FrontEnd 개발 언어의 흐름

jQuery 의 부흥

2014 년 처음 개발자가 되겠다고 마음을 먹었을 때, 처음 접하게 된 라이브러리는 jQuery 였다.

웹에서 작동하는 JavaScript 개념이 전혀 없어도 jQuery 와 함께라면 웹 화면을 만들 수 있었다.

처음 웹 개발을 시작한 주니어들에게 이해하기 쉬운 언어 규칙, 직관적인 방식으로

웹에 애니메이션 효과 혹은 css 효과를 줄 수 있는 매우 유용한 라이브러리였다.

jQuery 의 부흥과 더불어 트위터에서 jQuery 를 이용한 프런트 앤드 프레임워크인 Bootstrap 이 등장하게 되면서,

javascript 를 전혀 모르던 나도 부트스트랩과 jQuery 를 이용하여

그럴듯한 웹페이지를 만들 수 있었다.

프레임워크 3 대장 등장

업계에 들어오고 2 년 동안 나에겐 너무 편한 jQuery 를 난 여전히 사용하고 있었지만,

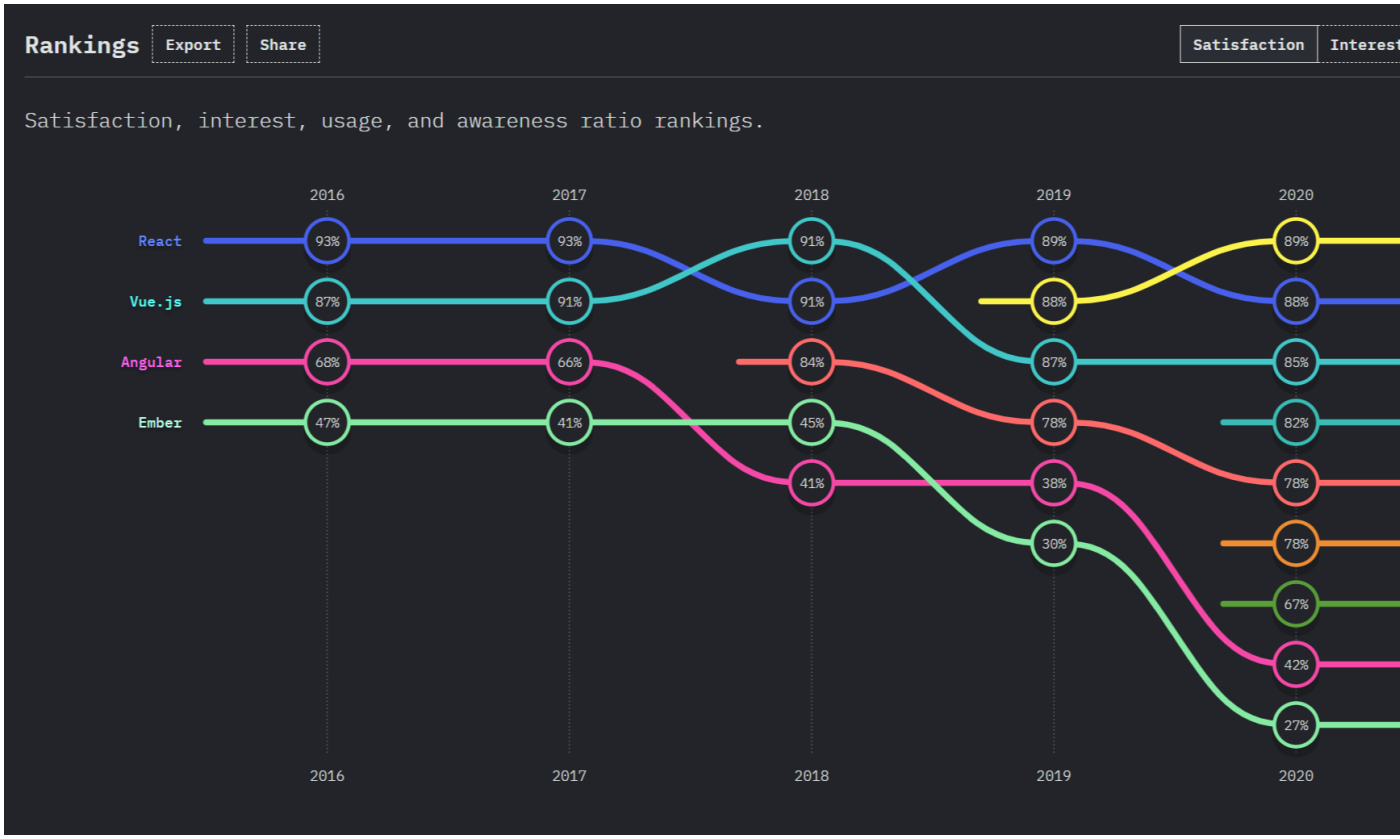
새로운 개발자 세계에서는 새로운 언어들이 등장하기 시작했다.

