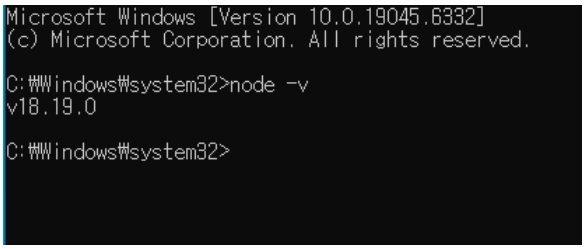
React

https://nodejs.org/ko/download (클릭시 다운로드 페이지 > Windows 설치 프로그램.msi)

(기초 Node.JS설치) > 시작프로그램 (npm 검색) > node - v (버전확인)



Node.js = 크롬 V8 자바스크립트 엔진에서 동작
⇒ 서버사이드 런타임 환경
⇒ (브라우저) 실행되던 자바스크립트가
⇒ PC나 서버환경에서도 실행할 수 있게 만든것

NPM *(Node Packge Manger)*

⇒ 사용하기 좋은 많은 모듈과 라이브러리를 제공

JDK (*Java Development Kit)*

⇒ Java 환경에서 돌아가는 프로그램을 개발하는 데 필요한 툴을 모아놓은 소프트웨어 패키지.

지금부터 Node.js 설치 이후에 React 개발을 시작하는 전 과정을 깔끔하게 정리해봤다.

📌React 설치 이후 전체 흐름 정리

01. 설치 확인

먼저 터미널에서 버전을 먼저 확인한다.

```
node -v
npm -v
```

버전이 나오면 정상입니다.

02. 작업 폴더 준비

원하는 위치에 프로젝트들을 모아둘 상위 폴더를 하나 만든다.
예시로 C 드라이브에 React 폴더를 쓰겠다.

```
mkdir C:\React
cd C:\React
```

03. 새 프로젝트 생성(CRA 기준)

Create React App으로 가장 간단히 시작한다.

```
npx create-react-app reacttest
cd reacttest
npm start
```

브라우저가 http://localhost:3000 을 열면 성공!!!

메모. Vite로 만들고 싶다면 아래 두 줄만 바꿔 쓰면 된다.

```
npm create vite@latest reacttest -- --template react
cd reacttest
npm install
npm run dev
```

04. VSCode에서 열기

VSCode를 켜고 File → Open Folder → C:\React\reactbasic 선택.

터미널 기본 경로를 이 폴더로 고정하고 싶다면 settings.json에 다음을 추가한다.

```
{
  "terminal.integrated.cwd": "C:\\React\\reactbasic"
}
```

05. 폴더 구조와 핵심 파일

src 폴더에 실제 소스 코드를 두며,

- src/index.js 가 진입점이다. 여기서 App 컴포넌트를 root에 렌더링한다.
- src/App.js 가 화면의 시작 컴포넌트이다.
- public/index.html 의 id="root" 요소에 App이 붙는다.

06. 컴포넌트 분리 예시

재사용 가능한 컴포넌트는 src/components 폴더에 둔다.

```
src/
  components/
    First.js
    Second.js
    Third.js
  App.js
```

예시 코드

```
// src/components/First.js
export default function First() {
  return <h2>First Component</h2>
}
```

App에서 불러오기

```
// src/App.js
import First from './components/First'
import Second from './components/Second'
import Third from './components/Third'

export default function App() {
  return (
    <div>
      <First />
      <Second />
      <Third />
    </div>
  )
}
```

여러 학습용 App_파일을 번갈아 실행하고 싶으면 index.js에서 렌더 대상을 바꾸거나 App.js 안에서 원하는 컴포넌트만 주석 해제하면 된다.

```
// src/index.js
import React from 'react'
import ReactDOM from 'react-dom/client'
import App_01 from './App_01_기본함수형컴포넌트'
```

```
const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(<App_01 />)
```

07. 스타일 적용 방법 두 가지

CSS 파일을 src 안에 두고 import 하는 방식

```
/* src/css/style.css */
.font { font-weight: 700; }
.color_red { color: red; }
```

```
// src/App.js
import './css/style.css'
```

public에 둔 CSS를 link로 여는 방식

```
<!-- public/index.html -->
<link rel="stylesheet" href="%PUBLIC_URL%/style.css" />
```

