뭐가 있을까요?
→ 로고 이미지 경로, 메뉴 이름(온라인, 오프라인, 기업교육, 내 강의실)이 있을 겁니다!
- 이 데이터는 <container/>나 <footer/> 컴포넌트에서는 쓰지 않겠죠!
- 즉, <header/> 컴포넌트에서만 쓰는 정보인 셈입니다.
- state는 한 컴포넌트에서만 사용하는 정보를 주로 넣어놓고 생성, 수정하는 데이터입니다.
- 생성도 수정도 오직 해당 컴포넌트 내에서만 이뤄집니다. 내꺼니까 생성도 수정도 자유롭죠!

- Props



props는 Component가 부모 Component로부터 받아온 데이터입니다.

이번엔 <container/> 컴포넌트를 예로 들어볼게요!



스파르타 웹/앱개발

왕초보 코딩교육의 명가

스파르타코딩클럽은 사전 경험이 전혀 없는 입문자를 위해 고안된 수업입니다. 필요한 것만 짧고 굵게 배우고, 내 것을 만들면서 기본기를 다집니다. 웹/앱 서비스의 작동 방식을 이해하고, 코딩 하는 사람으로서의 삶을 누리세요.

코딩 공부 어디서 뭐부터 시작해야하지?
더 이상 고민하지 마세요!

문의/상담은 여기로!

```
<container>
  <imagebanner/>
  <contents1/>
</container>
```

`<container/>` 컴포넌트는 두 개의 자식 컴포넌트를 가지고 있죠?

- `<container/>` 컴포넌트만 `<imagebanner/>` 컴포넌트한테 필요한 이미지 경로를 가지고 있다고 가정합니다. (state로 가지고 있다고 가정합니다!)
- 이 때 `<imagebanner/>` 컴포넌트는 자신의 부모인 `<container/>` 컴포넌트로부터 **이미지 경로**를 전달받아서 사용해야겠죠?
- `<container/>`가 가지고 있는 이미지 경로를 `<imagebanner/>`에게 전달해주면, 이 **이미지 경로**가 `<imagebanner/>` 컴포넌트의 props가 됩니다.
- 다시 말해, 부모 컴포넌트로부터 전달 받은 데이터를 props라고 합니다.
- 그럼 부모 컴포넌트가 가지고 있는 데이터를 `<imagebanner/>` 컴포넌트가 추가 하거나 수정할 수 있을까요?

Props로 받은 데이터는 수정할 수 없습니다! 남의 것이니까요!

04. Component (2)

▼ 10) 함수형 Component



컴포넌트가 뭔지 이제 직접 만들어 봅시다!

- VSCode를 열고, src 폴더 안에 BucketList.js 파일을 하나 만듭니다.



리액트 코딩 룰 1:

폴더는 소문자로 시작하는 카멜케이스를 사용

JS파일, 컴포넌트 이름은 대문자로 시작하는 카멜케이스를 사용

아래 코드를 BucketList.js에 붙여 넣어보세요.

▼ [코드스니펫] - 함수형 컴포넌트

```
//BucketList.js
// 리액트 패키지를 불러옵니다.
import React from 'react';

// 함수형 컴포넌트는 이렇게 쓸 수도 있고
// function BucketList(props){
//     return (
//         <div>버킷 리스트</div>
//     );
// }

// 이렇게 쓸 수도 있어요. =>가 들어간 함수를 화살표 함수라고 불러요.
// 저희는 앞으로 화살표 함수를 사용할거예요.
// 앳 () 안에 props! 부모 컴포넌트에게 받은 데이터입니다.
// js 함수가 값을 받아오는 것과 똑같이 받아오네요.
const BucketList = (props) => {

    // 컴포넌트가 뿌려줄 ui 요소(리액트 엘리먼트라고 불러요.)를 반환해줍니다.
    return (
        <div>
            버킷 리스트
        </div>
    );
}

// 우리가 만든 함수형 컴포넌트를 export 해줍니다.
// export 해주면 다른 컴포넌트에서 BucketList 컴포넌트를 불러다 쓸 수 있어요.
export default BucketList;
```

- 그리고 App.js로 돌아가서 BucketList 컴포넌트를 불러와 봅시다.