현재는 Angular, React, Vue.js 총 3 개가 대한민국에서

개발자들이 주로 사용하는 프런트 앤드 프레임워크로 자리 잡게 되었다.

프런트 앤드 개발자 사이에선 몇 년 전부터 끊임없이 어떠한 프레임워크가 좋은지에 대한 논쟁이 진행되었고,

2021 년이 된 지금 필자의 개인적인 생각으로는 대한민국 내에서는 **React 가 승자** 가 되었다.



state of Javascript Survey 2020

물론 현재 떠오르고 있는 **Svelte** 가 이후 어떠한 영향을 줄지는 아직 미지수다.

2021 년 Svelte 에 기대를 걸어 Svelte 를 주력으로 공부해볼까도 생각했지만,

이미 React 가 한국 프런트 앤드에 많은 영향을 미치고 있다고 판단해서

주력 언어인 Vue.js 정도까진 아니더라도 기본적인 React 공부를 시작해보려고 한다.

React 공부 방법

오늘부터 차근히 공부를 시작할 부분은 React 공식 홈페이지에서 제공해주는 자습서를 따라 하는 걸로 결정했다.

나의 컴퓨터에 코드를 직접 작성하여, React 의 자습서에서 만드는 틱택토 게임을 만들어 보려고 한다.

(틱택토가 무엇인가 봤더니 그냥 오목 같은 게임이었다.)

ko.reactjs.org/tutorial/tutorial.html

자습서: React 시작하기 – React

A JavaScript library for building user interfaces
ko.reactjs.org

React 란?

리액트는 사용자 인터페이스를 만들 위한 Javascript 라이브러리이며 처음 만든 곳은 페이스북이다.

React 는 **컴포넌트 기반**으로 이루어져 있다. 복잡한 UI 를 컴포넌트로 관리하여 돔과 별개로 상태 관리가 가능하다.

또한 vue 와 동일하게 데이터가 변경될 때마다 효율적으로 페이지가 갱신되고 랜더링 된다.

이러한 기능으로 좀 더 역동 적인 페이지를 만드는 것이 가능할 것으로 보인다.

가장 큰 장점은 리액트를 배울 경우 추후 **React Native** 를 이용하면 모바일 웹도 개발이 가능하니 일석 이조다.

React 프로젝트 생성하기

먼저, React 프로젝트 코드를 작성하기 위해서는 최신 버전의 Node.js 가 필요하다.

Node.js 가 없을 경우, 아래 링크를 통해서 Node.js 를 먼저 설치하도록 하자.

들어갈 경우, 안정성이 보장된 LTS 버전을 다운로드한 뒤 설치하면 된다.

nodejs.org/en/

Node.js

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
nodejs.org

노드 설치 후 cmd 창에 node -v 명령어를 입력 시 현재 설치된 node 버전이 보인다면,

설치가 완료된 것이다.