~~08. 자주 설치하는 기본 패키지 (※ 실무에서 거의 필수)~~

라우팅이 필요하면 (라우터 개념 공부할것)

```
npm install react-router-dom
```

API 호출이 필요하면 (axios 개념 공부할것)

```
npm install axios
```

스타일을 컴포넌트 단위로 관리하려면 (스타일 컴포넌트 공부할것)

```
npm install styled-components
```

폼 관리와 검증이 필요하면 (hook 공부할것)

```
npm install react-hook-form yup
```

아이콘이 필요하면 터미널에서 명령어를 치면 된다. (아이콘 불러오는법 공부할것)

```
npm install react-icons
```

09. npm 스크립트 이해

package.json의 scripts에 다음이 있다.

```
"start": "react-scripts start",
"build": "react-scripts build",
```

```
"test": "react-scripts test",
"eject": "react-scripts eject"
```

개발은 npm start, 배포용 번들은 npm run build 이다.

10. 자주 만나는 오류와 빠른 해결

- ENOENT Could not read package.json 이 보이면 현재 폴더에 package.json이 없다는것
  package.json이 있는 프로젝트 폴더로 이동해서 실행한다.

```
cd C:\React\reacttest
npm start
```

- Missing script: start 가 보이면 package.json의 scripts에 start가 없다는것.
- CRA로 다시 만들었는지 확인하거나 scripts를 추가한다.
- react-scripts not found 가 보이면 의존성 설치가 빠졌다는 소리.

```
npm install
```

- Port 3000 is already in use 가 보이면 다른 프로세스가 3000번 포트를 쓰는 중입니다.
  y를 눌러 다른 포트로 열거나 점유 프로세스를 종료한다.

여기까지가 Node.js 설치 이후에 바로 써먹는 표준 흐름이다.

⚛️ React 핵심 개념 정리

1. 기본 개념

1. **컴포넌트 (Component)**
   - UI를 작은 단위로 나눠서 조립
   - 재사용 가능 (버튼, 카드, 네비게이션 등)
2. **JSX (JavaScript + XML)**
   - 자바스크립트 안에서 HTML처럼 UI 구조 작성
   - 코드 가독성 ↑, 작성 편의성 ↑
3. **Props (Properties)**
   - 부모 → 자식 컴포넌트로 값 전달
   - 읽기 전용 데이터 (수정 불가)
4. **State (상태)**
   - 컴포넌트 내부에서 관리하는 동적 데이터
   - 값이 변하면 자동으로 리렌더링
5. **Virtual DOM (가상 DOM)**
   - 변경된 부분만 실제 DOM에 반영 → 성능 최적화

6. **단방향 데이터 흐름 (One-Way Data Flow)**
    - 데이터는 부모 → 자식으로만 전달
    - 예측 가능성 ↑, 관리 쉬움
7. **Router (라우터)**
    - SPA(Single Page Application)에서 페이지 전환 구현
    - 대표 라이브러리: **React Router**
    - URL에 따라 다른 컴포넌트 렌더링

---

2. Hooks (함수형 컴포넌트 핵심)

1. **useState** → 상태 관리
2. **useEffect** → 사이드 이펙트 처리 (데이터 fetch, 구독 등)
3. **useContext** → 전역 데이터 공유 (props drilling 방지)
4. **useReducer** → 복잡한 상태 관리 (Redux 대체 가능)
5. **useRef** → DOM 직접 제어, 값 유지
6. **useMemo** → 값 메모이제이션 → 불필요한 연산 방지
7. **useCallback** → 함수 메모이제이션 → 불필요한 재생성 방지
8. **useLayoutEffect** → DOM 업데이트 직후 동기 실행
9. **Custom Hook** → 재사용 가능한 로직 분리

👏 한 줄 요약

React = 컴포넌트 기반 UI 라이브러리로, JSX + Virtual DOM + 단방향 데이터 흐름을 바탕으로 동작하며, Props/State로 데이터 관리, Router로 화면 전환, Hooks로 상태와 로직을 유연하게 제어한다.

