

1. 리액트 시작

작성일시: 2022년 5월 4일 오후 1:59 참고 도서: 리액트를 다루는 기술

1.1 왜 리액트인가?

- 자바스크립트

한때 자바스크립트는 웹 브라우저에서 간단한 연산과 시각적인 효과를 주는 단순한 스크립트 언어에 불과했다. 하지만, 현재는 영역을 확장하여 서버 사이드 뿐만 아니라 모바일, 데스크탑, 애플리케이션에서도 엄청난 활약을 한다.

- 프레임 워크

- 3대 프론트엔드 + 급 부상 중인 프레임 워크

1. Angular
2. Vue.js
3. React.js
4. 급 부상 중인 프레임 워크 ⇒ Svelte.js
- 5.

이 프레임 워크들은 MVC(Model-View-Controller)아키텍처, MVVM(Model-View Model)아키텍처를 사용한다. ⇒ 백엔드에서 주로 사용된다.

1.1.1 리액트 이해

- 리액트란?

1. **자바스크립트 라이브러리**로 사용자 인터페이스를 만드는데 사용한다. 오직 V(view)만 신경 쓰는 라이브러리이다.

- **MVC** 아키텍처의 구조

Model → data=(DataBase)

View → 화면

Controller→data + 화면

1. **컴포넌트**는 재사용이 가능한 API로, 특정 부분이 어떻게 생길지 정하는 선언체이다.
2. **렌더링**은 사용자 화면에 뷰(HTML DOM)을 보여주는 것이다.

1.1.1.1 초기 렌더링(=기본동작 원리)

어떤 UI관련 프레임워크, 라이브러리를 사용하던지 맨 처음 어떻게 보일지를 정하는 초기 렌더링이 필요하다. 리액트에서는 render함수(함수형 component)가 있다.

```
render() {...}
```

1.1.1.2 조화과정

리액트 업데이트(화면갱신) 과정을 “조화과정(reconciliation)을 거친다” 라고 표현한다. 그 이유는 컴포넌트에서 데이터 변화가 있을 때, 변화에 따라 뷰(HTML DOM)가 변형되는 것처럼 보이지만, 새로운 요소를 갈아끼우기 때문이다. → 이 작업 또한 render함수가 맡아서 한다.

1.2 리액트의 특징

Virtual DOM

1.2.1.1 DOM이란?

1. DOM은 Document Object Model의 약자이다.
2. 객체로 문서 구조를 표현하는 방법으로 XML이나 HTML로 작성한다. → Tag에 의한 구조 표현
3. DOM은 동적 UI에 최적화 되어있지 않음으로 자바스크립트를 사용하여 동적으로 만들 수 있다.

- **DOM이 느려지는 이유는?**

웹 브라우저 단에서 DOM에 변화(수정)이 이뤄질 때마다, 웹 브라우저가 CSS 연산, 레이아웃 구성을 하면서 페이지를 리페인트 하는 과정에서 시간이 허비된다. 또한, 이벤트에서 성능을 다 걸어버리면 성능이 저하된다.

- **DOM이 느려지는 해결방법?**

DOM을 최소한으로 조작하여 작업을 처리하는 방법(useEffect 사용)으로 개선 할 수 있다. 또한, Virtual DOM 방식으로 DOM 업데이트를 추상화함으로 처리 횟수를 최소화하고 효율적으로 진행한다.

1.2.1.2 Virtual DOM

- 실제 DOM(브라우저에 노출되는 element)에 접근하여 조작하는 대신, 이를 추상화한 자바스크립트 객체(DOM을 메모리에 복사한 것)를 구성하여 사용한다.
 - 브라우저에 노출(표시)되고 있는 element ⇒ 수정이 될 때마다, 계속 새로 전부다 그려서 느려짐
 - DOM메모리 복사 ⇒ 메모리에서 수정을 하여 완성되면 복사된 것을 한 번에 보내서 한번만 그릴 수 있다.

1.2.2기타 특징

- 리액트는 오직 뷰(HTML DOM)만 담당한다. ⇒ 화면제어만 담당

1.3 작업환경

1.3.1 Node.js 와 npm

리액트와 Node.js는 직접적인 연관은 없지만, 프로젝트를 개발하는데 필요한 주요 도구들이 Node.js를 사용하기 때문에 설치.

