

React

REACT -

목차

01. REACT	5
> 01. react기초	5
> 02. 리액트 설치	10
> 03. JSX의 기본구조	35
> 04. 관리프로그램 GUI만들기	46
> 05.prop를 이용해 관리자 프로그램 업그레이드	52
1. 변수 선언 키워드	55
1) var (ES5 및 이전 버전)	55
2) let (ES6 도입)	55
3) const (ES6 도입)	56
> 06.화살표함수	57
3. 매개변수가 없을 때	59
4. 함수 본문이 하나의 표현식일 때	59
5. 함수 본문이 여러 줄일 때	59
JavaScript 변수 선언 키워드와 함수 범위(Function Scope) 관련 내용	61
1. 함수 범위(Function Scope)란?	61
2. var는 "함수 범위"를 가짐	62
3. let과 const는 "블록 범위(Block Scope)"를 가짐	62
4. 함수 내부에서 var, let, const 차이점 정리	63
5. 함수 내부에서 선언하지 않은 변수 (자동 전역 변수)	63
📌 1. 배열 선언 및 초기화	65
✓ 배열을 생성하는 방법	65
📌 2. 배열 요소 접근 및 변경	65
📌 3. 배열의 주요 속성과 메서드	66
◆ 배열 길이 (length)	66
◆ 배열 끝에 요소 추가 (push) / 삭제 (pop)	66
◆ 배열 앞에 요소 추가 (unshift) / 삭제 (shift)	66
◆ 특정 위치에 요소 추가/제거 (splice)	67
📌 4. 배열 순회 (반복문)	67
◆ for 반복문	67
◆ forEach() 메서드	67
◆ for...of 반복문 (ES6)	68
📌 5. 배열 변형 및 활용	68
◆ map() - 배열의 각 요소 변환	68
◆ filter() - 조건에 맞는 요소만 추출	68
◆ reduce() - 배열의 값을 하나로 합치기	68
📌 6. 배열과 문자열 변환	69

◆ join() - 배열을 문자열로 변환	69
◆ split() - 문자열을 배열로 변환	69
📌 7. 배열 정렬	69
◆ 오름차순 정렬 (sort())	69
◆ 내림차순 정렬	69
✓ 정리	70
✓ JSON (JavaScript Object Notation) 기초 설명	78
📌 1. JSON의 특징	78
📌 2. JSON 문법	78
✓ JSON 데이터 구조	78
✓ JSON 기본 예제	78
📌 3. JSON 데이터 타입	79
📌 4. JSON과 JavaScript 객체 변환	79
✓ JavaScript 객체 → JSON 변환 (JSON.stringify())	79
✓ JSON → JavaScript 객체 변환 (JSON.parse())	80
✓ 공통점	80
● 차이점	81
> 07. 이벤트 추가하기	82
◆ 1. 버튼 클릭 이벤트 (onClick)	85
◆ 2. 입력 필드 변경 이벤트 (onChange)	86
◆ 3. 마우스 이벤트 (onMouseEnter, onMouseLeave)	87
◆ 4. 키보드 이벤트 (onKeyDown, onKeyUp)	88
◆ 5. 폼 제출 이벤트 (onSubmit)	89
⌚ 정리	90
배열 구조 분해 할당 (Destructuring Assignment)	102
1. 기본 개념	102
예제	102
2. useState에서의 배열 구조 분해 할당	102
이와 동일한 동작을 하는 코드 (배열 구조 분해 할당 없이)	103
3. 배열 구조 분해 할당의 특징	103
1) 필요한 값만 할당 가능	103
2) 기본값 설정 가능	103
4. 정리	103
> 08.STATE 개념 이해하기	105
> 09.GUI CREATE,UPDATE 추가	115
> 09.GUI 최종(DELETE 추가)	122
> 10.컴포넌트에 값 세팅하기 (READ변경)	130
> 10.Create변경하기	140
> 11. UPDATE변경	143
> 12.Delete변경하기	159
> 13. 최종 결과	162
스프레드 연산자 (...)란?	198
1. 배열에서 ... 스프레드 연산자 사용	198
✓ 배열 복사	198

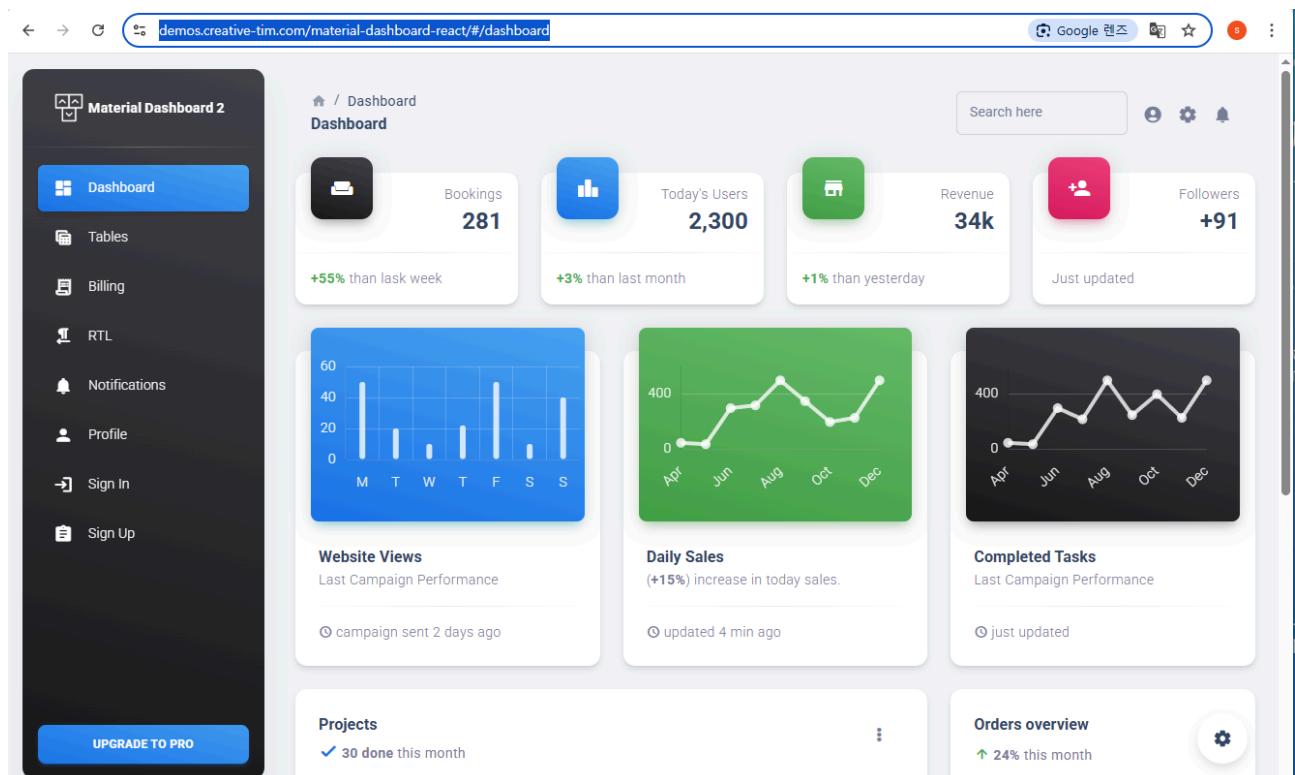
✓ 배열에 새로운 요소 추가	198
✓ 배열 합치기	199
2. 객체에서 ... 스프레드 연산자 사용	199
✓ 객체 복사	199
✓ 객체 속성 추가 및 업데이트	199
🚀 핵심 정리	200
> 14. 다양한 흑 사용법	201
기본 흑	201
설명	211
기본 문법	212
주요 개념	212
사용 예제	212
1. GET 요청	212
2. POST 요청	213
3. Error Handling	214
비동기/대기 (async/await)	214
결론	215
사용자 정의 흑 (Custom Hook)	217
주요 포인트:	218
> 15. 단어장 만들기	220
> 16.	221
> 17.쉽게 개선선	221
> 18.마지막 학생생	228
가장 간단한 조건부 렌더링 예제	229
동작 설명:	229
핵심 원리:	229
> 19.	251
> 20.	252
> 21.	252
> 22.	252
> 23.	252
> 24.	252
> 25.	252
> 26.	252
> 27.	253
> 28.	253
> 29.	253
> 30.	253

01. REACT

> 01. react 기초

React는 사용자 인터페이스(UI)를 구축하기 위한 JavaScript 라이브러리

<https://demos.creative-tim.com/material-dashboard-react/#/dashboard>



jQuery는 주로 DOM 조작과 이벤트 처리에 중점을 둔 라이브러리이며, **Angular**, **React**, **Vue**는 더 현대적인 프레임워크나 라이브러리로 복잡한 애플리케이션 구조를 관리하고 상태를 처리하는 데 중점을 둡니다.

Angular, React, Vue와의 비교

- **모듈화:** **Angular**, **React**, **Vue**는 컴포넌트 기반 아키텍처를 지원하여 코드의 재사용성과 유지 보수성을 높입니다. **jQuery**는 주로 직접 DOM 조작에 의존합니다.
- **상태 관리:** **Angular**, **React**, **Vue**는 복잡한 상태 관리와 데이터 바인딩을 지원합니다. **jQuery**는 상태 관리 기능이 부족합니다.
- **테스트:** **Angular**, **React**, **Vue**는 테스트 용이성을 고려하여 설계되었으며, **jQuery**는 테스트 관련 기능이 제한적입니다.
- **프레임워크/라이브러리의 목표:** **Angular**, **React**, **Vue**는 전체 애플리케이션 구조를 관리하기 위해 설계된 반면, **jQuery**는 특정 기능을 간편하게 구현하는 데 중점을 둡니다.

angular가 1등 이었으나 복잡한 사용법으로 **react**에게 1위 자리를 빼았겼고, 현재 클래스 기반 복잡한 **react** 프로그램 개발에서 함수 기반 **react** 프로그램 방식으로 변경 되었다.

더 쉬운 사용 방법으로 뷰가 확장세를 넓혀가고 있는 실정이고, 아직 **React**은 어려운 프로그램에 속한다. 앞으로 많은 변화가 예상되므로, 깊이 있는 공부 보다는 심플한 형태로 개발하는 것이 낫다.

React는 실질적으로 웹에서 **싱글 페이지 애플리케이션(SPA)**을 만드는 용도로 사용된다. 주로 **동적인 사용자 인터페이스(UI)**를 구축하는 데 활용되며, 웹에서 실행되는 애플리케이션 제작에 사용된다.

1. React는 웹 애플리케이션을 컴포넌트 기반으로 개발하는 라이브러리

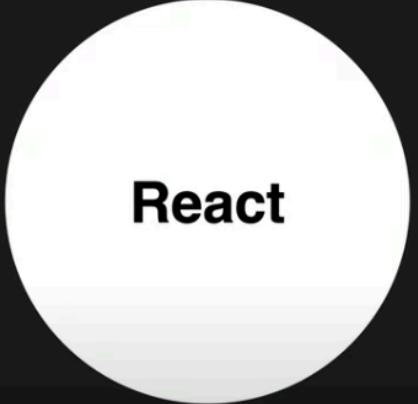
```
<header>
  <h1>
    <a href="/">web</a>
  </h1>
</header>
<nav>
  <ol>
    <li><a href="/read/1">html</a></li>
    <li><a href="/read/2">css</a></li>
    <li><a href="/read/3">javascript</a></li>
  </ol>
</nav>
<article>
  <h2>Welcome</h2>
  Hello, WEB
</article>
```

상위과 같은 html 코드 하위 이미지 같이 매핑 시켜서 컴포넌트로 만든다.

```
<Header></Header>    ==> <header>
                                <h1>
                                    <a href="/">web</a>
                                </h1>
                            </header>
<Nav></Nav>        ==> <nav>
                                <ol>
                                    <li><a href="/read/1">ht
                                    <li><a href="/read/2">cs
                                    <li><a href="/read/3">jav
                                </ol>
                            </nav>
<Article></Article> ==> <article>
                                <h2>Welcome</h2>
                                Hello, WEB
                            </article>
```

리액트를 사용하면 심플하게 컴포넌트화 해서 화면과 프로그램을 만들 수 있다.

```
<Header></Header>
<Nav></Nav>
<Article></Article>
```

The React logo is a white circle with the word "React" written in a bold, black, sans-serif font.

React

class vs function

과거에 리액트는 클래스 형태로 만들어졌는데 최근에는 함수 형태로 만들어져 더욱 쉬워져서 더 많이 사용하게 되었다. 1-2년 전 책들만 하더라도 모두 클래스 기반 리액트로 구현되어 있다.

리액트 자습서

<https://ko.legacy.reactjs.org/>

리액트 에디터

<https://stackblitz.com/edit/react-ldxeqc?file=src%2FApp.js>

> 02. 리액트 설치

✓ npm과 npx 는 ? 🚀

- ◆ 1. npm (Node Package Manager)

👉 패키지를 설치하고 관리하는 도구
(**Node.js**와 함께 제공됨)

✓ 하는 일 :

- 패키지(라이브러리) 설치, 업데이트, 삭제
- 프로젝트의 의존성 관리 (**package.json**)
- 전역(-g) 또는 로컬(**node_modules**)에 패키지 설치

✓ 예제:

```
npm install express      # 프로젝트에 express 설치  
npm install -g nodemon # 전역(Global)으로 nodemon 설치  
npm update react        # 설치된 react 패키지 업데이트  
npm uninstall axios     # axios 패키지 제거
```

◆ 2. npx (Node Package Executor)

👉 설치 없이 패키지를 실행하는 도구
(npm 5.2.0부터 기본 포함됨)

✓ 하는 일:

- 패키지를 설치하지 않고 바로 실행
- 특정 버전의 패키지를 다운로드하여 실행
- 프로젝트 내 패키지를 실행 (node_modules/.bin 대신 사용)

✓ 예제:

```
npx create-react-app my-app # 설치 없이 React 프로젝트 생성  
npx eslint myfile.js       # eslint를 설치 없이 실행  
npx cowsay "Hello World!" # 설치 없이 실행 후 메시지 출력
```

🚀 결론 (차이점 요약)

구분

npm

npx

역할

패키지 설치 및 관리

패키지를 설치 없이 실행

설치 필요 여부 패키지를 설치해야 사용 가능 설치 없이 바로 실행 가능

사용 예시

`npm install axios`

`npx create-react-app`

`my-app`

- ✓ `npm`은 패키지를 설치하고 관리하는 도구
- ✓ `npx`는 설치 없이 패키지를 바로 실행하는 도구

😊 쉽게 말해서

- 📦 `npm` = "패키지 설치"
- ⚡ `npx` = "설치 없이 실행"

리액트에서 NPM과 NPX 사용 용도

1. `npm`을 사용하는 경우

`npm`은 주로 패키지 관리와 관련된 작업을 처리합니다. **React** 프로젝트에서는 주로 다음과 같이 사용됩니다:

- ◆ 패키지 설치
 - **React**와 관련된 라이브러리나 도구들을 프로젝트에 설치할 때 사용됩니다.
 - 예: **React**, **React DOM**, 기타 필요한 라이브러리들을 설치

`npm install react react-dom # React와 ReactDOM을 설치`

`npm install axios # HTTP 요청을 위한 axios 설치`

```
npm install react-router-dom # React에서 라우팅을 위한 라이브러리 설치
```

- ◆ 의존성 관리

- 설치한 패키지는 **node_modules** 폴더에 저장되고, 이 정보는 **package.json**에 기록됩니다. 이 파일을 통해 나중에 다른 개발자가 프로젝트를 사용할 때 동일한 패키지를 설치할 수 있게 됩니다.

```
npm install # `package.json`에 기록된 모든 패키지 설치
```

- ◆ 스크립트 실행

- **npm**을 사용하여 프로젝트 내에서 정의된 스크립트를 실행할 수 있습니다. 예를 들어, **start**, **build** 등의 명령어를 **package.json**의 **scripts** 섹션에 정의하여 실행합니다.

```
npm start # React 애플리케이션을 실행 (개발 모드)
```

```
npm run build # 배포용 빌드 파일을 생성
```

2. **npx**를 사용하는 경우

npx는 설치 없이 패키지를 바로 실행하는 데 사용됩니다. **React** 프로젝트에서 주로 사용되는 경우는 **create-react-app** 같은 초기 설정을 할 때입니다.

- ◆ React 프로젝트 생성

- **React** 애플리케이션을 처음 시작할 때 **npx**를 사용하여 ****create-react-app****을 실행합니다. **create-react-app**은 **React** 프로젝트를 쉽게 설정해주는 도구로, 이 도구를 설치하지 않고도 **npx**로 실행할 수 있습니다.

```
npx create-react-app my-app # `create-react-app`을 실행하여 새로운 React 프로젝트 생성
```

이 명령어는 `create-react-app`을 전역에 설치하지 않고도 바로 사용할 수 있게 해줍니다. (즉, 매번 최신 버전의 `create-react-app`을 실행할 수 있게 해줍니다.)

- ◆ `npx`로 실행되는 기타 도구들

- `npx`는 다른 많은 도구들도 설치 없이 실행할 수 있게 해줍니다. 예를 들어, `eslint`, `prettier`, `storybook` 등을 설치 없이 실행할 수 있습니다.

```
npx eslint myfile.js # eslint를 설치 없이 실행하여 코드 검사
```

```
npx prettier --write . # prettier를 설치 없이 실행하여 코드 포맷팅
```

- ◆ React 개발 시 `npm`과 `npx` 사용 요약

기능	<code>npm</code> 사용	<code>npx</code> 사용
프로젝트 생성	-	<code>npx create-react-app my-app</code>
라이브러리 설치	<code>npm install react react-dom</code>	-
프로젝트 실행 (개발 모드)	<code>npm start</code>	-
빌드	<code>npm run build</code>	-

패키지 관리

npm install <패키지>

-

설치 없이 패키지 실행

-

npx <패키지>



결론

- **npm**: 패키지 설치, 관리, 실행에 사용됩니다. **React** 라이브러리와 도구들을 설치할 때 주로 사용합니다.
- **npx**: 설치 없이 패키지 실행에 사용됩니다. **React** 프로젝트를 시작할 때 **create-react-app**을 실행하거나, 일시적으로 실행할 도구들을 사용할 때 유용합니다.

둘 다 **React** 개발을 할 때 상호 보완적으로 사용됩니다! 😊

npm과 npx 설치하기

node를 설치하면 자동으로 npm과 npx가 자동 설치된다.

nodejs.org

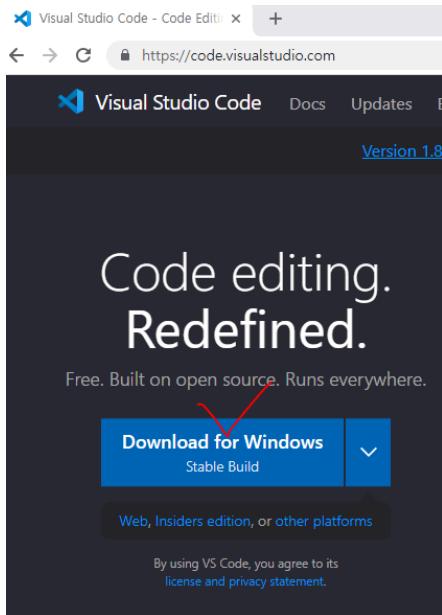
비주얼 스튜디오 코드 터미널에서 아래 이미지처럼 개발 환경을 만들어 보자
윈도우+R cmd node 입력해서 설치 여부 확인

node버전이 출력되면 node가 정상 설치된것이고 npm과 npx를 사용할 수 있는 상태이다.

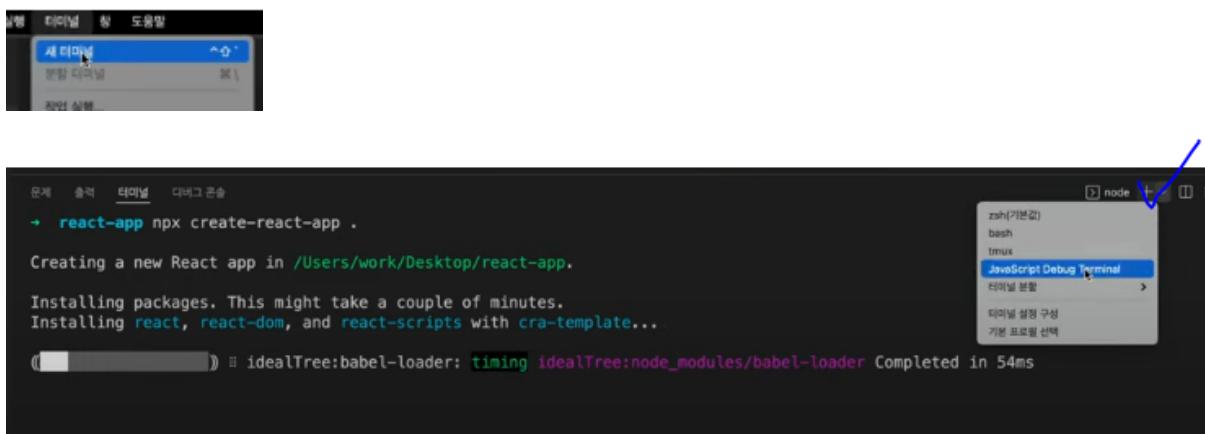
사용중 문제 발생시 관리자 권한으로 실행

visual studio code 설치

<https://code.visualstudio.com/>



바탕화면에 react폴더를 만들고 해당 폴더를 visual studio code에서 선택한 다음 터미널에 새터미널로 연다.



새터미널에서 다음과 같은 npx를 사용한다. 터미널에 문제가 발생하면 왼쪽 상단에서 다른 것을 선택해서 해보자. 파워셀이 제대로 동작하지 않는 경우가 있다. 되도록 관리자 권한으로 실행, 리부팅, 캐시정리, npm재설치 등을 진행해 보자.

- npx로 **create-react-app** 새 프로젝트 생성하기

react 폴더 안의 ch01 폴더를 만들고 터미널로 해당 폴더로 이동한다.

```
cd ch01
```

바탕화면에 폴더를 만들고 이동해서 npx create-react-app . 으로 현 폴더에 react 프로젝트를 생성해 보자.

```
npx create-react-app . 현재 폴더에 설치
```

```
npx create-react-app ch01 ch01에 설치
```

비주얼 스튜디오 코드를 관리자 권한으로 실행했는데도 관리자 권한 문제 문제가 날수 있다.

```
PS C:\Users\park\Desktop\react> npx create-react-app
npx : 이 시스템에서 스크립트를 실행할 수 없으므로 C:\Program Files\nodejs\npx.ps1 파일을 로드할 수 없습니다. 자세한 내용은 about_Execution_Policies(https://go.microsoft.com/fwlink/?LinkID=135170)를 참조하십시오.
위치 줄:1 문자:1
+ npx create-react-app
+ ~~~
+ CategoryInfo          : 보안 오류: (: [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\park\Desktop\react> Get-ExecutionPolicy -List
```

관리자 권한으로 Visual Studio Code를 실행해도 해결되지 않았다면, 다른 방법을 시도해 볼 수 있습니다.

1. PowerShell에서 실행 정책 확인:

먼저 PowerShell에서 현재 실행 정책을 확인하려면 다음 명령어를 입력합니다:

```
Get-ExecutionPolicy -List
```



2. 실행 정책을 변경:

RemoteSigned로 설정되어 있어야 npx가 실행됩니다. 실행 정책을 변경하려면 아래 명령어를 실행하세요:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

- - 그런 다음, Y를 입력하여 변경을 승인합니다.

3. 정책 변경 후 Visual Studio Code 다시 실행:

- 실행 정책을 변경한 후, Visual Studio Code를 종료하고, 다시 관리자 권한으로 열어서 터미널에서 npx create-react-app 명령어를 실행해 보세요.

버전에 문제가 발생하면 다음과 같은 에러가 날 수 있다.

```
PS C:\Users\park\Desktop\react> npx create-react-app ch01
You are running `create-react-app` 5.0.1, which is behind the latest release (5.1.0).
We recommend always using the latest version of create-react-app if possible.

The latest instructions for creating a new app can be found here:
https://create-react-app.dev/docs/getting-started/

npm notice New major version of npm available! 9.5.1 -> 11.2.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.2.0
npm notice Run npm install -g npm@11.2.0 to update!
npm notice
```

node.org에서 node 프로그램을 최신으로 설치한 후 해보자. 설치 불가능 시 다음 방법을 사용해 보자.

그래도 해결이 안되면 다음 방법을 참고해 보자.

메시지는 정보 제공과 몇 가지 제안이 포함된 내용입니다:

create-react-app 버전이 오래됨: 현재 사용 중인 **create-react-app** 버전은 5.0.1로, 최신 버전인 5.1.0보다 뒤쳐져 있습니다. 최신 버전을 사용하는 것이 좋습니다. 글로벌로 **create-react-app**을 업데이트하려면 아래 명령어를 실행하세요:

```
npm install -g create-react-app
```

npm의 새 버전: **npm**의 새로운 주요 버전(11.2.0)이 출시되었습니다. **npm**을 업데이트하려면 다음 명령어를 실행하세요:

```
npm install -g npm@11.2.0
```

작업중 다음과 같은 에러가 날수 있다.

