



## 리액트 기초 개념 알아보기

# 1. 리액트 앱 실행 복습하기

# 1 리액트 앱 실행 복습하기

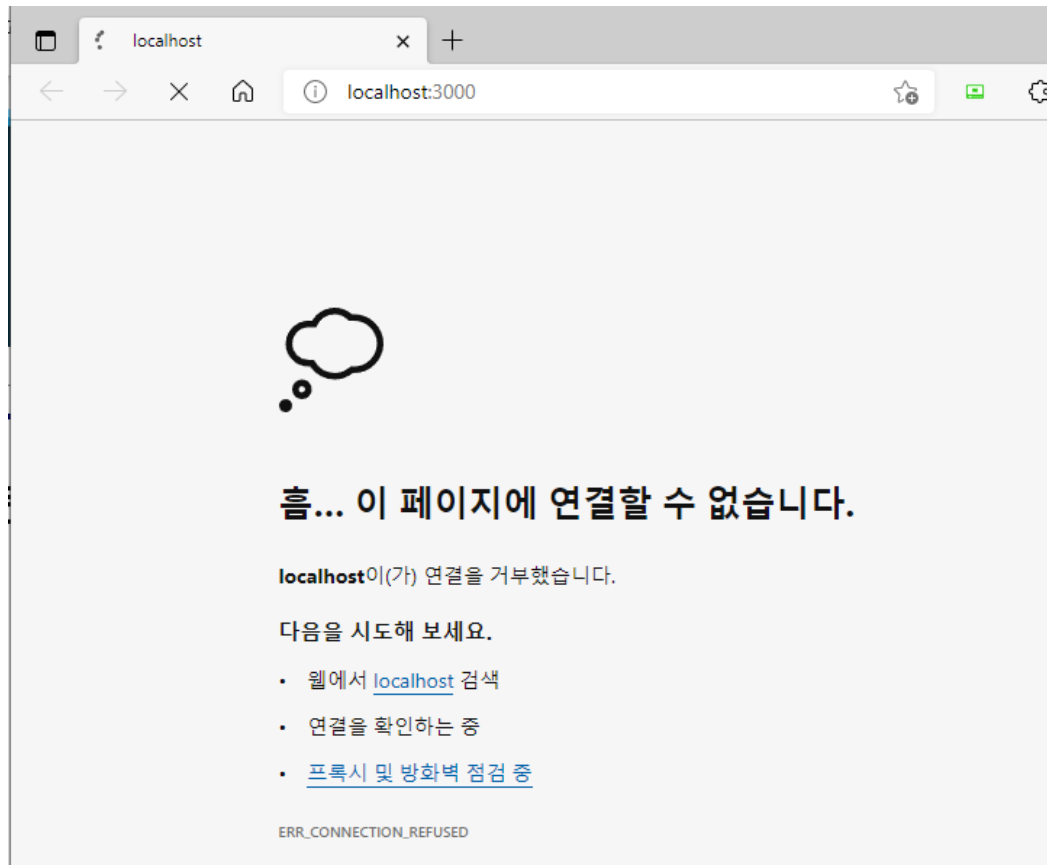
## ❖ 리액트 앱 다시 실행하기

- 리액트 앱은 터미널에서 리액트 앱의 루트폴더(/movie\_app\_2021)로 이동한 다음, `npm start`라는 명령어를 입력하면 자동으로 브라우저가 켜지면서 리액트 앱이 실행된다.
- 이때 터미널을 종료하면 안 된다.
- 터미널이 유지돼야 리액트 앱의 실행이 유지된다.
- 터미널을 종료하면 리액트 앱이 종료된다.
- 터미널을 종료하면 리액트 앱이 종료되니까 코드를 수정하고 저장해도 브라우저에 반영되지 않는다.

# 1 리액트 앱 실행 복습하기

## ❖ 터미널 종료하고 브라우저 새로 고침 하기

- 리액트 앱을 실행했던 터미널을 종료하고 브라우저를 새로고침 해보자.



# 1 리액트 앱 실행 복습하기

## ❖ 리액트 앱 다시 실행하고 코드 수정하기

- 리액트 앱을 다시 실행하고 **App.js** 파일을 열어 수정하고 저장하자.
- **App()** 함수의 반환값이 많아지면 소괄호로 감싸야 한다.
- 소괄호를 입력하고, **h1** 엘리먼트를 추가하고, 느낌표를 지우자.

1	
2	function App() {
3	return (
4	<div>
5	<h1>Hello</h1>
6	</div>
7	);
8	}
9	
10	export default App;

# 1 리액트 앱 실행 복습하기

## ❖ 리액트 앱 다시 실행하고 코드 수정하기

- 코드를 수정한 결과가 웹 브라우저에 잘 반영 되었는지 확인해 보자.
- 오류가 없고, Hello!!!!가 Hello로 바뀌면(h1 스타일로 바뀌고) 된다.
- 이 방법을 계속 반복하면서 프로젝트를 진행할 것이다.



Hello

## 2 리액트 기초 개념:컴포넌트

### ❖ App.js 파일로 컴포넌트의 정의 알아보기

- App.js 파일을 열고 App() 함수와 App() 함수가 반환하는 값을 보자.

1		App 컴포넌트를 정의했고
2	function App() {	
3	return (	
4	<div>	
5	<h1>Hello</h1>	App 컴포넌트는 HTML을 반환한다.
6	</div>	
7	);	
8	}	
9		
10	export default App;	

- App( ) 함수가 정의되어 있고, 이 함수가 <div><h1>Hello</h1></div>를 반환하고 있다.
- 이것이 바로 App 컴포넌트를 정의한 것이다.
- App() 함수가 반환한 HTML이 리액트 앱 화면에 그려지는 것이다.
- 이 과정을 자세히 알아보자.