▼ [코드스니펫] - 컴포넌트 불러오기

```
//App.js

import React from 'react';
import './App.css';
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from './BucketList';

function App() {

    return (
        <div className="App">
            <h1>내 버킷리스트</h1>
            { /* 컴포넌트를 넣어줍니다. */ }
            <BucketList/>
        </div>
    );
}

export default App;
```

- 이제 크롬으로 돌아가 컴포넌트가 나오는 지 확인해봅시다.

내 버킷리스트

버킷 리스트

05. Component (3)

▼ 11) 클래스형 Component



이번엔 함수형 컴포넌트를 클래스형 컴포넌트로 바꿔볼게요!

App.js를 클래스형으로 바꾸고, BucketList 컴포넌트에 데이터를 넘겨줘 봅시다!

- App.js를 클래스형으로 바꿔봅시다.

아래 코드를 App.js에 붙여 넣고 브라우저에서 함수형일 때와 동일하게 동작하는 걸 확인해봅시다.

▼ [코드스니펫] - 클래스형 컴포넌트

```
//App.js

import React from 'react';
import './App.css';
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from './BucketList';

// 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'],
    };
  }

  // 랜더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    return (
      <div className="App">
        <h1>내 버킷리스트</h1>
        { /* 컴포넌트를 넣어줍니다. */ }
        <BucketList/>
      </div>
    );
  }
}
```

```

    </div>
  );
}
}

export default App;

```

▼ 12) Component에서 Component로 데이터를 넘겨주자!

👉 App 컴포넌트가 가지고 있는 state를 BucketList에 넘겨줍니다.

- state 값 사용하기

아래와 같이 입력해보고 콘솔을 확인해봅시다.

```

//App.js

...
render() {
  console.log(this.state);
  return (
    <div className="App">
      <h1>내 버킷리스트</h1>
      { /* 컴포넌트를 넣어줍니다. */ }
      <BucketList />
    </div>
  );
}

```

```

▼ {list: Array(3)} ⓘ
  ► list: (3) ["영화관 가기", "매일 책읽기", "수영 배우기"]
  ► __proto__: Object

```



알면 덜 찜찜해지는, 몰라도 되는 이야기 :

this 키워드는 깊이 들어가면 **context 객체**라고 부르는 친구와 연관이 있어요.

우리는 함수나 클래스 안에서 사용하면 this를 쓴 위치(위의 경우에는 App 클래스)에 있는 값을 가지고 온다고 생각합시다.

ex) App 클래스 안에서 쓰면, this.[변수명]은 App 클래스 안에 있는 값을 가지고 옵니다.

- App 컴포넌트에서 BucketList 컴포넌트로 state를 넘겨주자

컴포넌트에 props를 넘겨줄 때는 아래와 같이 넘겨줍니다.

```

//App.js

...
render() {
  // this 키워드를 통해 state에 접근할 수 있어요.
  console.log(this.state);

  return (
    <div className="App">
      <h1>내 버킷리스트</h1>
      { /* 컴포넌트를 넣어줍니다. */ }
      { /* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */ }
      <BucketList list={this.state.list}/>
    </div>
  );
}

```

잘 넘어갔는 지 확인해볼까요?

BucketList.js 파일을 열고, 아래와 같이 입력합니다.

```
//BucketList.js

const BucketList = (props) => {

  console.log(props);

  // 컴포넌트가 뿌려줄 ui 요소(리액트 엘리먼트라고 불러요.)를 반환해줍니다.
  return (
    <div>
      버킷 리스트
    </div>
  );
}

export default BucketList;
```

이제 브라우저를 열어 콘솔에 찍히는 지 확인해봅시다.

좋아요! 이제 App에서 BucketList로 데이터를 전달해줬네요! 😊

```
▼ {list: Array(3)} App.js:22
  ► list: (3) ["영화관 가기", "매일 책읽기", "수영 배우기"]
  ► __proto__: Object

▼ {list: {...}} BucketList.js:17
  ► list: {list: Array(3)}
  ► __proto__: Object
```

06. Quiz_버킷리스트 페이지를 만들어봅시다!

▼ 13) 📁 버킷리스트 뼈대 잡기(1)

▼ Q. 퀴즈설명: 코드를 수정해서 아래 모양을 만들어보세요!

