

# React

## #1. The Basic of React

React.js : interactive 한 UI

React.dom : React Elements 를 HTML안에 삽입

ReactDOM.render()

### JSX

```
const element = <h1>Hello, world!</h1>;
```

JSX라 하며 JavaScript를 확장한 문법

JSX는 React “엘리먼트(element)” 를 생성

React에서는 본질적으로 렌더링 로직이 UI 로직(이벤트가 처리되는 방식, 시간에 따라 state가 변하는 방식, 화면에 표시하기 위해 데이터가 준비되는 방식 등)과 연결된다는 사실을 받아들인다.

EX)

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

JSX도 표현식이다.

컴파일이 끝나면, JSX 표현식이 정규 JavaScript 함수 호출이 되고 JavaScript 객체로 인식된다.

즉, JSX를 `if` 구문 및 `for` loop 안에 사용하고, 변수에 할당하고, 인자로서 받아들이고, 함수로부터 반환할 수 있다.

#### !!렌더링 주의사항!!

직접 만든 컴포넌트를 렌더링해서 다른 곳에서 사용할 때는 항상 대문자로 시작해야 한다.

(컴포넌트의 첫 글자는 대문자!! 만일 소문자면 React아 JSX는 HTML tag라고 생각한다.)

JSX 생성 규칙

## BABEL 이란?

<h3>Total clicks: {counter}</h3> React에서는 JSX에 변수를 전달할 때 {}를 사용하여 전달 가능!  
함수형으로 리액트 엘리먼트 생성

```
<body>
  <div id="root"></div>
</body>
<script src="https://unpkg.com/react@17.0.2/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script type="text/babel">
  const root = document.getElementById("root");
  const Title = () => (
    // JSX를 사용
    <h3 id="title" onMouseEnter={() => console.log("mouse enter")}>
      Hello I'm title
    </h3>
  );
  // const h3 = React.createElement(
  //   "h3",
  //   {
  //     onMouseEnter: () => console.log("mouse enter"),
  //   },
  //   "Hello I'm a h3!!"
  // );
  const Button = () => (
    //JSX를 사용
    <button
      style={{ backgroundColor: "tomato" }}
      onClick={() => console.log("i'm clicked")}
    >
      Click Me
    </button>
  );
  // const btn = React.createElement(
  //   "button",
  //   {
  //     onClick: () => console.log("i'm clicked!"),
  //   },
  //   "click me"
  // );
  const Container = () => (
    <div>
      <Title />
      <Button />
    </div>
  );
  // const container = React.createElement("div", null, [title, button]);
  ReactDOM.render(<Container />, root);
</script>
```

## STATE

React에서는 UI에서 바뀐 부분만 업데이트해준다. (개발자도구사용시)

rerendering 하는 가장 좋은 방법

react에서는 component를 새로 rendering할 때, 전체를 전부 재생성할 필요없이 바뀐부분만 새로 생성할 수 있게 도와준다.

데이터가 변경될 때마다 rendering 해주는 걸 안해도된다.

React.DOM.render(<App />, root); 를 호출 안해줘도됨

React.useState() 함수

```
const [data, setData] = React.useState(data의 초기값);
let counter = 0;
function countUp() {
  //code
```

```

}

const data = React.useState(0); 위와 동일
state가 바뀌면 react component를 새로운 값을 가지고 refresh 해준다.

```

배열에 있는 요소 꺼내는 방법

```

const x = [1, 2, 3] 배열 생성
const [a, b, c] = x // 배열의 요소를 하나하나씩 순차적으로 이름 부여
실행결과 a = 1, b = 2, c = 3

```

state function

```

<body>
  <div id="root"></div>
</body>
<script src="https://unpkg.com/react@17.0.2/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script type="text/babel">
  const root = document.getElementById("root");
  function App() {
    const [counter, setCounter] = React.useState(0);
    const onClick = () => {
      // setCounter(counter + 1);
      setCounter((current) => current + 1); // setCounter(counter + 1) 과 동일하지만 위의 방법 보다 안전하다. 리액트가 이 current가 확실히 현재값
    };
    return (
      <div>
        <h3>Total clicks: {counter}</h3>
        <button onClick={onClick}>Click me</button>
      </div>
    );
  }
  ReactDOM.render(<App />, root);
</script>