```
4 packages are looking for funding
  run `npm fund` for details
PS C:\Users\park\Desktop\react> npm install -g npm@11.2.0
npm ERR! code EBADENGINE
npm ERR! engine Unsupported engine
npm ERR! engine Not compatible with your version of node/npm: npm@11.2.0
npm ERR! notsup Not compatible with your version of node/npm: npm@11.2.0
npm ERR! notsup Required: {"node":"^20.17.0 || >=22.9.0"}
npm ERR! notsup Actual: {"npm":"9.5.1","node":"v18.16.1"}

npm ERR! A complete log of this run can be found in:
npm ERR!     C:\Users\park\AppData\Local\npm-cache\_logs\2025-03-10T16_35_10_349Z-debug-0.log
PS C:\Users\park\Desktop\react> []
```

1. Node.js 최신 버전 설치:

- [Node.js 공식 웹사이트](#)에서 최신 LTS 버전(현재는 20.x 이상)을 다운로드하여 설치합니다.

2. npm 업데이트:

Node.js를 최신 버전으로 업데이트한 후, 아래 명령어로 npm을 업데이트합니다:

```
npm install -g npm@11.2.0
```

이렇게 하면 npm 버전 호환성 문제가 해결되고, 최신 버전으로 업데이트할 수 있습니다.

```
→ react-app npx create-react-app .

Creating a new React app in /Users/work/Desktop/react-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

```
cd ch01
npm start
```

설치가 끝나고 `npm start`을 입력하면 우리가 만든 리액 페이지가 3000포트로 접속할 수 있도록 실행된다.

```
문제    출력    터미널    디버그 콘솔
? Something is already running on port 3000. Probably:
  /usr/local/Cellar/node/16.6.2/bin/node /Users/work/dev/project/seoma
d 79533)
  in /Users/work/dev/project/seomal/static/app

Would you like to run the app on another port instead? > (Y/n)
```

이미 사용하고 있는 포트가 있으면 상의 이미지 y를 눌러서 새로운 포트번호를 자동발급되거나 입력받을수 있다.

정상적으로 설치되었다면 브라우저에 다음과 같은 이미지를 확인할 수 있다.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Compiled successfully!
You can now view ch01 in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.35.106:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
webpack compiled successfully
```

상위처럼 실행 중이 아니라면 localhost:3000 주소 요청시 에러 페이지가 나온다.

컨트롤 +c 를 이용해서 실행중인 서버를 종단 할 수 있다.

요청 에러 발생시 현재 서버가 동작중인지 확인해 보자.

문제 발생시 다음을 확인해 보자.

예러시 확인해볼 사항 관리자 권한인지 확인해 본다.

한글폴더가 있는지 확인해 본다.

리부팅하고 실행해 본다.

npm 캐시 정리

npm 캐시를 정리해서 캐시와 관련된 문제를 해결할 수 있습니다.

npm cache clean --force

npm 재설치

npm을 재설치해서 문제를 해결할 수 있습니다.

npm install -g npm

visual studio 의 ch01파일을 열어 다음 파일들을 확인해 보자.

React 프로젝트의 **public** 폴더는 애플리케이션에서 외부 파일을 저장하고 제공하는 용도로 사용됩니다. 이 폴더에 있는 파일들은 빌드 프로세스를 거치지 않고 그대로 배포되며, 클라이언트에서 직접 접근할 수 있습니다. 주요 용도는 다음과 같습니다:

- 정적 파일 저장: 이미지, 폰트, 아이콘 등 React 컴포넌트와 별도로 관리되는 파일을 저장합니다.
- 직접 접근 가능: URL을 통해 외부에서 직접 접근할 수 있습니다

`index.html`이 실제 시작 실행 파일이다. `index.html`에 `<div id="root"></div>`가 다른 파일(`index.js`)에서 렌더링된 `html`이 들어 가는 부분이다. 핵심 파일 이지만 사용할 일이 없는 파일이다.

```
EXPLORER ... index.html < ch01 > public > index.html > html > head
REACT-APP
ch01
  node_modules
  public
    favicon.ico
  index.html
  logo192.png
  logo512.png
  manifest.json
  robots.txt
src
.gitignore
package-lock.json
package.json
README.md

22  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
23  work correctly both with client-side routing and a non-root public URL.
24  Learn how to configure a non-root public URL by running `npm run build`.
25
26  -->
27  <title>React App</title>
28  </head>
29  <body>
30    <noscript>You need to enable JavaScript to run this app.</noscript>
31    <div id="root"></div>
32    <!--
33      This HTML file is a template.
34      If you open it directly in the browser, you will see an empty page.
35
36      You can add webfonts, meta tags, or analytics to this file.
37      The build step will place the bundled scripts into the <body> tag.
38
39      To begin the development, run `npm start` or `yarn start`.
40      To create a production bundle, use `npm run build` or `yarn build`.
41    -->
42  </body>
43 </html>
44
```

index.html: React 애플리케이션의 기본 HTML 템플릿, **root** div를 포함.

index.js: React 앱을 브라우저에 렌더링하는 엔트리 포인트.

App.js: 애플리케이션의 주요 컴포넌트로, UI와 로직을 정의.

App.js를 다음과 같이 변경해 보자.

```
import logo from './logo.svg';
import './App.css';
function App() {
  return (
    <div className="App">
      Hello React?
    </div>
  );
}
export default App;
```

```

index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import App from './App';
4 import reportWebVitals from './reportWebVitals';
5
6 ReactDOM.render(
7   <React.StrictMode>
8     <App />
9   </React.StrictMode>,
10  document.getElementById('root')
11 );
12
13 // If you want to start measuring
14 // performance in your app, pass a
15 // function to log results (for example:
16 // reportWebVitals(console.log))
17 // or send to an analytics endpoint. Learn
18 // more: https://bit.ly/CRA-vitals
19 reportWebVitals();

```

```

App.js
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       Hello React!
8     </div>
9   );
10
11 export default App;
12
13

```

React App

Hello React!

Elements

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root">

<div class="App">Hello React!</div> == \$0

</div>

<!--

This HTML file is a template.

If you open it directly in the browser,

you will see an empty page.

You can add webfonts, meta tags, or

analytics to this file.

The build step will place the bundled

scripts into the <body> tag.

To begin the development, run `npm

start` or `yarn start`.

To create a production bundle, use `npm

run build` or `yarn build`.

-->

html body div#root div.App

Styles Computed Layout Event Listeners

```

index.js
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6
7 ReactDOM.render(
8   <React.StrictMode>
9     <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12  );
13
14 // If you want to start measuring
15 // performance in your app, pass a
16 // function to log results (for example:
17 // reportWebVitals(console.log))
18 // or send to an analytics endpoint.
19 // Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();

```

```

App.js
1 import logo from './logo.svg';
2 import './App.css'; // Circled in red
3
4 function App() {
5   return (
6     <div className="App">
7       Hello React!
8     </div>
9   );
10
11 export default App;
12
13

```

React App

Hello React!

Elements Console

<script defer src="/static/js/bundle.js"></script>

<style>...</style>

<!--

This HTML file is a template.

If you open it directly in the browser,

you will see an empty page.

html head style

css를 사용하고 싶다면 해당 css안에서 작업해야 한다.

index.css와 **app.css**는 React 프로젝트에서 스타일을 관리하는 데 사용

index.css

전체 애플리케이션에 적용되는 전역 스타일을 정의하는 데 사용됩니다.

글로벌하게 적용되므로, 어느 컴포넌트에서든지 이 스타일이 적용됩니다.

app.css

App 컴포넌트에 특정한 스타일을 정의하는 데 사용됩니다.

코드 정리방법

ctrl+ A 모두 선택

alt +shift + f 코드정리

index.css에 기술하면 app.js에서 div 색상이 변경된다. 모든 html 파일의 태그에 css가 적용된다.

```
div{  
    background-color: blue;  
}
```

app.css에 기술하면 app.js에서 만 적용 된다.

```
div{  
    background-color: red;  
}
```

모듈 시스템 3가지 핵심 문법 정리

1. Default Export 2. Named Export 3. Import

하나의 파일을 분리해서 여러개의 파일로 기술하는 방식을 의미한다.

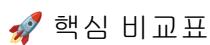
1. **Default Export** 임포트한 파일에서 대표기능 한개만 가져다가 쓸때 사용
2. **Named Export** 임포트한 파일에서 여러개의 기능을 가져다가 쓸때 사용
3. **Import** 임포트한 파일을 전체 적용해 가져다가 쓸때 사용

```
import { render, screen } from '@testing-library/react';

import App from './App';

import logo from './logo.svg'; //뒤에 있는 이미지를 식별자 logo로 사용한다.

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



구분

주요 특징

사용 예시

가져오기 방식

| | | | |
|----------------|------------------------|---------------------------------------|---|
| Default | 모듈의 주요 기능, 이름
자유 변경 | <code>export default</code>
컴포넌트 | <code>import 별명 from</code>
'경로' |
| Named | 다중 항목 내보내기,
이름 고정 | <code>export { 유ти1,
유티2 }</code> | <code>import { 이름 } from</code>
'경로' |
| CSS | 스타일시트 적용 | 여러 css 기술 | <code>import './App.css'</code> |

1. Default Export

문법 설명: 모듈에서 하나의 기본 값만 `export default`로 내보냄

// 내보내기

```
const 이름1 = 값;
```

```
export default 이름1;
```

```
export default function 이름2() {}
```

```
function 이름3() {}
```

```
export default 이름3
```

// 가져오기

```
import 사용자지정이름(이름1) from '모듈경로';
```

2. Named Export

문법 설명: 여러 값을 이름을 붙여 `export`로 내보냄

// 내보내기

```
export const 이름 = 값;
```

```
export function 이름() {}  
  
const 항목1 = 값;  
function 항목2() {}  
export { 항목1, 항목2 }; // 선언후 한번에 export할 수 있다.  
  
// 가져오기  
import { 이름1, 이름2 } from '모듈경로';
```

✓ 3. CSS Import

문법 설명: JavaScript나 React 파일에서 CSS 파일을 불러와 스타일을 적용 js 파일이면 해당 파일의 내용을 그대로 가져온다

```
// CSS 파일 가져오기  
import '파일경로/파일이름.css';  
import '파일경로/파일이름.js';
```

Default/Named Export 확장 설명 (CSS import 포함)

1. Default/Named Export 확장 예제

1.1 Default Export를 함수 선언과 함께 바로 사용

```
// utils.js  
  
// Default export (함수 선언과 동시에 내보내기)  
  
export default function mainUtil() {  
  
    return "MAIN FUNCTION";  
  
}  
  
// Named exports  
  
export function helper1() {  
  
    return "HELPER 1";
```

```
}

export function helper2() {
    return "HELPER 2";
}
```

1.2 Import 사용 예제

```
// 📁 app.js

import primaryFunction, { helper1, helper2 as h2 } from './utils.js';

console.log(primaryFunction()); // "MAIN FUNCTION"

console.log(helper1());          // "HELPER 1"

console.log(h2());               // "HELPER 2" (별칭 사용)
```

2. CSS 파일 import 방법

2.1 기본 CSS import

```
// 📁 component.js

import './styles.css'; // CSS 파일을 통으로 import (웹팩 등 모듈 번들러 필요)

import './styles.js'; // js 파일을 통으로 import (웹팩 등 모듈 번들러 필요)
```

2.2 CSS Module 사용 (React 예제)

```
import styles from './Button.module.css'; // CSS Module import

export default function Button() {
    return (
        <button className={styles.primary}>
            Click Me
        </button>
    );
}
```

```
</button>  
);  
}
```

4. 핵심 비교표 (업데이트 버전)

| 특징 | Default Export | Named Export | CSS Import |
|---------------------|--|---|---|
| 문법 | <code>export default</code>
<code>function</code> | <code>export function</code> | <code>import './file.css'</code> |
| import
방식 | <code>import XXX</code>
<code>from...</code> | <code>import { XXX }</code>
<code>from...</code> | <code>import styles from...</code> (CSS
Modules) |
| 이름 변경 | 자유롭게 변경 가능 | <code>as</code> 로 별칭 지정 필요 | CSS 클래스명 그대로 사용 |
| 사용 예시 | 주 컴포넌트/메인
기능 | 유тил 함수, 상수 | 스타일 적용 |
| 파일당
개수 | 1개 | 여러 개 | 여러 개 import 가능 |

번외 설명: 다양한 Import/Export 패턴

1. 모듈 전체 import

```
import * as mathUtils from './utils/mathUtils';

console.log(mathUtils.add(2, 3));
```

2. 이름 변경하여 import

```
import { add as sum } from './utils/mathUtils';

console.log(sum(2, 3));
```

3. CSS 모듈 (CSS-in-JS)

```
// 웹팩 css-loader 설정이 되어있다면

import styles from './styles/main.module.css';

const element = document.createElement('div');

element.className = styles['main-container']; // 고유한 클래스명이 생성됨
```

4. 동적 import (코드 스플리팅)

```
// 필요할 때만 모듈 로드

button.addEventListener('click', async () => {

  const math = await import('./utils/mathUtils');

  console.log(math.multiply(4, 5));

});
```

5. .js 생략

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

> 03. JSX의 기본구조

React JSX의 기초 문법을 설명해드리겠습니다. JSX는 JavaScript 코드 내에서 HTML과 유사한 문법을 사용하여 React 컴포넌트를 정의하고 사용할 수 있게 해줍니다.

"React 컴포넌트(JSX 사용)는 .jsx, 순수 JavaScript는 .js로 TypeScript 컴포넌트 .tsx 확장자 순수 TypeScript는 .ts를 사용

1. jsx 기본구조

1. React은 컴포넌트로 구성되어 있다.
2. 컴포넌트는 하나의 자바스크립트 파일에 하나의 함수로 만든다.
3. 하단에 **export default** 함수이름;을 넣는다.
4. 컴포넌트는 화면 구성을 위한 데이터 조작 부분을 메소드안에 기술한다.
5. 메소드안에 기술된 자바스크립트 코드를 가지고 리턴 값에 화면을 표현한다.
6. 컴포넌트 이름은 반드시 대문자로 시작해야 한다.

1. 컴포넌트 중심 구조

- "React는 컴포넌트(UI 조각)로 구성된다"
 - 모든 화면은 독립된 컴포넌트의 조합으로 생성됩니다.
 - 예: Header, Button, Card 등

2. 파일 구성 규칙

- "1파일 = 1컴포넌트 = 1함수"

//  Button.jsx

```
function Button() { // 컴포넌트 함수
  return <button>Click</button>;
}

• export default Button; // 필수 내보내기
```

3. 컴포넌트 작동 원리

- "로직(JS) + 화면(JSX)을 함께 작성"

```
function Counter() {
  // 1. 데이터 조작 (JavaScript)
  const count=0;

  // 2. 화면 반환 (JSX)
  return (
    <div>
      <p>{count}</p>
    </div>
  );
}
```

컴포넌트

코드

화면

```
import logo from './logo.svg';
import './App.css';

function App() {
  const element = <h1>Hello, world!</h1>

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

export default App;
```

4. 컴포넌트 네이밍 규칙

- "반드시 대문자로 시작 (PascalCase)"
 - Button (⭕) / button (✖)
 - 소문자로 시작하면 HTML 태그로 인식됩니다.
-

5. 필수 포함 요소

| 요소 | 설명 | 예시 |
|----------------|----------------------|----------------------------|
| 함수 선언 | 컴포넌트 로직과 JSX 포함 | function Component() {} |
| export default | 외부에서 사용 가능하도록 내보냄 | export default Component; |
| JSX 반환 | 화면을 정의하는 XML-like 구문 | return <div>Content</div>; |

2. JSX 표현식

JSX 내부에서는 중괄호 {}를 사용하여 JavaScript 표현식을 삽입할 수 있습니다.

```
const name = 'John';
const element = <h1>Hello, {name}!</h1>;
```

```
import logo from './logo.svg';
import './App.css';

function App() {
  const name = 'John';
  const element = <h1>Hello, {name}!</h1>

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

export default App;
```

3. JSX 요소 닫기

모든 JSX 태그는 반드시 닫혀야 합니다. 단일 태그는 self-closing 태그로 작성해야 합니다.

// 잘못된 예시

```
// const element = ;
```

// 올바른 예시

```
const element = ;
```

4. JSX 속성

JSX에서 속성은 HTML과 비슷하게 작성되지만, 일부 속성 이름은 JavaScript의 관습에 따라 작성됩니다. 예를 들어, class 대신 `className`, for 대신 `htmlFor`를 사용합니다.

`javascript` 키워드와 겹치면 다른 식별자를 사용한다.

```
const element = <div className="container"></div>;
// ✅ 'class' 대신 'className'
const label = <label htmlFor="username">Username</label>;
// ✅ 'for' 대신 'htmlFor'
```

5. 주석

JSX 내에서 주석: JSX 내에서 주석은 중괄호 안에 `/**/`를 이용해서 주석을 만든다.

한줄주석 // 은 제공되지 않는다.

```
const element = ( <div> /* This is a comment in JSX */ <p>Hello, World!</p> </div> );
```

6. 루트는 반드시 하나여야 한다.

두개 이상 리턴해야 할 경우 `<></>`안에 기술한다.

Fragments: JSX에서 두 개 이상의 요소를 반환하려면 반드시 하나의 부모 요소로 감싸야 합니다.

그러나 때로는 부모 요소를 추가하지 않고 여러 요소를 그룹화하고 싶을 때가 있습니다. `React Fragment`를 사용하여 부모 요소 없이 여러 요소를 그룹화할 수 있습니다.

```
const element = ( <> <p>Paragraph 1</p> <p>Paragraph 2</p> </> );
```

```
import logo from './logo.svg';
import './App.css';
function App() {
  const name = 'John';
  const element = <h1>Hello, {name}!</h1>;
  return (
    <>
      <div className="App">
        {element}
      </div>
      <div className="App">
        {element}
      </div>
    </>
  );
}
export default App;
```

7. 삼항연산자 사용 가능

조건부 렌더링: JSX 내에서 조건부로 요소를 렌더링하려면 삼항 연산자나 논리 연산자를 사용할 수 있습니다.

jsx

```
const isLoggedIn = true;
const element = isLoggedIn ? <p>Welcome, User!</p> : <p>Please log in</p>;
```

8. JSX로 컴포넌트 만들어 사용하기

컴포넌트는 단순히 함수를 정의하고 JSX를 반환해서 만들수 있다.

```
function Greeting() {
  return <h1>Hello, world</h1>;
}
```

함수로 컴포넌트를 만들어 원할때 사용할 수 있다.

```
import logo from './logo.svg';
import './App.css';

function Greeting() {
  return <h1>Hello, world</h1>;
}

function App() {
  const name = 'John';
  const element = <h1>Hello2, {name}!</h1>;
  return (
    <div className="App">
      {element}
      <Greeting/>
      <Greeting/>
    </div>
  );
}

export default App;
```

9. Props

컴포넌트에 데이터를 전달하기 위해 `props`를 사용합니다. `props`는 컴포넌트의 속성으로 전달됩니다.

컴포넌트 속성으로 선언한 속성 이름으로 속성의 값을 읽어 올수 있다. `props.속성 이름`하면 속성의 값에 접근한다.

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

```

// 사용 예시
<Greeting name="Alice" />

import React from 'react';
import './App.css';

// Greeting 컴포넌트 정의
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// App 컴포넌트 정의
function App() {
  return (
    <div className="App">
      <Greeting name="Alice" />
      <Greeting name="Tom" />
    </div>
  );
}

export default App;
문제1)
<Greeting name="Tom" age=15/>일때 “Tom의 나이는 15입니다.” 가 출력 되도록 컴포넌트를
만들어 보자.

```

10. 자식 요소

컴포넌트에 자식 요소를 포함할 수 있습니다.

```

function Welcome(props) {
  return (
    <div>
      <h1>Hello, {props.name}</h1>
      {props.children}
    </div>
  );
}

```

```

// 사용 예시
<Welcome name="Alice">
  <p>This is a child element.</p>
</Welcome>

```

11. 조건부 렌더링

JSX 내에서 조건부로 요소를 렌더링할 수 있습니다.

```
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign up.</h1>;
  }
}

import React from 'react';
import './App.css';

// Greeting 컴포넌트 정의
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign up.</h1>;
  }
}

// App 컴포넌트 정의
function App() {
  const userisLoggedIn = true; // 로그인 상태 변수(true/false로 변경 가능)
  return (
    <div className="App">
      <Greeting isLoggedIn={userisLoggedIn} />
    </div>
  )
}
```

```
 );
}
export default App;
```

12. 리스트 렌더링

여러 요소를 렌더링할 때는 `map` 함수를 사용합니다. REACT에서 반복되는 태그를 만들 때 유일한 값을 가지는 `key` 속성값을 추가해 줘야 한다.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

// 사용 예시

```
const numbers = [1, 2, 3, 4, 5];
<NumberList numbers={numbers} />
```

13. 인라인 스타일링

react에서 여러가지 css 사용하기

1. Inline Styles

인라인 스타일을 사용하면 JavaScript 객체로 스타일을 직접 지정할 수 있습니다.

JSX에서 스타일을 적용할 때는 객체를 사용합니다.

```
const divStyle = {
  color: 'blue',
  backgroundColor: 'lightgrey'
};
const element = <div style={divStyle}>Styled element</div>

import React from 'react';
import './App.css';

// 스타일 객체 정의
const divStyle = {
  color: 'blue',
  backgroundColor: 'lightgrey'
};

// App 컴포넌트 정의
function App() {
  return (
    <div className="App">
      <div style={divStyle}>Styled element</div>
    </div>
  );
}
export default App;
```

다음 코드를 읽고 이해해 보자.

```
import "./App.css";
function App() {
  const name = "Tom";
  const naver = {
    name: "네이버",
    url: "https://naver.com",
  };
  return (
    <div className="App">
      <h1
        style={{
```

```

        color: "#f0f",
        backgroundColor: "green",
    //}
>
Hello, {name}.<p>{2 + 3}</p>
</h1>
<a href={naver.url}>{naver.name}</a>
</div>
);
}
export default App;

```

2. 일반 CSS 파일 사용

가장 기본적인 방법으로, CSS 파일을 작성하고 컴포넌트에서 이를 임포트하여 사용하는 방식입니다.

App의 하위 컨트롤러에도 영향을 미친다.

```

/* styles.css */
.container {
  background-color: lightblue;
  padding: 20px;
}

// App.js
import React from 'react';
import './styles.css';
function App() {
  return <div className="container">Hello, World!</div>;
}
export default App;

```

현 상태에서 App에서 사용하는 하위 App2 컴포넌트에 .container를 사용하는 DIV를 추가하면 APP div와 하위에 있는 App2 div 둘다 .container의 영향을 받는다.

// App.js 를 변경하고 실행해 보자.

```

import React from 'react';
import './styles.css';
function App2() {

```

```
    return <div className="container">Hello, World!</div>;
}
function App() {
  return <div>
    <div className="container">Hello, World!</div>
    <App2/>
  </div>;
}
export default App;
```

3. CSS Modules

.module.css는 특정한 기능을 수행하기 위해 약속된 확장자입니다. React에서 CSS Modules을 사용할 때, .module.css 확장자를 가진 파일을 사용합니다.

css

```
/* App.module.css */
.container {
  background-color: lightblue;
  padding: 20px;
}
```

```
// App.js
import React from 'react';
import styles from './App.module.css';

function App() {
  return <div className={styles.container}>Hello, World!</div>;
}
export default App;

현 상태에서 App에서 사용하는 하위 App2 컴포넌트에 .container를 사용하는 DIV를 추가하면
App.module.css 에 정의한 내용은 App에 있는 div에만 영향을 주고 하위에 있는 App2 div에는
영향을 주지 않는다.

// App.js 변경
import React from 'react';
import styles from './App.module.css';

function App2() {
  return <div className="container">Hello, App2!</div>;
}
function App() {
  return <div>
    <div className={styles.container}>Hello, App!</div>
    <App2/>
  </div>;
}
export default App;
```

> 04. 관리프로그램 GUI 만들기

이거 하지 말고 하단의 코드를 사용하자.



```
import React from 'react';
```

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

```
import logo from './logo.svg';

function Header() {
  return (
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <div style={{ display: 'flex', alignItems: 'center' }}>
        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
        
        <h1 style={{ margin: 0 }}>Student Info</h1>
      </div>
    </header>
  );
}

function Nav() {
  return (
    <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <ol>
        <li>Alice</li>
        <li>Bob</li>
        <li>Charlie</li>
        <li>Dave</li>
      </ol>
    </nav>
  );
}
```

```
function Article() {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Select Menu</h2>  
      <p>Welcome to the Student Info App</p>  
    </article>  
  );  
}
```

```
function Create() {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Create</h2>  
      <form>  
        <p><input type="text" placeholder="name" /></p>  
        <p><input type="text" placeholder="username" /></p>  
        <p><input type="number" placeholder="age" /></p>  
        <p><input type="number" placeholder="height" /></p>  
        <p><input type="date" placeholder="join date" /></p>  
        <p><input type="submit" value="Create" /></p>  
      </form>  
    </article>  
  );  
}
```

```
function Update() {  
  return (
```

```
<article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

  <h2>Update</h2>

  <form>

    <p><input type="text" defaultValue="name" placeholder="name" /></p>

    <p><input type="text" defaultValue="username" placeholder="username" /></p>

    <p><input type="number" defaultValue="age" placeholder="age" /></p>

    <p><input type="number" defaultValue="height" placeholder="height" /></p>

    <p><input type="date" defaultValue="join date" placeholder="join date" /></p>

    <p><input type="submit" value="Update" /></p>

  </form>

</article>

);

}
```

```
function Footer() {

  return (

    <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>

      <ul>

        <li><a href="/create">Create</a></li>

        <li><a href="/update">Update</a></li>

      </ul>

    </footer>

  );
}


```

```
function App() {

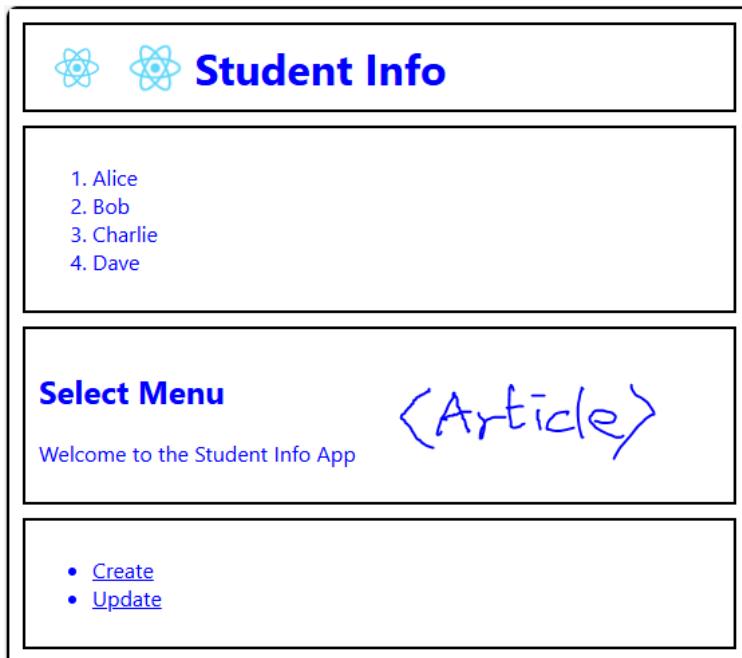
  return (

```

```
<div style={{ border: '2px solid black', padding: '10px' }}>  
  <Header />  
  <Nav />  
  <Create />  
  <Footer /> /* Footer 컴포넌트를 여기에서 추가합니다 */  
</div>  
);  
}  
  
export default App;
```

React 16 이하 → import React from 'react'; 필수

React 17 이상 → JSX가 정상적으로 동작하면 제거 가능



```
function App() {  
  return (  
    <div style={{ border: '2px solid black', padding: '10px' }}>  
      <Header />  
      <Nav />  
      <Article /> /* Create를 Article 컴포넌트로 변경 */  
    </div>  
  );  
}  
  
export default App;
```

```
<Footer />
</div>
);
}

export default App;
```

> 05. prop를 이용해 관리자 프로그램 업그레이드

부모 컴포넌트(**Parent Component**): 다른 컴포넌트를 포함하고 있는 컴포넌트

자식 컴포넌트(**Child Component**): 부모 컴포넌트에 의해 렌더링되고, 부모 컴포넌트로부터 전달된 데이터를 받아 사용하는 컴포넌트입니다.

