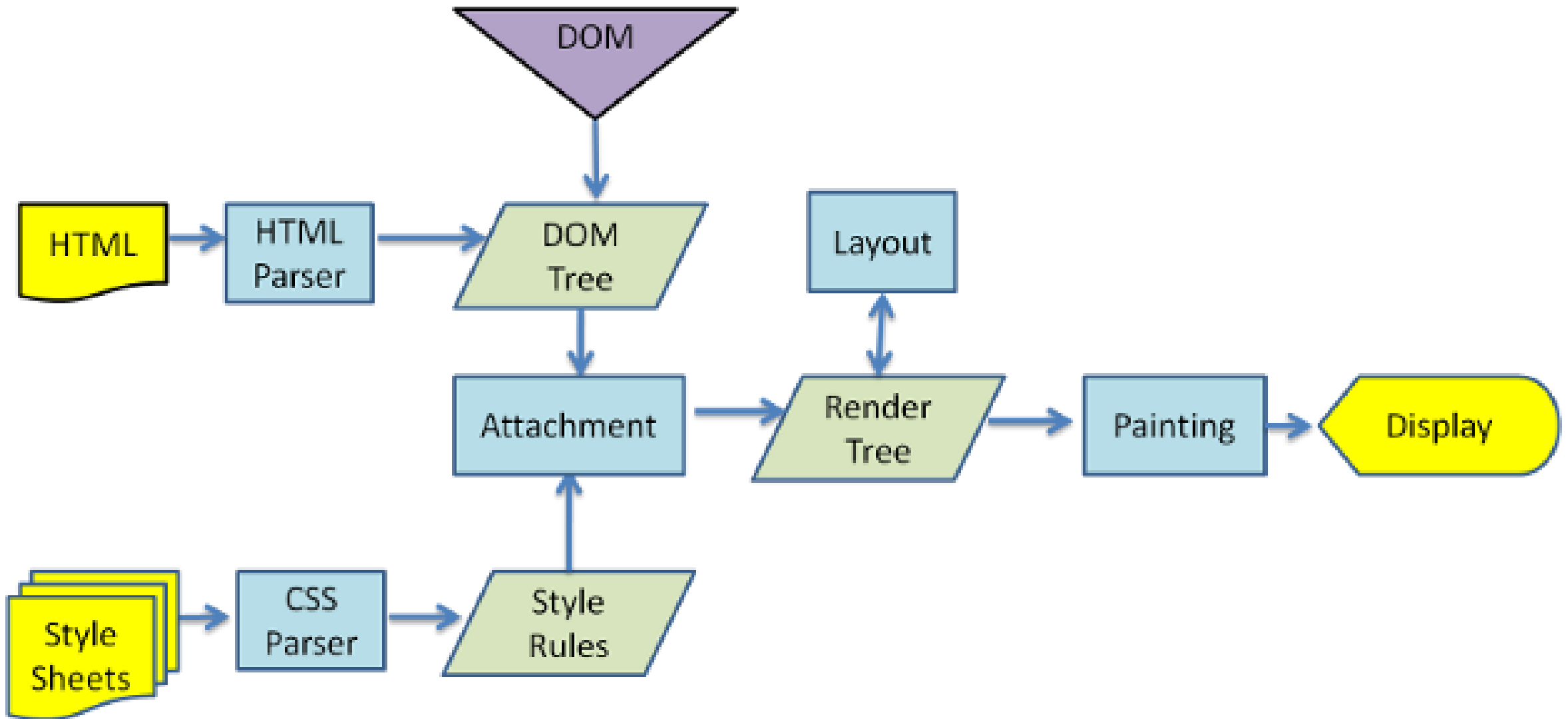
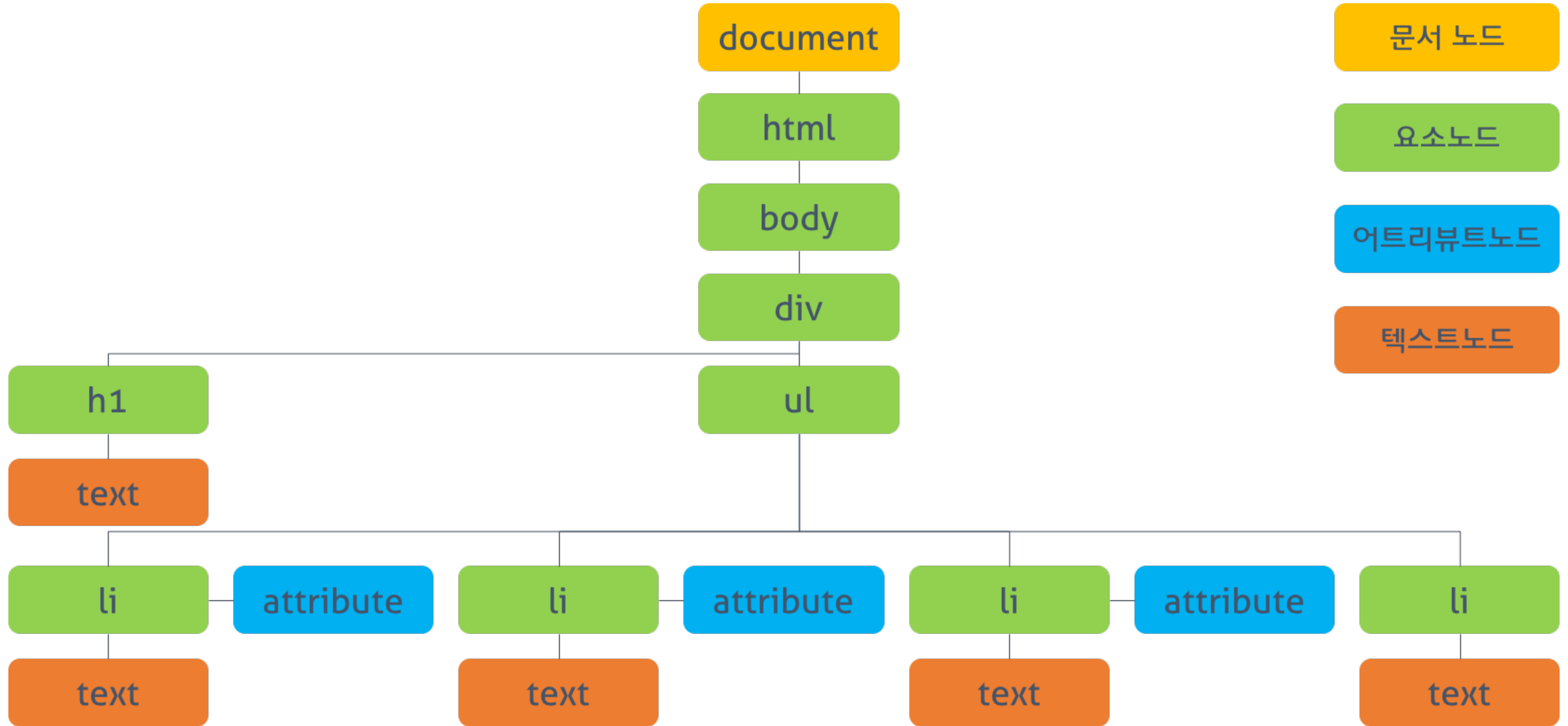