📘 App_01_기본함수형컴포넌트.js 정리

💡 개념 정리

- **함수형 컴포넌트 장점**
    - 선언이 간단하고 메모리 자원을 덜 사용
    - 리액트 v16.8 이전에는 `state`, 생명주기 함수 사용 불가
    - v16.8에서 **Hooks**가 도입 `useState`, `useEffect` 등 사용 가능
    - 반드시 `return()` 안에서 JSX 반환
- **Hooks (❄ 공부 필요)**
    - 함수형 컴포넌트에서 상태 관리와 라이프사이클을 다룰 수 있게 해줌
    - 대표적으로

- `useState` → 컴포넌트 상태 관리
- `useEffect` → 컴포넌트 렌더링 이후 동작 정의

📝 코드 예제

a. 기본 함수형 컴포넌트

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Hello React!!</h1>
      <p>리액트</p>
    </div>
  )
}

export default App
```

b. 화살표 함수형 컴포넌트

```
import React from 'react'

const App = () => {
  return (
    <div>
      <h1>Hello React!!</h1>
      <p>Arrow Function</p>
    </div>
  )
}

export default App
```

👉 이렇게 두 가지 방식(기본 함수형, 화살표 함수형) 모두 동일하게 동작하고,
팀 규칙에 따라 주로 **화살표 함수형 컴포넌트**를 많이 사용한다.

📘 App_02_기본클래스형컴포넌트.js 정리

클래스형은 `state`와 생명주기 메서드를 사용할 수 있고 임의 메서드를 정의해 로직을 분리하기 좋다.
함수형 대비 문법이 길지만 기존 코드나 특정 라이브러리와의 호환 때문에 여전히 쓰인다.
렌더링 결과는 `render()`에서 JSX로 반환한다.

📝 코드 예제

```
/*
  클래스형 컴포넌트
  : state 기능, 생명주기함수, 임의의 메서드를 정의해서 사용 가능
  (코딩규칙) 클래스 리턴이 없으므로 render()함수를 이용 = 그려주는 개념의 함수
*/
import React, { Component } from 'react'  // 리액트와 Component 가져오기

class App extends Component {            // App 컴포넌트 선언
  render() {                             // 화면에 그릴 내용 정의
    return (
      <div>
        <h2>클래스형 컴포넌트</h2>
      </div>
    )
  }
}

export default App                       // 밖에서 사용할 수 있게 내보내기
```

💡 개념 정리

클래스형 컴포넌트는 리액트가 제공하는 Component를 상속해 만드는 화면 단위다.
화면에 무엇을 그릴지는 클래스 안의 render 메서드가 정한다.
지금 단계에서는 render가 JSX를 반환한다는 점만 이해하면 된다.
state나 생명주기는 다음에 따로 배우면 된다.

📘 App_03_JSX기초.js 정리

```
/* 기본 JSX 문법
   : HTML + 자바스크립트를 기본으로 한 리액트만의 문법
     자바스크립트 확장 문법으로, XML의 형식을 따름
     작성한 코드는 브라우저에 실행되기 전 -> 코드가 바인딩되는 과정에서 일단 JS형태로 변환됨
     => 리액트는 실제로 자바스크립트 객체임

   컴포넌트에 여러가지 요소가 있다면 (반드시)하나의 부모태그로 코딩해야 함
   (기본형식)XML 폼을 따르기 때문에 모든 태그는 반드시 닫아줘야 함 */

/* 1. 기본 함수형 */
// import React from 'react'
// function App() {
//     return(
//         <div>
//             <h1>리액트 JSX 문법</h1>
```

```
//           <ol>
//             <li>가독성이 좋고, 코드가 익숙하며, HTML태그를 사용할 수 있음</li>
//             <li>XML폼을 따르고 유사하지만, 실제로는 자바스크립트 객체임</li>
//             <li>
//               Virtual DOM에서 컴포넌트 변화를 감지해 낼때 효율적으로 비교할 수 있도록 <br/>
//               컴포넌트 내부는 반드시 하나의 DOM 트리 구조로 이루어져야 함(=최상위 부모태그 1개 안에
JSX코딩)
//             </li>
//             <li>JSX 모든 태그는 닫아줘야 함 = 닫지 않으면 오류 발생</li>
//             <li>브라우저 실행전 바인딩 과정에서 자바스크립트로 변환된 후 브라우저에서 해석</li>
//           </ol>
//       </div>
//   )
// }
// export default App

/* 1-1. 최상위 부모태그 = Fragment 사용
   : 리액트 v16.8이후 사용가능
    Fragment = DOM에 별도의 노드를 추가하지 않고 여러 자식을 하나로 그룹화 해줌 */
// import React, { Fragment } from 'react'
// function App() {
//   return(
//     <Fragment>
//       <h1>Fragment 테스트</h1>
//     </Fragment>
//   )
// }
// export default App