App와 **ChildComponent**에서 App는 부모 컴포넌트 **childcomponent**는 자식 컴포넌트이다. **prop**(속성)은 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달하는 방법이다.

<ChildComponent name="존" age={30} /> 다음과 같이 속성에 값을 넣어 전송하면 **function ChildComponent(props) {** 와 같이 자식 컴포넌트 메소드로 선언된 매개변수 **props**를 통해서 **<p>이름: {props.name}</p>** 과 같은 방법으로 원하는 속성값을 출력할 수 있다.

```
import React from 'react';

function ChildComponent(props) {

  return (
    <div>
      <p>이름: {props.name}</p>
      <p>나이: {props.age}</p>
    </div>
  );
}

export default ChildComponent;
```

```
</div>

);

}

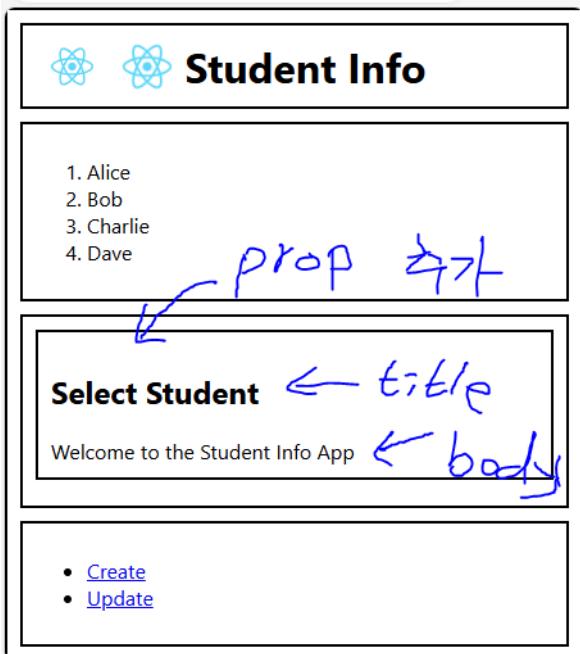
function App() {
  return (
    <div>
      <ChildComponent name="존" age={30} />
      <ChildComponent name="앨리스" age={25} />
      <ChildComponent name="밥" age={40} />
      <ChildComponent name="찰리" age={22} />
    </div>
  );
}

export default App;
```

Article컴포넌트 개선하기

prop를 이용해서 학생관리 프로그램을 변경해 보자.

Article함수에 prop를 추가하고 App함수의 Article 컴포넌트에 title과 body속성을 추가해서 다음과 같은 결과를 출력해 보자.



App의 내용은 다음과 같고 Article 컴포넌트를 만들어 보자.

```

function App() {
  return (
    <div style={{ border: '2px solid black', padding: '10px' }}>
      <Header />
      <Nav />
      <Article title="Select Student" body="Welcome to the Student Info App" />
      <Footer />
    </div>
  );
}

function Article(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>{props.title}</h2>
      {props.body}
    </article>
  );
}

```

1. 변수 선언 키워드

1) `var` (ES5 및 이전 버전)

- 함수 범위(Function Scope)를 가짐
- 재선언(같은 이름의 변수 선언) 가능
- 선언 전에 사용할 수 있음 (호이스팅 시 `undefined` 할당)
- `var`는 최신 JavaScript(ES6+)에서는 잘 사용되지 않음

```
javascript
```

복사편집

```
console.log(a); // undefined (호이스팅 발생)
```

```
var a = 10;
```

```
console.log(a); // 10
```

```
var b = 20;
```

```
var b = 30; // 재선언 가능
```

```
console.log(b); // 30
```

2) **let** (ES6 도입)

- 블록 범위(Block Scope)를 가짐 ({ } 내부에서만 유효)
- 재선언 불가능 (같은 이름의 변수 중복 선언 불가)
- 선언 전에 사용할 수 없음 (TDZ - Temporal Dead Zone 발생)

```
javascript
```

복사편집

```
console.log(x); // ReferenceError (TDZ로 인해 접근 불가)
```

```
let x = 10;
```

```
console.log(x); // 10
```

```
let y = 20;
```

```
let y = 30; // SyntaxError: Identifier 'y' has already been declared
```

3) **const** (ES6 도입)

- 블록 범위(Block Scope)를 가짐 (let과 동일)
- 반드시 선언과 동시에 초기화해야 함

- 재할당 불가능 (변경 불가능한 상수 변수)

javascript

복사편집

```
const z = 100;  
  
z = 200; // TypeError: Assignment to constant variable.
```

```
const pi; // SyntaxError: Missing initializer in const declaration
```

☞ 단, `const`는 객체(Object)와 배열(Array)일 때 내부 값은 변경할 수 있음

javascript

복사편집

```
const person = { name: "Alice" };  
  
person.name = "Bob"; // 가능  
  
console.log(person); // { name: "Bob" }
```

함수에 대한 설명

```
function add(a, b) {  
  
    return a + b;  
  
}
```

> 06. 화살표함수

```
// 1. 매개변수 없을 때  
  
const 함수명 = () => { 실행문 };  
  
// 2. 매개변수 1개일 때 (괄호 생략 가능)
```

```
const 함수명 = 매개변수 => { 실행문 };

// 3. 매개변수 여러 개일 때

const 함수명 = (매개변수1, 매개변수2) => { 실행문 };

// 4. 리턴값만 있을 때 (중괄호 & return 생략 가능)

const 함수명 = (a, b) => a + b;
```

//화살표함수

/*화살표 함수는 JavaScript에서 함수를 간단하게 만드는 방법입니다. 기존의 함수를 좀 더 짧고 간결하게 작성할 수 있게 도와줍니다. 두 방식의 결과의 차이점은 없습니다.

기존 함수:

```
function greet(name) {
    return "안녕하세요, " + name + "님!";
}
```

화살표 함수:

```
var greet = name => "안녕하세요, " + name + "님!";
```

다음은 화살표 함수 설명하는 내용입니다.

함수 선언: `function` 키워드 대신 `=>` 기호를 사용하여 함수를 선언합니다.

인자(매개변수): 화살표 앞쪽에 인자가 들어 간다. 인자가 없으면(), 여러 개의 인자가 있을 때는 괄호로 감싸줍니다. 예를 들어 `(x, y)=>`, 하나이면 중괄호를 생략할 수 있다. 예를 들어, `x =>`

함수 본문: 함수 본문은 화살표(`=>`) 다음에 나옵니다. 본문이 한줄일 경우 중괄호 `{}`로 감싸지 않아도 됩니다.

리턴 값: 본문이 한줄일 경우 별도의 `return` 키워드를 사용하지 않아도 됩니다.

간단한 예제로 화살표 함수를 설명하겠습니다:

```
// 기존 함수
```

```
function add(x, y) {  
    return x + y;  
}  
  
// 화살표 함수로 변환  
  
var add = (x, y) => x + y;
```

위 예제에서, `add` 함수를 화살표 함수로 변환하면 더 간결한 코드가 됩니다. 화살표 함수는 주로 익명 함수로 사용되며, 함수를 변수에 할당하거나 배열의 `map`, `filter`, `reduce`와 같은 고차 함수와 함께 사용되어 코드를 더 읽기 쉽게 만들어줍니다. */

```
var result=arr1.filter(x=>x%2==1);  
//(x,y)=>x+y;           //다음 3개는 같은 의미이다.  
//(x,y)=>{x=x+y;return x;};
```

자바스크립트 화살표 함수(Arrow Function) 예제 10개를 준비했습니다. 초급부터 중급 수준까지 다양하게 넣었어요:

- ◆ 1. 두 수 더하기

javascript

복사편집

```
const add = (a, b) => a + b;  
console.log(add(2, 3)); // 5
```

- ◆ 2. 인사말 반환

javascript

복사편집

```
const greet = name => `Hello, ${name}!`;  
console.log(greet("철수")); // Hello, 철수!
```

- ◆ 3. 배열의 제곱값 만들기 (`map` 사용)

javascript

복사편집

```
const numbers = [1, 2, 3, 4];
```

```
const squares = numbers.map(num => num * num);
console.log(squares); // [1, 4, 9, 16]
```

- ◆ 4. 짝수만 필터링

javascript

복사편집

```
const nums = [1, 2, 3, 4, 5, 6];
const evens = nums.filter(n => n % 2 === 0);
console.log(evens); // [2, 4, 6]
```

- ◆ 5. 객체 반환 (소괄호로 감싸기)

javascript

복사편집

```
const createUser = (name, age) => ({ name, age });
console.log(createUser("영희", 25)); // { name: '영희', age: 25 }
```

- ◆ 6. 배열의 총합 계산 (**reduce** 사용)

javascript

복사편집

```
const arr = [10, 20, 30];
const sum = arr.reduce((acc, cur) => acc + cur, 0);
console.log(sum); // 60
```

- ◆ 7. 문자열 길이 반환

javascript

복사편집

```
const getLength = str => str.length;
console.log(getLength("화살표 함수")); // 6
```

- ◆ 8. **setTimeout** 사용

javascript

복사편집

```
setTimeout(() => {
  console.log("3초 뒤 출력");
}, 3000);
```

- ◆ 9. 삼항 연산자와 함께 사용

javascript

복사편집

```
const isAdult = age => age >= 18 ? "성인" : "미성년자";
console.log(isAdult(20)); // 성인
```

- ◆ 10. 다중 매개변수, 중괄호와 **return** 사용

javascript

복사편집

```
const calculateArea = (width, height) => {
  return width * height;
};
console.log(calculateArea(5, 4)); // 20
```

화살표함수는 일반 함수를 간결하게 만들기 위해서 사용한다.

```
function 함수이름(매개변수1, 매개변수2, ...) {
  // 함수 본문
  // return 리턴할 값
}
```

```
const 함수이름 = (매개변수1, 매개변수2, ...) => {
    // 함수 본문
    // return 리턴할 값
};

// 두 예제 비교

// 일반 함수
function add(a, b) {
    return a + b;
}

// 화살표 함수
const add = (a, b) => a + b;
```

1. 기본 형태

화살표 함수:

```
const greet = (name) => {
    console.log(`Hello, ${name}!`);
};

greet('Alice'); // 출력: Hello, Alice!
```

일반 함수:

```
function greet(name) {  
  console.log(`Hello, ${name}!`);  
}  
  
greet('Alice'); // 출력: Hello, Alice!
```

2. 매개변수가 하나일 때

화살표 함수:

```
const square = x => x * x;  
  
console.log(square(5)); // 출력: 25
```

일반 함수:

```
function square(x) {  
  return x * x;  
  
}  
  
console.log(square(5)); // 출력: 25
```

3. 매개변수가 없을 때

화살표 함수:

```
const sayHello = () => console.log('Hello, world!');  
  
sayHello(); // 출력: Hello, world!
```

일반 함수:

```
function sayHello() {  
  console.log('Hello, world!');  
  
}  
  
sayHello(); // 출력: Hello, world!
```

4. 함수 본문이 하나의 표현식일 때

화살표 함수:

```
const add = (a, b) => a + b;  
console.log(add(3, 4)); // 출력: 7
```

일반 함수:

```
function add(a, b) {  
    return a + b;  
}  
console.log(add(3, 4)); // 출력: 7
```

5. 함수 본문이 여러 줄일 때

화살표 함수:

```
const multiply = (a, b) => {  
    const result = a * b;  
    return result;  
};  
console.log(multiply(6, 7)); // 출력: 42
```

일반 함수:

```
function multiply(a, b) {  
    const result = a * b;  
    return result;  
}  
console.log(multiply(6, 7)); // 출력: 42
```

JavaScript 변수 선언 키워드와 함수 범위(**Function Scope**) 관련 내용

JavaScript에서 변수를 선언하는 방법(**var**, **let**, **const**)은 **함수 범위(Function Scope)와 블록 범위(Block Scope)**에 영향을 미칩니다. 여기서는 함수 범위와 관련된 내용을 추가하여 설명하겠습니다.

1. 함수 범위(Function Scope)란?

함수 범위(Function Scope)는 변수가 선언된 함수 내부에서만 접근할 수 있는 범위를 의미합니다.
즉, 함수 내부에서 선언된 변수는 함수 외부에서 접근할 수 없습니다.

javascript

복사편집

```
function myFunction() {  
    var a = 10; // 함수 내부에서 선언된 변수  
    let b = 20;  
    const c = 30;  
    console.log(a, b, c); // 10, 20, 30  
}  
  
myFunction();  
  
console.log(a); // ReferenceError: a is not defined  
console.log(b); // ReferenceError: b is not defined  
console.log(c); // ReferenceError: c is not defined
```

✓ 결론: 함수 내부에서 선언된 변수는 함수가 종료되면 사라지며, 함수 외부에서는 접근할 수 없음.

2. var는 "함수 범위"를 가짐

`var`로 선언된 변수는 **함수 범위(Function Scope)**를 가집니다.
즉, `if`, `for` 등의 블록 `{}` 내부에서 선언해도 함수 전체에서 접근할 수 있습니다.

javascript

복사편집

```
function example() {  
    if (true) {
```

```
    var testVar = "I am inside if block";  
}  
  
console.log(testVar); // "I am inside if block" (함수 전체에서 접근 가능)  
}  
  
example();  
  
console.log(testVar); // ReferenceError: testVar is not defined (함수 외부에서는  
접근 불가능)
```

⚠ 문제점: `if` 블록 안에서 선언했지만, 함수 전체에서 접근할 수 있음 → 의도치 않은 변경 위험

3. `let`과 `const`는 "블록 범위(Block Scope)"를 가짐

`let`과 `const`는 **블록 범위(Block Scope)**를 가지므로 `{}` 내부에서만 유효합니다.

javascript

복사편집

```
function example() {  
  
    if (true) {  
  
        let testLet = "I am inside if block";  
  
        const testConst = "I am also inside if block";  
  
    }  
  
    console.log(testLet); // ReferenceError: testLet is not defined  
  
    console.log(testConst); // ReferenceError: testConst is not defined  
}  
  
example();
```

✓ 결론: `let`과 `const`는 블록 `{}` 내부에서만 접근 가능하여, `var`보다 안전함.

4. 함수 내부에서 `var`, `let`, `const` 차이점 정리

| 선언 키워드 | 함수 범위(Function Scope) | 블록 범위(Block Scope) | 중복 선언 | 호이스팅 시 초기값 |
|--------------------|-----------------------|--------------------|-------|------------------------|
| <code>var</code> | ✓ 있음 | ✗ 없음 | ✓ 가능 | <code>undefined</code> |
| <code>let</code> | ✗ 없음 | ✓ 있음 | ✗ 불가능 | TDZ (오류 발생) |
| <code>const</code> | ✗ 없음 | ✓ 있음 | ✗ 불가능 | TDZ (오류 발생) |

✓ `let`과 `const`를 사용하면 함수 내에서 블록 범위를 유지할 수 있어, 의도치 않은 변수 변경을 방지할 수 있음.

⚠ `var`는 함수 전체에서 접근 가능하므로, 되도록 사용하지 않는 것이 좋음.

5. 함수 내부에서 선언하지 않은 변수 (자동 전역 변수)

함수 내부에서 `var`, `let`, `const` 없이 변수를 선언하면 **자동으로 전역 변수(Global Variable)**가 됩니다.

이는 버그의 원인이 될 수 있으므로 사용하지 않는 것이 좋습니다.

javascript

복사편집

```
function myFunction() {  
    globalVar = "I am global!"; // var, let, const 없이 선언 (자동 전역 변수)  
}  
  
myFunction();  
  
console.log(globalVar); // "I am global!" (전역 변수가 되어버림)
```

✓ 해결 방법: `let`, `const`를 사용하여 명시적으로 선언해야 함.

```
import React from 'react';
```

이 구문은 **React** 컴포넌트를 정의하고 **JSX**를 사용할 수 있도록 하기 위해 추가됩니다.

📌 1. 배열 선언 및 초기화

✓ 배열을 생성하는 방법

`javascript`

코드 복사

```
// 방법 1: 대괄호([]) 사용 (가장 일반적)
```

```
let arr1 = [1, 2, 3, 4, 5];
```

```
// 방법 2: Array() 생성자 사용  
let arr2 = new Array(1, 2, 3, 4, 5);
```

```
// 방법 3: 빈 배열 생성 후 값 추가  
let arr3 = [];  
arr3.push(1);  
arr3.push(2);
```

✓ []을 사용하는 것이 가장 일반적이며, new Array()는 권장되지 않습니다.

📌 2. 배열 요소 접근 및 변경

배열 요소는 0부터 시작하는 인덱스(index)를 사용해 접근합니다.

javascript

코드 복사

```
let fruits = ["Apple", "Banana", "Cherry"];  
  
// 요소 접근  
console.log(fruits[0]); // Apple  
console.log(fruits[1]); // Banana  
  
// 요소 변경  
fruits[1] = "Blueberry";  
console.log(fruits); // ["Apple", "Blueberry", "Cherry"]
```

✓ 배열의 인덱스는 0부터 시작합니다.

📌 3. 배열의 주요 속성과 메서드

- ◆ 배열 길이 (**length**)

javascript

코드 복사

```
let numbers = [10, 20, 30, 40];
console.log(numbers.length); // 4
```

- ◆ 배열 끝에 요소 추가 (**push**) / 삭제 (**pop**)

javascript

코드 복사

```
let arr = [1, 2, 3];
```

```
arr.push(4); // 마지막에 추가
console.log(arr); // [1, 2, 3, 4]
```

```
arr.pop(); // 마지막 요소 제거
console.log(arr); // [1, 2, 3]
```

- ◆ 배열 앞에 요소 추가 (**unshift**) / 삭제 (**shift**)

javascript

코드 복사

```
let arr = [2, 3, 4];
```

```
arr.unshift(1); // 맨 앞에 추가
console.log(arr); // [1, 2, 3, 4]
```

```
arr.shift(); // 맨 앞 요소 제거
console.log(arr); // [2, 3, 4]
```

- ◆ 특정 위치에 요소 추가/제거 (**splice**)

javascript

코드 복사

```
let arr = ["a", "b", "c", "d"];
```

```
// 1번 인덱스부터 2개 제거
arr.splice(1, 2);
console.log(arr); // ["a", "d"]
```

```
// 1번 인덱스에 "x", "y" 추가  
arr.splice(1, 0, "x", "y");  
console.log(arr); // [ "a", "x", "y", "d" ]
```

📌 4. 배열 순회 (반복문)

- ◆ **for** 반복문

```
javascript  
코드 복사  
let arr = ["Apple", "Banana", "Cherry"];  
  
for (let i = 0; i < arr.length; i++) {  
    console.log(arr[i]);  
}
```

- ◆ **forEach()** 메서드

```
javascript  
코드 복사  
arr.forEach((item, index) => {  
    console.log(index, item);  
});
```

- ◆ **for...of** 반복문 (ES6)

```
javascript  
코드 복사  
for (let fruit of arr) {  
    console.log(fruit);  
}
```

5. 배열 변형 및 활용

- ◆ **map()** - 배열의 각 요소 변환

javascript

코드 복사

```
let numbers = [1, 2, 3, 4];
let squared = numbers.map(num => num * num);
console.log(squared); // [1, 4, 9, 16]
```

- ◆ **filter()** - 조건에 맞는 요소만 추출

javascript

코드 복사

```
let numbers = [10, 15, 20, 25, 30];
let filtered = numbers.filter(num => num > 15);
console.log(filtered); // [20, 25, 30]
```

- ◆ **reduce()** - 배열의 값을 하나로 합치기

javascript

코드 복사

```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // 10
```

6. 배열과 문자열 변환

- ◆ **join()** - 배열을 문자열로 변환

javascript

코드 복사

```
let words = ["Hello", "world"];
console.log(words.join(" ")); // "Hello world"
```

- ◆ **split()** - 문자열을 배열로 변환

javascript

코드 복사

```
let text = "apple,banana,grape";
let fruitArray = text.split(",");
console.log(fruitArray); // ["apple", "banana", "grape"]
```

7. 배열 정렬

- ◆ 오름차순 정렬 (**sort()**)

javascript

코드 복사

```
let numbers = [4, 2, 7, 1];
numbers.sort((a, b) => a - b);
console.log(numbers); // [1, 2, 4, 7]
```

- ◆ 내림차순 정렬

javascript

코드 복사

```
numbers.sort((a, b) => b - a);
console.log(numbers); // [7, 4, 2, 1]
```

정리

메서드

설명

push() 배열 끝에 요소 추가

pop() 배열 끝 요소 제거

`unshift()` 배열 앞에 요소 추가

`shift()` 배열 앞 요소 제거

`splice()` 특정 위치에 요소
추가/제거

`map()` 각 요소 변환 (새 배열
반환)

`filter()` 조건에 맞는 요소만
필터링

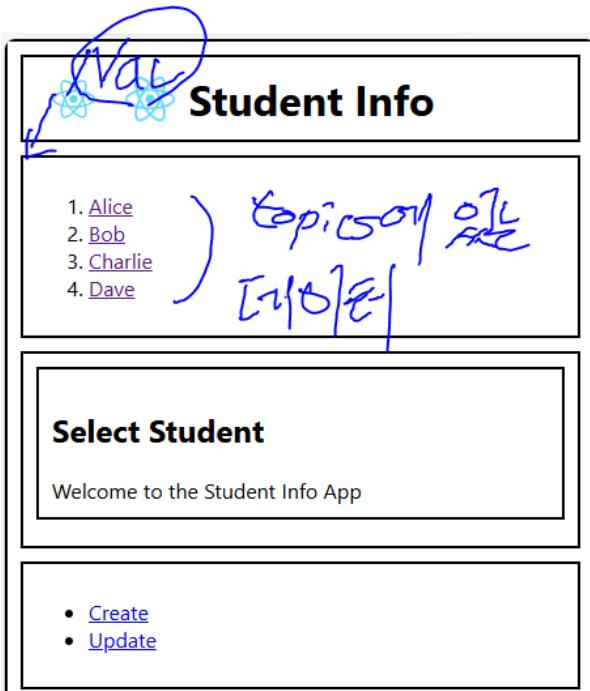
`reduce()` 배열 값을 하나로 총약

`sort()` 배열 정렬

✓ 배열을 다룰 때는 `map()`, `filter()`, `reduce()` 등의 고차 함수가 매우 유용합니다.



Nav에 `props`를 추가하여 다음과 같은 화면을 만들어 보자.



하나의 li는 다음과 같다.

```
<li key={t.id}>
  <a
    id={t.id}
    href={'/read/' + t.id}>
    {t.name}
  </a>
</li>
```

상위 key 속성은 반드시 유니크한 값을 가지고 있어야 한다.

React에서 key는 동적인 리스트 아이템을 렌더링할 때 사용되는 중요한 속성입니다. key는 React가 가상 DOM을 관리하고 업데이트하는 데 도움을 주는 유일한 식별자입니다. 이를 통해 React는 어떤 항목이 변경, 추가, 또는 제거되었는지를 파악하여 효율적으로 업데이트를 수행할 수 있습니다.