리액트

브라우저의 Workflow



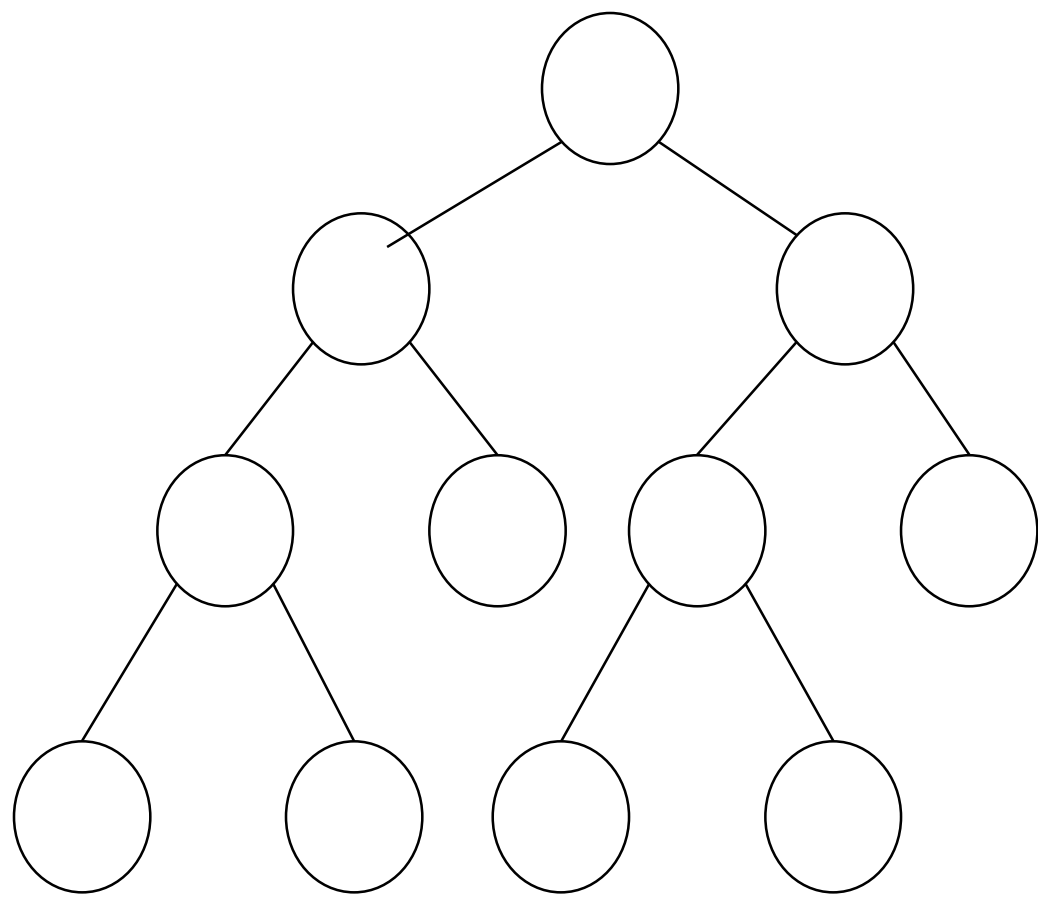
DOM(Document Object Model : 문서객체모델) Tree