## 2 리액트 기초 개념:컴포넌트

### ❖ index.js 파일로 컴포넌트의 사용 알아보기

- index.js 파일을 다시 한번 열어 보자.
- 그리고 `<App />`이라고 입력한 내용을 보자.

1	<code>import React from 'react';</code>	
2	<code>import ReactDOM from 'react-dom';</code>	
4	<code>import App from './App';</code>	
5		이게 바로 App 컴포넌트를 임포트하여 사용한 것이다.
6	<code>ReactDOM.render(</code>	
7	<code>&lt;App /&gt;,</code>	
8	<code>document.getElementById('root')</code>	
9	<code>);</code>	

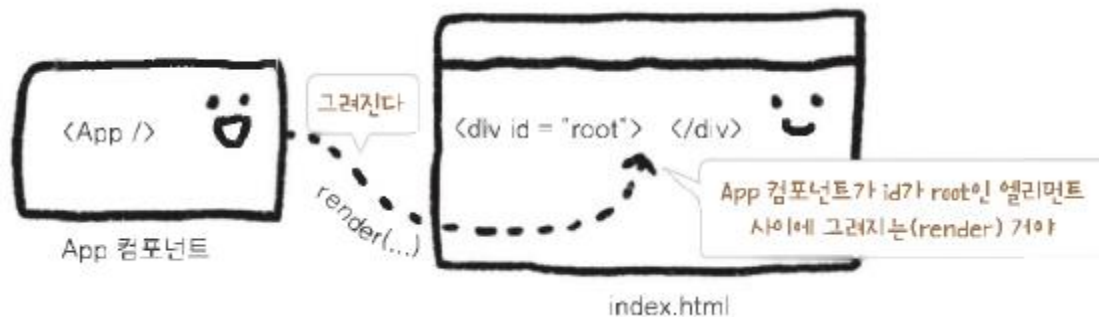
- App 컴포넌트 생김새가 마치 HTML 태그 같다.
- 하지만 HTML에는 저런 태그가 없다.
- 실제로 `<App />`은 HTML의 태그가 아니다.



## 2 리액트 기초 개념:컴포넌트

### ❖ index.js 파일로 컴포넌트의 사용 알아보기

- `<App />`을 `ReactDOM.render()` 함수의 첫 번째 인자로 전달하면 App 컴포넌트가 반환하는 것들을 화면에 그릴 수 있다.
- App 컴포넌트가 그려질 위치는 `ReactDOM.render()` 함수의 두 번째 인자로 전달하면 된다.
- 함수를 그대로 해석해서 'App 컴포넌트는 아이디가 root인 엘리먼트에 그려질 것이다.' 정도로 이해하면 된.
- 아이디가 root인 엘리먼트는 `index.html`에 있는데 이것도 곧 살펴볼 것이다.



## 2 리액트 기초 개념:컴포넌트

### ❖ index.js 파일로 컴포넌트의 사용 알아보기

- 리액트는 `<App />`과 같은 표시를 컴포넌트로 인식하고, 그 컴포넌트가 반환하는 값을 화면에 그려준다.
- 그래서 컴포넌트를 사용할 때 `<App />`가 아니라 `App`이라고 입력하면 오류가 발생한다.
- 여기서 집중할 내용은 리액트는 컴포넌트와 함께 동작하고, 리액트 앱은 모두 컴포넌트로 구성된다는 사실이다.
- 그러니 앞으로 영화 앱을 클론 코딩할 때는 컴포넌트를 많이 만들고 또 사용할 것이다.
- 그리고! 컴포넌트를 정의하거나 사용할 때는 그냥 모두 '컴포넌트'라고 부를 것이다.