반복되는 엘리멘트가 생성되면 반드시 key를 보유하고 있어야 한다.

```
function App() {
  const students=[

    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },

    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },

    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },

    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ];
}
```

```
return (
  <div style={{ border: '2px solid black', padding: '10px' }}>
    <Header />
    <Nav students={students}/>
    <Article title="Select Student" body="Welcome to the Student Info App" />
    <Footer /> /* Footer 컴포넌트를 여기에서 추가합니다 */
  </div>
);
}
```

```
function Nav(props) {
```

```
const list = props.students.map((t) => (
  <li key={t.id}>
    <a
      id={t.id}
      href={'/read/' + t.id}
    >
      {t.name}
    </a>
  </li>
));
console.log(list)

return (
  <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
    <ol>{list}</ol>
  </nav>
);
}
```

변경된 전체코드

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

function Header() {
  return (
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <div style={{ display: 'flex', alignItems: 'center' }}>
        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
        
      </div>
      <h1 style={{ margin: 0 }}>Student Info</h1>
    </div>
  </header>
);

function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a
        id={t.id}
        href={'/read/' + t.id}
      >
        {t.name}:{t.age}
      </a>
    </li>
  ));
  console.log(list)
  return (
    <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <ol>{list}</ol>
    </nav>
  );
}

function Article(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>{props.title}</h2>
      {props.body}
    </article>
  );
}
```

```
);

}

function Create() {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>Create</h2>
            <form>
                <p><input type="text" placeholder="name" /></p>
                <p><input type="text" placeholder="username" /></p>
                <p><input type="number" placeholder="age" /></p>
                <p><input type="number" placeholder="height" /></p>
                <p><input type="date" placeholder="join date" /></p>
                <p><input type="submit" value="Create" /></p>
            </form>
        </article>
    );
}

function Update() {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>Update</h2>
            <form>
                <p><input type="text" defaultValue="name" placeholder="name" /></p>
                <p><input type="text" defaultValue="username" placeholder="username" /></p>
                <p><input type="number" defaultValue="age" placeholder="age" /></p>
                <p><input type="number" defaultValue="height" placeholder="height" /></p>
                <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
                <p><input type="submit" value="Update" /></p>
            </form>
        </article>
    );
}

function Footer() {
    return (
        <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>
```

```
<ul>
  <li><a href="/create">Create</a></li>
  <li><a href="/update">Update</a></li>
</ul>
</footer>
);
}

function App() {
  const students=[
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ];
}

return (
  <div style={{ border: '2px solid black', padding: '10px' }}>
    <Header />
    <Nav students={students}/>
    <Article title="Select Student" body="Welcome to the Student Info App" />
    <Footer /> {/* Footer 컴포넌트를 여기에서 추가합니다 */}
  </div>
);
}

export default App;
```

JSON (JavaScript Object Notation)

기초 설명

JSON은 데이터를 저장하고 전송하기 위한 경량 데이터 형식입니다.

 **JavaScript**의 객체 표기법을 기반으로 하지만, 대부분의 프로그래밍 언어에서 사용 가능합니다.

1. JSON의 특징

- ✓ 텍스트 기반 데이터 형식 (UTF-8 지원)
- ✓ 가볍고 가독성이 좋음
- ✓ 구조화된 데이터 저장 및 전송 가능

- ✓ 프로그래밍 언어와 무관하게 사용 가능 (Java, JavaScript, Python, PHP 등에서 사용)
 - ✓ API 응답 형식으로 많이 사용됨 (REST API, AJAX, GraphQL 등)
-

📌 2. JSON 문법

✓ JSON 데이터 구조

JSON은 키-값 쌍 (**Key-Value Pair**)으로 구성됩니다.

- 객체(**Object**) → {} 중괄호 사용
- 배열(**Array**) → [] 대괄호 사용

✓ JSON 기본 예제

json

코드 복사

```
{  
  "name": "Alice",  
  "age": 25,  
  "isStudent": false,  
  "skills": ["JavaScript", "Python", "C++"],  
  "address": {  
    "city": "Seoul",  
    "zipCode": "12345"  
  }  
}
```

- ✓ 문자열은 항상 "(큰따옴표)"로 감싸야 함
 - ✓ 숫자, 불리언, 배열, 객체 등의 데이터 타입 지원
-

📌 3. JSON 데이터 타입

JSON에서 사용할 수 있는 데이터 타입은 다음과 같습니다.

| 데이터 타입 | 예제 |
|---------------|--------------------------------------|
| 문자열 (String) | "name": "Alice" |
| 숫자 (Number) | "age": 25 |
| 불리언 (Boolean) | "isStudent": false |
| 배열 (Array) | "skills": ["JavaScript", "Python"] |
| 객체 (Object) | "address": { "city": "Seoul" } |
| null 값 | "nickname": null |

📌 4. JSON과 JavaScript 객체 변환

✓ JavaScript 객체 → JSON 변환 (`JSON.stringify()`)

javascript
코드 복사

```
let person = {
  name: "Alice",
  age: 25,
  isStudent: false
};
```

```
let jsonString = JSON.stringify(person);
console.log(jsonString);
// '{"name":"Alice", "age":25, "isStudent":false}' (문자열 형태)
```

✓ `JSON.stringify()` 는 객체를 JSON 문자열로 변환

✓ JSON → JavaScript 객체 변환 (`JSON.parse()`)

javascript
코드 복사

```
let jsonData = '{"name": "Alice", "age": 25, "isStudent": false}';
```

```
let obj = JSON.parse(jsonData);
console.log(obj.name); // Alice
console.log(obj.age); // 25
```

✓ `JSON.parse()` 는 JSON 문자열을 JavaScript 객체로 변환

JSON과 JavaScript 객체는 서로 비슷하지만 중요한 차이점이 있어요.

✓ 공통점

- 둘 다 키-값(**key-value**) 구조를 가지고 있음
- 데이터를 표현하는데 사용됨
- 중괄호 `{}`를 사용하여 객체를 나타냄

● 차이점

| 차이점 | JSON | JavaScript 객체 |
|-----------|---|---|
| 형식 | 문자열(String) | 객체(Object) |
| 데이터 타입 | 문자열, 숫자, 불리언, 배열, null ,
객체만 가능 | 함수, undefined ,
Symbol 등도 가능 |
| 속성 이름 | 항상 큰따옴표(" ")로 감싸야 함 | 큰따옴표 없이도 가능 |
| 사용 가능 여부 | 언어나 환경에 관계없이 사용 가능
(Python, Java, C 등) | JavaScript에서만 사용 가능 |
| 메서드 포함 여부 | 값으로 메서드(함수) 저장 불가 | 가능 |

> 07. 이벤트 추가하기

부모 컨트롤러에서 자식 컨트롤러에 이벤트를 전송할 수 있다.

속성으로 함수를 받아서 화면의 jsx의 태그 이벤트안에서 호출하면 이벤트 핸들러를 전달해 주는것 처럼 사용할 수 있다.

부모 컨트롤러에서 속성으로 이벤트 핸들러를 다음과 같이 넘긴다.

`<ChildComponent handleClick={handleClick} />`

`handleClick`은 이벤트 핸들러 메소드이고 해당 이벤트 핸들러 메소드를 자식 컨트롤러에서 `prop`에서 찾아 이벤트에 연결한다.

`<button onClick={prop.handleClick}>클릭하세요</button>`

자식컨포넌트의 버튼을 클릭하면 부모에서 넘어온 이벤트 핸들러 메소드가 실행된다.

자바스크립트 간단 이벤트 처리 예제

```
<!DOCTYPE html>
```

```
<html lang="ko">
```

```
<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Click 이벤트 예제</title>

</head>

<body>

    <h1>Click 이벤트 예제</h1>

    <button onclick="changeText()">클릭하세요</button>

    <p id="text">버튼을 클릭해 보세요!</p>




<script>

    function changeText() {

        console.log("버튼이 클릭되었습니다.");

    }

</script>

</body>

</html>
```

```
function ChildComponent() {

    const handleButtonClick = () => {

        console.log("Button clicked!"); // 고정된 기능 추가

    };

}
```

```
return (
  <div>
    <button onClick={handleButtonClick}>클릭하세요</button>
  </div>
);
}
```

고정 이벤트

```
import React from "react";
import ChildComponent from "./ChildComponent";

function App() {
  return (
    <div>
      <h1>부모 컴포넌트</h1>
      <ChildComponent />
    </div>
  );
}

export default App;
```

그때 그때 이벤트 핸들러 변경하기

```
import React from 'react';
function ChildComponent(prop) {
  return (
    <div>
      {/* prop 객체를 통해 handleClick 함수에 접근합니다. */}
      <button onClick={prop.handleClick}>클릭하세요</button>
    </div>
  );
}

function App() {
  // 이벤트 핸들러
  const handleClick = () => {
    alert('버튼이 클릭되었습니다!');
  };
  const handleClick1 = () => {
    alert('handleClick1 이벤트 핸들러');
  };
  return (
    <div>
      {/* ChildComponent에 handleClick 함수를 prop으로 전달합니다. */}
      <ChildComponent handleClick={handleClick} />
      <ChildComponent handleClick={handleClick} />
      <ChildComponent handleClick={handleClick1} />
    </div>
  );
}

```

이벤트

이벤트(event)는 사용자의 행동(버튼 클릭, 입력 필드 변경, 키보드 입력 등)에 반응하는 기능을 말합니다.

React에서는 이벤트 핸들러(event handler) 함수를 만들어서 이벤트를 처리할 수 있습니다.

◆ 1. 버튼 클릭 이벤트 (onClick)

버튼을 클릭하면 handleClick 함수가 실행됩니다.

jsx

복사편집

```
import React from "react";

function App() {
  const handleClick = () => {
    alert("버튼이 클릭되었습니다! 🎉");
  };

  return (
    <div>
      <h1>이벤트 기초 예제</h1>
      <button onClick={handleClick}>클릭하세요</button>
    </div>
  );
}

export default App;
```

✓ onClick={handleClick} → 버튼을 클릭할 때 handleClick 함수 실행

✓ alert("버튼이 클릭되었습니다!") → 클릭하면 팝업 메시지 표시

◆ 2. 입력 필드 변경 이벤트 (onChange)

사용자가 입력하면, 입력한 값을 실시간으로 화면에 표시하는 예제입니다.

jsx

복사편집

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

function App() {
  const [text, setText] = useState("");

  const handleChange = (event) => {
    setText(event.target.value);
  };

  return (
    <div>
      <h1>입력 필드 이벤트</h1>
      <input type="text" onChange={handleChange} placeholder="입력하세요" />
      <p>입력한 값: {text}</p>
    </div>
  );
}

export default App;
```

✓ useState를 사용하여 입력값(text)을 저장

✓ onChange={handleChange} → 입력할 때마다 handleChange 실행

✓ event.target.value → 사용자가 입력한 값을 가져와 text에 저장

◆ 3. 마우스 이벤트 (onMouseEnter, onMouseLeave)

마우스를 올리거나 벗어날 때 배경색이 변하는 예제입니다.

jsx

복사편집

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

function App() {
  const [bgColor, setBgColor] = useState("white");

  return (
    <div>
      <h1>마우스 이벤트</h1>
      <div
        style={{ width: "200px", height: "100px", backgroundColor:
        bgColor, textAlign: "center", lineHeight: "100px" }}
        onMouseEnter={() => setBgColor("lightblue")}
        onMouseLeave={() => setBgColor("white")}
      >
        마우스를 올려보세요!
      </div>
    </div>
  );
}

export default App;
```

onMouseEnter → 마우스를 올리면 배경색 변경

onMouseLeave → 마우스를 벗어나면 원래 색으로 변경

◆ 4. 키보드 이벤트 (onKeyDown, onKeyUp)

사용자가 키보드를 누를 때, 어떤 키를 눌렀는지 출력하는 예제입니다.

```
jsx
복사편집
import React, { useState } from "react";

function App() {
  const [key, setKey] = useState("");

  const handleKeyDown = (event) => {
    setKey(event.key);
  };

  return (
    <div>
      <h1>키보드 이벤트</h1>
      <input type="text" onKeyDown={handleKeyDown} placeholder="키를
눌러보세요" />
      <p>입력한 키: {key}</p>
    </div>
  );
}

export default App;
```

onKeyDown → 사용자가 키를 누를 때 `handleKeyDown` 실행

event.key → 누른 키 값을 가져와 `key` 상태에 저장

◆ 5. 폼 제출 이벤트 (onSubmit)

폼을 제출할 때 입력한 데이터를 확인하는 예제입니다.

jsx

복사편집

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

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

  const handleSubmit = (event) => {
    event.preventDefault(); // 기본 폼 제출 동작 방지
    alert(`제출된 이름: ${name}`);
  };

  return (
    <div>
      <h1>폼 제출 이벤트</h1>
      <form onSubmit={handleSubmit}>
        <input type="text" value={name} onChange={(e) =>
          setName(e.target.value)} placeholder="이름 입력" />
        <button type="submit">제출</button>
      </form>
    </div>
  );
}

export default App;
```

-
- event.preventDefault() → 페이지 새로고침 방지
 - onSubmit={handleSubmit} → 제출 버튼 클릭 시 실행

🎯 정리

| 이벤트 종류 | 설명 | 예제 |
|-----------------------------|------------------------|-----------|
| onClick | 클릭 시 실행 | 버튼 클릭 |
| onChange | 입력값 변경 시 실행 | 텍스트 입력 필드 |
| onMouseEnter / onMouseLeave | 마우스 올릴 때 / 벗어날 때
실행 | 배경색 변경 |
| onKeyDown / onKeyUp | 키보드 누를 때 / 뗄 때 실행 | 키 입력 감지 |
| onSubmit | 폼 제출 시 실행 | 폼 데이터 확인 |

이제 React에서 이벤트를 쉽게 다룰 수 있을 거예요! 🎉

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

// 폼 컴포넌트
function FormComponent({ name, setName, handleSubmit }) {
  return (
    <div>
      <h1>폼 제출 이벤트</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
          placeholder="이름 입력"
        />
        <button type="submit">제출</button>
      </form>
    </div>
  );
}

// 메인 컴포넌트
```

```
function App() {
  const [name, setName] = useState("");
  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`제출된 이름: ${name}`);
  };
  return (
    <div>
      <FormComponent name={name} setName={setName} handleSubmit={handleSubmit}>
    />
    </div>
  );
}
export default App;
```

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

function App() {
  const [students, setStudents] = useState([
    { id: 1, name: "Alice", age: 21 },
    { id: 2, name: "Bob", age: 22 },
    { id: 3, name: "Charlie", age: 23 },
  ]);
  const [selectedStudent, setSelectedStudent] = useState(null);
  const [newStudent, setNewStudent] = useState({ name: "", age: "" });

  return (
    <div style={{ padding: "20px" }}>
      <h1>학생 목록</h1>
      <ul>
        {students.map((student) => (
          <li key={student.id}>
            <button onClick={() => setSelectedStudent(student)}>
              {student.name}
            </button>
          </li>
        )));
      </ul>

      {selectedStudent && (

```

```
<div style={{ marginTop: "20px", border: "1px solid black", padding: "10px" }}>
  <h2>학생 정보</h2>
  <p>이름: {selectedStudent.name}</p>
  <p>나이: {selectedStudent.age}세</p>
</div>
)

<div style={{ marginTop: "20px" }}>
  <h2>새 학생 추가</h2>
  <input
    type="text"
    placeholder="이름"
    value={newStudent.name}
    onChange={(e) => setNewStudent({ ...newStudent, name: e.target.value })}>
  />
  <input
    type="number"
    placeholder="나이"
    value={newStudent.age}
    onChange={(e) => setNewStudent({ ...newStudent, age: e.target.value })}>
  />
  <button
    onClick={() => {
      const newId = students.length + 1;
      setStudents([...students, { id: newId, name: newStudent.name, age: newStudent.age }]);
      setNewStudent({ name: "", age: "" });
    }}
  >
    추가
  </button>
</div>
</div>
);

export default App;
```

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

const quizData = [
  { question: "지구는 태양을 돋다.", answer: "0" },
  { question: "1 + 1 = 3 이다.", answer: "X" },
  { question: "코끼리는 날 수 있다.", answer: "X" },
  { question: "물은 얼면 부피가 줄어든다.", answer: "X" },
];

function App() {
  const [currentIndex, setCurrentIndex] = useState(0);
  const [score, setScore] = useState(0);
  const [showResult, setShowResult] = useState(false);

  const handleAnswer = (userAnswer) => {
    if (userAnswer === quizData[currentIndex].answer) {
      setScore(score + 1);
      alert("정답입니다! ✓");
    } else {
      alert("틀렸습니다! ✗");
    }
  }

  if (currentIndex + 1 < quizData.length) {
    setCurrentIndex(currentIndex + 1);
  } else {
    setShowResult(true);
  }
}

export default App;
```

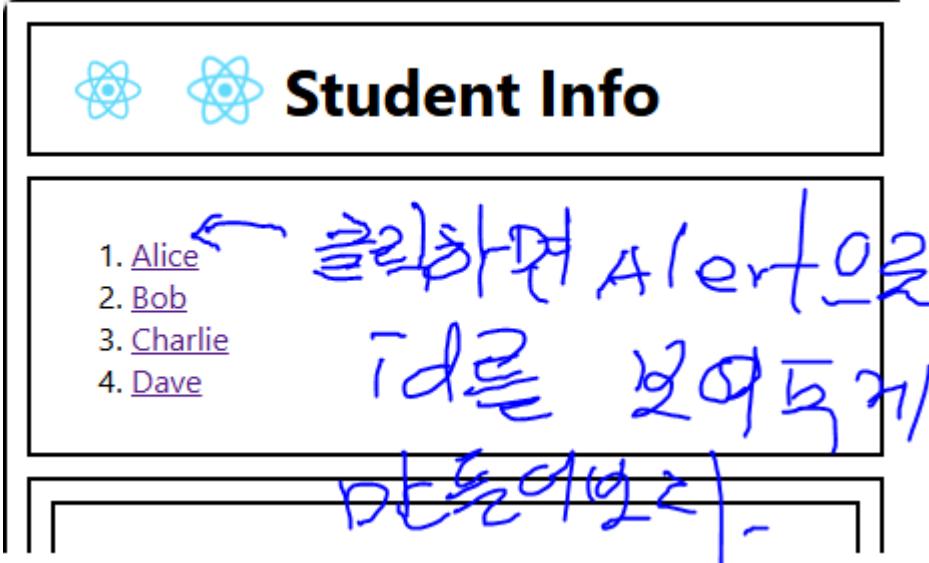
```
        }

    );

    return (
        <div style={{ textAlign: "center", padding: "20px" }}>
            <h1>OX 퀴즈 게임 🏆</h1>

            {!showResult ? (
                <>
                    <h2>{quizData[currentIndex].question}</h2>
                    <button
                        onClick={() => handleAnswer("O")}
                        style={{ marginRight: "10px", padding: "10px 20px", fontSize: "18px" }}
                    >
                        O
                    </button>
                    <button
                        onClick={() => handleAnswer("X")}
                        style={{ padding: "10px 20px", fontSize: "18px" }}
                    >
                        X
                    </button>
                </>
            ) : (
                <div>
                    <h2>게임 종료!</h2>
                    <p>당신의 점수: {score} / {quizData.length}</p>
                    <button onClick={() => { setCurrentIndex(0); setScore(0);
setShowResult(false); }}>
                        다시 시작 ⚡
                    </button>
                </div>
            )
        );
    );
}

export default App;
```



```
function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a
        id={t.id}
        href={`/read/${t.id}`}
        onClick={(event) => {
          event.preventDefault();
          props.onChangeMode(event.target.id);
        }}
      >
        {t.name}
      </a>
    </li>
  )));
  return (
    <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <ol>{list}</ol>
    </nav>
  );
}
```

App의 Nav컴포넌트 변경

```
<Nav
  students={students}
  onChangeMode={(id) => {
    alert(id);
 }}
/>
```

적용한 전체 코드

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

function Header() {
  return (
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
```

```
<div style={{ display: 'flex', alignItems: 'center' }}>
  <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
  
</div>
<h1 style={{ margin: 0 }}>Student Info</h1>
</div>
</header>
);
}
function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a
        id={t.id}
        href={'/read/' + t.id}
        onClick={(event) => {
          event.preventDefault();
          props.onChangeMode(event.target.id);
        }}
      >
        {t.name}
      </a>
    </li>
  ));
  return (
    <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <ol>{list}</ol>
    </nav>
  );
}

function Article(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>{props.title}</h2>
      {props.body}
    </article>
  );
}
```

```
function Create() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Create</h2>
      <form>
        <p><input type="text" placeholder="name" /></p>
        <p><input type="text" placeholder="username" /></p>
        <p><input type="number" placeholder="age" /></p>
        <p><input type="number" placeholder="height" /></p>
        <p><input type="date" placeholder="join date" /></p>
        <p><input type="submit" value="Create" /></p>
      </form>
    </article>
  );
}

function Update() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Update</h2>
      <form>
        <p><input type="text" defaultValue="name" placeholder="name" /></p>
        <p><input type="text" defaultValue="username" placeholder="username" /></p>
        <p><input type="number" defaultValue="age" placeholder="age" /></p>
        <p><input type="number" defaultValue="height" placeholder="height" /></p>
        <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
        <p><input type="submit" value="Update" /></p>
      </form>
    </article>
  );
}
```

```
function Footer() {
  return (
    <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>
      <ul>
        <li><a href="/create">Create</a></li>
        <li><a href="/update">Update</a></li>
      </ul>
    </footer>
  );
}

function App() {
  const students=[
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ];
  return (
    <div style={{ border: '2px solid black', padding: '10px' }}>
      <Header />
      <Nav
        students={students}
        onChangeMode={(id) => {
          alert(id);
        }}
      />
      <Article title="Select Student" body="Welcome to the Student Info App" />
      <Footer /> {/* Footer 컴포넌트를 여기에서 추가합니다 */}
    </div>
  );
}

export default App;
```

배열 구조 분해 할당 (**Destructuring Assignment**)

```
javascript
복사편집
const [count, setCount] = useState(0);
```

이 코드에서 사용된 배열 구조 분해 할당(**Destructuring Assignment**) 문법을 설명하겠습니다.

1. 기본 개념

배열 구조 분해 할당은 배열의 요소를 변수에 쉽게 할당할 수 있도록 해주는 **JavaScript** 문법입니다.

예제

```
javascript
복사편집
const arr = [1, 2];
const [a, b] = arr;

console.log(a); // 1
console.log(b); // 2
```

- `arr` 배열의 첫 번째 요소(1)가 `a`에, 두 번째 요소(2)가 `b`에 할당됩니다.
-

2. useState에서의 배열 구조 분해 할당

React의 `useState` 혹은 배열을 반환하는데, 이를 배열 구조 분해 할당을 이용하여 변수에 저장합니다.

```
javascript
복사편집
const [count, setCount] = useState(0);
```

- `useState(0)`는 [현재 상태 값, 상태 변경 함수] 형태의 배열을 반환합니다.
- 배열 구조 분해 할당을 사용하여:
 - `count`는 현재 상태 값
 - `setCount`는 상태를 변경하는 함수로 저장됩니다.

이와 동일한 동작을 하는 코드 (배열 구조 분해 할당 없이)

```
javascript
복사편집
const [count, setCount] = useState(0);

const stateArray = useState(0);
const count = stateArray[0];
const setCount = stateArray[1];
```

위처럼 `useState(0)`가 반환한 배열을 직접 변수에 저장할 수도 있지만, 배열 구조 분해 할당을 사용하면 훨씬 간결하게 코드를 작성할 수 있습니다.

3. 배열 구조 분해 할당의 특징

1) 필요한 값만 할당 가능

javascript

복사편집

```
const [count] = useState(0);
console.log(count); // 현재 상태 값 (0)
```

- `setCount`를 사용하지 않을 경우 생략 가능하지만, 일반적으로 `setCount`도 함께 사용해야 하므로 생략하지 않는 것이 좋습니다.

2) 기본값 설정 가능

javascript

복사편집

```
const [a = 10, b = 20] = [];
console.log(a); // 10
console.log(b); // 20
```

- 빈 배열일 경우 기본값이 할당됩니다.
-

4. 정리

- 배열 구조 분해 할당은 배열의 요소를 개별 변수로 쉽게 저장할 수 있는 문법입니다.
- `useState`는 `[현재 상태, 상태 변경 함수]` 형태의 배열을 반환하므로, 배열 구조 분해 할당을 사용하면 편리합니다.
- `const [count, setCount] = useState(0);`는 `useState`의 반환값을 각각 `count`(현재 값)와 `setCount`(변경 함수)로 나누어 저장하는 코드입니다.

즉, 배열 구조 분해 할당을 이용하면 배열에서 원하는 값을 추출하여 가독성 좋고 간결한 코드를 작성할 수 있습니다! 

> 08.STATE 개념 이해하기

state는 컴포넌트의 상태를 관리하는 객체입니다. 컴포넌트의 **state**는 컴포넌트의 데이터나 UI 상태를 저장하고, **state**가 변경되면 컴포넌트가 다시 렌더링됩니다.

state를 사용하지 않고 일반 변수를 사용하면 변수의 값이 변경되더라도 화면 갱신이 이루어지지 않는다.

`prop`은 외부에서 값을 설정해서 디자인에 적용하는 방법이고, `state`는 내부에서 값을 변경해서 디자인에 적용하는 방법이다.