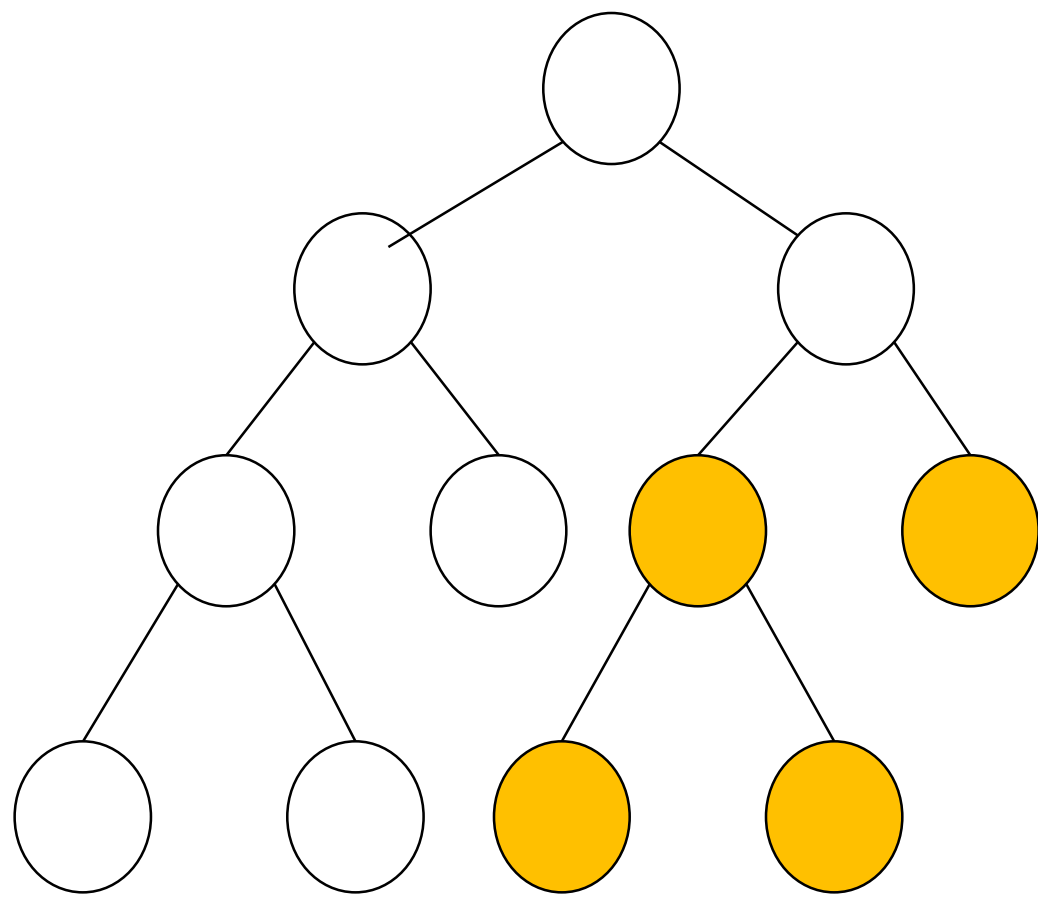
리액트

- 페이스북이 만든 사용자 UI구축을 위한 JS 라이브러리임
- 리액트는 Virtual DOM(Document Object Model : 문서객체모델)을 사용함
- Virtual DOM 방식을 사용하여 DOM 업데이트를 추상화함으로써 DOM 처리 횟수를 최소화하고 효율적으로 진행함
- 절차 1 : 데이터를 업데이트하면 전체 UI를 Virtual DOM에 리렌더링함
- 절차 2 : 이전 Virtual DOM에 있던 내용과 현재 내용을 비교함
- 절차 3 : 바뀐 부분만 실제 DOM에 적용함

자바스크립트를 사용하여 두 가지 뷰를 **비교**한 후, **둘의 차이를 알아내 최소한의 연산으로 DOM 트리를 업데이트함**



이전 DOM 트리



새로운 DOM 트리(Virtual DOM)

컴포넌트 : HTML 문서를 기능이나 목적에 따라 분할함

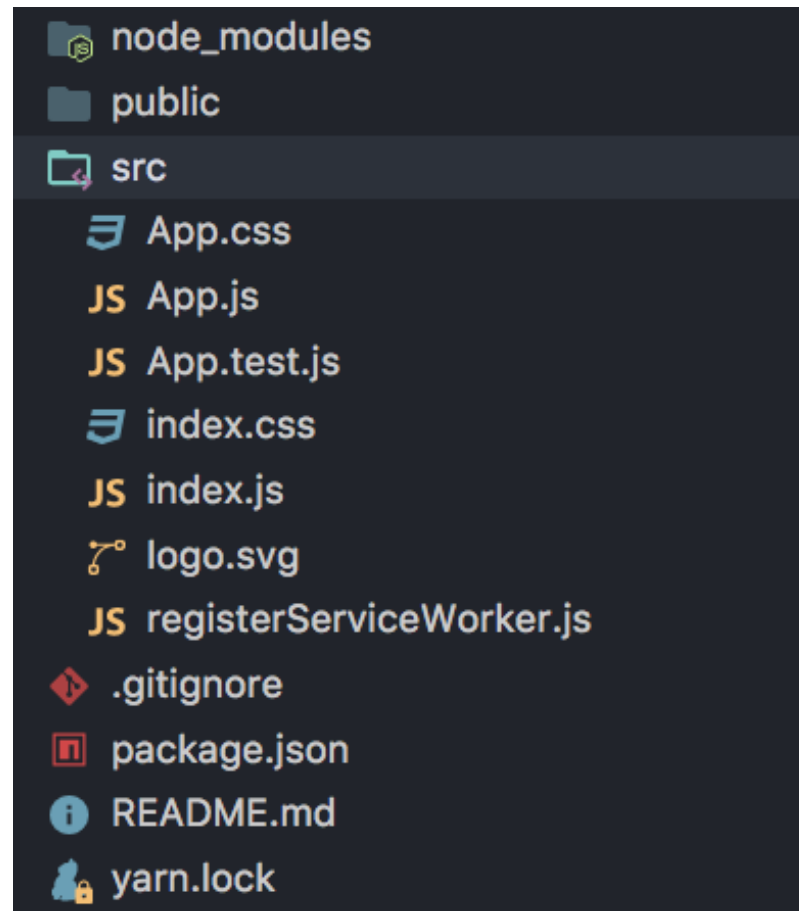
- 컴포넌트는 HTML을 반환(return)하는 함수
- 가독성이 좋음
- 재사용성 높음
- 유지보수 용이