```

jsx에서의 html 사용법

```

function App() {
  return (
    <div>
      <h1 className="hi">Super Converter</h1>
      <label htmlFor="minutes">Minutes</label>
      <input id="minutes" type="number" placeholder="Minutes" />
      <label htmlFor="hour">Hour</label>
      <input id="hour" type="number" placeholder="Hour" />
    </div>
  );
}

```

기존 html에서 class와 label의 for를 사용할 때와는 다르게 jsx에서는 className, htmlFor를 사용하여야 한다.

why? ) class와 for는 javascript에 미리 선점된 단어이기 때문

input을 이용한 state

```

<body>
  <div id="root"></div>
</body>
<script src="https://unpkg.com/react@17.0.2/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

```

```

<script type="text/babel">
  const root = document.getElementById("root");
  function App() {
    const [minutes, setMinutes] = React.useState();
    const onChange = (event) => {
      setMinutes(event.target.value);
    };
    return (
      <div>
        <h1 className="hi">Super Converter</h1>
        <label htmlFor="minutes">Minutes</label>
        <input
          value={minutes}
          id="minutes"
          type="number"
          placeholder="Minutes"
          onChange={onChange}
        />
        <h4>you want to convert {minutes}</h4>
        <label htmlFor="hour">Hour</label>
        <input id="hour" type="number" placeholder="Hour" />
      </div>
    );
  }
  ReactDOM.render(<App />, root);
</script>

```

## flip conversion

```

<body>
  <div id="root"></div>
</body>
<script src="https://unpkg.com/react@17.0.2/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script type="text/babel">
  const root = document.getElementById("root");
  function App() {
    const [minutes, setMinutes] = React.useState();
    const onChange = (event) => {
      setMinutes(event.target.value);
    };
    return (
      <div>
        <h1 className="hi">Super Converter</h1>
        <label htmlFor="minutes">Minutes</label>
        <input
          value={minutes}
          id="minutes"
          type="number"
          placeholder="Minutes"
          onChange={onChange}
        />
        <h4>you want to convert {minutes}</h4>
        <label htmlFor="hour">Hour</label>
        <input id="hour" type="number" placeholder="Hour" />
      </div>
    );
  }
  ReactDOM.render(<App />, root);
</script>

```

## converter

```

const root = document.getElementById("root");
function App() {
  const [amount, setAmount] = React.useState();
  const [flipped, setFlipped] = React.useState(false);

  const onChange = (event) => {

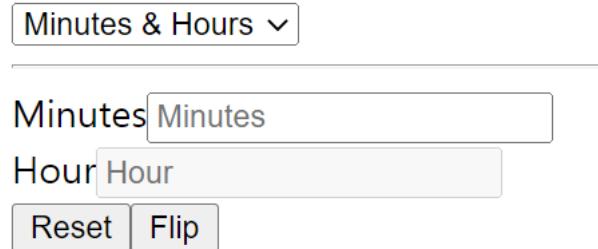
```

```

        setAmount(event.target.value);
    };
    const reset = () => setAmount(0);
    const onFlip = () => {
        setFlipped((current) => !current);
        reset();
    };
    return (
        <div>
            <h1 className="hi">Super Converter</h1>
            <div>
                <label htmlFor="minutes">Minutes</label>
                <input
                    value={flipped ? amount * 60 : amount} //input값을 state로 연결해주는 이유는 input값을 외부에서도 수정해주기 위해서
                    id="minutes"
                    type="number"
                    placeholder="Minutes"
                    onChange={onChange}
                    disabled={flipped}
                />
            </div>
            <h4>you want to convert {amount}</h4>
            <div>
                <label htmlFor="hour">Hour</label>
                <input
                    value={flipped ? amount : Math.round(amount / 60)}
                    id="hour"
                    type="number"
                    placeholder="Hour"
                    onChange={onChange}
                    disabled={!flipped}
                />
            </div>
            <button onClick={reset}>Reset</button>
            <button onClick={onFlip}>{flipped ? "Turn Back" : "Flip"}</button>
        </div>
    );
}
ReactDOM.render(<App />, root);
</script>