리액트에는 여러 종류의 툈이 있는데 `state`처럼 값이 변경되는 것을 감시하는 기능이 있는 것들을 툈이라고 한다.

`state`가 적용 안되는 예제

```
import React from 'react';

const MyComponent = () => {
  let count = 0; // useState를 사용하지 않음

  const increment = () => {
    count += 1; // 값은 증가하지만, 리렌더링이 발생하지 않음
    console.log(count); // 콘솔에는 증가된 값이 보이지만, UI에는 반영되지 않음
  };

  return (
    <div>
      <p>Count: {count}</p> {/* UI가 업데이트되지 않음 */}
      <button onClick={increment}>Increment</button>
    </div>
  );
};

export default MyComponent;
```



```

import React, { useState } from 'react';
const MyComponent = () => {
  // 상태 변수와 상태를 업데이트하는 함수 선언
  const [count, setCount] = useState(0);
  // 상태를 업데이트하는 함수
  const increment = () => {
    setCount(count + 1);
  };
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
};

function App() {

  return (
    <div>
      <MyComponent/>
    </div>
  );
}

export default App;

```

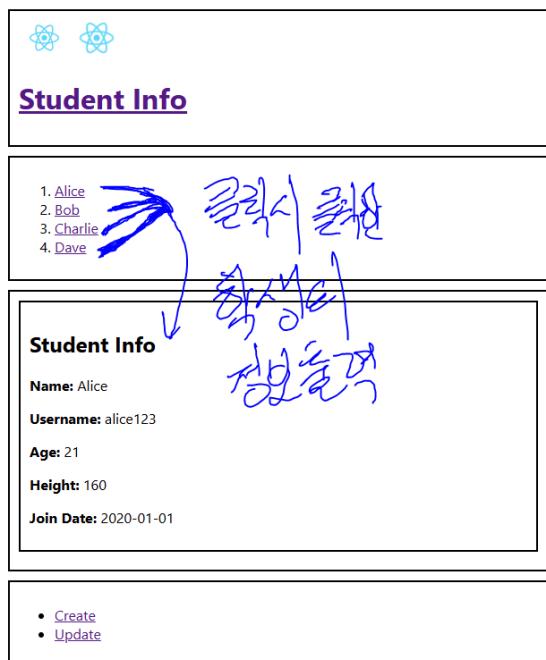
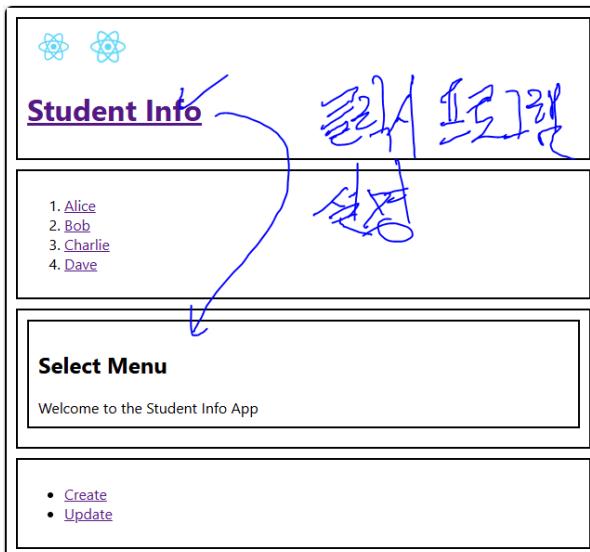
사용방법

1. useState를 import한다.
2. useState를 선언한다.
3. set함수를 사용해서 값을 변경한다.

```

const [count, setCount] = useState(0);
setCount(count + 1);

```



전체 코드

```

import React,{useState} from 'react';

import './App.css';

import logo from './logo.svg';

function Header(props) {
  return (
    <div>
      <img alt="Logo" src={logo}>
      <h1>Student Info</h1>
    </div>
  );
}

function App() {
  const [name, setName] = useState('Alice');
  const [username, setUsername] = useState('alice123');

  const handleNameChange = (e) => {
    setName(e.target.value);
  };

  const handleUsernameChange = (e) => {
    setUsername(e.target.value);
  };

  const handleCreateClick = () => {
    alert(`Create button clicked for ${name}`);
  };

  const handleUpdateClick = () => {
    alert(`Update button clicked for ${name}`);
  };

  return (
    <div>
      <h2>Student Info</h2>
      <p>Name: ${name}</p>
      <p>Username: ${username}</p>
      <input type="text" value={name} onChange={handleNameChange}/>
      <input type="text" value={username} onChange={handleUsernameChange}/>
      <button onClick={handleCreateClick}>Create</button>
      <button onClick={handleUpdateClick}>Update</button>
    </div>
  );
}

export default App;
  
```

```
        <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
          <div style={{ display: 'flex', alignItems: 'center' }}>
            <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
            
          </div>
          <h1 style={{ margin: 0 }}><a href="/" onClick={(event)=>{
            event.preventDefault();
            props.onChangeMode();
          }}>Student Info</a></h1>
        </div>
      </header>
    );
}

function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a
        id={t.id}
        href={'/read/' + t.id}
        onClick={(event) => {
          event.preventDefault();
          props.onChangeMode(event.target.id);
        }}
      >
        {t.name}
      </a>
    </li>
  ));
  return (
    <div style={{ display: 'flex', justify-content: 'space-around', width: '100%' }}>
      {list}
    </div>
  );
}
```

```
        </a>

    </li>

));
return (
<nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
    <ol>{list}</ol>
</nav>
);
}
```

```
function Article(props) {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>{props.title}</h2>
            {props.body}
        </article>
    );
}
```

```
function Create() {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Create</h2>  
      <form>  
        <p><input type="text" placeholder="name" /></p>  
        <p><input type="text" placeholder="username" /></p>  
        <p><input type="number" placeholder="age" /></p>  
        <p><input type="number" placeholder="height" /></p>  
        <p><input type="date" placeholder="join date" /></p>  
        <p><input type="submit" value="Create" /></p>  
      </form>  
    </article>  
  );  
}
```

```
function Update() {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Update</h2>  
      <form>  
        <p><input type="text" defaultValue="name" placeholder="name" /></p>  
        <p><input type="text" defaultValue="username" placeholder="username" /></p>  
        <p><input type="number" defaultValue="age" placeholder="age" /></p>  
        <p><input type="number" defaultValue="height" placeholder="height" /></p>  
        <p><input type="date" defaultValue="join date" placeholder="join date" /></p>  
        <p><input type="submit" value="Update" /></p>  
      </form>  
    </article>  
  );  
}
```

```

function Footer() {
  return (
    <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>
      <ul>
        <li><a href="/create">Create</a></li>
        <li><a href="/update">Update</a></li>
      </ul>
    </footer>
  );
}

}

```

```

function App() {

  const [mode, setMode]=useState('MENU');

  const students=[

    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ];

  let contentSelectMenu=<Article title="Select No" body="Welcome to the Student Info

```

```
App" />;

switch(mode){

  case "MENU"://상단 클릭시

    contentSelectMenu=<Article title="Select Top Menu" body="Welcome to the Student
Info App" />;

    break;

  case "READ"://학생데이터 클릭시

    contentSelectMenu=<Article title="Select Student" body="Welcome to the Student
Info App" />;

    break;

  case "CREATE"://학생생성화면

    contentSelectMenu=<Article title="CREATE Student" body="Welcome to the Student
Info App" />;

    break;

  case "UPDATE"://학생수정화면

    contentSelectMenu=<Article title="UPDATE Student" body="Welcome to the Student
Info App" />;

    break;

}

// const [mode,setMode]=useState('MENU');

return (

```

```

<div style={{ border: '2px solid black', padding: '10px' }}>

  <Header onChangeMode={

    ()=>{

      setMode('MENU');

    }/>

  <Nav

    students={students}

    onChangeMode={(id) => {

      setMode('READ');

      alert(id);

    }}

  />

  {contentSelectMenu}

  <Footer /> {/* Footer 컴포넌트를 여기에서 추가합니다 */}

</div>

);

}

export default App;

```

> 09.GUI CREATE,UPDATE 추가

CREATE와 UPDATE를 클릭 하였을 때 화면이 변경되도록 구현해 보자.

```

import React, { useState } from 'react';

import './App.css';

import logo from './logo.svg';

```

```

function Header(props) {
  return (
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <div style={{ display: 'flex', alignItems: 'center' }}>
        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
        
      </div>
      <h1 style={{ margin: 0 }}>
        <a href="/" onClick={(event) => {
          event.preventDefault();
          props.onChangeMode();
        }}>
          >
          Student Info
        </a>
      </h1>
    </div>
  </header>
);
}


```

```

function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a

```

```
        id={t.id}

        href={'/read/' + t.id}

        onClick={(event) => {
            event.preventDefault();
            props.onChangeMode(event.target.id);
        }}

    >

    {t.name}

</a>

</li>

));

return (

<nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

<ol>{list}</ol>

</nav>

);

}

function Article(props) {

return (

<article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

<h2>{props.title}</h2>

{props.body}

</article>

);

}
```

```
function Create() {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>Create</h2>
            <form>
                <p><input type="text" placeholder="name" /></p>
                <p><input type="text" placeholder="username" /></p>
                <p><input type="number" placeholder="age" /></p>
                <p><input type="number" placeholder="height" /></p>
                <p><input type="date" placeholder="join date" /></p>
                <p><input type="submit" value="Create" /></p>
            </form>
        </article>
    );
}

function Update() {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>Update</h2>
            <form>
                <p><input type="text" defaultValue="name" placeholder="name" /></p>
                <p><input type="text" defaultValue="username" placeholder="username" /></p>
                <p><input type="number" defaultValue="age" placeholder="age" /></p>
                <p><input type="number" defaultValue="height" placeholder="height" /></p>
                <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
            </form>
        </article>
    );
}
```

```
<p><input type="submit" value="Update" /></p>

</form>

</article>

);

}

function Footer(props) {

return (

<footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>

<ul>

<li>

<a

  href="/create"

  onClick={(event) => {

    event.preventDefault();

    props.onChangeMode('CREATE');

  }}

>

Create

</a>

</li>

<li>

<a

  href="/update"

  onClick={(event) => {

    event.preventDefault();

    props.onChangeMode('UPDATE');

  }}

>
```

```
        }}

      >

      Update

    </a>

  </li>

</ul>

</footer>

);

}

function App() {

  const [mode, setMode] = useState('MENU');

  ...

  const students = [
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ];

  let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />

  switch (mode) {

    case 'MENU':
```

```
        contentSelectMenu = <Article title="Select Top Menu" body="Welcome to the Student
Info App" />;
        break;

    case 'READ':
        contentSelectMenu = <Article title="Select Student" body="Welcome to the Student
Info App" />;
        break;

    case 'CREATE':
        contentSelectMenu = <Create />;
        break;

    case 'UPDATE':
        contentSelectMenu = <Update />;
        break;

    default:
        break;
    }

    return (
        <div style={{ border: '2px solid black', padding: '10px' }}>
            <Header onChangeMode={() => setMode('MENU')} />
            <Nav
                students={students}
                onChangeMode={(id) => {
                    setMode('READ');
                    alert(id);
                }}
            />
            {contentSelectMenu}
            <Footer onChangeMode={setMode} /> {/* Footer에서 setMode를 props로 전달 */}
        
```

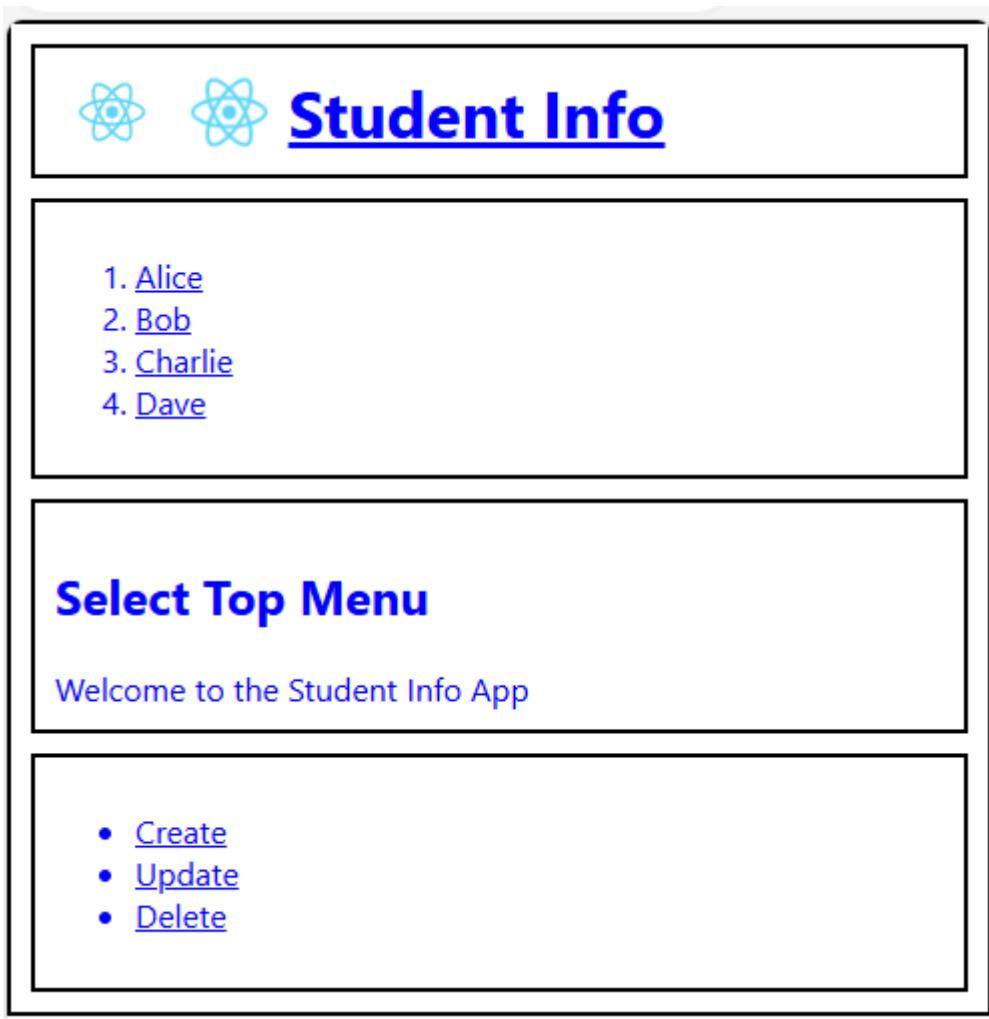
```
</div>

);

}

export default App;
```

> 09.GUI 최종(DELETE 추가)



delete를 클릭시 화면에 표시하는 부분을 추가 하였다.

```
import React, { useState } from 'react';

import './App.css';

import logo from './logo.svg';

function Header(props) {

  return (

    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

      <div style={{ display: 'flex', alignItems: 'center' }}>

        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
```

```

        <h1 style={{ margin: 0 }}>
            <a href="/" onClick={(event) => {
                event.preventDefault();
                props.onChangeMode();
            }}>
                >
                Student Info
            </a>
        </h1>
    </div>
</header>
);
}

```

```

function Nav(props) {
    const list = props.students.map((t) => (
        <li key={t.id}>
            <a id={t.id}>
                href={'/read/' + t.id}
                onClick={(event) => {
                    event.preventDefault();
                    props.onChangeMode(event.target.id);
                }}
            </a>
        </li>
    ));
}

```

```
        })
      >
      {t.name}
    </a>
  </li>
);
return (
<nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
  <ol>{list}</ol>
</nav>
);
}

function Article(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>{props.title}</h2>
      {props.body}
    </article>
  );
}

function Create() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
```

```
    })>

    <h2>Create</h2>

    <form>

        <p><input type="text" placeholder="name" /></p>
        <p><input type="text" placeholder="username" /></p>
        <p><input type="number" placeholder="age" /></p>
        <p><input type="number" placeholder="height" /></p>
        <p><input type="date" placeholder="join date" /></p>
        <p><input type="submit" value="Create" /></p>

    </form>

</article>

);

}

function Update() {

    return (

        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

            <h2>Update</h2>

            <form>

                <p><input type="text" defaultValue="name" placeholder="name" /></p>
                <p><input type="text" defaultValue="username" placeholder="username" /></p>
                <p><input type="number" defaultValue="age" placeholder="age" /></p>
                <p><input type="number" defaultValue="height" placeholder="height" /></p>
                <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
                <p><input type="submit" value="Update" /></p>

            </form>

    )

}
```

```
</article>

);

}

function Delete(props) {

return (

<article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

<h2>Confirm Delete</h2>

<p>Are you sure you want to delete this student?</p>

</article>

);

}

function Footer(props) {

return (

<footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>

<ul>

<li>

<a

href="/create"

onClick={(event) => {

event.preventDefault();

props.onChangeMode('CREATE');

}}>

Create

</a>

</li>


```

```
<li>
  <a
    href="/update"
    onClick={(event) => {
      event.preventDefault();
      props.onChangeMode('UPDATE');
    }}
  >
    Update
  </a>
</li>

<li>
  <a
    href="/delete"
    onClick={(event) => {
      event.preventDefault();
      props.onChangeMode('DELETE');
    }}
  >
    Delete
  </a>
</li>
</ul>
</footer>
);

}

function App() {
```

```
const [mode, setMode] = useState('MENU');

const students = [
  { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
  { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
  { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
  { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
];

let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />

switch (mode) {
  case 'MENU':
    contentSelectMenu = <Article title="Select Top Menu" body="Welcome to the Student Info App" />;
    break;
  case 'READ':
    contentSelectMenu = <Article title="Select Student" body="Welcome to the Student Info App" />;
    break;
  case 'CREATE':
    contentSelectMenu = <Create />;
    break;
  case 'UPDATE':
    contentSelectMenu = <Update />;
}
```

```

        break;

    case 'DELETE':
        contentSelectMenu = <Delete />;
        break;

    default:
        contentSelectMenu = <Article title="Select default" body="Welcome to the Student Info App" />;
    }

    return (
        <div style={{ border: '2px solid black', padding: '10px' }}>
            <Header onChangeMode={() => setMode('MENU')} />
            <Nav
                students={students}
                onChangeMode={(id) => {
                    setMode('READ');
                    alert(id);
                }}
            />
            {contentSelectMenu}
            <Footer onChangeMode={setMode} /* Footer에서 setMode를 props로 전달 */ />
        </div>
    );
}

export default App;

```

> 10. 컴포넌트에 값 세팅하기 (READ변경)

app.js 컴포넌트를 다음과 같이 변경한다.

mode 화면 선택, students 학생정보들, id 현재선택된 학생의 id저장, setNextId 새로 추가한 학생의 id기록

```
const [mode, setMode] = useState('MENU');

const students = [
  { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
  { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
  { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
  { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
];

let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />;
```

다음과 같이 변경

```
const [mode, setMode] = useState('MENU');

const [id, setId] = useState(null);

const [nextId, setNextId] = useState(5);

const [students, setStudents] = useState([
  { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
  { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
  { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
  { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
]);

let contentSelectMenu = <Article title="Select No" body="Welcome to the Student App"
```

```
/>;
```

기존 읽을때 발생하는 코드

```
case 'READ':  
    contentSelectMenu = <Article title="Select Student" body="Welcome to the Student  
Info App" />;  
    break;
```

변경된 코드

```
case 'READ':  
  
    const student = students.find((student) => student.id === parseInt(id));  
  
    if (student) {  
  
        contentSelectMenu = (  
  
            <article style={{ border: '2px solid black', padding: '10px', marginBottom:  
'10px' }}>  
  
                <h2>Student Info</h2>  
  
                <p><strong>Name:</strong> {student.name}</p>  
  
                <p><strong>Username:</strong> {student.username}</p>  
  
                <p><strong>Age:</strong> {student.age}</p>  
  
                <p><strong>Height:</strong> {student.height}</p>  
  
                <p><strong>Join Date:</strong> {student.joinDate}</p>  
  
            </article>  
        );  
  
    }  
  
    break;
```

기존 클릭이벤트

<Nav

```
    students={students}

    onChangeMode={(id) => {
        setMode('READ');

        alert(id);

    }}
/>
```

변경된 코드

<Nav

```
    students={students}
    onChangeMode={(id) => {
        setMode('READ');
        setId(id);
    }}
/>
```

Read 컴포넌트 생성

```
function Read(props) {
    const { student } = props;
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>Student Info</h2>
            <p><strong>Name:</strong> {student.name}</p>
            <p><strong>Username:</strong> {student.username}</p>
            <p><strong>Age:</strong> {student.age}</p>
            <p><strong>Height:</strong> {student.height}</p>
            <p><strong>Join Date:</strong> {student.joinDate}</p>
        </article>
    );
}
```

```
}
```

화면 표시 switch문에서 생성한 Read컴포넌트 사용

```
case 'READ':  
    const student = students.find((student) => student.id === parseInt(id));  
    if (student) {  
        contentSelectMenu = <Read student={student} />;  
    }  
    break;
```

전체코드

```
import React, { useState } from 'react';  
import './App.css';  
import logo from './logo.svg';  
  
function Header(props) {  
    return (  
        <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
            <div style={{ display: 'flex', alignItems: 'center' }}>  
                <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />  
                  
                <h1 style={{ margin: 0 }}>  
                    <a  
                        href="/"  
                        onClick={(event) => {  
                            event.preventDefault();  
                            props.onChangeMode();  
                        }}>  
                        Student Info  
                    </a>  
                </h1>
```

```
        </div>
    </header>
);
}

function Nav(props) {
    const list = props.students.map((t) => (
        <li key={t.id}>
            <a
                id={t.id}
                href={'/read/' + t.id}
                onClick={(event) => {
                    event.preventDefault();
                    props.onChangeMode(event.target.id);
                }}
            >
                {t.name}
            </a>
        </li>
    )));
    return (
        <nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <ol>{list}</ol>
        </nav>
    );
}

function Article(props) {
    return (
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
            <h2>{props.title}</h2>
            {props.body}
        </article>
    );
}
```

```
function Create() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Create</h2>
      <form>
        <p><input type="text" placeholder="name" /></p>
        <p><input type="text" placeholder="username" /></p>
        <p><input type="number" placeholder="age" /></p>
        <p><input type="number" placeholder="height" /></p>
        <p><input type="date" placeholder="join date" /></p>
        <p><input type="submit" value="Create" /></p>
      </form>
    </article>
  );
}
```

```
function Update() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Update</h2>
      <form>
        <p><input type="text" defaultValue="name" placeholder="name" /></p>
        <p><input type="text" defaultValue="username" placeholder="username" /></p>
        <p><input type="number" defaultValue="age" placeholder="age" /></p>
        <p><input type="number" defaultValue="height" placeholder="height" /></p>
        <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
        <p><input type="submit" value="Update" /></p>
      </form>
    </article>
  );
}

function Delete(props) {
  return (
```

```
        <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
          <h2>Confirm Delete</h2>
          <p>Are you sure you want to delete this student?</p>
        </article>
      );
    }

function Read(props) {
  const { student } = props;
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Student Info</h2>
      <p><strong>Name:</strong> {student.name}</p>
      <p><strong>Username:</strong> {student.username}</p>
      <p><strong>Age:</strong> {student.age}</p>
      <p><strong>Height:</strong> {student.height}</p>
      <p><strong>Join Date:</strong> {student.joinDate}</p>
    </article>
  );
}

function Footer(props) {
  return (
    <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>
      <ul>
        <li>
          <a
            href="/create"
            onClick={(event) => {
              event.preventDefault();
              props.onChangeMode('CREATE');
            }}
          >
            Create
          </a>
        </li>
        <li>
```

```

        <a
          href="/update"
          onClick={(event) => {
            event.preventDefault();
            props.onChangeMode('UPDATE');
          }}
        >
          Update
        </a>
      </li>
      <li>
        <a
          href="/delete"
          onClick={(event) => {
            event.preventDefault();
            props.onChangeMode('DELETE');
          }}
        >
          Delete
        </a>
      </li>
    </ul>
  </footer>
);
}

function App() {
  const [mode, setMode] = useState('MENU');
  const [id, setId] = useState(null);
  const [nextId, setNextId] = useState(5);
  const [students, setStudents] = useState([
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160,
joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate:
'2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180,
joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate:
'2017-07-20' },
  ])
}

```

```
]);
let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />

switch (mode) {
  case 'MENU':
    contentSelectMenu = <Article title="Select Top Menu" body="Welcome to the Student Info App" />;
    break;
  case 'READ':
    const student = students.find((student) => student.id === parseInt(id));
    if (student) {
      contentSelectMenu = <Read student={student} />;
    }
    break;

  case 'CREATE':
    contentSelectMenu = <Create />;
    break;
  case 'UPDATE':
    contentSelectMenu = <Update />;
    break;
  default:
    case 'DELETE':
      contentSelectMenu = <Delete />;
      break;
}
return (
  <div style={{ border: '2px solid black', padding: '10px' }}>
    <Header onChangeMode={() => setMode('MENU')} />
    <Nav
      students={students}
      onChangeMode={(id) => {
        setMode('READ');
        setId(id);
      }}
    />
    {contentSelectMenu}
  
```

```
<Footer onChangeMode={setMode} /> {/* Footer에서 setMode를 props로 전달
*/}

</div>
);
}

export default App;
```

> 10.Create변경하기

기존 create컴포넌트를 삭제하고 새로운 형태로 추가한다.

```
function Create() {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }>
      <h2>Create</h2>
      <form>
        <p><input type="text" placeholder="name" /></p>
        <p><input type="text" placeholder="username" /></p>
        <p><input type="number" placeholder="age" /></p>
        <p><input type="number" placeholder="height" /></p>
        <p><input type="date" placeholder="join date" /></p>
        <p><input type="submit" value="Create" /></p>
      </form>
    </article>
  );
}
```

다음으로 변경

```
function Create(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }>
      <h2>Create</h2>
      <form
        onSubmit={(event) => {
          event.preventDefault();
          const name = event.target.name.value;
          const username = event.target.username.value;
          const age = event.target.age.value;
          const height = event.target.height.value;
          const joinDate = event.target.joinDate.value;
          props.onCreate({ name, username, age, height, joinDate });
        }}
      >
        <p>
          <input type="text" name="name" placeholder="name" />
        </p>
      
```

```

<p>
    <input type="text" name="username" placeholder="username" />
</p>
<p>
    <input type="number" name="age" placeholder="age" />
</p>
<p>
    <input type="number" name="height" placeholder="height" />
</p>
<p>
    <input type="date" name="joinDate" placeholder="join date" />
</p>
<p>
    <input type="submit" value="Create" />
</p>
</form>
</article>
);
}

```

switch부분 화면 변경

```

case 'CREATE':
    contentSelectMenu = <Create />;
    break;

```

를 다음으로 변경

```

case 'CREATE':
    contentSelectMenu = (
        <Create
            onCreate={({ name, username, age, height, joinDate }) => {
                const newStudent = { id: nextId, name, username, age, height, joinDate };
                const newStudents = [...students, newStudent];
                setStudents(newStudents);
                setMode('READ');
                setId(nextId);
                setNextId(nextId + 1);
            }}
        />
    );
    break;

```

전체 코드는 이전 전체 코드에서 상위 2부분을 변경하자.