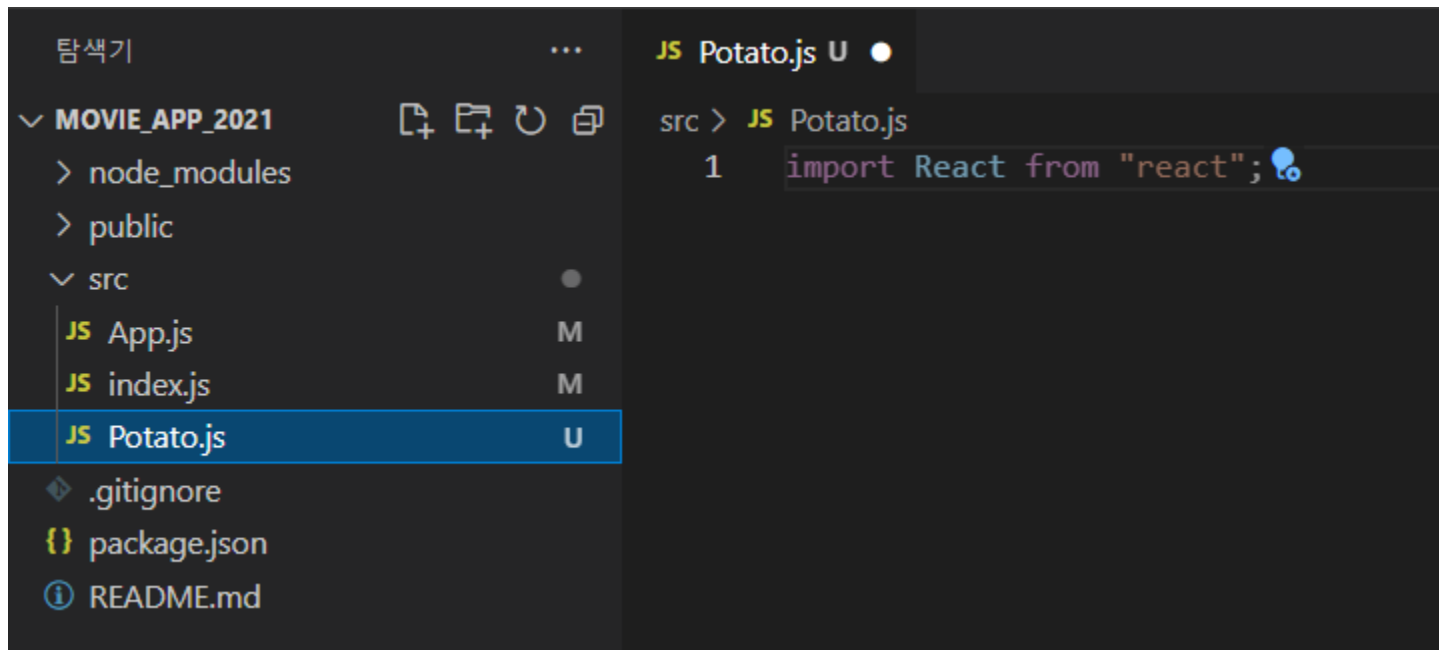
### 3 리액트 기초 개념: JSX

- ❖ 우리는 아직 컴포넌트를 만든 적이 없다.
- ❖ 컴포넌트는 어떻게 만들까?
- ❖ 컴포넌트는 자바스크립트와 HTML을 조합한 JSX라는 문법을 사용해서 만든다.
- ❖ 하지만 JSX는 새로운 문법은 아니다.
- ❖ 왜냐고?
- ❖ JSX는 HTML과 자바스크립트를 조합한 거니까.
- ❖ 그래서 JSX 문법은 따로 설명하지 않을 것이다.
- ❖ 컴포넌트를 만들다 보면 자연스럽게 JSX 문법을 어떻게 사용해야 하는지 알게 될 것이다.
- ❖ 그러면 당장 JSX로 컴포넌트를 만들어 보자.
- ❖ 이번에 만들 컴포넌트의 이름은 감자(Potato)이다.

### 3 리액트 기초 개념: JSX

#### ❖ Potato 컴포넌트 만들기

- src 폴더 안에 Potato.js라는 이름의 새 파일을 만든다.
- 파일 이름에서 첫 번째 글자는 반드시 대문자로 하고, 파일을 열고 맨 위에 `import React from 'react';`를 입력한다.



- 이 코드를 입력해야 리액트가 JSX를 이해할 수 있다.
- 이제 컴포넌트를 정의하면 된다.

# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 만들기

- 다음과 같이 `Potato()` 함수를 작성해 보자.

1	<code>import React from "react";</code>
2	
3	<code>function Potato() {</code>
4	
5	<code>}</code>

- `Potato` 컴포넌트의 기본 틀이 완성되었다.
- 컴포넌트를 작성할 때 중요한 규칙은 이름은 대문자로 시작해야 한다는 점이다.
- 이제 컴포넌트가 반환할 값을 입력하면 된다.
- 이 때 컴포넌트가 반환할 값은 **JSX**로 작성한다.

# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 만들기

- 다음을 참고하여 **Potato** 컴포넌트가 **JSX**를 반환하도록 만들자.

1	import React from "react";
2	
3	function Potato() {
4	return <h3>I love potato</h3>;
5	}

HTML이 아니라 JSX이다.

# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 만들기

- 이제 마지막이다.
- 마지막 줄에 **export default Potato;** 를 추가하자.

1	import React from "react";
2	
3	function Potato() {
4	return <h3>I love potato</h3>;
5	}
6	
7	export default Potato;

- **export default Potato;**를 추가하면 다른 파일에서 Potato 컴포넌트를 사용할 수 있다.
- 자! Potato 컴포넌트를 완성했다!
- 이제 완성한 컴포넌트를 사용해보자.

# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 사용하기

- `index.js` 파일을 열어서 **Potato** 컴포넌트를 어떻게 사용할지 잠시 코드를 살펴보자.

1	<code>import React from 'react';</code>
2	<code>import ReactDOM from 'react-dom';</code>
4	<code>import App from './App';</code>
5	
6	<code>ReactDOM.render(</code>
7	<code>&lt;App /&gt;,</code>
8	<code>document.getElementById('root')</code>
9	<code>);</code>

여기에 `<Potato />`로 추가해볼까?

- **App** 컴포넌트가 사용된 부분에 **Potato** 컴포넌트를 추가하면 될 것 같은데?
- 한번 시도해보자.