내 버킷리스트

영화관 가기
매일 책읽기
수영 배우기

▼ [코드스니펫] - 버킷리스트 컴포넌트 만들기

```
//BucketList.js
// 리액트 패키지를 불러옵니다.
import React from 'react';

// 함수형 컴포넌트는 이렇게 쓸 수도 있고
// function Bucketlist(props){
//   return (
//     <div>버킷 리스트</div>
//   );
// }

// 이렇게 쓸 수도 있어요. =>가 들어간 함수를 화살표 함수라고 불러요.
// 저희는 앞으로 화살표 함수를 사용할거예요.
```

```
// 와 () 안에 props! 부모 컴포넌트에게 받아온 데이터입니다.
// js 함수가 값을 받아오는 것과 똑같이 받아오네요.
const BucketList = ({list}) => {

  // Quiz 1: my_list에 ['a', 'b', 'c'] 대신 부모 컴포넌트가 넘겨준 값을 넣으려면 어떻게 해야할까요?
  const my_lists = ['a', 'b', 'c'];

  // 컴포넌트가 뿌려줄 ui 요소(리액트 엘리먼트라고 불러요.)를 반환해줍니다.
  return (
    <div>
      {
        // js의 내장 함수 중 하나인 map입니다. 리스트의 갯수만큼 => 오른쪽 구문을 반복해요.
        // 자세한 사용법은 아래 링크를 확인해주세요.
        // https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map
        my_lists.map((list, index) => {
          // 콘솔을 확인해봅시다 :)
          console.log(list);
          return (<div key={index}>{list}</div>);
        })
      }
    </div>
  );
}

// 우리가 만든 함수형 컴포넌트를 export 해줍니다.
// export 해주면 다른 컴포넌트에서 BucketList 컴포넌트를 불러다 쓸 수 있어요.
export default BucketList;
```



힌트:

1. 부모 컴포넌트가 넘겨준 값은 props로 받아옵니다.
2. 자바스크립트의 내장함수 중 하나인 Map은 for문하고 비슷한 친구죠!

▼ A. 함께하기(완성본)