/* 1-2. (JSX)에서 자바스크립트 사용 */
// import React from 'react'
// function App() {
//   const name = 'mcssam';
//   const library = 'React.js';
//   return(
//     <div>
//       <h1>나의 이니셜은 {name}</h1>
//       <p>
//         <strong>{library}</strong>는 현재 가장 많이 사용되고 있는 <br/>
//         자바스트립트 라이브러리입니다.
//       </p>
//     </div>
//   )
// }
// export default App

/* 1-3. (JSX)조건문 사용 = 삼항조건연산자를 대체하여 사용 */
// import React from 'react'
```

```
// function App() {
//     const name = 'mcssam';
//     return(
//         <div>
//             <h1>JSX내에 삼항조건연산자 사용</h1>
//             <p>
//                 JSX 삼항조건연산자 형식 : <br/>
//                 {/* { name==='mcssam'?('김쌤의 닉네임은 = ' + name):('닉네임 다름') } */}
//                 {/* 템플릿 문자열 */}
//                 { name==='mcssam'?(`김쌤의 닉네임은 = ${name}`) : (`닉네임 다름`) }
//             </p>
//         </div>
//     )
// }
// export default App

/* 2. 클래스형 */
/* 1-4. (JSX)안에서 조건문 사용 */
// import React, { Component } from 'react'
// class App extends Component {
//     render() {
//         return(
//             <div>
//                 {/* 삼항조건연산자 */}
//                 { 1+1 === 2 ? (<div>정답!!</div>) : (<div>오답!!</div>) }
//                 {/* 조건&&참 : 참인 경우만 실행 */}
//                 { 2+2 === 4 && (<div>&amp;&amp;참인 경우만 실행!!</div>) }
//                 {/* 조건||거짓 : 거짓인 경우만 실행 */}
//                 { 2+2 === 5 || (<div>|| 거짓인 경우만 실행!!</div>) }
//             </div>
//         )
//     }
// }
// export default App

/* 1-5. DOM요소에 변수 사용하여 인라인 스타일 적용 */
// import React from 'react'
// function App() {
//     const name = 'mcssam';
//     const jsxStyle = { backgroundColor: '#00f', fontSize: 24, padding: 8, color: '#fff' }
//     return(
//         <div style={jsxStyle}>
//             {name}
//         </div>
//     )
// }
// export default App
```

```jsx
/* 1-6. DOM 요소 외부 스타일 적용 */
import React from 'react'
import './css/style.css'
function App() {
  const str = '외부 스타일 적용';
  return(
    <div className='font color_red'>
      {str}
    </div>
  )
}
export default App
```

JSX는 HTML처럼 보이지만 자바스크립트로 변환되어 동작하는 문법이다.
모든 태그는 꼭 닫아야 한다. 컴포넌트의 반환값은 반드시 하나의 최상위 부모 요소로 감싸야 한다.

✨기본 함수형

- 함수에서 return으로 JSX를 돌려준다.
- 가독성이 좋고 작성이 간단하다.
- 리스트나 여러 태그를 쓸 때는 하나의 부모로 감싼다.

예제 1-1 Fragment 사용

- 불필요한 DOM을 만들지 않고 자식 요소를 묶을 때 `<Fragment>` 또는 빈 태그 `<>...</>`를 쓴다.
- 최상위 부모가 필요하지만 실제 DOM 요소를 추가하고 싶지 않을 때 유용하다.

예제 1-2 JSX에서 자바스크립트 사용

- 중괄호 `{}` 안에서 변수와 표현식을 쓸 수 있다.
- 템플릿 문자열 `` `Hello ${name}` ``을 활용하면 문자열 합치기가 편하다.
- JSX 안에서 주석은 `{/* ... */}` 형태로 작성한다.

예제 1-3 JSX 조건부 렌더링

- 삼항 연산자 `조건 ? 표현식1 : 표현식2` 로 분기한다.
- 참일 때만 보여주려면 `조건 && 표현식` 패턴을 쓴다.
- 거짓일 때만 보여주려면 `조건 || 대체표현식` 패턴을 응용할 수 있다.
- JSX 안에서는 if 문을 직접 쓸 수 없으므로 이런 식의 표현식으로 처리한다.

✨ 클래스형 함수

- 클래스형 컴포넌트의 `render()` 안에서도 위와 같은 삼항, `&&`, `||` 패턴을 그대로 사용한다.

- `render()`가 반환하는 JSX가 화면에 그려진다.

예제 1-5 인라인 스타일 적용

- 객체 형태로 스타일을 준다 예시 `{ backgroundColor: '#00f', fontSize: 24 }`
- CSS 프로퍼티는 `background-color` 대신 `backgroundColor`처럼 카멜케이스를 쓴다.
- 숫자는 px로 간주되고 색상은 문자열로 준다.

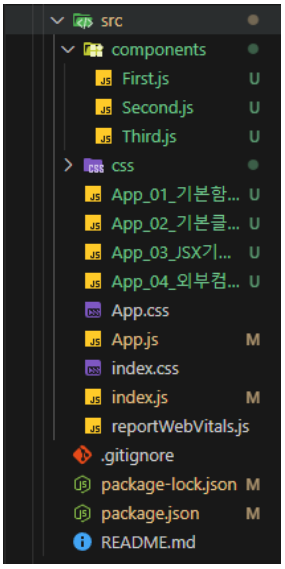
예제 1-6 외부 CSS 적용

- `src` 안에 CSS 파일을 두고 `import './css/style.css'` 로 불러온다.
- 여러 클래스를 적용할 때는 `className="font color_red"` 처럼 공백으로 구분한다.
- `public` 폴더의 CSS를 쓰고 싶다면 `public/index.html`에 `<link>`로 연결한다.

💡 한줄 요약

태그는 반드시 닫고 최상위 부모 하나로 감싸며 조건과 자바스크립트 표현은 `{}` 안에서 처리한다 스타일은 인라인 객체 또는 외부 CSS import로 적용한다

📘 App_04_외부컴포넌트삽입.js 정리