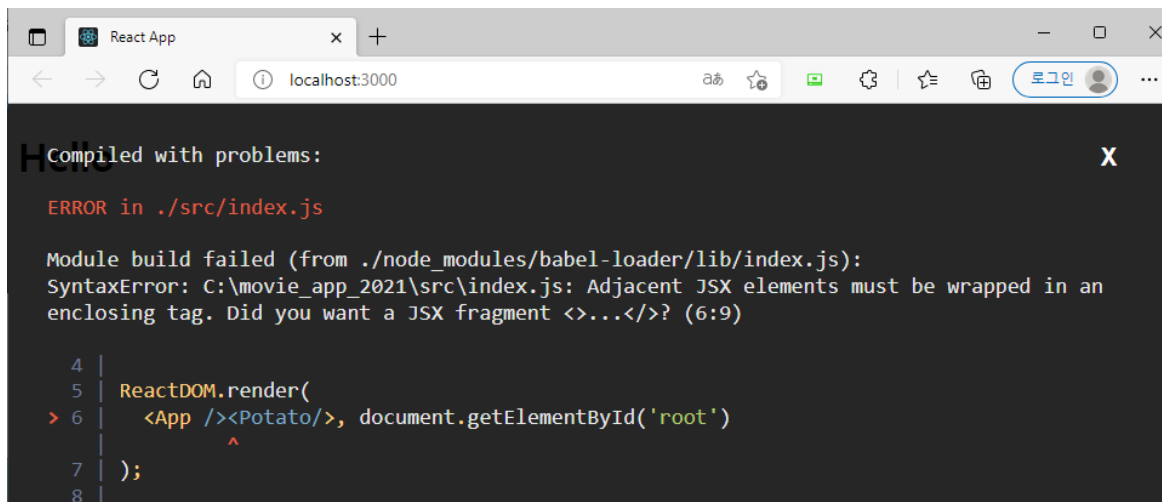


# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 사용하기

- 코드를 다음과 같이 수정하고 저장해 보자.

1	import React from 'react';
2	import ReactDOM from 'react-dom';
4	import App from './App';
5	
6	ReactDOM.render( 7     <App /> <Potato />, 8     document.getElementById('root') 9   );



```
Compiled with problems:
ERROR in ./src/index.js

Module build failed (from ./node_modules/babel-loader/lib/index.js):
SyntaxError: C:\movie_app_2021\src\index.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (6:9)

   4 |
   5 | ReactDOM.render(
>  6 |   <App /><Potato />, document.getElementById('root')
     |         ^
   7 | );
   8 |
```

# 3 리액트 기초 개념: JSX

## ❖ Potato 컴포넌트 사용하기

- 오류가 발생한다.
- 오류 메시지를 읽어 보면 '인접한JSX 요소는 반드시 하나의 태그로 감싸야 합니다.'라고 말하고 있다.
- 리액트는 최종으로 단 한 개의 컴포넌트를 그려야 하는데 지금은 두 개의 컴포넌트를 그리려 해서 오류가 발생한 것이다.
- 그러니 Potato 컴포넌트는 App 컴포넌트 안에 넣어야 한다.
- index.js 파일을 원래대로 돌려놓자.
- <Potato />를 삭제하자.
- 이제 Potato 컴포넌트를 App 컴포넌트에서 사용할 수 있도록 импорт 해보자.

### 3 리액트 기초 개념: JSX

#### ❖ App 컴포넌트에 Potato 컴포넌트 импорт하기

- 다음과 같이 수정하면 App 컴포넌트에 Potato 컴포넌트를 импорт할 수 있다.

./는 현재 파일이 있는 폴더라는 뜻이다.

1	import Potato from './Potato';
2	
3	function App() {
4	return (
5	<div>
6	<h1>Hello</h1>
7	</div>
8	);
9	}
10	
11	export default App;

- `import Potato from './Potato'` ; 에서 ./는 같은 경로(폴더)라는 것을 의미한다.

# 3 리액트 기초 개념: JSX

## ❖ App 컴포넌트에 Potato 컴포넌트 импорт하기

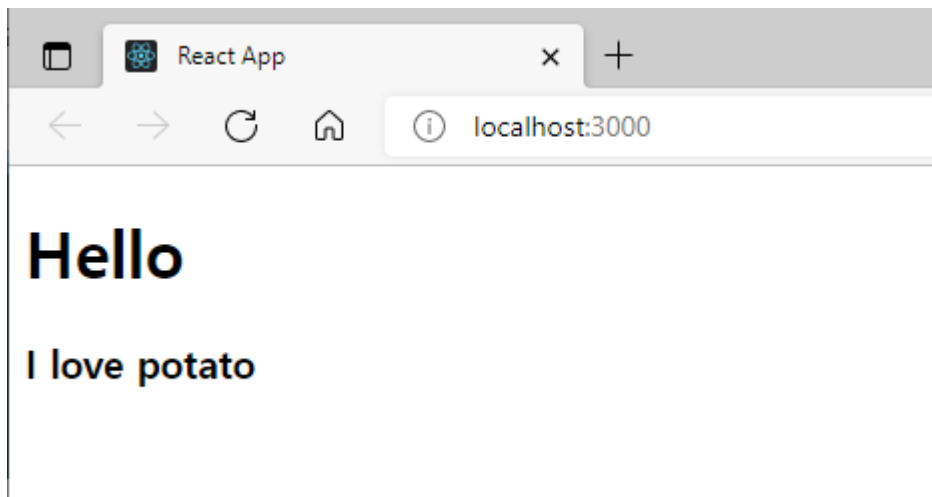
- **Potato.js**와 **App.js**는 같은 폴더에 있다.
- 그래서 **./**을 사용해서 **Potato** 컴포넌트를 импорт할 수 있다.
- 이제 우리는 **App** 컴포넌트에 **Potato** 컴포넌트를 포함시켜 사용할 것이다.
- **App** 컴포넌트가 반환할 값으로 **<Potato />**를 추가하고, 코드를 수정하고 저장한 다음 결과를 확인해 보자.