▼ [코드스니펫] - 버킷리스트(완성)

```
//BucketList.js
// 리액트 패키지를 불러옵니다.
import React from 'react';

// 함수형 컴포넌트는 이렇게 쓸 수도 있고
// function Bucketlist(props){
//   return (
//     <div>버킷 리스트</div>
//   );
// }

// 이렇게 쓸 수도 있어요. =>가 들어간 함수를 화살표 함수라고 불러요.
// 저히는 앞으로 화살표 함수를 사용할거예요.
// 와 () 안에 props! 부모 컴포넌트에게 받아온 데이터입니다.
// js 함수가 값을 받아오는 것과 똑같이 받아오네요.
const BucketList = (props) => {

  // Quiz 1: my_list에 ['a', 'b', 'c'] 대신 부모 컴포넌트가 넘겨준 값을 넣으려면 어떻게 해야할까요?
  const my_lists = props.list;

  console.log(props);
  // 컴포넌트가 뿌려줄 ui 요소(리액트 엘리먼트라고 불러요.)를 반환해줍니다.
  return (
    <div>
      {
        // js의 내장 함수 중 하나인 map입니다. 리스트의 갯수만큼 => 오른쪽 구문을 반복해요.
        // 자세한 사용법은 아래 링크를 확인해주세요.
        // https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map
        my_lists.map((list, index) => {
          // 콘솔을 확인해봅시다 :)
          console.log(list);
          return (<div key={index}>{list}</div>);
        })
      }
    </div>
  );
}
```

```

    </div>
  );
}

// 우리가 만든 함수형 컴포넌트를 export 해줍니다.
// export 해주면 다른 컴포넌트에서 BucketList 컴포넌트를 불러다 쓸 수 있어요.
export default BucketList;

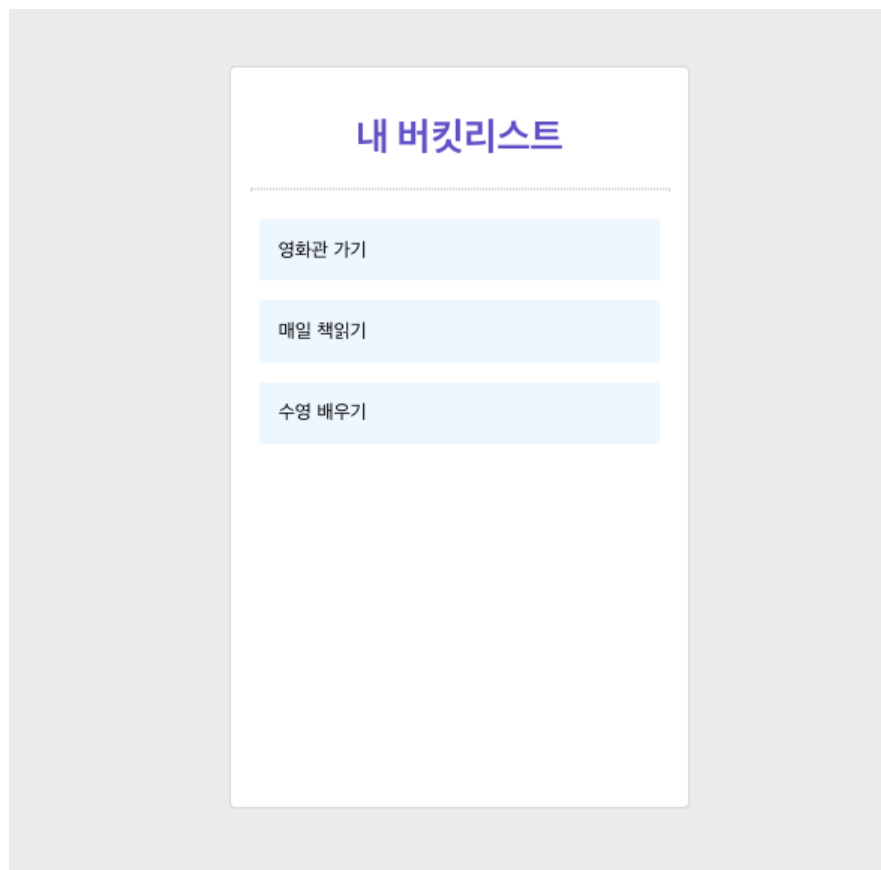
```

07. 화면을 예쁘게! React에서 CSS 사용하기

▼ 14) CSS in React



이번에는 src폴더에 style.css 파일을 만들고 그 안에 스타일을 정의하고, 그 파일을 불러다 사용하는 방법을 배워볼게요!



▼ (1) CSS 파일 만들기



class 대신 className을 주는 거 잊지 않으셨죠?

아래에 만들어진 파일을 공유하지만, 꼭 혼자서도 만들어 보세요.

▼ [코드스니펫] - App.js

```

import React from 'react';
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from './BucketList';

// 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'],
    };
  }

  // 렌더 함수 안에 리액트 엘리먼트를 넣어줍니다!
  render() {
    // this 키워드를 통해 state에 접근할 수 있어요.
    console.log(this.state);

    return (
      <div className="App">
        <div className="container">
          <h1>내 버킷리스트</h1>
          <hr className="line"/>
          {/* 컴포넌트를 넣어줍니다. */}
          {/* <컴포넌트 명 [props 명]={넘겨줄 것(리스트, 문자열, 숫자, ...)}> */}
          <BucketList list={this.state.list} />
        </div>
      </div>
    );
  }
}

export default App;

```

▼ [코드스니펫] - BucketList.js

```

// 리액트 패키지를 불러옵니다.
import React from "react";

// 함수형 컴포넌트는 이렇게 쓸 수도 있고
// function BucketList(props){
//   return (
//     <div>버킷 리스트</div>
//   );
// }

// 이렇게 쓸 수도 있어요. =>가 들어간 함수를 화살표 함수라고 불러요.
// 저희는 앞으로 화살표 함수를 사용할거예요.
// 앳 () 안에 props! 부모 컴포넌트에게 받아온 데이터입니다.
// js 함수가 값을 받아오는 것과 똑같이 받아오네요.
const BucketList = ({ list }) => {

  const my_lists = list;
  // 컴포넌트가 뿌려줄 ui 요소(리액트 엘리먼트라고 불러요.)를 반환해줍니다.
  return (
    <div>
      {
        // js의 내장 함수 중 하나인 map입니다. 리스트의 갯수만큼 => 오른쪽 구문을 반복해요.
        // 자세한 사용법은 아래 링크를 확인해주세요.
        // https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map
        my_lists.map((list, index) => {
          // 콘솔을 확인해봅시다 :)
          console.log(list);
          return (
            <div className="list-item" key={index}>
              {list}
            </div>
          );
        })
      }
    </div>
  );
}