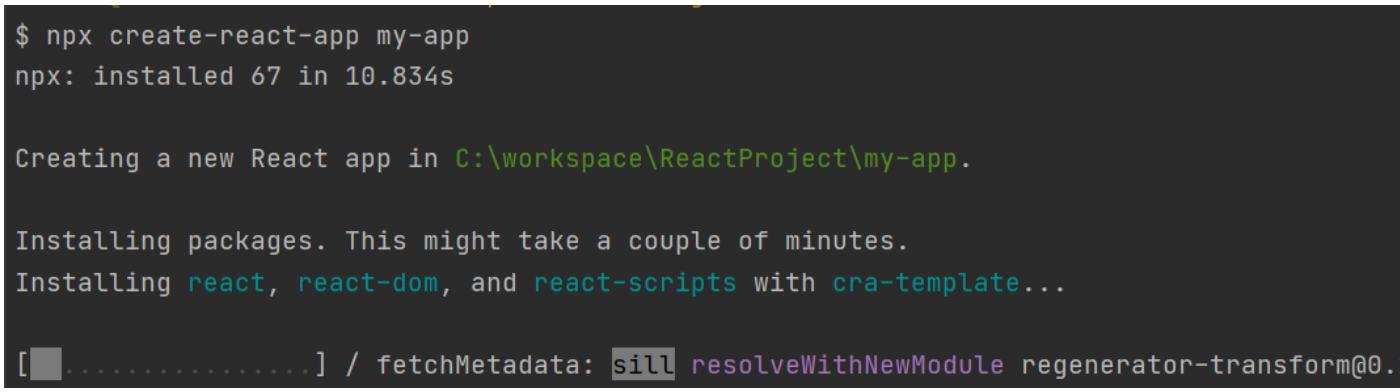
node version 확인하기

자 이제, Create React App 명령어를 이용해 React 개발 환경을 만들어 주도록 한다.

해당 명령어로 만든 React 환경은 react 가 정상적으로 동작할 수 있는 환경을 자동적으로 구축해준다.

```
npx create-react-app my-app
```

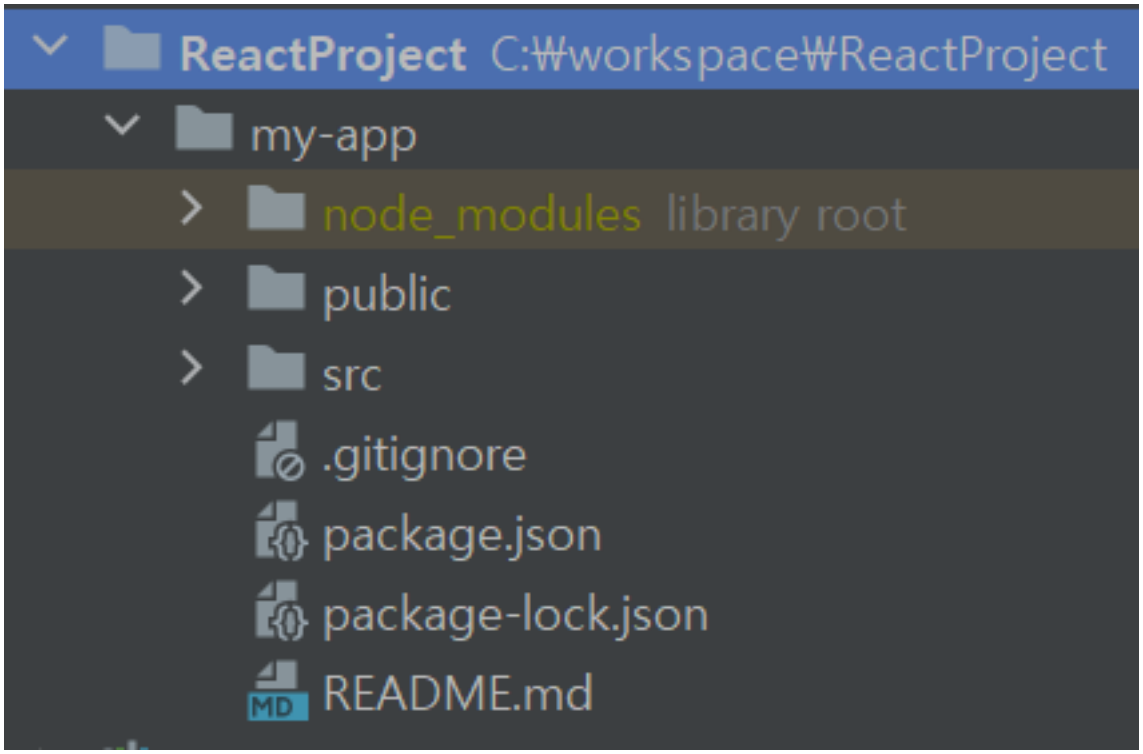
해당 명령어 입력 시 자동적으로 무언가가 깔리면서, 프로젝트 환경이 구성된다.