1	<code>import Potato from "./Potato";</code>
2	
3	<code>function App() {</code>
4	<code>  return (</code>
5	<code>    &lt;div&gt;</code>
6	<code>      &lt;h1&gt;Hello&lt;/h1&gt;</code>
7	<code>      &lt;Potato /&gt;</code>
8	<code>    &lt;/div&gt;</code>
9	<code>  );</code>
10	<code>}</code>
11	
12	<code>export default App;</code>

# 3 리액트 기초 개념: JSX

## ❖ App 컴포넌트에 Potato 컴포넌트 импорт하기

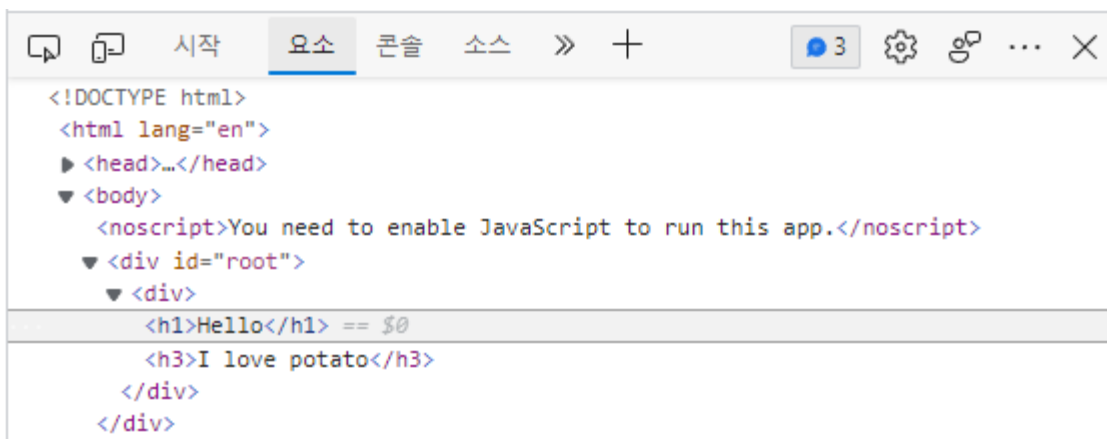
- App 컴포넌트에 JSX로 작성한 `<div><h1>Hello</h1></div>`와 `<Potato />`가 Hello, I love potato로 출력되었다.
- `<Potato />`는 어떻게 출력된 걸까?
- 개발자 도구에서 확인해 보자.



# 3 리액트 기초 개념: JSX

## ❖ 개발자 도구에서 Potato 컴포넌트 살펴보기

- 개발자 도구를 실행한 다음 [요소] 탭을 열어 코드를 살펴보자.
- **body** 엘리먼트 왼쪽에 있는 ► 표시가 보이는가?
- 그걸 누르면 아래와 같이 코드를 펼쳐 볼 수 있다.



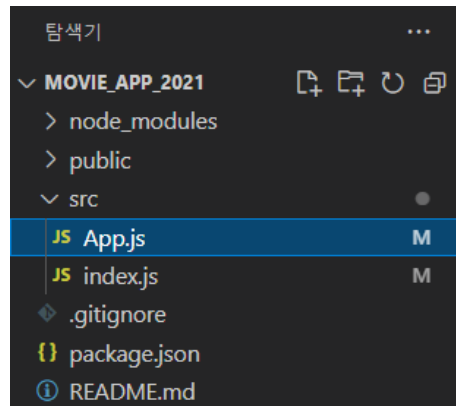
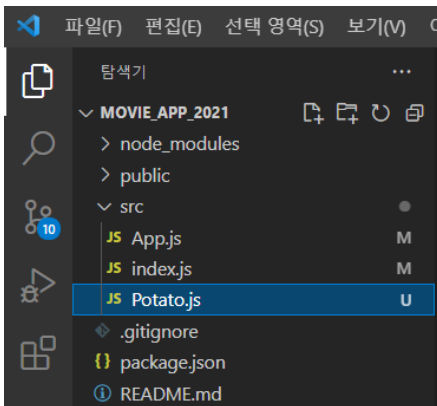
```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div>
        <h1>Hello</h1> == $0
        <h3>I love potato</h3>
      </div>
    </div>
```

- 리액트가 `<Potato />`를 해석해서 `<h3>I love potato</h3>`로 만들었다.
- 이게 컴포넌트와 JSX가 리액트에서 동작하는 방식이다.
- 컴포넌트는 JSX로 만들고, JSX는 자바스크립트와 HTML을 조합한 문법을 사용한다는 말이 이제 약간 이해가 되는가?
- 다음 실습으로 넘어가기 전에 파일을 정리하자.

### 3 리액트 기초 개념: JSX

#### ❖ Potato.js 파일 삭제하고 App.js 파일 수정하기

- Potato.js 파일을 삭제하고 App.js 파일에서 Potato 컴포넌트를 import하는 코드를 지우자.



1	<b>import Potato from "./Potato";</b>
2	
3	function App() {
4	return (
5	<div>
6	<h1>Hello</h1>
	...)

### 3 리액트 기초 개념: JSX

#### ❖ **Potato.js** 파일 삭제하고 **App.js** 파일 수정하기

- **App.js** 파일을 저장하면 오류 메시지가 나타난다.
- 오류 메시지의 내용은 '**App.js** 파일에 **Potato**라는 것이 정의되지 않아서 컴파일에 실패했다'는 내용이다.
- 아까 **Potato.js** 파일을 삭제하고, **App.js** 파일에서 **import Potato from './Potato';**를 지웠다.
- 하지만 여전히 **App.js** 파일에서는 **Potato** 컴포넌트를 사용하고 있다.
- 그러니 오류가 발생한 것이다.
- 이제 이 오류를 해결하기 위해 **App.js** 파일 안에 **Potato** 컴포넌트를 만든 다음 **Potato** 컴포넌트를 사용해 보자.