```jsx
import React from 'react'
import './css/style.css'
import First from './components/First'
import Second from './components/Second'
import Third from './components/Third'
function App() {
  const str = '외부 스타일 적용'
  return (
    <div>
      <div className="font color_red">
        {str}
      </div>

      {/* 분리한 컴포넌트 불러오기 */}
      <First />
      <Second />
      <Third />

    </div>
  )
}

export default App
```

`import` **- 프로그래밍에서 불러오다 , 안으로 가져오다 라는 뜻을 의미**

`export` **- 프로그래밍에서 밖으로 내보내다 라는 뜻을 의미**

import 설명

`import React from 'react'` JSX를 사용하기 위한 기본 임포트이다.

`import './css/style.css'` src/css/style.css를 번들에 포함시켜 클래스 스타일을 적용

`import First/Second/Third` src/components 폴더의 분리된 컴포넌트를 가져온다.

렌더 구조

최상위 div 하나로 감싸며 안쪽의

`<div className="font color_red">`가 외부 CSS 클래스를 동시에 적용하고 문자열 `str`을 출력.

그 아래에서 `<First />`, `<Second />`, `<Third />` 순서로 자식 컴포넌트를 렌더링한다.

전제 조건

`src/css/style.css` 파일에 `.font` 와 `.color_red` 클래스가 정의돼 있어야 한다.

`src/components/First.js`, `Second.js`, `Third.js` 파일이 존재하고
각각 기본 내보내기(default export)여야 한다.

```jsx
import React from 'react'

function First() {
  return (
    <div>
      <h2>First Component</h2>
      <p>첫 번째 컴포넌트입니다.</p>
    </div>
  )
}

export default First
```

```jsx
import React from 'react'

function Second() {
  return (
    <div>
      <h2>Second Component</h2>
      <p>두 번째 컴포넌트입니다.</p>
    </div>
  )
}

export default Second
```

```jsx
import React from 'react'

function Third() {
  return (
    <div>
      <h2>Third Component</h2>
      <p>세 번째 컴포넌트입니다.</p>
    </div>
  )
}

export default Third
```

폴더 구조

```
src/
  App_04_외부컴포넌트삽입.js
  components/
```

```
    First.js
    Second.js
    Third.js
  css/
    style.css
```

index.js 연결

`import App from './App_04_외부컴포넌트삽입'` 후 `root.render(<App />)` 로 렌더링한다.

📘 App_05_Props컴포넌트.js 정리

```
import React from 'react'
import './css/style.css'
// import First from './components/First'
// import Second from './components/Seco
nd'
// import Third from './components/Third'
import Props from './components/Props'

  function App() {

  return (
    <div>

      {/* 분리한 컴포넌트 불러오기 */}
      {/* <First /> */}
      {/* <Second /> */}
      {/* <Third /> */}
      <Props name="김재돌" nickname="mc
ssam"/> {/* Props 전달 */}