한참 뒤, 성공했다는 메시지가 뜨면,

해당 명령어를 입력 한 폴더 하위에 my-app 이라는 폴더가 생성된 것을 확인할 수 있다.

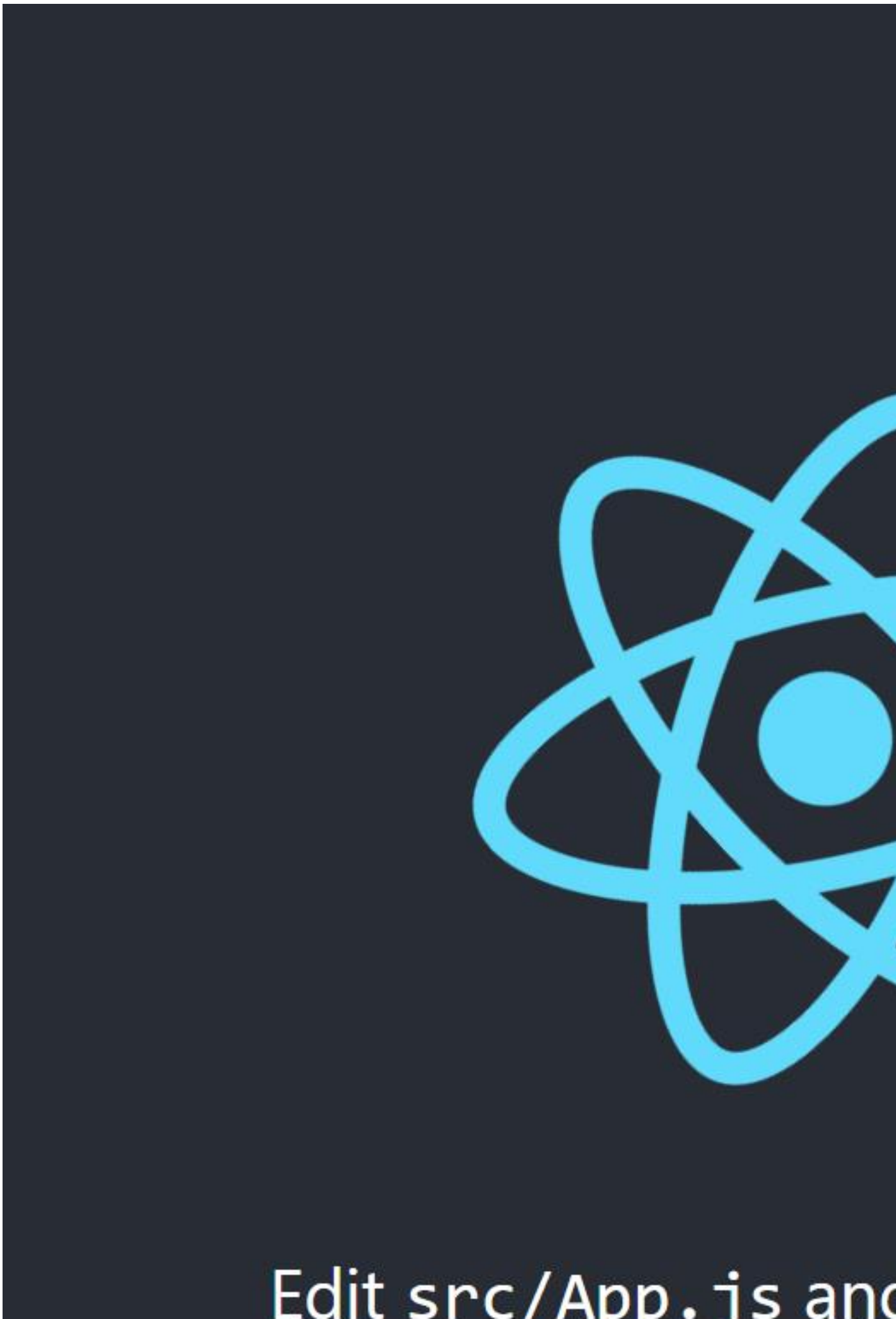
```
Success! Created my-app at C:\workspace\Rea
Inside that directory, you can run several

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for p

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build depe
    and scripts into the app directory. If
```