### 3 리액트 기초 개념: JSX

#### ❖ App 컴포넌트 안에 Potato 컴포넌트 만들기

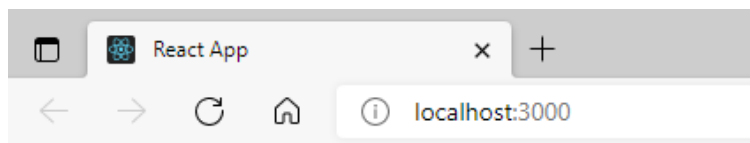
- 다음과 같이 App 컴포넌트를 수정해서 Potato 컴포넌트를 만들어 보자.
- 별거 없어 App.js 파일에 Potato()함수를 만들면 된다.

1	
2	<code>function Potato(){</code>
3	<code>  return &lt;h1&gt;I like potato&lt;/h1&gt;;</code>
4	<code>}</code>
5	
6	<code>function App() {</code>
7	<code>  return (</code>
8	<code>    &lt;div&gt;</code>
9	<code>      &lt;h1&gt;Hello&lt;/h1&gt;</code>
10	<code>      &lt;Potato /&gt;</code>
11	<code>    &lt;/div&gt;</code>
12	<code>  );</code>
13	<code>}</code>
14	
15	<code>export default App;</code>

### 3 리액트 기초 개념: JSX

#### ❖ App 컴포넌트 안에 Potato 컴포넌트 만들기

- 파일을 저장하면 리액트 앱이 다시 정상으로 작동한다.
- App.js 파일 안에 Potato 컴포넌트를 만들었고, Potato 컴포넌트를 App 컴포넌트 안에서 사용했다.
- 왜 App.js 파일에 Potato 컴포넌트를 포함시키는지 궁금할 것이다.
- 그냥 여러 파일을 이동하며 코드 작업을 하기 싫어서 이다.
- 2개의 파일을 이동하면서 강의하는 게 불편하기도 하고.
- 앞으로 짧은 코드로 작성할 수 있는 컴포넌트는 모두 App 컴포넌트 안에 작성할 것이다.
- 이제 파일이 준비 되었으니 다음 개념을 공부해 보자.



Hello

I like potato

## 4 리액트 기초 개념: props

- ❖ 이제 props라는 개념을 배울 차례이다.
- ❖ props는 간단히 말하자면 컴포넌트에서 컴포넌트로 전달하는 데이터를 말한다.
- ❖ 다들 함수의 매개변수라는 개념을 알고 있을 것이다.
- ❖ 매개변수를 이용하면 함수를 효율적으로 재사용할 수 있다.
- ❖ 컴포넌트의 props도 비슷하다.
- ❖ props를 사용하면 컴포넌트를 효율적으로 재사용할 수 있다.

## 4 리액트 기초 개념: props

### ❖ 컴포넌트 여러 개 사용해 보기

- 만약 우리가 만든 영화 앱에 영화 목록이 있고, 그 영화 목록을 컴포넌트로 표현한다고 해보자.
- 그런 상황을 가정하기 위해 **Potato** 컴포넌트의 이름을 **Movie**로 바꿔보자.

1		컴포넌트를 정의할 때의 이름과
2	function <b>Movie</b> () {	
3	return <h1>I like potato</h1>;	
4	}	
5		
6	function App() {	
7	return (	
8	<div>	
9	<h1>Hello</h1>	
10	< <b>Movie</b> />	컴포넌트를 사용할 때의 이름을 모두 바꿔야 한다.
11	</div>	
12	);	
13	}	
14		
15	export default App;	

## 4 리액트 기초 개념: props

### ❖ 컴포넌트 여러 개 사용해 보기

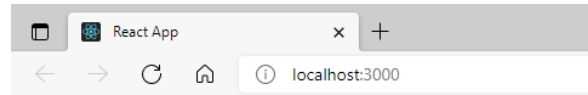
- **Movie** 컴포넌트 1 개가 영화 목록 1개라고 생각해 보자.
- 만약 영화 목록을 20개 그리려면 어떻게 해야 할까?
- **Movie** 컴포넌트를 여러 개 늘어 놓으면 될까?
- 다음과 같이 **Movie** 컴포넌트를 20개 복사해서 붙여 넣어 보자.

	...
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Movie />
11	<Movie />
	...
28	<Movie />
29	<Movie />
30	</div>
31	);
	...

## 4 리액트 기초 개념: props

### ❖ 컴포넌트 여러 개 사용해 보기

- 앱을 실행하면 **I like potato**가 20개 출력될 것이다.
- 그리고 이미 느꼈겠지만... 이런 코드를 작성하다 보면 좀 비효율적이라는 생각이 들 것이다.
- 왜냐하면 **Movie** 컴포넌트 20개를 손으로 직접 입력하고 있으니까.
- 또 20개의 **Movie** 컴포넌트가 20개나 되는데 출력하는 값이 모두 **I like potato**로 같다는 것도 문제이다.
- 컴포넌트가 서로 다른 값을 출력해야 영화 앱의 영화 목록을 구현할 수 있을 텐데 말이다.
- 그래서 컴포넌트로 데이터를 보내는 방법을 배워야 한다.
- 그 방법이 **props**이다.