작업환경 설정

- **Node.js** : 기존에는 자바스크립트 프로그램은 인터넷 웹브라우저 위에서만 실행할 수 있었음. 노드는 자바스크립트 프로그램을 웹브라우저 영역 뿐만 아니라, 웹서버, 모바일 애플리케이션, 데스크톱 애플리케이션 영역에서도 실행할 수 있게 해줌
리엑트는 웹브라우저에서 실행되는 코드이므로 Node.js와 직접적인 연관은 없지만, 프로젝트를 개발하는데 필요한 주요 도구들이 Node.js를 사용하기 때문에 설치함
Node.js 설치 : <https://nodejs.org/ko/download/> (Windows 용 설치) (설치확인 `node -v`)
- **Npm** : Node.js의 패키지 매니저 도구임. Npm으로 수많은 개발자가 만든 패키지(재사용 가능한 코드)를 설치하고 설치한 패키지의 버전을 관리할 수 있음.
npm 실행 후 `npm -v`
- **Yarn** : npm보다 빠르고 효율적인 캐시 시스템과 기타 부가 기능을 제공함.
Yarn 설치 : <https://yarnpkg.com/en/docs/install#windows-stable>
- **Visual Studio Code** 설치 : 유용한 확장프로그램(ESLint, Reactjs Code Snippets, Prettier-Code formatter)

yarn strat

- 리액트 개발 전용 서버 구동(Visual Studio Code에서 보기 - 터미널 메뉴 클릭)
- 프로젝트 생성하기 : **yarn create react-app 프로젝트명(hello-react)**
- 프로젝트 디렉터리 이동한 후 서버 구동 :
cd hello-react (실행할 프로젝트 폴더로 이동)
yarn start (리액트실행)
- 프로그램이 자동으로 열리 않을 때 :
주소창에 <http://localhost:3000/> 입력



ESLint : 문법검사 도구

- VS 코드 화면 [보기] 메뉴 - [문제] 클릭하면 오류메시지 출력
빨간색 표시줄은 치명적인 오류이므로 반드시 수정해야 함

Prettier(코드스타일 자동정리 도구)

- ▶ F1 - format 입력하고 Enter 치면 보기좋게 정리됨
세미콜론(;)이 빠진 곳에는 세미콜론이 자동으로 추가됨
작은 따옴표는 큰 따옴표로 변경됨
- ▶ 저장할때 자동으로 코드 정리하기
설정아이콘 클릭 - [설정] - format on save 검색하여 찾아지면 체크 표시하기

Reactjs Code Snippet

- ▶ 리액트 코드 단축키 자동완성
rsc 엔터(함수형 컴포넌트) rcc 엔터(클래스형 컴포넌트)

컴포넌트 파일 파헤치기(src/index.js)

- App 컴포넌트를 불러와서 ReactDOM.render() 함수를 사용하여 index.html의 id가 'root'인 DOM을 찾아 그리도록 설정함(렌더링)

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './index.css';
```

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

```
import registerServiceWorker from './registerServiceWorker';
```

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

```
registerServiceWorker();
```

함수형 컴포넌트 vs 클래스형 컴포넌트

```
import React from "react";  
import "./App.css";
```

```
function App() {  
  const name = "리액트";  
  return <div className="react">{name}</div>  
}
```

리액트 v16.8 업데이트 이후 Hooks라는 기능이 도입되면서 함수형컴포넌트도 state와 라이프사이클 API 사용이 가능해짐.
함수형 컴포넌트는 클래스형 컴포넌트에 비해 선언도 편하고 메모리 자원도 덜 사용하므로, Hooks와 함께 사용을 권장하고 있음

```
import React, { Component } from "react";  
import "./App.css";
```

```
class App extends Component {  
  render() {  
    const name = " 리액트";  
    return <div className="react">{name}</div>;  
  }  
}
```

결과는 똑같지만 클래스형 컴포넌트는 state 기능 및 라이프사이클 기능을 사용할 수 있으며, 임의 메서드를 정의할 수 있음

컴포넌트 파일 파헤치기(src/App.js)

```
import React, { Component } from 'react';  
import logo from './logo.svg';  
import './App.css';
```

리액트를 불러와서 사용할 수 있게 해줌
내부의 다른 Component 를 불러옴(import)

▶ 컴포넌트 종류 : 클래스형 컴포넌트 vs 함수형 컴포넌트

▶ 클래스형 컴포넌트 : render() 함수 필요하며, 내부에서는 JSX를 return해 주어야 함

```
class App extends Component {
```

```
  render() {
```

```
    return (
```

JSX : 자바스크립트의 확장 문법이며 XML과 매우 비슷함. 작성한 코드는 브라우저에서 실행되기 전에 코드가 번들링(연결)되는 과정에서 일반 자바스크립트 형태로 변환됨

```
    )
```

```
  }
```

```
}
```

```
export default App;  컴포넌트를 다른 곳에서 사용할 수 있도록 내보내기 함
```

JSX : 자바스크립트에서 사용하는 html

- 보기 쉽고 익숙하며, HTML 태그를 사용할 수 있음
- XML 형식이지만 실제로는 자바스크립트 객체임

JSX 문법

- 컴포넌트에 여러 요소가 있다면 반드시 부모 요소 하나로 감싸야 함.

```
import React from 'react';
function App() {
  return (
    <div>
      <h1>리액트 안녕</h1>
      <h2>잘 작동하나?</h2>
    </div>
  )
}
```

```
import React, { Fragment } from 'react';
function App() {
  return (
    <Fragment>
      <h1>리액트 안녕</h1>
      <h2>잘 작동하나?</h2>
    </Fragment>
  )
}
```

cf) 리액트 v16 이상

```
import React from 'react';
function App() {
  return (
    <>
      <h1>리액트 안녕</h1>
      <h2>잘 작동하나?</h2>
    </>
  )
}
```

- Virtual DOM에서 컴포넌트 변화를 감지해 낼 때 효율적으로 비교할 수 있도록 컴포넌트 내부는 하나의 DOM 트리 구조로 이루어져야 한다.

JSX에서는 모든 태그를 닫지 않으면 오류 발생

```
function App() {  
  const name = '리액트';  
  return (  
    <>  
      <div className="react">{name}</div>  
      <input> </input>  
    </>  
  )  
}
```

```
function App() {  
  const name = '리액트';  
  return (  
    <>  
      <div className="react">{name}</div>  
      <input />  
    </>  
  )  
}
```

JSX에서 주석 작성하기 `{/* ~ ~ */}`

```
function App() {  
  const name = '리액트';  
  return (  
    <>  
      {/* 주석은 이렇게 작성합니다. */}  
      <div className="react">{name}</div>  
      <input> </input>  
    </>  
  )  
}
```