설치 완료가 된 내용 중 npm start 를 이용하면, 리액트 프로젝트를 돌려볼 수 있다.

localhost:3000 으로 웹사이트에 접속해보면, 기본적으로 세팅된 화면이 뜨는 걸 확인할 수 있다.

```
$ npm start

> my-app@0.1.0 start C:\workspace\ReactProj
> react-scripts start

i ｢wds｣: Project is running at http://192.1
i ｢wds｣: webpack output is served from
i ｢wds｣: Content not from webpack is served
i ｢wds｣: 404s will fallback to /
Starting the development server...
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.35.241:3

Note that the development build is not opti
To create a production build, use npm run b
```


Edit src/App.js and

npm run start 후, localhost:3000 접속 화면

하지만, 이제 우린 자습서의 내용을 추가해서 넣을 예정이니,

기존 my-app 에 src/폴더에서 index.css 와 index.js 만 남기고 모두 삭제해주고,

먼저 맛보기로 Hello, world 를 띄우는 화면을 만들어 보도록 하겠다.

Hello World

index.js 페이지에 아래 코드를 복사해서 넣어보자.

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';



/*hello world 출력하기*/

ReactDOM.render(

    <h1>Hello, world!</h1>,

    document.getElementById('root')

);
```

import 기능을 통해 react 와 reactDOM 을 이용한다고 선언했다.

reactDOM 패키지를 주입할 경우, DOM 에서 이용할 수 있는 특화된 메서들을 호출해서 사용할 수 있고,

그 안에 들어갈 엘리먼트들을 reactDOM 에서 관리한다.

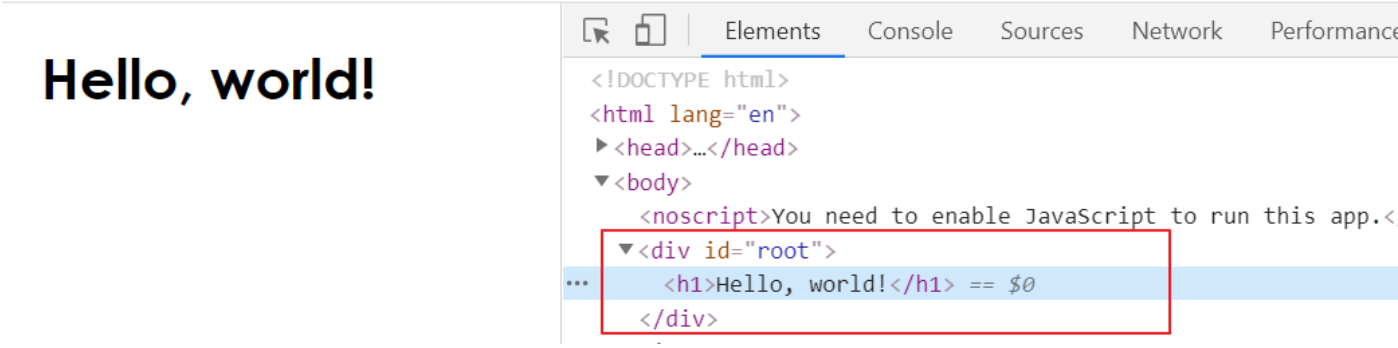
그중 render 라는 메서드를 이용할 경우, react 에서 만든 element 를 cotainer 에 랜더링 하는 게 가능하다.

reactdom.render() 사용법