```

```

    </div>
  );
};

// 우리가 만든 함수형 컴포넌트를 export 해줍니다.
// export 해주면 다른 컴포넌트에서 BucketList 컴포넌트를 불러다 쓸 수 있어요.
export default BucketList;

```

▼ [코드스니펫] - style.css

화면과 똑같지 않아도 괜찮아요! 원하는 모양으로 예쁘게 만들어주세요.

```

.App {
  background-color: #eee;
  height: 100vh;
  width: 100vw;
  display: flex;
}

.container{
  background-color: #fff;
  width: 50vw;
  max-width: 350px;
  margin: auto;
  height: 80vh;
  padding: 16px;
  border: 1px solid #ddd;
  border-radius: 5px;
}


.container > h1 {
  color: slateblue;
  text-align: center;
}

.container > .line {
  margin: 16px 0px;
}

.list-item {
  padding: 16px;
  margin: 8px;
  background-color: aliceblue;
}

```

▼ (2) CSS 파일을 가져다 쓰려면?

 import를 하자!

```

//App.js

import React from 'react';
import logo from './logo.svg';
// BucketList 컴포넌트를 import 해줍니다.
// import [컴포넌트 명] from [컴포넌트가 있는 파일경로];
import BucketList from './BucketList';
// 이 import를 통해 css 파일을 불러다 씁니다!
import './style.css';

...

```




엇? App.js에서는 import를 했는데, 왜 BucketList.js에서는 style.css를 import하지 않나요?

- 자식 컴포넌트는 부모 컴포넌트에 속해 있으니까요!
- 즉, 자식 컴포넌트는 return에 들어 있는 리액트 요소를 부모 컴포넌트에 가져다 줍니다.
- 부모와 같은 스타일 파일을 쓸 때는 부모 컴포넌트에만 import하면 됩니다.

08. 화면을 예쁘게! - styled-components

▼ 15) styled-components 설치하기

- 패키지 설치하기

```
yarn add styled-components
```

▼ 16) styled-components란?



컴포넌트 스타일링 기법 중, 제가 가장 좋아하는 방식입니다! 😎

왜 좋아하냐구요? 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!

▼ 17) styled-components를 적용해보기

```
//App.js

import React from "react";
import BucketList from "./BucketList";
import "./style.css";
import styled from "styled-components";

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

    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };
  }

  render() {
    console.log(this.state.list);
    return (
      <div className="App">
        <MyStyled>
          <p>im here!!!</p>
        </MyStyled>
        { /* <div className="container">
          <h1>내 버킷리스트</h1>
          <hr className="line"/>
          <BucketList list={this.state.list} />
        </div> */ }
      </div>
    );
  }
}

// function App() {
```



```
// return (
//   <div className="App">
//     <BucketList/>
//   </div>
// );
// }

// scss 문법 1: 네스팅! 내가 포함하고 있는 요소에 접근할 수 있어요. ex)내 것{ 자식 것: {}}
// scss 문법 2: &는 나 자신!
const MyStyled = styled.div`
  width: 50vw;
  height: 150px;
  background-color: ${props => (props.bg_color ? "red" : "purple")};
  p {
    color: blue;
  }
  &:hover{
    background-color: yellow;
  }
`;


export default App;
```



알면 덜 찜찜한 이야기

 <- 이렇게 생긴 따옴표( 키와 함께 있습니다!)를 백틱이라고 불러요.