Hello

I like potato

I like potato

I like potato

I like potato

## 4 리액트 기초 개념: props

### ❖ props로 컴포넌트에 데이터 전달하기

- 컴포넌트의 이름을 **Movie**에서 **Food**로 변경하자.
- 그리고 **Movie** 컴포넌트는 모두 삭제하자.

1	
2	function <b>Food</b> () {
3	return <h1>I like potato</h1>;
4	}
5	
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	< <b>Food</b> />
11	</div>
12	);
13	}
14	
15	export default App;

## 4 리액트 기초 개념: props

### ❖ props로 컴포넌트에 데이터 전달하기

- 아직 영화 데이터를 다루지 않으니까 잠시 음식을 주제로 리액트 앱을 만들어 보다.
- **props**는 컴포넌트로 데이터를 보낼 때 사용할 수 있다고 했다.
- 이제 **props**를 이용하여 **Food** 컴포넌트에 데이터를 보낼 것이다.
- `<Food />`를 `<Food fav="kimchi" />`로 수정해 보자.
- **fav** props의 값으로 "kimchi"를 추가하는 것이다.

	...
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Food fav= "kimchi" /> — fav는 favorite의 줄임말이다.
11	</div>
12	);
13	}
14	
15	export default App;



## 4 리액트 기초 개념: props

### ❖ props로 컴포넌트에 데이터 전달하기

- 이게 바로 props를 이용하여 Food 컴포넌트에 데이터를 보내는 방법이다.
- Food 컴포넌트에 사용한 props의 이름은 fav이고, fav에 " kimchi " 라는 값을 담아 Food 컴포넌트에 보낸 것이다.
- Props에는 불리언 값(true, false), 숫자, 배열과 같은 다양한 형태의 데이터를 담을 수 있다.
- 여기서 주의할 점은 'props에 있는 데이터는 문자열인 경우를 제외하면 모두 중괄호({})로 값을 감싸야 한다는 점'이다.
- 정말 그럴까?
- 예제 코드를 작성해 보자.

## 4 리액트 기초 개념: props

### ❖ Food 컴포넌트에 props 전달하기

- 다음과 같이 Food 컴포넌트에 something, papapapa props를 추가해보자.
- 그리고 수정한 파일을 저장한다.

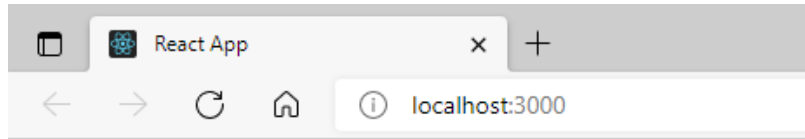
	...
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Food fav="kimchi" something={true} papapapa={['hello', 1, 2, 3, 4, true]} />
11	</div>
	...

- 리액트 앱 실행에 전혀 문제가 없다.
- 코드에서 보듯 문자열이 아닌 값인 true와 ['hello', 1, 2,...] 배열은 중괄호로 감싸 전달하고 있다.
- 이 사실을 기억하고 넘어가자.

## 4 리액트 기초 개념: props

### ❖ Food 컴포넌트에 props 전달하기

- 파일을 저장하여 리액트 앱을 실행해 보자.



Hello

I like potato

- 그러면 아마 아무런 변화가 없을 것이다.
  - kimch라는 값이 보이거나 하지 않을 것이다.
- 왜 그럴까?
- Food 컴포넌트에 props 보내기만 했을 뿐 아직 사용하지 않았기 때문이다.
- 그럼 Food 컴포넌트에서 props를 사용하려면 어떻게 해야 할까?

## 4 리액트 기초 개념: props

### ❖ props 사용하기

- 일단 Food 컴포넌트의 인자로 전달된 props를 출력해 보자.
- Food 컴포넌트의 인자인 props는 potato로 바뀌어도 된다.

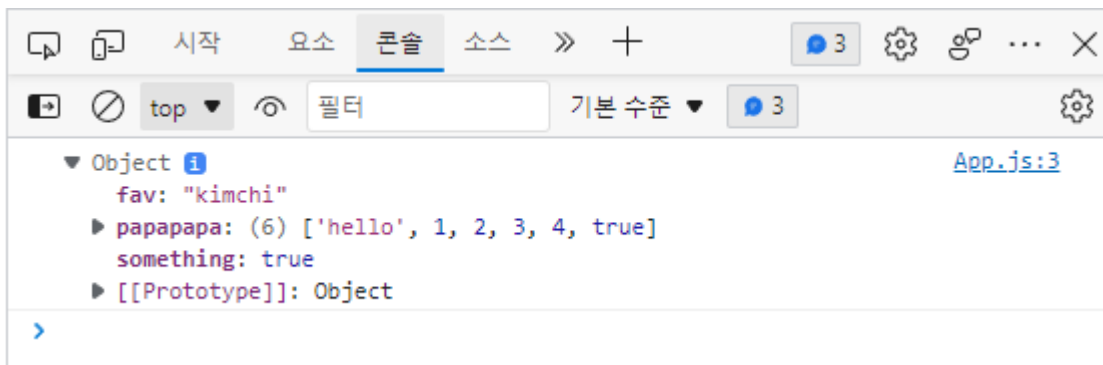
1	
2	function Food(props){
3	console.log(props)
	return <h1>I like potato</h1>;
4	}
5	
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Food fav="kimchi" something={true} papapapa={['hello', 1, 2, 3, 4, true]} />
	...

- 리액트 앱 화면은 여전히 아무런 변화가 없을 것이다.
- console.log() 함수는 개발자 도구의 [Console] 탭에만 영향을 주는 함수이기 때문이다.