    </div>
  )
}

export default App
```

```
function Props({ name, nickname }) {

 const point = { color: 'blue', fontSize: 30 }

 return (
   <div>
     <h1>부모 컴포넌트에서 전달받은 이름 : {na
me}</h1>
     <h3>부모 컴포넌트에서 전달받은 별명 : {ni
ckname}</h3>
     <p style={point}>{name} : {nickname}
</p>
   </div>
 )
}

export default Props
```

부모 컴포넌트에서 전달받은 이름 : 김재돌

부모 컴포넌트에서 전달받은 별명 : mcssam

김재돌 : mcssam

리액트에선 데이터 전달 방법이 딱 두가지로 구성 되어있다.

Props - JSX에서 속성명 과 속성값으로 부모가 자식에게 넘기는 것

- 상속개념이다.

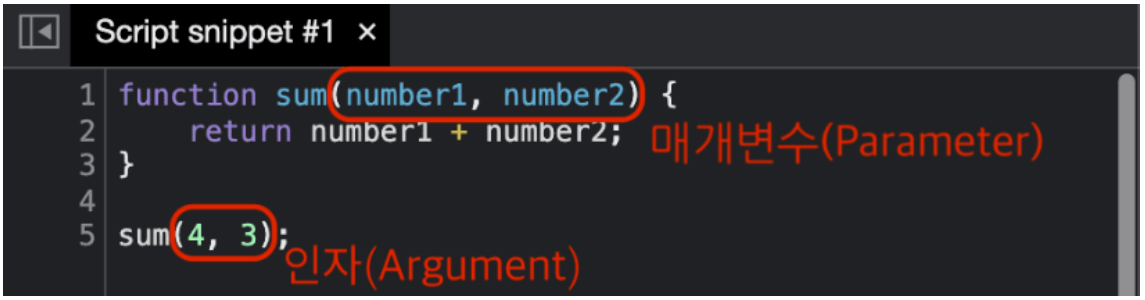
- 자식은 받은 props를 직접 바꿀 수 없다
- 부모가 내려줄 때 형태는 속성명:값이며 JSX에선 `<Child name="홍길동" age={20} />` 처럼 쓴다

State - 보관성 내부 데이터로 key와 value를 상태로 들고 있다가 전용 함수로만 바꾼다.

- 상속 개념이 아니다.
- 컴포넌트 자신이 들고 있는 내부 상태 값.
- 컴포넌트 안에서만 보관하고 `setState` 또는 `useState`로만 변경한다.
- 부모의 state를 자식에게 "전달"할 때도 결국 props로 내려가는 것이다.

데이터 흐름 요약

- 기본 흐름은 위에서 아래로 한 방향이다.
- 자식에서 부모로 값을 "올리고" 싶으면,
  부모가 콜백 함수를 props로 내려주고 자식이 그 함수를 호출한다.
- 여러 자식이 같은 값을 써야 하면 그 값을 더 위 부모로 끌어올려 한 곳에서 관리한다
- 중첩이 깊어 props 전달이 번거로우면 Context 또는 전역 상태 관리 라이브러리를 쓴다.



Javascirpt 매게변수와 인자 처럼, Parent()에서 **State** 를 리턴해서 **Props** 전달 Child 받고 실행.

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



function Parent() {
  const [userName, setUserName] = useState('홍길동') //초기값 홍길동
  const changeName = e => setUserName(e)  // state를 변경하는 콜백
  return <Child name={userName} onChangeName={changeName} /> // Child로 props 던짐
}

function Child({ name, onChangeName }) { // 매게변수 name과 onChangeName 받아
  return ( // 클릭시 onChangeName  실행 ('변경값') 인자로 변경
```

```
    <>
      <h3>{name}</h3>
      <button onClick={() => onChangeName('김현수')}>변경</button>
    </>
  )
}

function Props({ name, nickname }) {
  const point = { color: 'blue', fontSize: 30 }

  return (
    <div>
      <h1>부모 컴포넌트에서 전달받은 이름 : {name}</h1>
      <h3>부모 컴포넌트에서 전달받은 별명 : {nickname}</h3>
      <p style={point}>{name} : {nickname}</p>
      <Parent />
    </div>
  )
}

export default Props
```

코드흐름

- **Parent가 userName을 useState로 가진다.**
- **Parent가 name과 onChangeName을 Child에 props로 내려준다**
- **Child가 버튼 클릭 시 onChangeName('김현수') 호출**
- **Parent의 상태가 바뀌고 다시 렌더링된다.**

버튼을 누르면 Parent의 state가 `홍길동 → 김현수`로 바뀌고 그 값이 Child를 통해 화면에 즉시 반영된다.