- 개발을 하는데 사용되는 도구들
 1. ECMAScript 6(ES6) 2015년 업데이트한 자바스크립트 문법을 호환 시켜주는 ⇒ 바벨(Babel)
 2. 모듈화된 코드들을 한 파일로 합치고(번들링) 코드를 수정할 때마다 웹 브라우저를 리로딩 등을 해주는 ⇒ 웹팩(Webpack)

Node.js를 설치하면 Node.js 패키지 매니저 도구인 npm이 설치된다. ⇒ npm의 업그레이드된 도구가 yarn이다.

1.3.2 Yarn

yarn은 npm을 대체할 수 있는 도구로 npm보다 더 빠르며 효율적인 캐시 시스템과 기타 부가기능을 제공함.

```
$npm install -g yarn
```

1.3.3 create-react-app으로 프로젝트 생성하기

1. 터미널을 열고, 프로젝트를 만들고 싶은 디렉토리(경로)에서 다음 명령어 실행

```
$ yarn create react-app 프로젝트이름
```

1. 프로젝트 생성 완료 되었다면. 다음 명령어 실행하여 프로젝트 디렉터리로 이동 후, 리액트 전용 서버구동 명령어 입력하여 실행

```
$cd 프로젝트이름  
$yarn start
```

2. JSX

작성일시: 2022년 5월 4일 오후 1:59 참고 도서: 리엑트를 다루는 기술

2.1 코드 이해하기

src/App.js 파일 이해하기

```
/*src/App.js 기본 파일*/

import logo from './logo.svg';
import './App.css';
// import는 다른 js 파일을 불러와 사용할 수 있다. 일반 JS에는 없는 기능이다.
// 분할된 JS를 import를 이용하여 조립(번들러(웹팩)사용)을 한다. => 파일을 묶듯 연결을
해줌.

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

import

```
import logo from './logo.svg';
import './App.css';
// import는 다른 js 파일을 불러와 사용할 수 있다. 일반 JS에는 없는 기능이다.
// 분할된 JS를 import를 이용하여 조립(번들러(웹팩)사용)을 한다. => 파일을 묶듯 연결을
해줌.
```

웹팩의 로더(loader)기능

1. 로더는 여러가지 종류가 있다. ex) CSS파일, file-loader는 웹 폰트나 미디어 파일 등을 불러올 수 있다.
- 2. babel-loader는 자바스크립트 파일들을 불러오면서 최신 자바스크립트 문법들을 ES5문법으로 변환해준다.

작성(ES6) → 병합 → 변환(ES5) Node.js 웹팩 바벨 웹브라우저 ⇒ 문법 변환 진행 과정

함수 컴포넌트

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
```

2.2 JSX란?

JSX는 자바스크립트의 확장 문법이며, XML과 매우 비슷하게 생겼습니다. 코드가 번들링 되는 과정에서 바벨(babel)을 사용하여 일반 자바스크립트 형태의 코드로 변환된다.

```
/* JSX */
function App(){
  return(
    <div>
      Hello<b>react</b>
    </div> // 축약형 표현이다.
  );
}
```

```
/* 바벨을 통해 변환된 코드 */  
function App(){  
  return React.createElement("div", null, "Hello", React.createElement("b", null,  
    "react"));  
}
```

2.3 JSX의 장점

1. 보기 쉽고 익숙하다
2. 더욱 높은 활용도

src/index.js

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import './index.css';  
import App from './App';  
import reportWebVitals from './reportWebVitals';  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>  
);
```

2.4 JSX문법

2.4.1 감싸인 요소

컴포넌트에 여러 요소가 있다면 반드시 부모 요소 하나로 감싸야한다. 아닐경우, 에러가 발생함.

에러가 발생하는 코드

```
function App(){  
  return(  
    <h1>리액트 안녕!</h1>  
    <h2>잘 작동하니?</h2>  
  )  
};  
export default App;
```

올바른 코드 사용

```
function App(){
  return(
    <div>
      <h1>리액트 안녕!</h1>
      <h2>잘 작동되니?</h2>
    </div>
  )
};
export default App;
```


```
01_src_app > src > JS App.js > default
1  function App(){
2    return(
3      <div>
4        <h1>리액트 안녕!</h1>
5        <h2>잘 작동되니?</h2>
6      </div>
7    )
8  };
9  export default App;
```

리액트 안녕!

잘 작동되니?