> 11. UPDATE변경

기존 update

```
function Update() {
```

```

    return (
      <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
        <h2>Update</h2>
        <form>
          <p><input type="text" defaultValue="name" placeholder="name" /></p>
          <p><input type="text" defaultValue="username" placeholder="username" /></p>
          <p><input type="number" defaultValue="age" placeholder="age" /></p>
          <p><input type="number" defaultValue="height" placeholder="height" /></p>
          <p><input type="date" defaultValue="join date" placeholder="join date" /></p>
          <p><input type="submit" value="Update" /></p>
        </form>
      </article>
    );
  }
}

변경한 update

```

```

function Update(props) {
  const { name, username, age, height, joinDate, onUpdate } = props;
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Update</h2>
      <form
        onSubmit={(event) => {
          event.preventDefault();
          const updatedName = event.target.name.value;
          const updatedUsername = event.target.username.value;
        }}
      >
    
```

```
        const updatedAge = event.target.age.value;

        const updatedHeight = event.target.height.value;

        const updatedJoinDate = event.target.joinDate.value;

        onUpdate({ name: updatedName, username: updatedUsername, age: updatedAge,
height: updatedHeight, joinDate: updatedJoinDate });

    }

>

<p>

    <input type="text" name="name" defaultValue={name} placeholder="name" />

</p>

<p>

    <input type="text" name="username" defaultValue={username}
placeholder="username" />

</p>

<p>

    <input type="number" name="age" defaultValue={age} placeholder="age" />

</p>

<p>

    <input type="number" name="height" defaultValue={height} placeholder="height"
/>

</p>

<p>

    <input type="date" name="joinDate" defaultValue={joinDate} placeholder="join
date" />

</p>

<p>

    <input type="submit" value="Update" />

</p>

</form>
```

```
</article>

);

}

case 'UPDATE':  
    contentSelectMenu = <Update />;  
    break;
```

다음으로 변경

```
case 'UPDATE':  
    if (id === null) {  
        setMode('ERROR');  
    } else {  
        const updateStudent = students.find((student) => student.id ===  
parseInt(id));  
        if (updateStudent) {  
            contentSelectMenu = (  
                <Update  
                    name={updateStudent.name}  
                    username={updateStudent.username}  
                    age={updateStudent.age}  
                    height={updateStudent.height}  
                    joinDate={updateStudent.joinDate}  
                    onUpdate={({ name, username, age, height, joinDate }) => {
```

```

        const updateStudents = students.map((student) =>

            student.id === parseInt(id) ? { ...student, name, username, age,
height, joinDate } : student

        );

        setStudents(updateStudents);

        setMode('READ');

    }}

/>

);

}

break;

```

switch문에 ERROR추가

```

case 'ERROR':

    contentSelectMenu = <Article title="Error" body="Please select a student
to update." />;

    break;

```

전체코드

```

import React, { useState } from 'react';

import './App.css';

import logo from './logo.svg';

```

```
function Header(props) {  
  return (  
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <div style={{ display: 'flex', alignItems: 'center' }}>  
        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />  
          
      </div>  
      <h1 style={{ margin: 0 }}>  
        <a href="/" onClick={(event) => {  
          event.preventDefault();  
          props.onChangeMode();  
        }}>  
          >  
          Student Info  
        </a>  
      </h1>  
    </div>  
  </header>  
);  
}  
  
function Nav(props) {
```

```
const list = props.students.map((t) => (
  <li key={t.id}>
    <a
      id={t.id}
      href={'/read/' + t.id}
      onClick={(event) => {
        event.preventDefault();
        props.onChangeMode(event.target.id);
      }}
    >
      {t.name}
    </a>
  </li>
));
return (
<nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
  <ol>{list}</ol>
</nav>
);
}

function Article(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>{props.title}</h2>
      {props.body}
    </article>
  );
}
```

```
</article>

);

}

function Create(props) {

  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Create</h2>
      <form
        onSubmit={(event) => {
          event.preventDefault();
          const name = event.target.name.value;
          const username = event.target.username.value;
          const age = event.target.age.value;
          const height = event.target.height.value;
          const joinDate = event.target.joinDate.value;
          props.onCreate({ name, username, age, height, joinDate });
        }}
      >
        <p>
          <input type="text" name="name" placeholder="name" />
        </p>
        <p>
          <input type="text" name="username" placeholder="username" />
        </p>
        <p>
          <input type="number" name="age" placeholder="age" />
        </p>
      
```

```

        <p>
          <input type="number" name="height" placeholder="height" />
        </p>
        <p>
          <input type="date" name="joinDate" placeholder="join date" />
        </p>
        <p>
          <input type="submit" value="Create" />
        </p>
      </form>
    </article>
  );
}


```

```

function Update(props) {
  const { name, username, age, height, joinDate, onUpdate } = props;
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Update</h2>
      <form
        onSubmit={(event) => {
          event.preventDefault();
          const updatedName = event.target.name.value;
          const updatedUsername = event.target.username.value;
          const updatedAge = event.target.age.value;
          const updatedHeight = event.target.height.value;
        }}
      >
    
```

```
        const updatedJoinDate = event.target.joinDate.value;

        onUpdate({ name: updatedName, username: updatedUsername, age: updatedAge,
height: updatedHeight, joinDate: updatedJoinDate });

    }

>

<p>

    <input type="text" name="name" defaultValue={name} placeholder="name" />

</p>

<p>

    <input type="text" name="username" defaultValue={username}
placeholder="username" />

</p>

<p>

    <input type="number" name="age" defaultValue={age} placeholder="age" />

</p>

<p>

    <input type="number" name="height" defaultValue={height} placeholder="height"
/>

</p>

<p>

    <input type="date" name="joinDate" defaultValue={joinDate} placeholder="join
date" />

</p>

<p>

    <input type="submit" value="Update" />

</p>

</form>

</article>

);
```

```
}

function Delete(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Confirm Delete</h2>
      <p>Are you sure you want to delete this student?</p>
    </article>
  );
}

function Read(props) {
  const { student } = props;
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Student Info</h2>
      <p><strong>Name:</strong> {student.name}</p>
      <p><strong>Username:</strong> {student.username}</p>
      <p><strong>Age:</strong> {student.age}</p>
      <p><strong>Height:</strong> {student.height}</p>
      <p><strong>Join Date:</strong> {student.joinDate}</p>
    </article>
  );
}

function Footer(props) {
```

```
return (

<footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>

<ul>

<li>

<a href="/create"

onClick={(event) => {

  event.preventDefault();

  props.onChangeMode('CREATE');

}}>

Create

</a>

</li>

<li>

<a href="/update"

onClick={(event) => {

  event.preventDefault();

  props.onChangeMode('UPDATE');

}}>

Update

</a>

</li>

<li>

<a
```

```
        href="/delete"

        onClick={(event) => {
            event.preventDefault();
            props.onChangeMode('DELETE');
        }}
    >

    Delete

</a>
</li>
</ul>
</footer>
);

}

function App() {
    const [mode, setMode] = useState('MENU');

    const [id, setId] = useState(null);

    const [nextId, setNextId] = useState(5);

    const [students, setStudents] = useState([
        { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
        { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
        { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
        { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
    ]);
}
```

```
let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />;

switch (mode) {

  case 'MENU':

    contentSelectMenu = <Article title="Select Top Menu" body="Welcome to the Student Info App" />;

    break;

  case 'READ':

    const student = students.find((student) => student.id === parseInt(id));

    if (student) {

      contentSelectMenu = <Read student={student} />;

    }

    break;

  case 'CREATE':

    contentSelectMenu = (
      <Create
        onCreate={({ name, username, age, height, joinDate }) => {
          const newStudent = { id: nextId, name, username, age, height, joinDate };
          const newStudents = [...students, newStudent];
          setStudents(newStudents);
          setMode('READ');
          setId(nextId);
          setNextId(nextId + 1);
        }}
      />
    );
  }
}
```

```
break;

case 'UPDATE':
  if (id === null) {
    setMode('ERROR');
  } else {
    const updateStudent = students.find((student) => student.id ===
parseInt(id));
    if (updateStudent) {
      contentSelectMenu = (
        <Update
          name={updateStudent.name}
          username={updateStudent.username}
          age={updateStudent.age}
          height={updateStudent.height}
          joinDate={updateStudent.joinDate}
          onUpdate={({ name, username, age, height, joinDate }) => {
            const updateStudents = students.map((student) =>
              student.id === parseInt(id) ? { ...student, name, username, age,
height, joinDate } : student
            );
            setStudents(updateStudents);
            setMode('READ');
          }}>
      />
    );
  }
}
```

```
        }

        break;

    case 'DELETE':
        contentSelectMenu = <Delete />;
        break;

    case 'ERROR':
        contentSelectMenu = <Article title="Error" body="Please select a student to update." />;
        break;

    default:
        contentSelectMenu = <Article title="default Error" body="Welcome to the Student Info App" />;
}

return (
<div style={{ border: '2px solid black', padding: '10px' }}>
    <Header onChangeMode={() => setMode('MENU')} />
    <Nav
        students={students}
        onChangeMode={(id) => {
            setMode('READ');
            setId(id);
        }}
    />
    {contentSelectMenu}
    <Footer onChangeMode={setMode} /> {/* Footer에서 setMode를 props로 전달 */}
</div>
);
```

```
}

export default App;
```

> 12. Delete 변경하기

다음 코드를 변경한다.

```
function Delete(props) {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Confirm Delete</h2>  
      <p>Are you sure you want to delete this student?</p>  
    </article>  
  );  
}
```

변경할 코드

```
function Delete(props) {  
  return (  
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>  
      <h2>Confirm Delete</h2>  
      <p>Are you sure you want to delete this student?</p>  
      <button  
        onClick={() => {  
          if (props.onConfirm) {  
            props.onConfirm();  
          }  
        }}>  
        Yes  
      </button>  
      <button  
        onClick={() => {  
          if (props.onCancel) {  
            props.onCancel();  
          }  
        }}>  
        No  
      </button>  
    </article>  
  );  
}
```

```

    props.onCancel();

}

};

>

No

</button>

</article>

);

}

```

다음 코드를 변경한다.

```

case 'DELETE':
  contentSelectMenu = <Delete />;
  break;

```

```

case 'DELETE':
  if (id === null) {
    setMode('ERROR');
  } else {
    const studentToDelete = students.find((student) => student.id ===
parseInt(id));
    if (studentToDelete) {
      contentSelectMenu = (
        <Delete
          onConfirm={() => {
            const newStudents = students.filter((student) => student.id !==
parseInt(id));

```

```
        setStudents(newStudents);

        setMode('MENU');

        setId(null);

    }

    onCancel={() => {

        setMode('READ');

    }}

    />

);

}

break;
```

> 13. 최종 결과

Delete에서 완성된 코드와 조금 다르다.

```
import React, { useState } from 'react';

import './App.css';

import logo from './logo.svg';
```

```

function Header(props) {
  return (
    <header style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <div style={{ display: 'flex', alignItems: 'center' }}>
        <img src={logo} alt="Logo" style={{ height: '40px', marginRight: '10px' }} />
        
      </div>
      <h1 style={{ margin: 0 }}>
        <a href="/" onClick={(event) => {
          event.preventDefault();
          props.onChangeMode();
        }}>
          >
          Student Info
        </a>
      </h1>
    </div>
  </header>
);
}


```

```

function Nav(props) {
  const list = props.students.map((t) => (
    <li key={t.id}>
      <a

```

```
    id={t.id}

    href={'/read/' + t.id}

    onClick={(event) => {
        event.preventDefault();
        props.onChangeMode(event.target.id);
    }}
    >
    {t.name}
</a>
</li>
));
return (
<nav style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
<ol>{list}</ol>
</nav>
);
}

function Article(props) {
return (
<article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
<h2>{props.title}</h2>
{props.body}
</article>
);
}
```

```
function Create(props) {
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Create</h2>
      <form
        onSubmit={(event) => {
          event.preventDefault();
          const name = event.target.name.value;
          const username = event.target.username.value;
          const age = event.target.age.value;
          const height = event.target.height.value;
          const joinDate = event.target.joinDate.value;
          props.onCreate({ name, username, age, height, joinDate });
        }}
      >
        <p>
          <input type="text" name="name" placeholder="name" />
        </p>
        <p>
          <input type="text" name="username" placeholder="username" />
        </p>
        <p>
          <input type="number" name="age" placeholder="age" />
        </p>
        <p>
          <input type="number" name="height" placeholder="height" />
        </p>
      </form>
    </article>
  );
}
```

```
<p>
  <input type="date" name="joinDate" placeholder="join date" />
</p>
<p>
  <input type="submit" value="Create" />
</p>
</form>
</article>
);
}

function Update(props) {
  const { name, username, age, height, joinDate, onUpdate } = props;
  return (
    <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
      <h2>Update</h2>
      <form>
        <onSubmit={(event) => {
          event.preventDefault();
          const updatedName = event.target.name.value;
          const updatedUsername = event.target.username.value;
          const updatedAge = event.target.age.value;
          const updatedHeight = event.target.height.value;
          const updatedJoinDate = event.target.joinDate.value;
          onUpdate({ name: updatedName, username: updatedUsername, age: updatedAge, height: updatedHeight, joinDate: updatedJoinDate });
        }}>
      </form>
    </article>
  );
}
```

```
>

<p>
  <input type="text" name="name" defaultValue={name} placeholder="name" />
</p>

<p>
  <input type="text" name="username" defaultValue={username}
placeholder="username" />
</p>

<p>
  <input type="number" name="age" defaultValue={age} placeholder="age" />
</p>

<p>
  <input type="number" name="height" defaultValue={height} placeholder="height"
/>
</p>

<p>
  <input type="date" name="joinDate" defaultValue={joinDate} placeholder="join
date" />
</p>

<p>
  <input type="submit" value="Update" />
</p>
</form>
</article>
);

}

function Delete(props) {
```

```
return (

  <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>

    <h2>Confirm Delete</h2>

    <p>Are you sure you want to delete this student?</p>

    <button

      onClick={() => {

        if (props.onConfirm) {

          props.onConfirm();
        }
      }}
    >

      Yes

    </button>

    <button

      onClick={() => {

        if (props.onCancel) {

          props.onCancel();
        }
      }}
    >

      No

    </button>

  </article>
);

}

function Read(props) {
```

```
const { student } = props;

return (
  <article style={{ border: '2px solid black', padding: '10px', marginBottom: '10px' }}>
    <h2>Student Info</h2>
    <p><strong>Name:</strong> {student.name}</p>
    <p><strong>Username:</strong> {student.username}</p>
    <p><strong>Age:</strong> {student.age}</p>
    <p><strong>Height:</strong> {student.height}</p>
    <p><strong>Join Date:</strong> {student.joinDate}</p>
  </article>
);

}
```

```
function Footer(props) {

  return (
    <footer style={{ border: '2px solid black', padding: '10px', marginTop: '10px' }}>
      <ul>
        <li>
          <a
            href="/create"
            onClick={(event) => {
              event.preventDefault();
              props.onChangeMode('CREATE');
            }}
          >
            Create
          </a>
        </li>
      </ul>
    </footer>
  );
}
```

```
</li>

<li>
  <a
    href="/update"
    onClick={(event) => {
      event.preventDefault();
      props.onChangeMode('UPDATE');
    }}
  >
    Update
  </a>
</li>

<li>
  <a
    href="/delete"
    onClick={(event) => {
      event.preventDefault();
      props.onChangeMode('DELETE');
    }}
  >
    Delete
  </a>
</li>
</ul>
</footer>
);
}
```

```
function App() {

  const [mode, setMode] = useState('MENU');

  const [id, setId] = useState(null);

  const [nextId, setNextId] = useState(5);

  const [students, setStudents] = useState([
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ]);

  let contentSelectMenu = <Article title="Select No" body="Welcome to the Student Info App" />;

  switch (mode) {
    case 'MENU':
      contentSelectMenu = <Article title="Select Top Menu" body="Welcome to the Student Info App" />;
      break;
    case 'READ':
      const student = students.find((student) => student.id === parseInt(id));
      if (student) {
        contentSelectMenu = <Read student={student} />;
      }
  }
}
```

```
        break;

    case 'CREATE':
        contentSelectMenu = (
            <Create
                onCreate={({ name, username, age, height, joinDate }) => {
                    const newStudent = { id: nextId, name, username, age, height, joinDate };
                    const newStudents = [...students, newStudent];
                    setStudents(newStudents);
                    setMode('READ');
                    setId(nextId);
                    setNextId(nextId + 1);
                }}
            />
        );
        break;

    case 'UPDATE':
        if (id === null) {
            setMode('ERROR');
        } else {
            const updateStudent = students.find((student) => student.id === parseInt(id));
            if (updateStudent) {
                contentSelectMenu = (
                    <Update
                        name={updateStudent.name}
                        username={updateStudent.username}
                        age={updateStudent.age}
                        height={updateStudent.height}
                );
            }
        }
    }
}
```

```
joinDate={updateStudent.joinDate}

onUpdate={({ name, username, age, height, joinDate }) => {

    const updateStudents = students.map((student) =>

        student.id === parseInt(id) ? { ...student, name, username, age,
height, joinDate } : student

    );

    setStudents(updateStudents);

    setMode('READ');

}};

/>

);

}

}

break;

case 'DELETE':


if (id === null) {

    contentSelectMenu = <Article title="Error" body="Please select a student to
delete." />;

    setMode('MENU');

} else {


    const studentToDelete = students.find((student) => student.id ===
parseInt(id));

    if (studentToDelete) {

        contentSelectMenu = (

<Delete

onConfirm={() => {

    const newStudents = students.filter((student) => student.id !==
parseInt(id));

    setStudents(newStudents);

}};
```

```

        setMode('MENU');

        setId(null);

    }

    onCancel={() => {
        setMode('READ');
    }
}

/>

);

}

break;

case 'ERROR':

    contentSelectMenu = <Article title="Error" body="Please select a student to
update." />;

    break;

default:

    contentSelectMenu = <Article title="default Error" body="Welcome to the Student
Info App" />;

}

```



```

return (
<div style={{ border: '2px solid black', padding: '10px' }}>
<Header onChangeMode={() => setMode('MENU')} />
<Nav
    students={students}
    onChangeMode={(id) => {
        setMode('READ');
        setId(id);
    }
}

```

```

        })
```

```

      />
```

```

      {contentSelectMenu}
```

```

<Footer onChangeMode={(mode) => {

```

```

      if (mode === 'DELETE' && id === null) {

```

```

        setMode('ERROR');

```

```

      } else {

```

```

        setMode(mode);

```

```

      }

```

```

    }} />

```

```

</div>

```

```

);

```

```

}

```

```

export default App;

```

자바스크립트 기본 예제

```

import React from 'react'; // 'react' 모듈에서 React를 가져옴
import { useState } from 'react'; // 'react' 모듈에서 useState를 가져옴

//import React from 'react';
export default function MyComponent() {
  return <div>Hello</div>;
}

```

```
//import { useState } from 'react';
export function add(a, b) {
  return a + b;
}
```

import ...: 모듈 전체를 가져오거나, 부수 효과를 위해 파일을 가져올 때 사용

변수 사용법

javaScript 변수 선언 키워드

1. **var** (ES5)

- 함수 범위(**Function Scope**)
- 재선언 가능
- 호이스팅 발생 (초기 값 **undefined**)
- 최신 JavaScript에서는 잘 사용되지 않음

```
console.log(a); // undefined (호이스팅 발생)
```

```
var a = 10;
```

```
console.log(a); // 10

var b = 20;

var b = 30; // 재선언 가능

console.log(b); // 30
```

2. `let` (ES6)

- 블록 범위(**Block Scope**)
- 재선언 불가능
- **TDZ(Temporal Dead Zone)**로 선언 전에 접근 불가

`javascript`

복사편집

```
console.log(x); // ReferenceError (TDZ로 인해 접근 불가)

let x = 10;

console.log(x); // 10

let y = 20;

let y = 30; // SyntaxError: Identifier 'y' has already been declared
```

3. `const` (ES6)

- 블록 범위(**Block Scope**)
- 선언과 동시에 초기화 필수

- 재할당 불가능

javascript

복사편집

```
const z = 100;  
  
z = 200; // TypeError: Assignment to constant variable.
```

```
const pi; // SyntaxError: Missing initializer in const declaration
```

📌 객체와 배열 내부 값 변경 가능

javascript

복사편집

```
const person = { name: "Alice" };  
  
person.name = "Bob"; // 가능  
  
console.log(person); // { name: "Bob" }
```

JavaScript의 블록(Block) 단위와 함수 단위 설명

1. 블록(Block) 단위 {}

블록(Block)은 중괄호 {}로 감싸진 코드 영역이며, 제어문(if, for 등) 또는 함수에서 사용됨.

블록 내부에서 선언된 변수의 **범위(Scope)**는 let과 const의 경우 **블록 범위(Block Scope)**를 가짐.

- ◆ 블록 예시 (if, for, while)

javascript

복사편집

```
if (true) {  
  
    let a = 10; // 블록 내부에서만 접근 가능  
  
    console.log(a); // 10  
  
}  
  
console.log(a); // ReferenceError (블록 밖에서 접근 불가)
```

javascript

복사편집

```
for (let i = 0; i < 3; i++) {  
  
    console.log(i); // 0, 1, 2  
  
}  
  
console.log(i); // ReferenceError (블록 밖에서 접근 불가)
```

✓ `var`는 블록을 무시하고 함수 범위를 가짐

javascript

복사편집

```
if (true) {  
  
    var b = 20;  
  
}  
  
console.log(b); // 20 (블록 밖에서도 접근 가능)
```

2. 함수(Function) 단위

함수 블록은 특정 작업을 수행하는 코드의 끝이며, 함수 내부에서 선언된 변수는 함수 내에서만 유효(함수 범위, Function Scope).

javascript

복사편집

```
function example() {  
  let x = 100;  
  
  console.log(x); // 100  
  
}  
  
example();  
  
console.log(x); // ReferenceError (함수 외부에서는 접근 불가)
```

함수 내 변수는 외부에서 접근 불가 (캡슐화 효과)

3. 블록 단위와 함수 단위 차이점

구분

블록 {}

함수 function

| | | |
|----|-----------------------|-----------------|
| 범위 | let, const → 블록 내부에서만 | 내부에서 선언된 변수는 함수 |
| | 접근 가능 | 내부에서만 유효 |

| | | | |
|-----|----------|---------|----------------|
| var | 블록을 무시하고 | 함수 단위에서 | 함수 전체에서만 접근 가능 |
| 사용 | 관리 | | |

예외 `var`로 선언 시 블록 밖에서도 `var`는 함수 전체에서만 유효
접근 가능

4. 결론

- `{}` 블록은 제어문(`if`, `for`, `while` 등)에서 사용되며 `let`, `const`는 블록 범위를 가짐
- `function`은 코드의 독립적인 실행 단위이며 함수 내부에서 선언된 변수는 **함수 범위(Function Scope)**를 가짐
- `var`는 블록을 무시하고 함수 단위에서 관리됨

✓ 정리표

| 키워드 | 범위 | 재선언 | 재할당 | 호이스팅 |
|--------------------|----|-----|-----|------------------------------|
| <code>var</code> | 함수 | ○ | ○ | ○ (<code>undefined</code>) |
| <code>let</code> | 블록 | × | ○ | × |
| <code>const</code> | 블록 | × | × | × |

📌 JavaScript 함수 기본 문법과 매개변수 예제

1. 함수 기본 문법

`javascript`

복사편집

```
function 함수이름(매개변수1, 매개변수2, ...) {  
    return 결과값; // 선택 사항  
}
```

2. 함수의 다양한 매개변수 사용 예제

✓ 1) 매개변수가 없는 함수

- 매개변수를 받지 않고, 내부에서 값을 직접 처리

javascript

복사편집

```
function sayHello() {  
    return "Hello, world!";  
}  
  
console.log(sayHello()); // "Hello, world!"
```

✓ 2) 기본적인 매개변수 사용

javascript

복사편집

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(3, 7)); // 10
```

3) 기본값이 있는 매개변수 (ES6+)

- 매개변수에 기본값을 설정하면 인자가 전달되지 않았을 때 사용됨

javascript

복사편집

```
function greet(name = "Guest") {  
  
    return `Hello, ${name}!`;  
  
}  
  
  
  
console.log(greet());           // "Hello, Guest!"  
  
console.log(greet("Alice"));   // "Hello, Alice!"
```

4) 나머지 매개변수(Rest Parameter, ES6+)

- 여러 개의 인자를 배열 형태로 받음

javascript

복사편집

```
function sum(...numbers) {  
  
    return numbers.reduce((acc, num) => acc + num, 0);  
  
}  
  
  
  
console.log(sum(1, 2, 3, 4, 5)); // 15  
  
console.log(sum(10, 20));      // 30
```

✓ 5) 매개변수에 함수(콜백 함수) 전달

- 콜백 함수는 다른 함수의 매개변수로 전달됨
- 비동기 처리나 이벤트 처리에 자주 사용

javascript

복사편집

```
function process(num, callback) {  
  const result = num * 2;  
  callback(result);  
}  
  
process(5, function(result) {  
  console.log("결과:", result); // 결과: 10  
});
```

✓ 6) 매개변수 구조 분해 할당 (Destructuring, ES6+)

- 객체나 배열에서 필요한 값만 추출하여 사용

javascript

복사편집

```
function introduce({ name, age }) {  
  return `이름: ${name}, 나이: ${age}`;  
}
```

```
const person = { name: "Alice", age: 25, city: "Seoul" };
```

```
console.log(introduce(person)); // "이름: Alice, 나이: 25"
```

📌 정리

방식

설명

예제

매개변수 없음

인자 없이 실행되는 함수

```
function sayHello() { return  
    "Hello!"; }
```

기본 매개변수

일반적인 매개변수 사용

```
function add(a, b) { return a  
+ b; }
```

기본값 설정

기본값 지정 (ES6)

```
function greet(name =  
    "Guest") {}
```

나머지 매개변수

여러 개의 값 받기

```
function sum(...numbers) {}
```

콜백 함수

함수 전달 가능

```
function process(num,  
callback) {}
```

구조 분해 할당

객체에서 값 추출

```
function introduce({ name,  
age }) {}
```

🚀 최신 **JavaScript**에서는 기본값, 나머지 매개변수, 구조 분해 할당을 적극 활용하는 것이 좋음!

화살표 함수

화살표함수는 일반 함수를 간결하게 만들기 위해서 사용한다.