09. Quiz_버킷리스트에 styled-components 적용하기

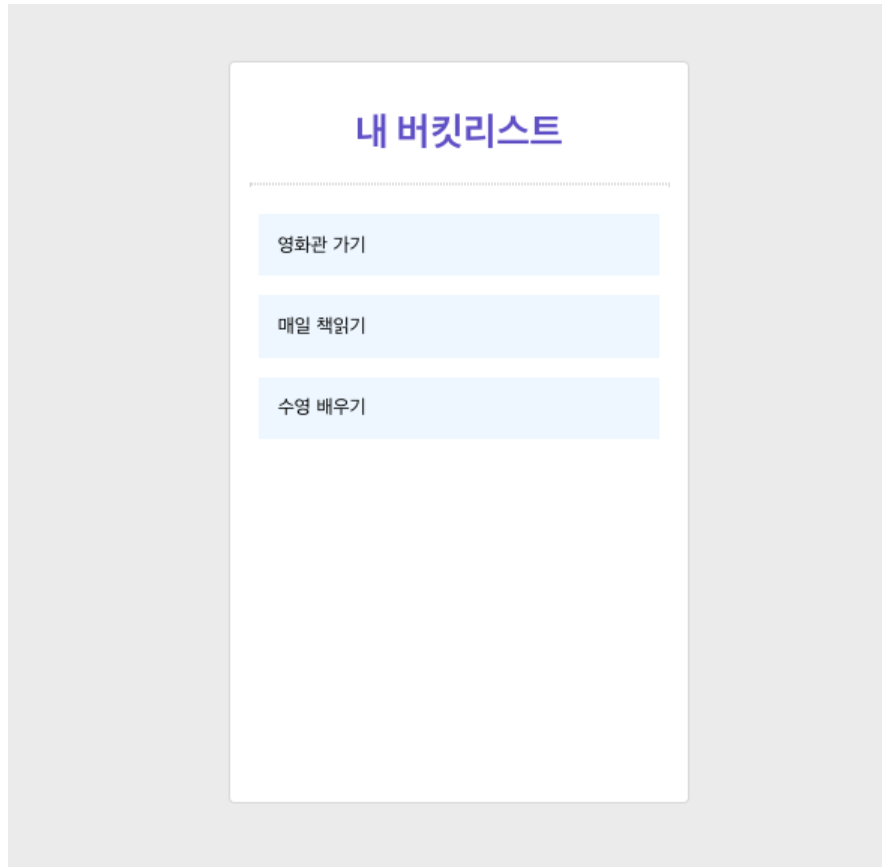
▼ 18)  버킷리스트에 styled-components를 적용해보기



버킷리스트에 입힌 스타일을 styled-components로 싹 바꿔봅시다!

▼ Q. 퀴즈설명

▼ 모습 보기



힌트: 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] - App.js

```
import React from "react";
// BucketList 컴포넌트를 import 해옵니다.
// import [컴포넌트명] 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"],
    };
  }

  // 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] - BucketList.js

```

// 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;

```

10. Ref! 리액트에서 돔요소를 가져오려면?

▼ 19) Ref



이제 컴포넌트와 리액트 요소를 다루는데에 조금 익숙해졌나요?
그런데 만약에, 내가 어떤 인풋박스에서 텍스트를 가져오고 싶으면 어떻게 접근해야할까요?
DOM에서 다루는 방법은 1주차에 배웠는데, 리액트에선 어떻게 하면 좋을까요?
(render()가 끝나고 가져오면 될까요? 아니면 mount가 끝나고?
아니, 그 전에 가상돔에서 가져오나? 아니면 DOM에서? 🤔)
→ 답은, **리액트 요소에서 가져온다!**

▼ React 요소를 가지고 오는 방법1 : React.createRef()

```
//App.js

import React from "react";
// 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;

```

▼ React 요소를 가지고 오는 방법2 : React.useRef()



useRef()는 리액트 훅 중 하나예요. 😊

리액트 혹은 사용방법이 무지무지 간단합니다! 순식간에 끝나버리니까 집중집중!