```
import { Fragment } from 'react';

function App(){
  return(
    <Fragment>
      <h1>리액트 안녕!</h1>
      <h2>잘 작동하니?</h2>
    </Fragment>
  )
};
export default App;
```

01_src_app > src >  App.js > ...

```

1  import { Fragment } from 'react';
2
3  function App(){
4    return(
5      <Fragment>
6        <h1>리액트 안녕!</h1>
7        <h2>잘 작동하니?</h2>
8      </Fragment>
9    )
10 };
11 export default App;
12
13

```

리액트 안녕!

잘 작동하니?

```

function App(){
  return(
    <>
      <h1>리액트 안녕!</h1>
      <h2>너도 잘 작동하니?</h2>
    </>
  )
};
export default App;

```

01_src_app > src >  App.js >  App

```

1  function App(){
2    return(
3      <>
4        <h1>리액트 안녕!</h1>
5        <h2>너도 잘 작동하니?</h2>
6      </>
7    )
8  };
9  export default App;
10

```

리액트 안녕!

너도 잘 작동하니?

2.4.2 자바스크립트 표현

JSX 안에서는 자바스크립트 표현식을 쓸 수 있다.

```

function App(){
  const name = '리액트';

```



```

    return (
      <>
        <h1>{name}안녕!</h1>
        <h2>너는 잘 작동되니?</h2>
      </>
    );
  }
  export default App;

```

01_src_app > src > JS App.js > default

```

1  function App() {
2    const name = '리액트';
3    return (
4      <>
5        <h1>{name}안녕!</h1>
6        <h2>너는 잘 작동되니?</h2>
7      </>
8    );
9  }
10 export default App;

```

리액트안녕!

너는 잘 작동되니?

2.4.3 If문 대신 조건부 연산자

JSX 내부의 자바스크립트 표현식에서는 if문을 사용할 수 없습니다. 그럼으로 JSX 밖에서 if문을 사용하여 사전에 값을 설정하거나, {} 안에서 조건부 연산자(삼항연산자)를 사용하면 된다.

삼항연산자를 사용한 코드

```

function App() {
  const name = '리액트';
  return (
    <div>
      {name=== '리액트'? (
        <h1>안녕!</h1>
      ):(
        <h2>너는 잘 작동되니?</h2>
      )}
    </div>
  );
}
export default App;

```

```
01_src_app > src > JS App.js > [default]
1  function App() {
2    const name = '리액트';
3    return (
4      <div>
5        {name=== '리액트'? (
6          <h1>안녕!</h1>
7        ):(
8          <h2>너는 잘 작동되니?</h2>
9        )}
10     </div>
11   );
12 }
13 export default App;
```

안녕!

2.4.4 AND 연산자(&&)를 사용한 조건부 렌더링

특정 조건을 만족할 때 보여주고, 만족하지 않을 때 아예 아무것도 렌더링을 하는 않는 상황을 조건부 연산자로 구현할 수 있다.

조건부 연산자를 이용한 코드

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

```
01_src_app > src > JS App.js > App
1  function App() {
2    const name = '리액트';
3    return <div>{name=== '리액트'?<h1>리액트입니다.</h1> : null}</div>
4  }
5  export default App;
```

리액트입니다.

AND연산자를 이용한 같은 코드

```
function App() {
  const name = '리액트';
  return <div>{name=== '리액트'&&<h1>리액트입니다.</h1>}</div>;
  //&& 연산자는 조건이 참일때만 실행됨
}
export default App;
```

```
01_src_app > src > App.js > default
1 function App() {
2   const name = '리액트';
3   return <div>{name=== '리액트'&&<h1>리액트입니다.</h1>}</div>;
4   //&& 연산자는 조건이 참일때만 실행됨
5 }
6 export default App;
```

리액트입니다.

&& 연산자로 조건부 렌더링을 할 수있는 이유는 리액트에서 false를 렌더링 할 때 null과 마찬가지로 아무것도 나타나지 않기 때문이다. 여기서 주의할 점은 falsy한 값인 0은 예외적으로 화면에 나타난다.