```
// function App() {  
//   const name = '리액트';  
//   return (  
//     <>  
//       <div className="react">{name}</div>  
//       {/* <input /> */}  
//     </>  
//   )  
// }
```

새로운 컴포넌트 추가하기

Reply.js

```
import React from 'react';  
function Reply() {  
  return (  
    <div>  
      <p>잘 작동하고 있어요.</p>  
    </div>  
  )  
}  
export default Bread;
```

App.js

```
import React from 'react';  
import Reply from './Reply';  
function App() {  
  return (  
    <div>  
      <h1>리액트 안녕</h1>  
      <h2>잘 작동하나?</h2>  
      <Reply />  
    </div>  
  )  
}
```


새로운 컴포넌트 추가하기

```
function Reply() {  
  return (  
    <div>  
      <p>잘 작동하고 있어요.</p>  
    </div>  
  )  
}
```

App.js

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

```
function App() {  
  return (  
    <div>  
      <h1>리액트 안녕</h1>  
      <h2>잘 작동하나?</h2>  
      <Reply />  
    </div>  
  )  
}
```

리액트에서 DOM 요소에 인라인스타일 적용

- Import React from 'react';

```
function App() {  
  return (  
    <div style = {{  
      backgroundColor: 'black',  
      color: 'aqua',  
      fontSize: '48px',  
      padding: 16  
    }}  
    >  
      리액트  
    </div>  
  );  
}  
  
export default App;
```

변수를 사용하여 인라인스타일 적용

- Import React from 'react';

```
function App() {  
  const name = '리액트';  
  const style = {  
    backgroundColor: 'black',  
    color: 'aqua',  
    fontSize: '48px',  
    padding: 16 // 단위 생략하면 px로 지정됨  
  };  
  return <div style={style}> {name} </div>;  
}  
  
export default App;
```

JSX 안에서 자바스크립트 표현 { }

```
import React from 'react';

Function App() {
  const name = '리액트';
  return (
    <>
      <h1> { name } 안녕</h1>
      <h2>잘 작동하나?</h2>
    </>
  )
}
```

cf) 변수 선언 키워드

블록 단위 범위(중복선언불가능) : let(바뀔수있는값), **const(바뀔수없는값)**
함수 범위(중복선언가능) : var

JSX 안에서 if문 대신 삼항조건연산자 사용

- If 문 사용할 수 없음. 대신 삼항조건연산자 사용

```
function App() {  
  const name = '리액트';  
  return (  
    <div>  
      { name === '리액트' ? (<h1>리액트입니다.</h1>) : (<h2>리액트가 아닙니다.</h2>) }  
    </div>  
  )  
}
```

```
function App() {  
  const name = '리액트';  
  return ( < div> { name === '리액트' ? <h1>리액트입니다.</h1> : null } </div> )  
}
```

```
function App() {  
  const name = '리액트';  
  return < div> { name === '리액트' && <h1>리액트입니다.</h1>    } </div>  
}
```

- 논리값 False 값은 화면에 출력되지 않으며, 0은 화면에 출력됨

조건부 렌더링

```
import React, { Component } from 'react';  
class App extends Component {  
  render() {  
    return (  
      <div>  
        { 1 + 1 === 2 ? (<div>맞아요!</div>) : (<div>틀려요!</div>) }  
      </div>  
    );  
  }  
}  
  
export default App;
```

조건부 렌더링

```
import React, { Component } from 'react';  
class App extends Component {  
  render() {  
    return (  
      <div>  
        { 1 + 1 === 2 && (<div>맞아요!</div>) }  
      </div>  
    );  
  }  
}  
  
export default App;
```

리액트에서 DOM 요소에 외부스타일 적용

- Import React from 'react';

import './App.css';

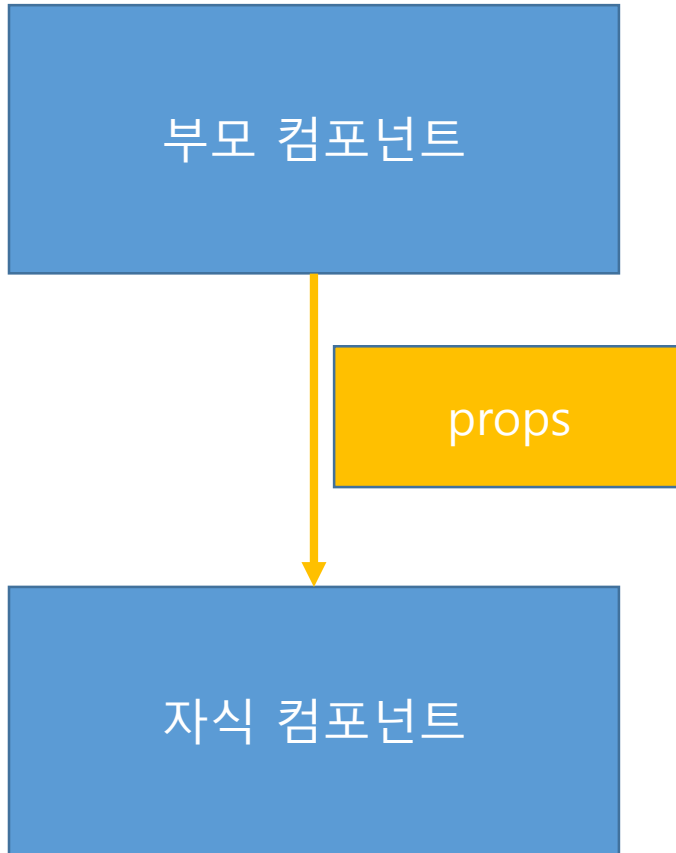
```
function App() {  
  const name = '리액트';  
  return (  
    <div className="react">  
      {name}  
    </div>  
  );  
}  
  
export default App;
```

```
.react {  
  background:aqua;  
  color:black;  
  font-size:48px;  
  font-weight:bold;  
  padding:16px;  
}
```

className 대신에 class 속성을 설정해도 스타일은 적용되기는 하지만 console 탭에서 확인해보면 경고메시지가 출력되므로 className만 사용할 것

컴포넌트 속성 설정하기(props) : 컴포넌트를 외부에서 조작할 때 사용함

- props : properties 의 줄임말로 상위 컴포넌트에서 하위 컴포넌트에 데이터를 전달할때 사용하는 속성, 자식 컴포넌트에 **this.props.속성명**으로 값을 렌더링함