```
function 함수이름(매개변수1, 매개변수2, ...) {  
    // 함수 본문  
    // return 리턴할값  
}  
  
const 함수이름 = (매개변수1, 매개변수2, ...) => {  
    // 함수 본문  
    // return 리턴할값  
};  
  
// 두 예제 비교  
// 일반 함수  
function add(a, b) {  
    return a + b;  
}  
  
// 화살표 함수  
const add = (a, b) => a + b;
```

1. 기본 형태

화살표 함수:

```
const greet = (name) => {
    console.log(`Hello, ${name}!`);
};

greet('Alice'); // 출력: Hello, Alice!
```

일반 함수:

```
function greet(name) {
    console.log(`Hello, ${name}!`);

}

greet('Alice'); // 출력: Hello, Alice!
```

2. 매개변수가 하나일 때

화살표 함수:

```
const square = x => x * x;

console.log(square(5)); // 출력: 25
```

일반 함수:

```
function square(x) {
    return x * x;

}

console.log(square(5)); // 출력: 25
```

3. 매개변수가 없을 때

화살표 함수:

```
const sayHello = () => console.log('Hello, world!');

sayHello(); // 출력: Hello, world!
```

일반 함수:

```
function sayHello() {  
  console.log('Hello, world!');  
}  
  
sayHello(); // 출력: Hello, world!
```

4. 함수 본문이 하나의 표현식일 때

화살표 함수:

```
const add = (a, b) => a + b;  
  
console.log(add(3, 4)); // 출력: 7
```

일반 함수:

```
function add(a, b) {  
  return a + b;  
}  
  
console.log(add(3, 4)); // 출력: 7
```

5. 함수 본문이 여러 줄일 때

화살표 함수:

```
const multiply = (a, b) => {  
  const result = a * b;  
  return result;  
};  
  
console.log(multiply(6, 7)); // 출력: 42
```

일반 함수:

```
function multiply(a, b) {  
  const result = a * b;  
  return result;  
}  
  
console.log(multiply(6, 7)); // 출력: 42
```

1. 배열 선언 방법

배열 리터럴([]) 사용 (가장 일반적)

javascript

복사편집

```
const numbers = [10, 20, 30, 40, 50];  
  
console.log(numbers); // [10, 20, 30, 40, 50]
```

Array 생성자 사용

javascript

복사편집

```
const arr1 = new Array(5); // 길이가 5인 빈 배열 생성  
  
console.log(arr1); // [ <5 empty items> ]
```

```
const arr2 = new Array(1, 2, 3, 4, 5); // 초기값 포함 배열 생성  
  
console.log(arr2); // [1, 2, 3, 4, 5]
```

2. 배열 요소 접근 (index)

인덱스는 **0**부터 시작

배열이름[인덱스]를 사용하여 요소에 접근 가능

javascript

복사편집

```
const fruits = ["Apple", "Banana", "Cherry"];
```

```
console.log(fruits[0]); // "Apple" (첫 번째 요소)
```

```
console.log(fruits[1]); // "Banana"
```

```
console.log(fruits[2]); // "Cherry"
```

3. 배열 길이 확인 (length 속성)

javascript

복사편집

```
const numbers = [1, 2, 3, 4, 5];
```

```
console.log(numbers.length); // 5
```

1. 배열의 값변경

```
const fruits = ["Apple", "Banana", "Cherry"];
```

```
fruits[1]='orange';  
  
console.log(fruits);
```

📌 2. JSON 문법

✓ JSON 데이터 구조

JSON은 키-값 쌍 (**Key-Value Pair**)으로 구성됩니다.

- 객체(**Object**) → {} 중괄호 사용
- 배열(**Array**) → [] 대괄호 사용

✓ JSON 기본 예제

json

코드 복사

```
{  
  "name": "Alice",  
  "age": 25,  
  "isStudent": false,  
  "skills": [ "JavaScript", "Python", "C++" ],  
  "address": {  
    "city": "Seoul",  
    "zipCode": "12345"  
  }  
}
```

- ✓ 문자열은 항상 "(큰따옴표)로 감싸야 함
- ✓ 숫자, 불리언, 배열, 객체 등의 데이터 타입 지원

📌 3. JSON 데이터 타입

JSON에서 사용할 수 있는 데이터 타입은 다음과 같습니다.

데이터 타입

예제

```
문자열 (String) "name": "Alice"  
숫자 (Number) "age": 25  
불리언 "isStudent": false  
(Boolean)  
배열 (Array) "skills": ["JavaScript",  
"Python"]  
객체 (Object) "address": { "city":  
"Seoul" }  
null 값 "nickname": null
```

📌 4. JSON과 JavaScript 객체 변환

✓ JavaScript 객체 → JSON 변환 (JSON.stringify())

javascript

코드 복사

```
let person = {  
    name: "Alice",  
    age: 25,  
    isStudent: false  
};
```

```
let jsonString = JSON.stringify(person);  
console.log(jsonString);  
// '{"name":"Alice", "age":25, "isStudent":false}' (문자열 형태)
```

✓ JSON.stringify() 는 객체를 JSON 문자열로 변환

✓ JSON → JavaScript 객체 변환 (JSON.parse())

javascript

코드 복사

```
let jsonData = '{"name":"Alice", "age":25, "isStudent":false}';
```

```
let obj = JSON.parse(jsonData);  
console.log(obj.name); // Alice
```

```
console.log(obj.age); // 25
```

- ✓ `JSON.parse()` 는 JSON 문자열을 JavaScript 객체로 변환

📌 구조 분해 할당(Destructuring Assignment)

구조 분해 할당은 배열이나 객체의 값을 개별 변수로 쉽게 추출할 수 있는 문법입니다.
배열과 객체에서 각각 다른 방식으로 사용됩니다.

1. 배열 구조 분해 할당

배열의 요소를 순서대로 변수에 할당합니다.

- ✓ 기본 사용법

`javascript`

복사편집

```
const fruits = ["Apple", "Banana", "Cherry"];
```

```
// 배열에서 순서대로 값을 꺼내 변수에 저장
```

```
const [first, second, third] = fruits;
```

```
console.log(first); // "Apple"
```

```
console.log(second); // "Banana"
```

```
console.log(third); // "Cherry"
```

- ✓ 일부 요소만 추출

`javascript`

복사편집

```
const numbers = [10, 20, 30];

// 첫 번째 값만 가져오고, 나머지는 무시

const [first] = numbers;

console.log(first); // 10
```

✓ 값 건너뛰기

javascript

복사편집

```
const colors = ["Red", "Green", "Blue", "Yellow"];

// 두 번째 값만 가져오기 (첫 번째 건너뛰기)

const [, secondColor] = colors;

console.log(secondColor); // "Green"
```

✓ 나머지 요소 추출 (rest 연산자)

javascript

복사편집

```
const animals = ["Dog", "Cat", "Rabbit", "Elephant"];
```

// 첫 번째 요소는 따로 변수에 저장하고, 나머지는 배열로 받기

```
const [firstAnimal, ...restAnimals] = animals;

console.log(firstAnimal); // "Dog"

console.log(restAnimals); // ["Cat", "Rabbit", "Elephant"]
```

2. 객체 구조 분해 할당

객체에서 원하는 속성을 이름을 기준으로 추출합니다.

기본 사용법

javascript

복사편집

```
const person = { name: "Alice", age: 25, city: "Seoul" };
```

// 객체에서 변수로 꺼내기

```
const { name, age, city } = person;
```

```
console.log(name); // "Alice"
```

```
console.log(age); // 25
```

```
console.log(city); // "Seoul"
```

변수 이름 변경 (: 사용)

javascript

복사편집

```
const user = { id: 1, username: "JohnDoe" };
```

```
// 변수명을 다른 이름으로 변경하여 할당
```

```
const { username: userName } = user;
```

```
console.log(userName); // "JohnDoe"
```

✓ 기본값 설정 (`undefined` 방지)

javascript

복사편집

```
const book = { title: "JavaScript Guide", author: "MDN" };
```

```
// 존재하지 않는 속성에는 기본값 할당
```

```
const { title, author, year = 2024 } = book;
```

```
console.log(year); // 2024 (기본값 사용)
```

✓ 나머지 속성 추출 (`rest` 연산자)

javascript

복사편집

```
const car = { brand: "Tesla", model: "Model 3", year: 2023 };
```

```
// brand만 가져오고, 나머지는 객체로 받기  
const { brand, ...rest } = car;  
  
  
  
console.log(brand); // "Tesla"  
console.log(rest); // { model: "Model 3", year: 2023 }
```

3. 함수에서 구조 분해 할당

구조 분해 할당을 함수 매개변수에서 사용할 수도 있습니다.

✓ 배열 매개변수

javascript

복사편집

```
function printColors([first, second]) {  
  console.log(first, second);  
}
```

```
printColors(["Red", "Green", "Blue"]); // "Red Green"
```

✓ 객체 매개변수

javascript

복사편집

```
function printUser({ name, age }) {  
  console.log(`이름: ${name}, 나이: ${age}`);  
}  
  
  
  
  
  
const user = { name: "Alice", age: 30 };  
  
printUser(user); // "이름: Alice, 나이: 30"
```

🚀 핵심 정리

- ✓ 배열 구조 분해 → 순서대로 값 할당
- ✓ 객체 구조 분해 → 이름으로 값 추출
- ✓ 기본값 설정 가능
- ✓ 함수 매개변수에도 사용 가능

📌 구조 분해 할당은 코드를 더 간결하고 가독성 좋게 만들어줍니다! 🚀

스프레드 연산자 (...)란?

스프레드 연산자 (...)는 배열이나 객체의 요소를 개별적으로 펼치는(**spread**) 역할을 합니다.
즉, 기존 값을 복사하여 새로운 배열이나 객체를 쉽게 만들 수 있습니다.

1. 배열에서 ... 스프레드 연산자 사용

배열의 요소를 개별적으로 펼쳐서 새로운 배열을 만들 수 있습니다.

✓ 배열 복사

javascript

복사편집

```
const students = ["Alice", "Bob", "Charlie"];
```

```
const newStudents = [...students];

console.log(newStudents); // ["Alice", "Bob", "Charlie"]
```

📌 설명

- ...students → ["Alice", "Bob", "Charlie"]가 펼쳐짐
 - newStudents는 students의 복사본이므로 원본을 변경해도 영향을 받지 않음
-

✓ 배열에 새로운 요소 추가

javascript

복사편집

```
const students = ["Alice", "Bob"];
const newStudent = "Charlie";

// 기존 배열에 새로운 요소 추가
const newStudents = [...students, newStudent];

console.log(newStudents); // ["Alice", "Bob", "Charlie"]
```

📌 설명

- ...students → 기존 배열의 요소("Alice", "Bob")를 펼침
 - newStudent("Charlie")를 추가하여 새로운 배열을 생성
 - 원본 배열(students)은 변경되지 않음
-

✓ 배열 합치기

javascript

복사편집

```
const groupA = ["Alice", "Bob"];
const groupB = ["Charlie", "David"];

const allStudents = [...groupA, ...groupB];

console.log(allStudents); // ["Alice", "Bob", "Charlie", "David"]
```

📌 설명

- ...groupA와 ...groupB를 펼쳐서 하나의 배열로 합침
-

2. 객체에서 ... 스프레드 연산자 사용

객체의 속성을 복사하거나 새로운 속성을 추가할 때 사용합니다.

✓ 객체 복사

javascript

복사편집

```
const student = { name: "Alice", age: 20 };
const copiedStudent = { ...student };

console.log(copiedStudent); // { name: "Alice", age: 20 }
```

📌 설명

- ...student를 사용하여 새로운 객체를 생성
 - 원본(student)을 변경해도 copiedStudent에는 영향 없음
-

✓ 객체 속성 추가 및 업데이트

javascript

복사편집

```
const student = { name: "Alice", age: 20 };

// 기존 객체 복사 + 새로운 속성 추가
const updatedStudent = { ...student, grade: "A" };

console.log(updatedStudent); // { name: "Alice", age: 20, grade: "A" }
```

📌 설명

- 기존 객체(student)의 속성을 유지하면서 grade 속성을 추가

javascript

복사편집

```
const student = { name: "Alice", age: 20 };

// 기존 속성 덮어쓰기
const updatedStudent = { ...student, age: 21 };

console.log(updatedStudent); // { name: "Alice", age: 21 }
```

📌 설명

- `age: 21`이 `student`의 기존 `age: 20`을 덮어씀
-

핵심 정리

- ✓ 배열에서 `...` → 배열 요소를 펼쳐서 복사, 추가, 합치기 가능
 - ✓ 객체에서 `...` → 객체의 복사, 새로운 속성 추가 및 업데이트 가능
 - ✓ 불변성 유지 → 원본 데이터를 직접 수정하지 않고 새로운 데이터 생성 가능
- ✓ ... 스프레드 연산자는 코드의 가독성과 유지보수성을 높이는 중요한 기능! 

> 14. 다양한 흑 사용법

훅(Hook)은 React 16.8부터 도입된 기능으로, 함수형 컴포넌트에서 상태(state)와 라이프사이클(lifecycle) 기능을 사용할 수 있도록 해줍니다. 흑을 사용하면 클래스 컴포넌트 없이도 상태 관리와 다양한 기능을 구현할 수 있습니다. 기본적으로 제공되는 흑과 사용자 정의 흑(Custom Hook)을 사용할 수 있습니다.

기본 흑

1. `useState`

- 설명: 함수형 컴포넌트에서 상태를 관리할 수 있도록 해주는 툴입니다.
- 변수에 값이 변경되면 화면을 다시 그린다.

사용법:

Welcome to the Counter App

Count: 4

Increment

```
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0); // 상태 변수와 업데이트 함수 선언
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

const App = () => {
  return (
    <div style={{ textAlign: 'center', padding: '20px' }}>
      <h1>Welcome to the Counter App</h1>
      <Counter />
    </div>
  );
};

export default App;
```

2. useEffect

`useEffect`는 **React** 컴포넌트에서 부작용(**side effects**)을 처리하는 **Hook**이에요.

👉 부작용이란? API 호출, 타이머, 이벤트 리스너 등 렌더링과 직접 관련 없는 작업

렌더링이란 화면을 다시 그려주는 작업을 의미한다.

① `useEffect` 기본 문법

```
jsx  
복사편집  
useEffect(() => {  
  // 실행할 코드  
  return () => {  
    // 정리(clean-up) 코드 (선택 사항)  
  };  
}, [의존성]);
```

- 의존성에 따라 실행되는 타이밍이 달라짐!
-

② 실행 방식 정리

| <code>useEffect</code> 형태 | 실행 시점 | 사용 예시 |
|--|----------------|------------------------------|
| <code>useEffect(() =>
{...})</code> | 매 렌더링마다 실행 | <code>console.log()</code> 등 |
| <code>useEffect(() =>
{...}, [])</code> | 처음 한 번만 실행 | API 호출, 이벤트 등록 |
| <code>useEffect(() =>
{...}, [변수])</code> | 특정 값이 변경될 때 실행 | <code>count</code> 가 변할 때 실행 |

👉 예제 코드

```
useEffect(() => {
  console.log('항상 실행됨');
});

useEffect(() => {
  console.log('처음 한 번만 실행됨');
}, []);

useEffect(() => {
  console.log('count가 변경될 때 실행됨');
}, [count]);
```

③ useEffect의 클린업 (정리 함수)

타이머, 이벤트 리스너 같은 것들은 꼭 정리해줘야 함!

```
jsx
복사편집
useEffect(() => {
  const interval = setInterval(() => {
    console.log('1초마다 실행됨');
  }, 1000);

  return () => {
    clearInterval(interval); // 언마운트 시 정리
    console.log('타이머 제거됨');
  };
}, []);
```

④ 실제 사용 예제 (API 호출)

```
jsx
복사편집
useEffect(() => {
  fetch('https://jsonplaceholder.typicode.com/posts')
```

```
.then(res => res.json())
.then(data => console.log(data));
}, []); // 처음 한 번만 실행
```

⌚ 핵심 요약

- 렌더링마다 실행 → `useEffect(() => {...})`
- 처음 한 번만 실행 → `useEffect(() => {...}, [])`
- 특정 값이 바뀔 때 실행 → `useEffect(() => {...}, [변수])`
- 타이머, 이벤트 리스너는 정리 필요 → `return () => {...}` 사용

`useState`는 변수에 값이 변경되면 화면을 다시그리기 위해서 사용하고 `useEffect`는 변수의 값이 변경되면 특정 로직을 실행하기 위해서 사용한다.

```
import React, { useState, useEffect } from 'react';

// SimpleEffectComponent 컴포넌트 정의
const SimpleEffectComponent = () => {
  useEffect(() => {
    console.log('컴포넌트가 화면에 나타났습니다!');
  });

  // 클린업 함수
  return () => {
    console.log('컴포넌트가 화면에서 사라졌습니다.');
  };
}, []); // 빈 배열로 인하여 처음 렌더링될 때만 이펙트 실행
// 내부적으로 생성해 놓는 작업이 있어서 생각과 다르게 실행된다.
```

```
return (
  <div>
    <h1>Simple Effect Component</h1>
    <p>콘솔에서 메시지를 확인해 보세요.</p>
  </div>
);
};

// App 컴포넌트 정의
const App = () => {
  const [show, setShow] = useState(true);

  return (
    <div>
      <button onClick={() => setShow(!show)}>
        {show ? 'Hide' : 'Show'} Simple Effect Component
      </button>
      {show && <SimpleEffectComponent />}
    </div>
  );
};

export default App;
```

title: Hello World! (11)

Button clicked: 11 times

Change Title

Title이 변경될때마다 실행되도록 구현한 useEffect예제 입니다.

```
import React, { useState, useEffect } from 'react';

const TitleChanger = () => {
  const [title, setTitle] = useState('Hello World!');
  const [clickCount, setClickCount] = useState(0);

  // 이 함수는 컴포넌트가 처음 렌더링될 때만 실행됩니다.
  useEffect(() => {
    document.title = title; // 브라우저의 탭 제목을 설정합니다.
    console.log('컴포넌트가 렌더링되었습니다!');
    // 클린업 함수 (선택적)
  });
}
```

```

        console.log('컴포넌트가 제거됩니다.');
    };
}, [title]); // title이 변경될 때마다 이 effect가 실행됩니다.

const handleClick = () => {
    setClickCount(clickCount + 1);
    setTitle(`Hello World! ${clickCount + 1}`); // 제목을 클릭 수에 따라
변경
};

return (
<div>
    <p>title: {title} </p>
    <p>Button clicked: {clickCount} times</p>
    <button onClick={handleClick}>Change Title</button>
</div>
);
};

export default TitleChanger;

```

App.js로 만들기

```

import React, { useState, useEffect } from 'react';

// TitleChanger 컴포넌트
const TitleChanger = () => {
    const [title, setTitle] = useState('Hello World!');
    const [clickCount, setClickCount] = useState(0);

```

```
useEffect(() => {
  document.title = title; // 브라우저의 탭 제목을 변경합니다.
  console.log('컴포넌트가 렌더링되었습니다!');
  return () => {
    console.log('컴포넌트가 제거됩니다.');
  };
}, [title]);

const handleClick = () => {
  setClickCount(clickCount + 1);
  setTitle(`Hello World! ${clickCount + 1}`); // 클릭 수에 따라 제목 변경
};

return (
  <div>
    <p>title: {title}</p>
    <p>Button clicked: {clickCount} times</p>
    <button onClick={handleClick}>Change Title</button>
  </div>
);
};

// App 컴포넌트
const App = () => {
  return (
    <div>
      <h1>Welcome to Title Changer App!</h1>
      <TitleChanger /> /* TitleChanger 컴포넌트 포함 */
    </div>
  );
};

export default App;
```

json 요청하기

<https://jsonplaceholder.typicode.com/users> 에 들어가 보면 test json 데이터를 확인할 수 있다.

```
import React, { useState, useEffect } from 'react';

// DataFetcher 컴포넌트 정의
const DataFetcher = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // 데이터 fetch
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setData(data))
      .catch(error => console.error('Error fetching data:', error));
  }, []); // 빈 배열을 의존성 배열로 전달하면, 컴포넌트가 처음 렌더링될 때만
        // 실행됩니다.

  return (
    <div>
      {data ? (
        <div>
          {data.map(user => (
            <p key={user.id}>{user.name}</p>
          )))
        </div>
      ) : (
        <div>
          ...
        </div>
      )}
    </div>
  );
}
```

```

        <p>Loading...</p>
    )
</div>
);
};

// App 컴포넌트 정의
const App = () => {
  return (
    <div style={{ padding: '20px' }}>
      <h1>User List</h1>
      <DataFetcher />
    </div>
  );
};

export default App;

```

설명

1. **data** 배열의 사용:

- **data**는 여러 사용자 객체로 이루어진 배열입니다. 따라서 `data.name`이 아닌 `data.map(user => user.name)`을 사용해야 합니다.

2. **map** 메소드:

- `data.map(user => ...)`을 사용하여 배열의 각 사용자 객체에 대해 `<p>` 요소를 생성합니다.
- `key={user.id}`를 추가하여 각 `<p>` 요소에 고유한 키를 부여합니다. 이는 React가 각 요소를 효율적으로 추적하는 데 필요합니다.

3. 예러 처리:

- `.catch(error => console.error('Error fetching data:', error))`를 추가하여 데이터를 가져오는 동안 발생할 수 있는 오류를 콘솔에 출력합니다.

이 수정된 코드로 `data` 배열의 각 사용자 이름을 웹 페이지에 렌더링할 수 있으며, 데이터가 로드되는 동안 "Loading..." 메시지를 표시합니다.

fetch관련 자바 스크립트 예제

`fetch`는 웹 브라우저에서 HTTP 요청을 보내고 응답을 받기 위한 API입니다. `fetch`는 JavaScript에서 비동기적으로 데이터를 가져오는 데 사용되며, 주로 API 호출, 데이터 요청, 서버와의 상호작용 등에 사용됩니다.

기본 문법

```
fetch(url, options)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // 또는 response.text() 등
  })
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('There has been a problem with your fetch operation:', error);
  });
}
```

주요 개념

1. **URL**: 요청을 보낼 서버의 URL입니다.
2. 옵션 객체: HTTP 메서드(GET, POST, PUT, DELETE 등), 헤더, 본문 데이터 등을 설정할 수 있습니다.
3. 응답 객체: 서버에서 반환된 응답을 나타내는 객체입니다. `response.json()`은 JSON 형태로 데이터를 파싱합니다.

사용 예제

1. GET 요청

서버로부터 데이터를 가져오는 기본적인 GET 요청 예제입니다.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log('Data:', data);
  })
  .catch(error => {
    console.error('Fetch error:', error);
  });
}
```

- 설명: `fetch`를 호출하여 서버에서 데이터를 요청합니다. 응답을 JSON으로 변환하고, 데이터를 콘솔에 출력합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

2. POST 요청

서버에 데이터를 전송하는 POST 요청 예제입니다.

```
fetch('https://api.example.com/data', {
  method: 'POST', // 요청 메서드
```

```

headers: {
  'Content-Type': 'application/json' // 요청 헤더
},
body: JSON.stringify({ key: 'value' }) // 요청 본문
})
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => {
  console.log('Response Data:', data);
})
.catch(error => {
  console.error('Fetch error:', error);
});

```

- 설명: `method`를 'POST'로 설정하고, `body`에 JSON 문자열을 포함시킵니다. 이 데이터를 서버에 전송하고, 응답을 처리합니다.