```
function App() {
  const number = 0;
  return number && <div>내용</div>;
}
export default App;
```

```
01_src_app > src > App.js > App
1 function App() {
2   const number = 0;
3   return number && <div>내용</div>;
4 }
5 export default App;
```

0

2.4.5 undefined를 렌더링하지 않기

리액트 컴포넌트 함수에서 undefined만 반환하여 렌더링을 하는 상황이 되면 코드는 오류를 발생시킨다.

undefined로 발생하는 오류 코드

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

```
function App() {
  const name = undefined;
  return name; //함수가 undefined를 리턴하면 에러이다 -> 즉, 반드시 Tag를 리턴해줘야 한다.
}

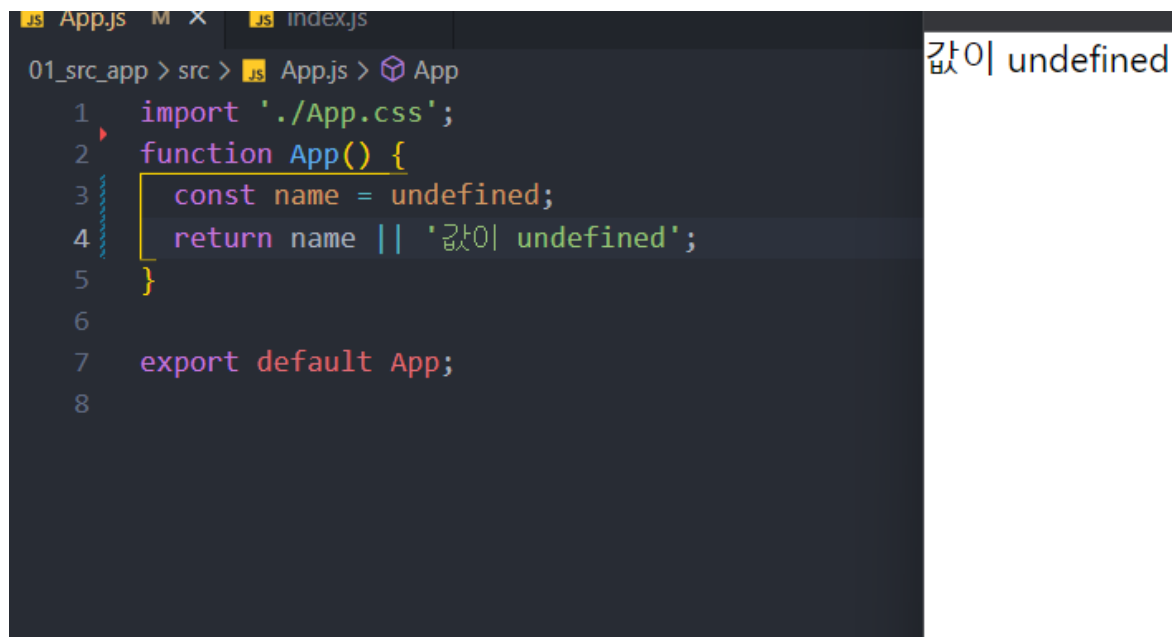
export default App;
```

OR(||) 연산자를 사용하여 오류 방지하기

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

function App() {
  const name = undefined;
  return name || '값이 undefined';
}

export default App;
```



name 값이 undefined일 때 보여줄 문구가 있을 때 사용하는 코드

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

function App() {
  const name = undefined;
  return <div>{name}</div>; //div로 출력함으로 에러가 나지 않고 name 값이 undefined가 됨.
}

export default App;
```

OR 조건 연산자를 이용한 코드

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

function App() {
  const name = undefined;
  return<div>{name || '리액트'}</div>;
}

export default App;
```