```

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

const BucketList = ({ list }) => {
  const my_lists = list;
  const my_wrap = React.useRef(null);

  console.log(my_wrap); // 콘솔로 확인해봐요!

  window.setTimeout(() => { // 1초 뒤에는?!
    console.log(my_wrap);
  }, 1000);

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

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

export default BucketList;

```

11. State 관리 (1)

▼ 20) 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 리렌더링 될 거구요. 😞

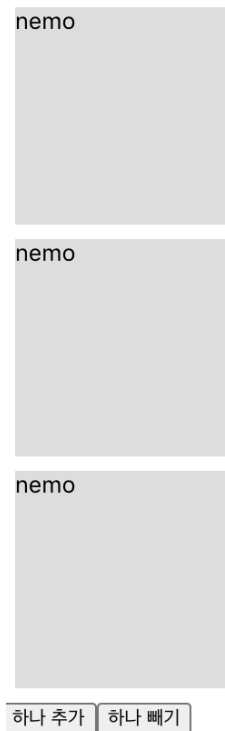
▼ 21) 클래스형 컴포넌트에서 state 관리 - setState()

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

- 새 프로젝트를 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.)



⚠️ 이번까지는 클래스형 컴포넌트를 쓸 거예요. App.js를 클래스형으로 바꾸는 게 조금 귀찮더라도, 같이 해봅시다. 둘 다 할 줄 알면 박박 좋으니까요! 😎

▼ (1) 새 CRA 만들기

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

▼ (2) 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;
```

▼ (3) 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, 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>
    );
  }
}
```



```

    );
  })}
</div>
);
}
}

export default App;

```

▼ (4) 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>
);

```

▼ (5) 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!');
  }
};

render(){ ... }

```

▼ (6) 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>
);

```

▼ (7) 완성본 코드

▼ [코드스니펫] - 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 없겠조! 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>
    );
  }
}

```

```

        </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;

```

12. State 관리 (2)

▼ 22) 함수형 컴포넌트에서 state 관리 - useState()

👉 이번에는 함수형 컴포넌트에서 어떻게 state를 쓸 수 있는 지 봅시다.
 함수형 컴포넌트는 클래스형처럼 자체적으로 state를 가지고 있지 않지만, 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 껍데기 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

```
//Nemo.js
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);
  return null;
};

export default Nemo;
```

▼ (4) 뷰를 만들고(=반환할 리액트 요소를 만들고),

```
//Nemo.js

...
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,

```
//Nemo.js

...
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);
};

return ( ... )
```

▼ (6) return (연결하자!

```
//Nemo.js

...
<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. 헛갈리지 않도록 꼭 손에 익혀야해요!

특히 제가 컴포넌트를 만들고 → state를 쓰는 순서! 뷰 먼저 → 그 다음은 state를 만들고(기본값도 잡아주고!) → state를 조작하는 무언가를 만들어서 → 연결한다!

이 순서가 중요해요!

13. 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 책읽기

수영 배우기

다시 **week-1** 폴더로 가서 코드를 작성해봅시다.

▼ [코드스니펫] - App.js

```
import React from "react";
import BucketList from "../BucketList";
import "./style.css";
import styled from "styled-components";

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

    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };

    this.text = React.createRef();
  }

  componentDidMount() {}

  render() {

    return (
      <div className="App">
        <Container>
          <Title>내 버킷리스트</Title>
```

```

        <Line />
        <BucketList list={this.state.list} />
      </Container>

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

const Container = styled.div`
  background-color: #fff;
  width: 50vw;
  max-width: 350px;
  margin: auto;
  height: 80vh;
  padding: 16px;
  border: 1px solid #ddd;
  border-radius: 5px;
`;

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

const Line = styled.hr`
  margin: 16px 0px;
`;

export default App;

```

▼ [코드스니펫] - BucketList.js

```

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

const BucketList = ({ list }) => {
  const my_lists = list;
  const my_wrap = React.useRef(null);

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

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] - 버킷리스트 추가하기 App.js 완성본

```
import React from "react";
import BucketList from "../BucketList";
import styled from "styled-components";

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

    this.state = {
      list: ["영화관 가기", "매일 책읽기", "수영 배우기"],
    };

    this.text = React.createRef();
  }

  componentDidMount() {}

  addBucket = () => {
    console.log(this.text.current.value);
    const new_item = this.text.current.value;
    // ... => 스프레드 문법
    // [...this.state.list, 넣고 싶었던 어떤 값]
    this.setState({ list: [...this.state.list, new_item] });
  }

  render() {
    return (
      <AppWrap className="App">
        <Container>
          <Title>내 버킷리스트</Title>
          <Line />
          <BucketList list={this.state.list} />
        </Container>

        <InputWrap>
          <input type="text" ref={this.text} />
          <button onClick={this.addBucket}>추가하기</button>
        </InputWrap>
      </AppWrap>
    );
  }
}

const AppWrap = styled.div`
  background-color: #eee;
  height: 100vh;
  width: 100vw;
  display: flex;
  flex-direction: column;
`;

const Container = styled.div`
  background-color: #fff;
  width: 50vw;
  max-width: 350px;
  margin: auto;
  height: 80vh;
  padding: 16px;
  border: 1px solid #ddd;
`;
```



```

    border-radius: 5px;
  `;

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

  const Line = styled.hr`
    margin: 16px 0px;
  `;

  const InputWrap = styled.div`
    background-color: #fff;
    width: 50vw;
    max-width: 350px;
    margin: auto;
    padding: 16px;
    border: 1px solid #ddd;
    border-radius: 5px;
  `;

  export default App;