## 4 리액트 기초 개념: props

### ❖ props 사용하기

- 개발자 도구를 실행해서 [Console] 탭을 눌러 보자.



## 4 리액트 기초 개념: props

### ❖ props 사용하기

- Food 컴포넌트에 전달한 props(fav, something, papapapa)를 속성으로 가지는 객체 (Object)가 출력되었다.
- 다음과 같은 과정으로 props가 전달된 것이다.

1		② 전달받은 props를 props라는 인자로 받아 출력
2	function Food(props){	
3	console.log(props)	
	return <h1>I like potato</h1>;	
4	}	
5		
6	function App() {	
7	return (	
8	<div>	① props에 있는 데이터를 객체로 변환하여 Food 컴포넌트(함수)에 전달
9	<h1>Hello</h1>	
10	<Food fav="kimchi" something={true} papapapa={['hello', 1, 2, 3, 4, true]}	
	/>	
	...	

- props를 사용하면 컴포넌트에 데이터를 쉽게 전달할 수 있다.
- 한 번만 더 실습해 보자.

## 4 리액트 기초 개념: props

### ❖ props 다시 한 번 사용하기

- 코드를 다음과 같이 수정 해보자.
- **something, papapapa props**는 사용하지 않을 거니까 지우면 된다.

1	
2	function Food(props){
3	console.log(props)
	return <h1>I like potato</h1>;
4	}
5	
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Food fav="kimchi" />
	...)

- 저장하면 콘솔에 {fav : "kimchi"} 만 출력될 것이다.
- 만약 문자열 " kimchi " 를 화면 그대로 출력하고 싶다면 어떻게 해야 할까?

## 4 리액트 기초 개념: props

### ❖ props 다시 한 번 사용하기

- Food 컴포넌트에 props에 있는 데이터 "kimchi"를 화면에 출력하려면 props.fav를 중괄호로 감싸서 사용하면 된다.

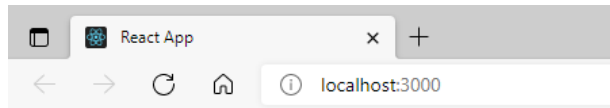
1	
2	function Food(props){
3	console.log(props)
	return <h1>I like {props.fav}</h1>;
4	}
5	
6	function App() {
7	return (
8	<div>
9	<h1>Hello</h1>
10	<Food fav="kimchi" />
	...



## 4 리액트 기초 개념: props

### ❖ props 다시 한 번 사용하기

- 객체에 있는 값을 사용하려면 점 연산자(.)를 사용한다.
- fav props의 값을 사용하려면 props.fav와 같이 점 연산자를 사용해야 한다.
- 리액트 앱을 실행하면 I like kimchi가 출력될 것이다.



Hello

I like kimchi

- 자! Food 컴포넌트에 props를 전달하는 방법도 알았으니 똑같은 Food 컴포넌트를 반복해서 사용하되, fav props의 값이 서로 다르게 코딩해 볼 것이다.
- 예를 들어, "ramen", "samgiopsal", "chukumf와 같은 데이터를 fav props에 담아 Food 컴포넌트에 전달할 수 있다.

## 4 리액트 기초 개념: props

### ❖ 구조 분해 할당으로 props 사용하기

- 자바스크립트 ES6의 문법 중 구조 분해 할당(destructuring-assignment)을 사용하면 점 연산자를 사용하지 않아도 된다.
- 다음은 Food 컴포넌트로 전달한 fav props를 Food 컴포넌트(함수)에서 { fav } = props;와 같은 방법으로 사용한 예이다.

<pre>function Food(props){   { fav } = props;   return &lt;h1&gt;I like {fav}&lt;/h1&gt;; }</pre>	<pre>function Food({ fav }){   return &lt;h1&gt;I like {fav}&lt;/h1&gt;; }</pre>
---	--

- props에 포함된 데이터의 개수가 적으면 점 연산자를 사용하여 props.fav와 같은 방법으로 사용해도 불편하지 않지만, props에 포함된 데이터의 개수가 많아지면 매번 props.fav와 같은 방법으로 사용하면 불편하다.
- 이런 경우 구조 분해 할당을 사용하면 편리하다.

## 4 리액트 기초 개념: props

### ❖ 여러 개의 컴포넌트에 props 사용하기

- 앞으로 props를 사용할 때는 대부분 구조 분해 할당을 사용할 것이다.
- Food 컴포넌트를 3개 추가하고 fav props의 값이 서로 다르도록 코드를 수정하자.

1		
2	function Food({fav}){	
3	return <h1>I like {fav}</h1>;	Hello
4	}	I like kimchi
5		
6	function App() {	I like ramen
7	return (	I like samgiopsal
8	<div>	
9	<h1>Hello</h1>	I like chukumi
10	<Food fav="kimchi" />	
11	<Food fav="ramen" />	
12	<Food fav="samgiopsal" />	
13	<Food fav="chukumi" />	
14	</div>	
	...	

## 4 리액트 기초 개념: props

### ❖ 여러 개의 컴포넌트에 props 사용하기

- 이번 액션에서는 Food 컴포넌트를 4개 사용해 각 컴포넌트에 전달한 fav props를 출력했다.
- 각각의 fav props에는 서로 다른 값이 들어 있으니까 같은 컴포넌트를 사용해도 서로 다른 문장이 출력된 것이다.
- 이걸 컴포넌트를 재사용한다고도 한다.
- 이 코드를 조금씩 발전시키면 영화 앱이 될 것이다.





**Thank You !**