```
ReactDOM.render(element, container [, callback])
```

위의 예시 코드는 우리는 reactDOM.render 의 엘리먼트에 <h1> 태그 안 Hello, world 를

HTML 파일에 root 라는 id 를 가지는 DOM 노드를 찾아 해당 엘리먼트를 추가하라고 코드를 만든 거다.



ReactDOM 에서 id 가 Root 인 DOM 을 찾아 hello, world!를 넣어줬다.

기본 html 파일 구조는 /public 폴더 밑 index.html 에 작성되어있다.

Hello, World! 를 컴포넌트로 바꿔보자.

React 의 가장 중요한 개념 중 하나는 **컴포넌트**를 주로 사용한다는 것이다.

즉 우리는 우리가 원하는 기능을 하나의 컴포넌트로 만든 뒤, 조합해서 사용하는 것이 가능한 게

React 의 컴포넌트 기능이다.

지금부터 우린 앞서 만든 hello, World 를 컴포넌트로 만들어서 화면에 보여주는 걸 해보도록 하겠다.

재사용이라는 개념은 처음 웹을 해본 사람들에게는 잘 모르겠지만,

이 재사용을 잘할 수 있는 코드를 만들어 놓아야 나중에 업무가 매우 줄어든다는 사실을 알 수 있다.

컴포넌트는 함수형과 클래스 형태가 있는데, 지금은 클래스를 이용하여 컴포넌트를 만들도록 하겠다.

아래의 코드를 복사해서 index.js 에 다시 넣어보자.

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

class HelloWorld extends React.Component {
    render() {
        return <h1>Hello, world!</h1>;
    }
}
ReactDOM.render(
    <HelloWorld/>,
    document.getElementById('root')
);
```

화면 상에서는 아까와 동일한 형태로 Hello, world! 가 보일 것이다.

하지만 이번에 작성한 코드는 Helloworld 라는 클래스를 만들어서 해당 클래스를 호출하는 방식이다.

이럴 경우 어떠한 화면에서 hello world 가 필요하다면, 해당 컴포넌트를 호출해서

사용하기만 하면 된다.

지금의 예제는 너무 활용도가 낮은 예제라 느낄 수는 없지만, 추후 좀 더 의미 있는 예제를 만들어봐야겠다.

react에서의 props 활용하기

저번에 React 컴포넌트에서 공부를 했다.

우리는 앞으로 React에서 컴포넌트를 만들어서 재사용을 할 예정이다.

이때 컴포넌트 호출 시 각각 필요한 데이터를 전달할 일이 생길 것인데 이때 이용하는 게 바로 props이다.

오늘은 React에 props라는 개념을 이용해서 React 컴포넌트에 데이터를 전달해 보기로 하겠다.

컴포넌트를 만드는 두 가지 방법

React에서 컴포넌트는 두 가지 방법으로 만들 수 있다.

JavaScript의 function모양과 비슷하게 작성하는 **함수 컴포넌트와**

Class를 만들 듯이 작성하는 **클래스 컴포넌트가** 있다.

저번 스터디에서는 클래스 컴포넌트 형태로 Hello, world를 화면에 출

력시켜보았다.

지난 시간에 만든 클래스 컴포넌트와 동일한 역할을 수행하는 함수형 컴포넌트로 한번 바꿔보겠다.

전체 코드

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';


//클래스 컴포넌트

class HelloWorld extends React.Component {

    render() {

        return <h1>Hello, world!</h1>;

    }

}


//함수컴포넌트

function FunctionHelloWorld() {

    return <h1>Hello, world</h1>;

}
```

```
//함수컴포넌트로 만든 엘리먼트로 추가해도 클래스컴포넌트로 넣었을 때와 동일한 결과가 나온다.