```

14. 끝 & 숙제 설명



아래 기획서를 보고, 퀴즈 시작하기 화면을 만들어보세요!

<App/> 아래에 <Start/>라는 자식 컴포넌트를 만들어서 해봅시다!

→ 숙제할 때는 꼭꼭 클래스형 컴포넌트를 쓸 필요 없어요! (해설 코드는 함수형 컴포넌트로 나갑니다.)

▼ 기획서(레이아웃) 보기

시작하기

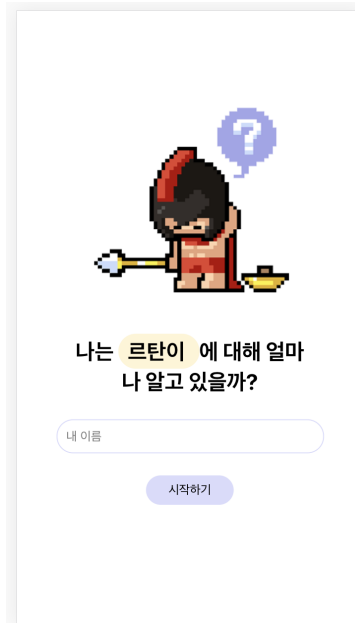
나는 이범규 에 대해서

얼마나 알고 있을까?

내 이름

시작하기

▼ 예시 화면



- 우리가 배운 내용으로 위 페이지를 만들어봅시다.
이미지도 넣어보고, 내 친구가 이름을 넣을 텍스트 입력 인풋과 시작하기 버튼을 만들어요.
[나는 {} 에 대해서...]부분에 {}는 state에 넣고 prop로 넘겨서 해보세요!

HW. 2주차 숙제 답안 코드

▼ [코드스니펫] - 2주차 숙제 답안 코드

전체 코드

▼ 이미지 파일



▼ App.js

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

```
import './App.css';
import Start from './Start';

function App() {
  const [name, setName] = React.useState("르탄이");

  return (
    <div className="App" style={{
      maxWidth: "350px",
      margin: "auto"
    }}>
      <Start name={name}/>
    </div>
  );
}

export default App;
```

▼ Start.js

```
import React from "react";
import img from "./scc_img01.png";
const Start = (props) => {
  console.log(props);
  return (
    <div
      style={{
        display: "flex",
        height: "100vh",
        flexDirection: "column",
        alignItems: "center",
        justifyContent: "center",
        padding: "16px",
        boxSizing: "border-box",
      }}
    >
      <img
        src={img}
        style={{
          width: "60vw",
          margin: "16px",
        }}
      />
      <h1 style={{ fontSize: "1.5em", lineHeight: "1.5" }}>
        나는 <span style={{
          backgroundColor: "#fef5d4",
          padding: "5px 10px",
          borderRadius: "30px"
        }}>{props.name}</span>에 대해 얼마나 알고 있을까?
      </h1>
      <input style={{
        border: "1px solid #dadafc",
        borderRadius: "30px",
        padding: "10px",
        width: "100%"
      }}/>
      <button style={{
        padding: "10px 36px",
        backgroundColor: "#dadafc",
        border: "#dadafc",
        borderRadius: "30px",
        margin: "36px 0px"
      }}>시작하기</button>
    </div>
  );
};

export default Start;
```

- 이미지는 src 폴더에 넣고 불러와볼게요!

Copyright © TeamSparta All rights reserved.