```

# Super Converter



## PROPS

컴포넌트 사용 시

컴포넌트에 props를 넣으면 React.js는 자동으로 props를 컴포넌트에 오브젝트로 주어지고 컴포넌트는 argument형태로 받는다.

부모컴포넌트에서 자식컴포넌트로 값 전달

```

<script type="text/babel">
    function Btn({ banana, big }) {

```

```

//props는 objects (부모 컴포넌트에서 보낸 모든 것들을 갖는)
return (
  <button
    style={{
      backgroundColor: "tomato",
      color: "white",
      border: 1,
      borderRadius: 10,
      padding: "10px 20px",
      font_size: big ? 18 : 12, // style내에서 삼항 연산자 사용 가능
    }}
  >
  {banana}
  </button>
);
}
function App() {
  return (
    <div>
      <Btn banana="Save" big={true} />
      <Btn banana="Confirm" big={false} />
    </div>
  );
}
const root = document.getElementById("root");
ReactDOM.render(<App />, root);
</script>

```

## memo

Btn onClick={changeValue} /> 이것은 이벤트 리스너를 붙인 것이 아닌, 컴포넌트에 onClick이라는 프롭을 전달한 것이다. 여기서 onClick은 단순히 props의 이름이다.

결국 컴포넌트에 onClick 이벤트리스너를 붙이고 싶다면 해당 컴포넌트의 props으로 이벤트 리스너를 보내고, 그걸 받아오면 해당 컴포넌트의 최상단 엘리먼트에 onClick 이벤트 리스너를 붙이면 된다

즉, 컴포넌트에는 HTML Element처럼 속성을 지정해줄 수 없다.

컴포넌트에 그러한 행위를 하는것은 그저 props를 전달해 주는 것 뿐이다.

## 리액트의 렌더링방식

불필요한 리렌더링을 MEMO로 막기

```

<script type="text/babel">
  function Btn({ text, onClick }) {
    //props는 objects 부모 컴포넌트에서 보낸 모든 것들을 갖는
    console.log(text, "was rendered");
    return (
      <button
        style={{
          backgroundColor: "tomato",
          color: "white",
          border: 1,
          borderRadius: 10,
          padding: "10px 20px",
        }}
        onClick={onClick}
      >
        {text}
      </button>
    );
  }
  const MemorizedBtn = React.memo(Btn);
  function App() {
    const [value, setValue] = React.useState("Save Changes");
    const changeValue = () => setValue("Revert Changes");
    return (
      <div>
        <MemorizedBtn text={value} onClick={changeValue} />
        <MemorizedBtn text="Continue" />
      </div>
    );
  }
  const root = document.getElementById("root");
  ReactDOM.render(<App />, root);
</script>

```

실행결과

Save Changes was rendered

```
Continue was renderd  
Revert Changes was renderd
```