```
return ( <MyName name="리액트" /> );
```

```
return (  
  <div>  
    안녕하세요! 제 이름은 {this.props.name}입니다.  
  </div>  
);
```

클래스형 컴포넌트에서 props 사용 예제

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

class Myname extends Component {
  render() {
    return (
      <div>
        안녕하세요! 이름은 {this.props.name}입니다.<br>
        좋아하는 숫자는 {this.props.favoriteNumber} 입니
        다.
      </div>
    );
  }
}

export default Myname;
```

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

export default function App() {
  return <MyName
    name='김은영'
    favoriteNumber=3
  />;
}

export default App;
```

함수형 컴포넌트에서 props 사용 예제

```
import React from "react";
const MyName = ({name, favoriteNumber}) => {
  return (
    <div>
      안녕하세요! 이름은 {name}입니다.<br>
      좋아하는 숫자는 {favoriteNumber} 입니다.
    </div>
  );
}
export default Myname;
```

초기 마운트 속도가 클래스형 컴포넌트에 비해 미세하게 빠름
불필요한 기능이 없기 때문에 메모리 자원도 덜 사용

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

export default function App() {
  return <MyName
    name='김은영'
    favoriteNumber='3'
  />;
}
export default App;
```

함수형 컴포넌트에서 props 사용 예제

```
import React from "react";
const MyName = props => {
  const { name, favoriteNumber } = props
  return (
    <div>
      안녕하세요! 이름은 {name}입니다.<br>
      좋아하는 숫자는 {favoriteNumber} 입니다.
    </div>
  );
}
export default Myname;
```

비구조화 할당 문법 사용
(방금 사용한 객체에서 값을 추출함)

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

export default function App() {
  return <MyName
    name='김은영'
    favoriteNumber='3'
  />;
}
export default App;
```

props 기본값 설정 : defaultProps(함수형)

```
import React from "react";
const MyName = ({name, favoriteNumber}) => {
  return (
    <div>
      안녕하세요! 이름은 {name}입니다.<br>
      좋아하는 숫자는 {favoriteNumber} 입니다.
    </div>
  )
};

MyName.defaultProps = {
  name: "기본이름",
  favoriteNumber: '기본숫자 '
};

export default MyName;
```

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

export default function App() {
  return <MyName />;
}

export default App;
```

React Developer Tools 크롬 확장프로그램 설치

props 기본값 설정 : defaultProps(클래스형)

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

class Myname extends Component {
  static defaultProps = {
  name: "홍길동",
  favoriteNumber:"100" }
  render() {
    return (
      <div>
        안녕하세요! 이름은 {this.props.name}입니다.<br>
        좋아하는 숫자는 {this.props.favoriteNumber} 입니다.
      </div>
    );
  }
}

export default Myname;
```

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

export default function App() {
  return <MyName />;
}

export default App;
```

프로젝트명 : idiya



꿀호떡



아이스크림 호떡



수플레 치즈 케이크



데블스 초코 케이크



햄앤치즈샌드위치



떠먹는 티라미수



떠먹는 롤케이크(플레인)



떠먹는 롤케이크(초코)

```
<div>
  <h1> 이디야베이커리 </h1>
  <ul>
    <li>
      <img src="" alt="">
      <span>꿀호떡</span>
    </li>
  </ul>
</div>
```

App.js(부모)

Menu.js(자식)

부모컴포넌트에 있는 데이터를 자식컴포넌트한테 전달
해서 리스트를 구성하기 위하여,

데이터를 배열로 저장하여 map() 으로 접근하고
필요한 값을 추출하여, 자식한테 전달할 props 설정함

배열 메소드로 데이터 추가, 삭제, 수정하기

.concat() : 기존 배열에 새로운 값을 추가해서 새로운 배열을 반환

```
var alphas = [ ' a ' , ' b ' , ' c ' ]  
var result = alphas.concat('d')  
console.log(result) // 결과: ['a', 'b', 'c', 'd']
```

...(스프레드) 연산자(기존배열복사)로 추가하기

```
var alphas = ['a', 'b', 'c']  
var result = [...alphas, 'd']  
console.log(result) // ['a', 'b', 'c', 'd']
```

.filter() : 기존 배열에서 조건에 맞는 값만 필터링해서 새로운 배열을 반환

```
var nums = [1, 2, 3, 4, 5]  
var result = nums.filter( n => n > 3 )  
console.log(result) // 결과: [4, 5]
```

```
var nums = [1, 2, 3, 4, 5]  
var result = nums.filter( n => n !== 3 )  
console.log(result) // 결과: [1, 2, 4, 5]
```

.map() : 기존 배열을 순차적으로 접근해서 수정하고 새로운 배열을 반환

```
var nums = [1, 2, 3, 4, 5]  
var result = nums.map( n => n*n ) // 화살표함수 사용  
console.log(result) // 결과: [1, 4, 9, 16, 25]
```

...(스프레드) 연산자 활용하여 수정하기

```
var objects = { a:1, b:1, c:3 }  
var result = { ...objects, b:2 }  
console.log(result) // {a:1, b:2, c:3}
```

slice(), filter() 참고 https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/slice

map() 참고 https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map

배열의 불변성을 지키면서 데이터 제거

```
const numbers = [1, 2, 3, 4, 5]
numbers.slice(0, 2) // 1, 2
numbers.slice(1, 3) // 2, 3
numbers.slice(3, 5) // 4, 5
numbers.slice(0, 2).concat(numbers.slice(3,5)) // 1, 2, 4, 5
[
  ...numbers.slice(0,2),
  10,
  // 1, 2, 10, 4, 5
  ...numbers.slice(3,5)
]
```