01_src_app > src > App.js > App

```
1 import './App.css';
2
3 function App() {
4   const name = undefined;
5   return<div>{name || '리액트'}</div>;
6 }
7
8 export default App;
```

리액트

2.4.6 인라인 스타일링

1. 리액트에서 DOM 요소에 스타일 적용시 문자열 형태가 아닌 객체 형태로 넣어줘야한다.
2. 이름은 - 문자를 없애고 카멜 표기법(camelCase)으로 작성해야한다. ex) backgroundColor

인라인 스타일링으로 작성된 코드

```
function App() {
  const name = '리액트';
  const style={
    // background-color는 backgroundColor와 같이 -가 사라지고 카멜 표기법으로 작성된
    다.
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: '48px', //font-size -> fontSize
    fontWeight: 'bold', // font-weight -> fontWeight
    padding: 16 //단위를 생략하면 px로 지정됩니다.
  };
  return<div style={style}>{name}</div>;
}

export default App;
```

```
01_src_app > src > Appjs > default
1 function App() {
2   const name = '리액트';
3   const style={
4     // background-color는 backgroundColor와 같이 -가 사라지고
5     backgroundColor: 'black',
6     color: 'aqua',
7     fontSize: '48px', //font-size -> fontSize
8     fontWeight: 'bold', // font-weight -> fontWeight
9     padding: 16 //단위를 생략하면 px로 지정됩니다.
10  };
11  return<div style={style}>{name}</div>;
12 }
13
14 export default App;
```

리액트

객체 선언 없이 바로 styled 값 지정

```
function App() {
  const name = '리액트';

  return<div style={{
    //객체 생성을 하지 않을 경우 바로 그 자리에 들어가기 때문에 이중괄호가 사용됨
    // background-color는 backgroundColor와 같이 -가 사라지고 카멜 표기법으로 작성된
    다.
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: '48px', //font-size -> fontSize
    fontWeight: 'bold', // font-weight -> fontWeight
    padding: 16 //단위를 생략하면 px로 지정됩니다.
  }}>{name}</div>;
}

export default App;
```

```
01_src_app > src > Appjs > App
1 function App() {
2   const name = '리액트';
3
4   return<div style={{
5     // background-color는 backgroundColor와 같이 -가 사라지고
6     backgroundColor: 'black',
7     color: 'aqua',
8     fontSize: '48px', //font-size -> fontSize
9     fontWeight: 'bold', // font-weight -> fontWeight
10    padding: 16 // 단위를 생략하면 px로 지정됩니다.
11    }}>{name}</div>;
12 }
13
14 export default App;
```

리액트

2.4.7 class 대신 className

- 일반 HTML에서 CSS 클래스를 사용할 경우 class라는 속성을 설정한다. 하지만, JSX에서는 class가 아닌 className으로 설정해줘야 한다.

JSX를 작성할 때 CSS 클래스를 설정하는 과정에서 className이 아닌 class값을 설정해도 스타일 적용은 되지만 개발자 도구에 오류가 발생됨 Warning: Invalid DOM property **class**. Did you mean **className**?

CSS 클래스 코드

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

className값을 지정한 코드

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

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

```
01_src_app > src > js App.js > [e] default
1  import './App.css';
2
3  function App(){
4    const name = '리액트';
5    return <div className="react">{name}</div>
6  }
7  export default App;
```