## CRA

### 리액트 프로젝트를 시작하는데 필요한 개발 환경을 세팅해주는 도구(Tool Chain)

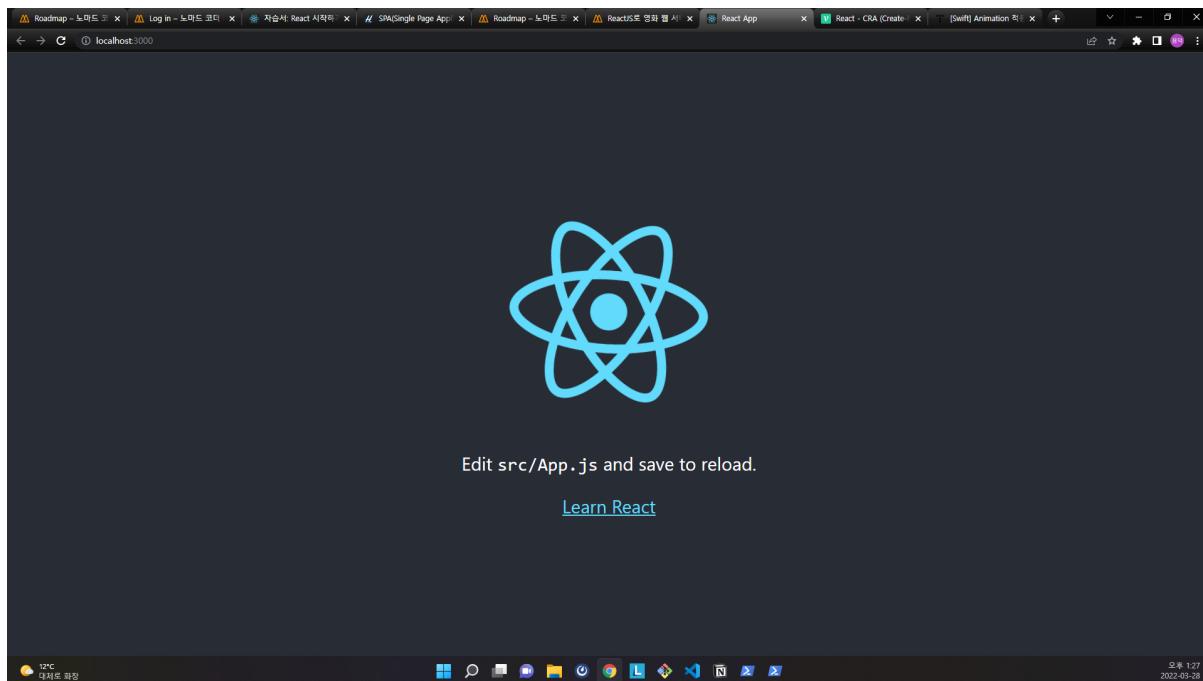
기본적으로 지금까지는 script 파일을 import해주는 방식으로 진행했지만, 이렇게 진행을 할 경우 개발자가 필요한 라이브러리를 직접 추가하는 번거로움이 발생한다.

Create-React-App은 이러한 React 프로젝트에 필요한 기본 설정들을 자동으로 세팅해주는 도구이다.

### CRA 설치

- CRA 프로젝트를 설치하기 위해서는 :

- Desktop - project 폴더 진입 `cd Desktop/Projects`
  - react-app 설치 `npx create-react-app my-project`
  - my-project 폴더 진입 `cd my-project`
  - 로컬 서버 띄우기 `npm start`
- npx on the first line is not a typo — it's a package runner tool that comes with npm 5.2+.
  - npm start 입력 시 <http://localhost:3000> 주소를 확인할 수 있다.