배열의 불변성을 지키면서 데이터 제거

```
const numbers = [1, 2, 3, 4, 5];  
numbers.filter( n => n>3 );           // 4, 5  
numbers.filter( n => n !== 3 )        // 1, 2, 4, 5
```

3을 9로 변경하기

```
[    ...numbers.slice(0,2),  
    9,                               // 1, 2, 9, 4, 5  
    ...numbers.slice(3,5)  
]
```

```
numbers.map( n => {  
    if (n===3) {  
        return 9                       // 1, 2, 9, 4, 5  
    }  
    return n  
} )
```

.SCSS 파일 제작하기

```
<form className="TodoInsert">
  <input placeholder="할 일을 입력하세요" />
  <button type="submit">추가</button>
</form>
```

.CSS

```
.TodoInsert { background: #495057;
  display: flex; flex: 1 }
input { background: none; outline: none;
  border: none; padding: 0.5rem;
  font-size: 1.125rem; line-height: 1.5;
  color: white; }
input::placeholder { color: #dee2e6; }
button { background: none; outline: none;
  border: none; background: #868e96;
  color: white; padding-left: 1rem;
  padding-right: 1rem; font-size: 1.5rem;
  display: flex; align-items: center;
  cursor: pointer;
  transition: 0.1s background ease-in; }
button:hover { background: #adb5bd; }
```

.scss 파일을 사용할 때는 node-sass 라이브러리 추가함
yarn add node-sass
import './*.scss';
버전 에러 뜨면 npm uninstall node-sass 언인스톨한 후
npm install node-sass@4.14.1

.SCSS

```
.TodoInsert { background: #495057; display: flex; flex: 1
  input { background: none; outline: none;
    border: none; padding: 0.5rem;
    font-size: 1.125rem; line-height: 1.5;
    color: white;
    &::placeholder { color: #dee2e6; }
  }
  button { background: none; outline: none;
    border: none; background: #868e96; color: white;
    padding-left: 1rem; padding-right: 1rem;
    font-size: 1.5rem; display: flex; align-items: center;
    cursor: pointer; transition: 0.1s background ease-in;
    &:hover { background: #adb5bd; }
  }
}
```

CSS 클래스를 조건부로 설정하기

```
<div className={cn('checkbox', { checked } )}>
  { checked ? 참 : 거짓 }
  <div>{text}</div>
</div>
```

```
.checkbox { cursor: pointer; flex: 1;
display: flex; align-items: center;
svg { font-size: 1.5rem; }
.text { margin-left: 0.5rem; flex: 1; }
```

// 체크되었을 때 보여줄 스타일

```
&.checked {
  svg { color: #22b8cf; }
  .text { color: #adb5bd; text-decoration: line-through; }
}
```

클래스명을 여러개 사용하거나 조건부로 설정할때
는 **classnames** 라이브러리 추가한 후 import 함
yarn add classnames
import cn from 'classnames';

checked 값이 true 이면 <MdCheckBox />



false 이면 <MdCheckBoxOutlineBlank />



반응형 스타일 라이브러리 설치

yarn add include-media

TodoTemplate.scss 파일 수정

```
@import '~include-media/dist/include-media';

.TODOTemplate {
  @include media('<=1239px') {      1239 이하 스타일추가    }
}
```

리액트 배포하기

yarn build

build 폴더 안에 있는 index.html 파일을 schedule.html 파일로 수정함

static 폴더와 schedule.html 파일을 현재 진행중인 프로젝트의 루트폴더로 옮김

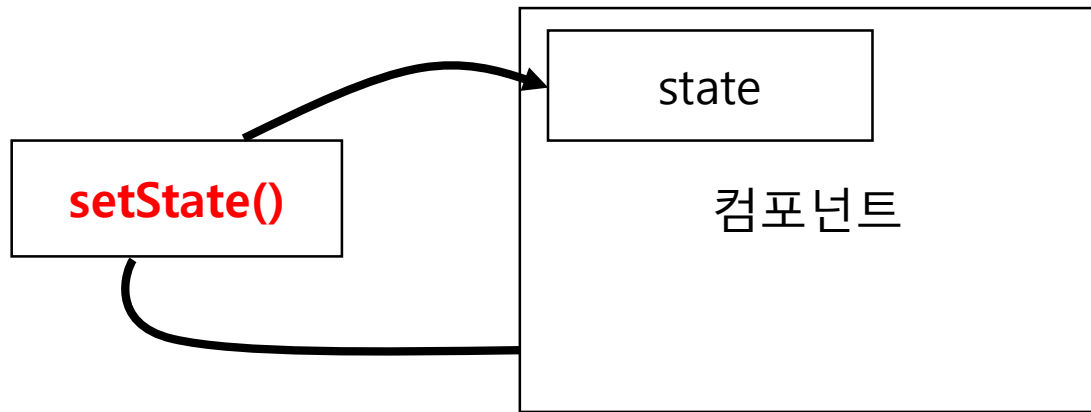
파일 경로 중 "/static~" 에 "static~" / 제거함

```
<link href="static/css/main.407ca8cf.chunk.css" rel="stylesheet" />
```

```
<script src="static/js/2.ce977c75.chunk.js"> </script>
```

```
<script src="static/js/main.6b3797ba.chunk.js"> </script>
```

state : 컴포넌트 내부에서 바뀔 수 있는 값



```
class Counter extends Component {
  state = {
    number: 0
  };
  render() {
    return (
      <div>
        <h1>카운터</h1>
        <div>값: {this.state.number}</div>
      </div>
    );
  }
}
```

- state는 자신 내부에서 변경할 수 있다.
- 변경할 때는 언제나 setState라는 함수를 사용한다.

```
class Counter extends Component {
  state = {
    number: 0
  };
  render() {
    const {number} = this.state;
    return (
      <div>
        <h1>카운터</h1>
        <div>값: {number}</div>
      </div>
    );
  }
}
```