리액트

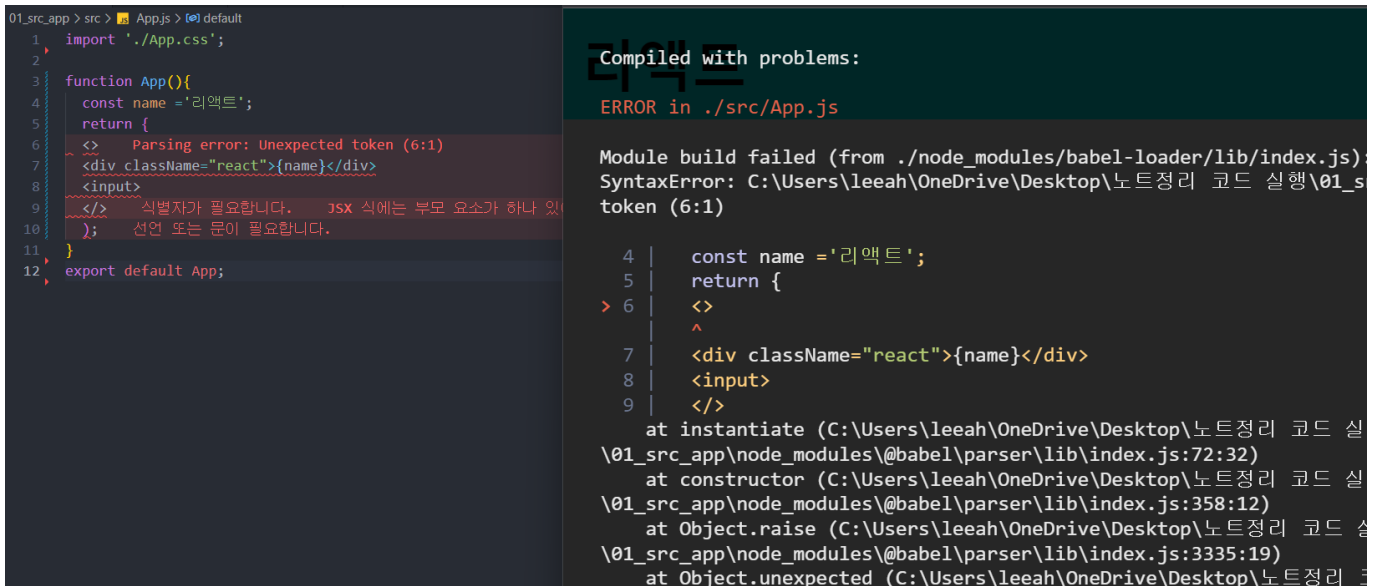
2.4.8 꼭 닫아야 하는 태그

HTML은 가끔 태그를 닫지 않는 상태로 작성해도 문제 없이 작동을 하지만, JSX에서는 태그를 닫지 않으면 오류가 발생 합니다. ex)

오류를 발생하는 단지 않은 태그

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

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



오류 해결을 위한 input 태그

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

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



```

01_src_app > src > App.js > ...
1  import './App.css';
2
3  function App(){
4      const name = '리액트';
5      return (
6          <>
7          <div className="react">{name}</div>
8          <input/>
9          </>
10         );
11     }
12     export default App;
13

```

리액트

태그 사이에 별도의 내용이 들어가지 않는 경우

```

import './App.css';

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

```

```

01_src_app > src > App.js > [default]
1  import './App.css';
2
3  function App(){
4      const name = '리액트';
5      return (
6          <>
7          <div className="react">{name}</div>
8          <input/>
9          </>
10         );
11     }
12     export default App;

```

리액트

2.4.9 주석

JSX 안에서 주석을 작성하는 방법은 일반 자바스크립트랑 다릅니다.

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

function App(){
  const name = '리액트';
  return (
    <>
    { /* 주석은 이렇게 작성합니다. 하지만 태그 밖에서는 사용할 수 없다 */}
    <div className="react">{name}</div>
    // 하지만 이런 주석이나
    /* 이런 주석은 페이지에 그대로 나타나게 됩니다. */
    <input/>
    </>
    // 하지만 태그 밖에서는 사용이 가능합니다.
    /* 이 주석 또한 가능합니다. */
  );
}
export default App;
```

```
01_src_app > src > Appjs > App
1  import './App.css';
2
3  function App(){
4    const name = '리액트';
5    return (
6      <>
7      { /* 주석은 이렇게 작성합니다. 하지만 태그 밖에서는 사용할 수 없다 */}
8      <div className="react">{name}</div>
9      // 하지만 이런 주석이나
10     /* 이런 주석은 페이지에 그대로 나타나게 됩니다. */
11     <input/>
12     </>
13     // 하지만 태그 밖에서는 사용이 가능합니다.
14     /* 이 주석 또한 가능합니다. */
15   );
16 }
17 export default App;
```

리액트

// 하지만 이런 주석이나 /* 이런 주석은 페이지에 그대로 나타나게 됩니다. */