## CRA 기본 폴더 및 파일 구성

### 1) node.modules

- CRA 를 구성하는 모든 패키지 소스 코드가 존재하는 폴더

### 2) package.json

- CRA 기본 패키지 외 추가로 설치된 라이브러리/패키지 정보(종류, 버전)가 기록되는 파일
- 모든 프로젝트마다 package.json 하나씩 존재
- `"dependencies"`
  - 리액트를 사용하기 위한 모든 패키지 리스트, 버전 확인 가능
  - 실제 코드는 `node.modules` 폴더에 존재
  - 왜 node.modules 와 package.json 에서 이중으로 패키지를 관리할까?
    - 실제 내가 작성한 코드, 내가 설치한 패키지는 내 로컬에만 존재
    - github 에 올릴 때 내가 작성한 코드와 함께 **package.json(추가로 설치한 패키지 정보)**을 넘긴다.
    - 다른 사람이 그것을 (pull) 받아서 npm install 만 입력하면 package.json 에 기록되어 있는 패키지의 이름과 버전 정보를 확인하여 자동으로 설치한다.
    - 이때, github 에 올릴 때, node.modules 는 올리지 않는다. (불필요한 용량 차지),
      - .gitignore 파일에 github 에 올리고 싶지 않은 폴더와 파일을 작성할 수 있다.
- 참고) 새로운 Library(package) 설치 시
  - 누군가 만든 소스코드를 다운받는 것
  - npm으로 설치 (ex. npm install slider)
  - 설치 시 node modules 에 자동으로 설치됨.
  - 하지만 package.json - dependencies 에 추가 자동으로 되는 건 아님.
  - 그래서, npm install slider —save
  - —save 까지 작성해야 dependencies 에 추가됨
  - (npm 버전이 업그레이드 됨에 따라 자동으로 추가되는 경우가 많지만 여전히 불안한 패키지들이 존재하기 때문에 패키지 설치 시 —save 까지 입력하는 것을 권장.)
- `"scripts"`
  - run : 프로젝트를 development mode(개발 모드) 실행을 위한 명령어. `npm run start`.
  - build : 프로젝트 production mode(배포 모드) 실행을 위한 명령어. 서비스 상용화.

### • 참고) package.json vs. package-lock.json

### 3) .gitignore

- `.gitignore` 파일에 github 에 올리고 싶지 않은 폴더와 파일을 작성할 수 있다.
- `push` 를 해도 `.gitignore` 파일에 작성된 폴더와 파일은 올라가지 않는다.

### 1) public - `index.html`

- `<div id="root"></div>`

### 2) src - `index.js`

- React의 시작 (Entry Point)
- `ReactDOM.render( <App /> , document.getElementById('root'))`
  - ReactDOM.render 함수의 인자는 두 개

- 첫 번째 인자는 화면에 보여주고 싶은 컴포넌트
- 두 번째 인자는 화면에 보여주고 싶은 컴포넌트의 위치
- (이름 함부로 수정하면 안 됨)

### 3) src - `App.js`

- 현재 화면에 보여지고 있는 초기 컴포넌트
- React Router 를 적용 후에는 `<Routes />` 컴포넌트가 최상위 컴포넌트로 그 자리에 위치하게 된다.

## 기타 폴더 구성

### 1) public 폴더

- index.html
- images - 이미지 파일 관리
- data - mock data 관리

### 2) src 폴더

- components - 공통 컴포넌트 관리
- pages - 페이지 단위의 컴포넌트 폴더로 구성
- styles 폴더
  - `reset.scss` - CSS 초기화
  - `common.scss` - 공통으로 사용하는 CSS 속성 정의 (ex. font-family, theme color)
- 참고) components vs. pages
  - 여러 페이지에서 동시에 사용되는 컴포넌트의 경우 components 폴더에서 관리함. (ex. Header, Nav, Footer)
  - 페이지 컴포넌트의 경우 pages 폴더에서 관리함.
  - 해당 페이지 내에서만 사용하는 컴포넌트의 경우 해당 페이지 컴포넌트 폴더 하위에서 관리함.

npm-start를 했을 시 사용되지 않은 components를 알려준다. (CRA의 기능)

```
WARNING in src\App.js
Line 1:8: 'Button' is defined but never used no-unused-vars
```

Button이라는 component가 정의되었고, import를 했지만 사용되지 않았다는 WARNING을 CRA에서 보냄.

## 실습 예제

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

```
import Button from "./Button";
import styles from "./App.module.css";
function App() {
  return (
    <div>
      <h1 className={styles.title}>Welcome Back!!!</h1>
      <Button text={"click"} />
    </div>
  );
}

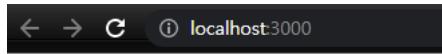
export default App;
```

```
        </div>
    );
}

export default App;
```

결과 화면

App.module.css / Button.module.css



Welcome Back!!!

click

```
.title {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 40px;
}
```

```
.title {
    background-color: brown;
    font-size: 18px;
}
```

ClassName을 Random으로 지정

```
<h1 class="App_title_7ikRp">Welcome Back!!!</h1> == $0
<button class="Button_title_dgaIS">click</button>
```