화살표 함수 사용하여 이벤트 등록하기

```
handleIncrease = () => {  
  this.setState({  
    number: this.state.number + 1  
  });  
}
```

```
handleDecrease = () => {  
  this.setState({  
    number: this.state.number - 1  
  });  
}
```

```
handleIncrease = function () {  
  this.setState({  
    number: this.state.number + 1  
  });  
}
```

```
handleDecrease = function () {  
  this.setState({  
    number: this.state.number - 1  
  });  
}
```

이벤트 연결하기

```
<button onClick={this.handleIncrease}>  
+  
</button>  
<button onClick={this.handleDecrease}>  
-  
</button>
```

```
<button onclick="handleIncrease()">  
+  
</button>  
<button onclick="handleDecrease()">  
+  
</button>
```

(리액트에서 이벤트 함수를 설정할때 주의사항)

이벤트이름 설정 할 때 camelCase 로 설정함 : onClick, onMouseDown, onChange

useState 사용하기(함수형 컴포넌트)

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

```
const Say = () => {
```

```
  const [message, setMessage] = useState("");
```

```
  const onClickEnter = () => setMessage("안녕하세요");
```

```
  const onClickLeave = () => setMessage("안녕히 가세요!");
```

```
  const [color, setColor] = useState("black");
```

```
  return (
```

```
    <div>
```

```
      <button onClick={onClickEnter}>입장</button>
```

```
      <button onClick={onClickLeave}>퇴장</button>
```

```
      <h1 style={{ color }}>{message}</h1>
```

```
      <button style={{ color: "red" }} onClick={() => setColor("red")}>빨간색</button>
```

```
      <button style={{ color: "green" }} onClick={() => setColor("green")}>초록색
```

```
    </button>
```

```
      <button style={{ color: "blue" }} onClick={() => setColor("blue")}>파란색
```

```
    </button>
```

```
  </div>
```

```
);
```

```
};
```

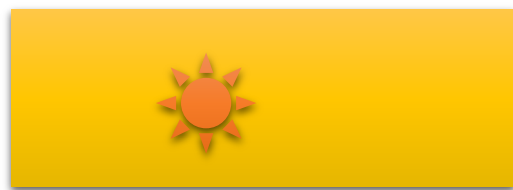
```
export default Say;
```

useState 함수의 인자에는 상태의 초기값 설정
한 컴포넌트에서 useState 여러 번 사용가능

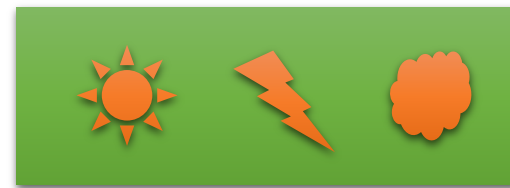
- 함수를 호출하면 배열이 반환됨.
- 배열 첫번째 원소는 현재상태,
- 배열 두번째 원소는 상태를 바꾸어주는 함수

API(Application Programing Interface)

- 인터페이스
 - 물건을 조작하기 위한 디자인과 작동 방식
 - 키보드, 마우스, 리모컨처럼 물리적인 형태를 갖추고 동작을 인식할 수 있는 방식까지 포함됨
 - 사람을 위한 인터페이스
- API는 한 프로그램이 다른 프로그램을 이용할 때 쓰는 인터페이스로 프로그램간의 커뮤니케이션을 담당
 - My 프로그램



기상청 날씨 정보 프로그램



axios : 웹서버와의 비동기통신을 위한 라이브러리

설치 : `npm i axios`

사용 : `async () => {`

▶ `const { data: { data: { movies } } } = await axios.get("API");`

리액트 배포하기

yarn build

build 폴더 안에 있는 index.html 파일을 schedule.html 파일로 수정함

static 폴더와 schedule.html 파일을 현재 진행중인 프로젝트의 루트폴더로 옮김

파일 경로 중 `"/static~"` 에 `"static~"` / 제거함

```
<link href="static/css/main.407ca8cf.chunk.css" rel="stylesheet" />
```

```
<script src="static/js/2.ce977c75.chunk.js"> </script>
```

```
<script src="static/js/main.6b3797ba.chunk.js"> </script>
```

비구조화 할당 : 배열이나 객체의 값을 새로운 변수에 할당

```
const 객체명 = { 키1 : 값1, 키2 : 값2, 키3 : 값3, ..... }
```

```
const { 키1, 키2, 키3 } = 객체명
```

```
const 배열명 = [ 값1, 값2, 값3, .... ]
```

```
const [변수1, 변수2, ...rest] = 배열명
```

```
const kim = {a:1, b:2, c:3}
```

```
const { a, c } = kim
```

```
console.log(a, c) // 1, 3 출력
```

```
const park = [ 1, 2, 3, 4, 5, 6 ]
```

```
const [d, , e, , ...rest] = park
```

```
console.log(d) // 1 출력
```

```
console.log(e) // 3 출력
```

```
console.log(...rest) // 5, 6 출력
```

컴포넌트 라이프사이클 메서드(생성과 파괴)

- mounting(생성), unmounting(파괴 : state를 통해 값이 변경됐거나, 페이지가 교체될때)
- constructor() : 마운트될때 가장 먼저 호출되는 함수
constructor(props) {
 super(props);
}
- componentDidUpdate() : setState()를 수행하고 render() 후 호출되는 함수
render() -> setState() -> render() -> componentDidUpdate() -> setState() -> render()
-> componentDidUpdate()
- componentWillUnmount() : 언마운트될 때 호출되는 함수
render() -> componentWillUnmount()

prop-types 를 통한 props 검증

- 설치 : `npm i prop-types`
 - `import PropTypes from "prop-types";`
 - 함수명.propTypes = { // 데이터타입이 틀리거나 필수요소인데
 props명: PropTypes.number.isRequired, // 누락되면 오류표시
};
- ```
function Food({name, rating}) {
 Food.propTypes = {
 name: PropTypes.string.isRequired,
 rating: PropTypes.number
 }
}
```