3. Error Handling

`fetch` 호출 시 발생할 수 있는 오류를 처리하는 예제입니다.

javascript

코드 복사

```

fetch('https://api.example.com/data')
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => {
  console.log('Data:', data);
})

```

```
.catch(error => {
  console.error('Fetch error:', error.message);
});
```

- 설명: `response.ok`를 사용하여 응답 상태가 성공적인지 확인하고, 오류가 발생하면 `catch` 블록에서 처리합니다.

비동기/대기 (`async/await`)

`fetch`와 `async/await`를 함께 사용하여 코드의 가독성을 높일 수 있습니다.

javascript

코드 복사

```
const fetchData = async () => {
  try {
    const response = await fetch('https://api.example.com/data');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    console.log('Data:', data);
  } catch (error) {
    console.error('Fetch error:', error.message);
  }
};

fetchData();
```

- 설명: `fetch`를 `await`하여 응답을 기다리고, JSON으로 변환 후 데이터를 처리합니다. 오류가 발생하면 `catch` 블록에서 처리합니다.

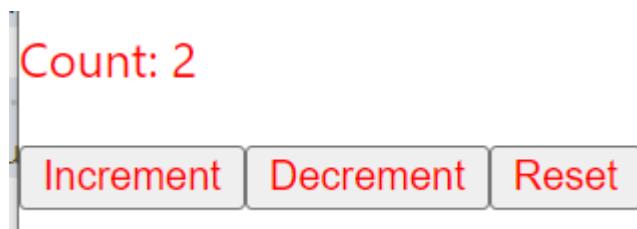
결론

`fetch`는 웹에서 데이터를 가져오거나 서버에 데이터를 보내는 데 유용한 API입니다. 비동기적으로 동작하며, `Promise` 기반의 처리를 지원하여 코드의 가독성을 높입니다. `async/await`와 함께 사용하면 더욱 깔끔한 코드를 작성할 수 있습니다.

useReducer

- 설명: `useState`보다 복잡한 상태 로직을 관리할 때 사용되는 툐입니다. 상태 업데이트 로직을 별도의 `reducer` 함수로 분리할 수 있습니다.

`useReducer`는 상태 관리 로직을 리듀서 함수로 분리하여 복잡한 상태 업데이트를 간단하고 예측 가능하게 처리할 수 있는 React 툐입니다.



다음과 같은 화면은 3개의 메소드로 `count`값을 조작하는 형태로 되어 있다.

이렇게 여러개의 데이터와 여러개의 메소드가 같은 데이터를 조작하는 용도로 사용되면 `useReducer`로 묶어서 표현할 수 있다.

```
import React, { useReducer } from 'react';

// 1. 리듀서 함수 정의
const reducer = (state, action) => {
  switch (action.type) {
```

```

    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    case 'RESET':
      return { count: 0 };
    default:
      return state;
  }
};

// 2. 초기 상태
const initialState = { count: 0 };

const Counter = () => {
  // 3. useReducer를 사용하여 상태와 디스패치 함수를 정의
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      {/* 4. 변경된 데이터를 그때그때 보여줄 state변수 기술*/}
      <p>Count: {state.count}</p>
      {/* 5. 디스패치 함수를 사용하여 액션을 보냄 */}
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>
      <button onClick={() => dispatch({ type: 'RESET' })}>Reset</button>
    </div>
  );
};

export default Counter;

```

`const [state, dispatch] = useReducer(reducer, initialState);`에서 `state`는 현재 상태를 나타내는 변수입니다.

`dispatch`는 상태를 업데이트하기 위해 액션을 보내는 함수입니다

1. `useReducer`에서 사용할 로직들을 구현한다.
2. 값변경시 화면에 변경된 값을 바로바로 표현해줄 변수를 생성한다.
3. 로직과 변수로 `useReducer`메소드를 이용해서 `state`와 `dispatch`를 얻는다.
4. 4. 변경된 데이터를 그때그때 보여줄 `state`변수 기술
5. 디스패치 함수를 사용하여 액션을 보냄

사용자 정의 툐 (Custom Hook)

사용자 정의 툐을 통해 공통된 로직을 재사용할 수 있습니다.

많이 사용하는 기능을 Hook으로 만들어서 필요할 때 사용한다. 다음 예제는 토글 기능을 가지고 있는 `useToggle` Hook를 만들어 보았다.

`useToggle`은 불리언 상태를 간단하게 토글할 수 있는 사용자 정의 툐입니다.

주요 포인트:

- 목적: 불리언 값을 `true`와 `false`로 쉽게 변경하는 기능을 제공합니다.
- 구성: 상태 값과 상태를 반전시키는 함수(토글 함수)를 반환합니다.
- 장점:
 - 코드 간결성: 상태를 간단하게 관리할 수 있음.
 - 재사용성: 여러 컴포넌트에서 반복 사용 가능.
 - 초기 값 설정: 초기 상태를 `true` 또는 `false`로 설정 가능.

App Component

Off

Toggle1

Off

Toggle2

```
import React, { useState } from 'react';
```

```
// 사용자 정의 흙: true/false 값을 토글
```

```
const useToggle = (initialValue = false) => {
```

```
  const [value, setValue] = useState(initialValue);
```

```
  const toggle = () => {
```

```
    setValue(prevValue => !prevValue);
```

```
  };
```

```
  return [value, toggle];
```

```
};
```

```
// Toggle 버튼을 포함한 컴포넌트
```

```
const ToggleComponent = () => {
```

```
  const [isToggled1, toggle1] = useToggle(); // 사용자 정의 흙 사용
```

```
  const [isToggled2, toggle2] = useToggle(); // 사용자 정의 흙 사용
```

```
return (<>
  <div>
    <h1>{isToggled1 ? 'On' : 'Off'}</h1>
    <button onClick={toggle1}>Toggle1</button>
  </div>
  <div>
    <h1>{isToggled2 ? 'On' : 'Off'}</h1>
    <button onClick={toggle2}>Toggle2</button>
  </div>
</>);
};
```

```
// App 컴포넌트
const App = () => {
  return (
    <div>
      <h1>App Component</h1>
      <ToggleComponent /> {/* ToggleComponent을 포함 */}
    </div>
  );
};

export default App;
```

> 15. 단어장 만들기

> 16.

> 17. 쉽게 개선선

```
import React, { useState } from 'react';

import './App.css';

const StudentApp = () => {

  const [mode, setMode] = useState('HOME');

  const [students, setStudents] = useState([
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ]);

  const [selectedId, setSelectedId] = useState(null);

  const [nextId, setNextId] = useState(5);

  const selectedStudent = students.find(s => s.id === selectedId);

  const handleCreate = (student) => {
    setStudents([...students, { ...student, id: nextId }]);
    setNextId(nextId + 1);
    setMode('SELECT');
  };

  const handleUpdate = (updatedStudent) => {
```

```

    setStudents(students.map(s => s.id === selectedId ? { ...s, ...updatedStudent } : s));
    setMode('SELECT');
}

const handleDelete = () => {
  if (selectedId) {
    setStudents(students.filter(s => s.id !== selectedId));
    setSelectedId(null);
    setMode('SELECT');
  }
};

return (
  <div className="app-container">
    <header>
      <h1>Student Info System</h1>
      <div className="menu-buttons">
        <button onClick={() => { setMode('CREATE'); setSelectedId(null); }}>CREATE</button>
        <button onClick={() => { setMode('SELECT'); setSelectedId(null); }}>SELECT</button>
        <button onClick={() => { selectedId ? setMode('UPDATE') : alert('수정할 학생을 선택하세요'); }}>UPDATE</button>
        <button onClick={() => { selectedId ? setMode('DELETE') : alert('삭제할 학생을 선택하세요'); }}>DELETE</button>
      </div>
    </header>
    <p className="mode-indicator">현재 페이지: {mode}</p>
  </div>
)

```

```
</header>

<main>

  {mode === 'HOME' && <p>메뉴를 선택해주세요.</p>}

  {mode === 'CREATE' && <CreateForm onCreate={handleCreate} />}

  {mode === 'SELECT' &&

    <StudentList

      students={students}

      onSelect={id => { setSelectedId(id); setMode('DETAIL'); }}

    />}

  {mode === 'DETAIL' &&

    (selectedStudent ? <StudentDetail student={selectedStudent} /> : <p>학생을
선택하세요.</p>)}

  {mode === 'UPDATE' &&

    (selectedStudent ? <UpdateForm student={selectedStudent}
onUpdate={handleUpdate} /> : <p>학생을 선택하세요.</p>)}

  {mode === 'DELETE' &&

    (selectedStudent ? (
      <div>
        <p>{selectedStudent.name} 학생을 삭제하시겠습니까?</p>
        <button onClick={handleDelete}>삭제</button>
        <button onClick={() => setMode('SELECT')}>취소</button>
      </div>
    ) : <p>학생을 선택하세요.</p>)}

</main>

</div>

);
```

```
};

const CreateForm = ({ onCreate }) => {

  const [form, setForm] = useState({ name: '', username: '', age: '', height: '', joinDate: '' });

  const handleSubmit = (e) => {
    e.preventDefault();
    onCreate(form);
    setForm({ name: '', username: '', age: '', height: '', joinDate: '' });
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>학생 추가</h2>
      <InputForm form={form} setForm={setForm} />
      <button type="submit">등록</button>
    </form>
  );
};

const UpdateForm = ({ student, onUpdate }) => {

  const [form, setForm] = useState({ ...student });

  const handleSubmit = (e) => {
    e.preventDefault();
    onUpdate(form);
  };
}
```

```
};

return (
  <form onSubmit={handleSubmit}>
    <h2>학생 수정</h2>
    <InputForm form={form} setForm={setForm} />
    <button type="submit">수정 완료</button>
  </form>
);
};

const InputForm = ({ form, setForm }) => {
  const handleChange = (e) => {
    const { name, value } = e.target;
    setForm(prev => ({ ...prev, [name]: value }));
  };

  return (
    <>
      <div><input name="name" placeholder="이름" value={form.name} onChange={handleChange} required /></div>
      <div><input name="username" placeholder="아이디" value={form.username} onChange={handleChange} required /></div>
      <div><input name="age" type="number" placeholder="나이" value={form.age} onChange={handleChange} required /></div>
      <div><input name="height" type="number" placeholder="키(cm)" value={form.height} onChange={handleChange} required /></div>
    </>
  );
};
```

```
        <div><input name="joinDate" type="date" placeholder="가입일"
value={form.joinDate} onChange={handleChange} required /></div>
    </>
);
};

const StudentList = ({ students, onSelect }) => (
<div>
    <h2>학생 목록</h2>
    <ul>
        {students.map(student => (
            <li key={student.id}>
                <button onClick={() => onSelect(student.id)}>
                    {student.id}. {student.name}
                </button>
            </li>
        ))}
    </ul>
</div>
);


```

```
const StudentDetail = ({ student }) => (
<div>
    <h2>학생 상세보기</h2>
    <p><strong>이름:</strong> {student.name}</p>
    <p><strong>아이디:</strong> {student.username}</p>
    <p><strong>나이:</strong> {student.age}</p>
```

```
<p><strong>키:</strong> {student.height}cm</p>
<p><strong>가입일:</strong> {student.joinDate}</p>
</div>
);

export default StudentApp;
```

> 18. 마지막 학생생

가장 간단한 조건부 렌더링 예제

html

Copy

Download

Run

```
<button onclick="showText()">눌러보세요</button>
```

```
<p id="message"></p>
```

```
<script>
```

```
let visible = false;
```

```
function showText() {
```

```
    visible = !visible; // true ↔ false 전환
```

```
// 조건부 렌더링
```

```
    document.getElementById("message").innerHTML =
```

```
        visible && "안녕하세요!" ;
```

```
}
```

```
</script>
```

동작 설명:

- 처음에는 빈 화면 (`visible = false`)

- 버튼 클릭시:

- `visible`이 `true`로 바뀜 → "안녕하세요!" 표시

- 다시 클릭하면 `visible`이 `false`로 바뀜 → 텍스트 사라짐

핵심 원리:

- 조건 `&&` 내용 → 조건이 `true`일 때만 내용이 실행됨

- `false && "안녕하세요!"` → 아무것도 표시 안됨

- `true && "안녕하세요!"` → "안녕하세요!" 표시


```
function App() {  
  const isShow = true; // 조건 변수 (true/false 변경해보세요)  
  
  return (  
    <div>  
      {isShow && <p>이 문구는 조건이 true일 때만 보입니다!</p>}  
    </div>  
  );  
}  
  
export default App;
```

```
import React, { useState } from 'react';  
import './App.css';  
  
const StudentApp = () => {  
  const [mode, setMode] = useState('HOME');  
  const [students, setStudents] = useState([  
    { id: 1, name: 'Alice', username: 'alice123', age: 21, height: 160, joinDate: '2020-01-01' },  
    { id: 2, name: 'Bob', username: 'bob123', age: 22, height: 170, joinDate: '2019-03-15' },  
    { id: 3, name: 'Charlie', username: 'charlie123', age: 23, height: 180, joinDate: '2018-05-10' },  
  ]);
```

```
    { id: 4, name: 'Dave', username: 'dave123', age: 24, height: 175, joinDate: '2017-07-20' },
  ]);

const [selectedId, setSelectedId] = useState(null);

const [nextId, setNextId] = useState(5);

const selectedStudent = students.find(s => s.id === selectedId);

const handleCreate = (student) => {
  setStudents([...students, { ...student, id: nextId }]);
  setNextId(nextId + 1);
  setMode('SELECT');
};

const handleUpdate = (updatedStudent) => {
  setStudents(students.map(s => s.id === selectedId ? { ...s, ...updatedStudent } : s));
  setMode('SELECT');
};

const handleDelete = () => {
  if (selectedId) {
    setStudents(students.filter(s => s.id !== selectedId));
    setSelectedId(null);
    setMode('SELECT');
  }
};
```

```
return (

  <div className="app-container">

    <header>

      <h1>Student Info System</h1>

      <div className="menu-buttons">

        <button onClick={() => { setMode('CREATE'); setSelectedId(null); }}>CREATE</button>

        <button onClick={() => { setMode('SELECT'); setSelectedId(null); }}>SELECT</button>

        <button onClick={() => { selectedId ? setMode('UPDATE') : alert('수정할 학생을 선택하세요'); }}>UPDATE</button>

        <button onClick={() => { selectedId ? setMode('DELETE') : alert('삭제할 학생을 선택하세요'); }}>DELETE</button>

      </div>

      <p className="mode-indicator">현재 페이지: {mode}</p>

    </header>

    <main>

      {mode === 'HOME' && <p>메뉴를 선택해주세요.</p>}

      {mode === 'CREATE' && <CreateForm onCreate={handleCreate} />}

      {mode === 'SELECT' && (
        <>
          <StudentList
            students={students}
            onSelect={id => setSelectedId(id)}>

```

```
        />

        {selectedStudent ? (
            <StudentDetail student={selectedStudent} />
        ) : (
            <p>학생을 선택하세요.</p>
        )}
    </>
)}
```



```
{mode === 'UPDATE' &&
(selectedStudent ? <UpdateForm student={selectedStudent}
onUpdate={handleUpdate} /> : <p>학생을 선택하세요.</p>)}
```

```
}
```



```
{mode === 'DELETE' &&
(selectedStudent ? (
<div>
    <p>{selectedStudent.name} 학생을 삭제하시겠습니까?</p>
    <button onClick={handleDelete}>삭제</button>
    <button onClick={() => setMode('SELECT')}>취소</button>
</div>
) : <p>학생을 선택하세요.</p>)}
}
```

```
</main>
```

```
</div>
```

```
);
```

```
};
```

```
const CreateForm = ({ onCreate }) => {
  const [form, setForm] = useState({ name: '', username: '', age: '', height: '', joinDate: '' });

  const handleSubmit = (e) => {
    e.preventDefault();
    onCreate(form);
    setForm({ name: '', username: '', age: '', height: '', joinDate: '' });
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>학생 추가</h2>
      <InputForm form={form} setForm={setForm} />
      <button type="submit">등록</button>
    </form>
  );
};

const UpdateForm = ({ student, onUpdate }) => {
  const [form, setForm] = useState({ ...student });

  const handleSubmit = (e) => {
    e.preventDefault();
    onUpdate(form);
  };
}
```

```

return (
  <form onSubmit={handleSubmit}>
    <h2>학생 수정</h2>
    <InputForm form={form} setForm={setForm} />
    <button type="submit">수정 완료</button>
  </form>
);

};

const InputForm = ({ form, setForm }) => {
  const handleChange = (e) => {
    const { name, value } = e.target;
    setForm(prev => ({ ...prev, [name]: value }));
  };

  return (
    <>
      <div><input name="name" placeholder="이름" value={form.name} onChange={handleChange} required /></div>
      <div><input name="username" placeholder="아이디" value={form.username} onChange={handleChange} required /></div>
      <div><input name="age" type="number" placeholder="나이" value={form.age} onChange={handleChange} required /></div>
      <div><input name="height" type="number" placeholder="키(cm)" value={form.height} onChange={handleChange} required /></div>
      <div><input name="joinDate" type="date" placeholder="가입일" value={form.joinDate} onChange={handleChange} required /></div>
    </>
  );
}

```

```
</>

);

};

const StudentList = ({ students, onSelect }) => (
  <div className="student-list">
    <h2>학생 목록</h2>
    <table>
      <thead>
        <tr>
          <th>ID</th>
          <th>이름</th>
          <th>아이디</th>
          <th>나이</th>
          <th>성별</th>
          <th>가입일</th>
          <th>선택</th>
        </tr>
      </thead>
      <tbody>
        {students.map(student => (
          <tr key={student.id}>
            <td>{student.id}</td>
            <td>{student.name}</td>
            <td>{student.username}</td>
            <td>{student.age}</td>
```

```
<td>{student.height}cm</td>

<td>{student.joinDate}</td>

<td>
    <button onClick={() => onSelect(student.id)}>선택</button>
</td>
</tr>
))}

</tbody>
</table>
</div>
);
```

```
const StudentDetail = ({ student }) => (
    <div className="student-detail">
        <h2>학생 상세보기</h2>
        <p><strong>이름:</strong> {student.name}</p>
        <p><strong>아이디:</strong> {student.username}</p>
        <p><strong>나이:</strong> {student.age}</p>
        <p><strong>키:</strong> {student.height}cm</p>
        <p><strong>가입일:</strong> {student.joinDate}</p>
    </div>
);
```

```
export default StudentApp;
```

```
.App {
```

```
    text-align: center;  
}  
  
.App-logo {  
    height: 40vmin;  
    pointer-events: none;  
}  
  
@media (prefers-reduced-motion: no-preference) {  
    .App-logo {  
        animation: App-logo-spin infinite 20s linear;  
    }  
}  
  
.App-header {  
    background-color: #282c34;  
    min-height: 100vh;  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    justify-content: center;  
    font-size: calc(10px + 2vmin);  
    color: white;  
}  
  
.App-link {
```

```
color: #61dafb;

}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

body {
  font-family: 'Noto Sans KR', sans-serif;
  background: #f5f7fa;
  margin: 0;
  padding: 0;
}

.container {
  width: 90%;
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

.header {
```

```
    text-align: center;  
  
    margin-bottom: 20px;  
  
}  
  
  
.mode-navigation {  
  
    display: flex;  
  
    justify-content: center;  
  
    gap: 10px;  
  
    margin-bottom: 20px;  
  
}  
  
  
.mode-navigation button {  
  
    padding: 10px 15px;  
  
    font-size: 14px;  
  
    background-color: #0066cc;  
  
    color: white;  
  
    border: none;  
  
    border-radius: 6px;  
  
    cursor: pointer;  
  
}  
  
  
.mode-navigation button:hover {  
  
    background-color: #005bb5;  
  
}  
  
  
.current-mode {
```

```
    text-align: center;
    margin-bottom: 20px;
}

.error {
    color: red;
    font-weight: bold;
}

.form, .list, .detail, .delete-confirm {
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}

.form input {
    display: block;
    width: 100%;
    margin-bottom: 15px;
    padding: 8px;
    font-size: 14px;
}

.form button {
    width: 100%;
```

```
padding: 10px;  
background-color: #28a745;  
border: none;  
color: white;  
border-radius: 6px;  
font-size: 16px;  
cursor: pointer;  
}
```

```
.form button:hover {  
background-color: #218838;  
}
```

```
.list-item {  
margin-bottom: 10px;  
}
```

```
.list-item button {  
width: 100%;  
padding: 10px;  
background-color: #f1f1f1;  
border: 1px solid #ccc;  
border-radius: 6px;  
cursor: pointer;  
}
```

```
.list-item.selected button {  
  background-color: #007bff;  
  color: white;  
}  
  
.danger {  
  background-color: #dc3545;  
  margin-right: 10px;  
}  
  
.danger:hover {  
  background-color: #c82333;  
}  
/* App.css */  
  
body {  
  margin: 0;  
  padding: 0;  
  font-family: 'Noto Sans KR', sans-serif;  
  background-color: #f8f9fa;  
}  
  
.app-container {  
  max-width: 900px;  
  margin: 40px auto;  
  padding: 20px;
```

```
background: white;  
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
border-radius: 12px;  
}
```

```
header {  
    text-align: center;  
    margin-bottom: 30px;  
}
```

```
header h1 {  
    margin-bottom: 10px;  
    font-size: 32px;  
    color: #343a40;  
}
```

```
.menu-buttons {  
    margin-bottom: 15px;  
}
```

```
.menu-buttons button {  
    margin: 0 8px;  
    padding: 10px 20px;  
    background-color: #6c5ce7;  
    color: white;  
    border: none;
```

```
border-radius: 8px;  
font-size: 16px;  
cursor: pointer;  
transition: background-color 0.3s;  
}
```

```
.menu-buttons button:hover {  
background-color: #5a4fcf;  
}
```

```
.mode-indicator {  
margin-top: 10px;  
font-size: 18px;  
color: #555;  
}
```

```
main {  
margin-top: 20px;  
}
```

```
form {  
display: flex;  
flex-direction: column;  
gap: 12px;  
}
```

```
form input {  
    padding: 10px;  
    font-size: 16px;  
    border: 1px solid #ccc;  
    border-radius: 8px;  
}  
  
form button {  
    padding: 10px;  
    background-color: #00b894;  
    color: white;  
    font-size: 16px;  
    border: none;  
    border-radius: 8px;  
    cursor: pointer;  
    transition: background-color 0.3s;  
}  
  
form button:hover {  
    background-color: #019875;  
}  
  
.student-list {  
    margin-top: 20px;  
}
```

```
.student-list table {  
    width: 100%;  
    border-collapse: collapse;  
    background: #fff;  
    border-radius: 12px;  
    overflow: hidden;  
    box-shadow: 0 2px 6px rgba(0, 0, 0, 0.1);  
}  
  
/*
```

```
.student-list th,  
.student-list td {  
    padding: 12px 15px;  
    text-align: center;  
    border-bottom: 1px solid #eee;  
}  
  
/*
```

```
.student-list th {  
    background-color: #6c5ce7;  
    color: white;  
}  
  
/*
```

```
.student-list tr:hover {  
    background-color: #f1f3f5;  
}  
  
/*
```

```
.student-list button {
```

```
padding: 6px 12px;  
background-color: #0984e3;  
color: white;  
border: none;  
border-radius: 6px;  
cursor: pointer;  
transition: background-color 0.3s;  
}
```

```
.student-list button:hover {  
background-color: #74b9ff;  
}
```

```
.student-detail {  
margin-top: 20px;  
padding: 20px;  
background: #f1f3f5;  
border-radius: 12px;  
}
```

```
.student-detail p {  
margin: 8px 0;  
font-size: 18px;  
}
```

```
form {  
    width: 500px; /* ★ 폼 너비 늘리기 */  
    margin: 40px auto; /* 가운데 정렬 + 위아래 여백 */  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    padding: 80px; /* ★ 폼 안 여백 넓게 */  
    padding-left: 40px; /* 가운데 정렬되지 않아서 왼쪽에 padding으로 가운데 정렬함함 */  
    border: 1px solid #ddd;  
    border-radius: 12px;  
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);  
    background: #fafafa;  
}
```

```
form h2 {  
    margin-bottom: 30px;  
}
```

```
form div {  
    width: 100%;  
    margin-bottom: 20px;  
}
```

```
form input {  
    width: 100%;
```

```
padding: 14px; /* ★ 입력창 padding도 키워서 여유 있게 */
font-size: 16px;
border: 1px solid #ccc;
border-radius: 8px;
}
```

```
form button {
margin-top: 20px;
padding: 14px 28px;
font-size: 16px;
background-color: #4CAF50;
color: white;
border: none;
border-radius: 8px;
cursor: pointer;
transition: background-color 0.3s ease;
}
```

```
form button:hover {
background-color: #45a049;
}
```

> 20.

> 21.

> 22.

> 23.

> 24.

> 25.

> 26.

> 27.

> 28.

> 29.

> 30.