ReactDOM.render(

    <FunctionHelloWorld/>,

    document.getElementById('root')

);
```

함수형 컴포넌트로 만든 FunctionHelloWorld를 ReactDom.render로 호출 시

마찬가지로 페이지에 Hello, world가 쓰이는 걸 볼 수 있다.

컴포넌트 네이밍 만들 시 주의사항

React 컴포넌트는 반드시 대문자로 시작해야 한다.

대문자로 작성하지 않을 경우, 해당 컴포넌트는 정상 작동하지 않는다.

ReactDom.render 시 엘리먼트 자리에 소문자로 시작하는 것은 DOM태그로 인식하기 때문에,
사용자가 직접 작성한 컴포넌트를 엘리먼트 자리에 넣어 실행할 경우에는 대문자로 시작해야 정확히 인식한다.

```
▶Warning: <functionHelloWorld /> is using incorrect inde
casing. Use PascalCase for React components, or lowercase
HTML elements.
    at functionHelloWorld
```

소문자로 컴포넌트을 만들면 동작하지 않는다.

Props 활용해보기

그럼 이제 우리가 만든 함수형 컴포넌트를 보자.

Javascirpt에 익숙한 분들에겐 함수형 파라미터인 매개변수가 익숙할 것인데,

React의 props라는 개념은 즉 함수에 파라미터인 매개변수와 비슷한 역할이다.

만약 우리가 위에서 만든 함수 컴포넌트에서 Hello, World가 아닌 특정 사람 이름을 넣어서

인사하는 구문으로 바꾸려면 어떻게 해야 할까?

먼저 익숙한 함수 컴포넌트에서 props를 이용해 이름을 직접 받아보도록 하겠다.

전체 코드

```
import React from 'react';

import ReactDOM from 'react-dom';
```

```
import './index.css';


function FunctionHelloWorld(obj) {
    console.log(obj);
    return <h1>Hello, {obj.name}</h1>;
}


ReactDOM.render(
    <FunctionHelloWorld name="JiNa"/>,
  document.getElementById('root')
);
```

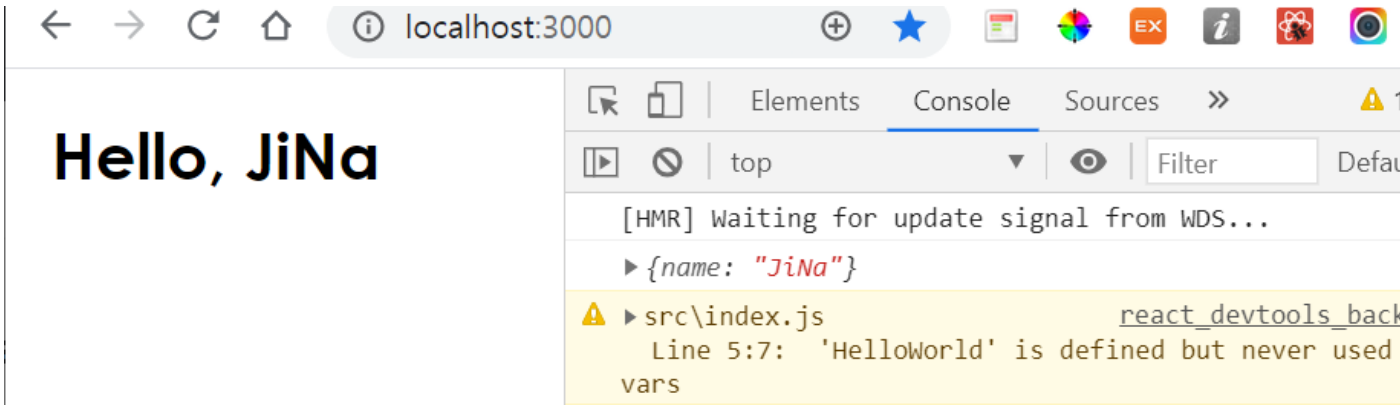
기존 FunctionHellWorld 컴포넌트 매개변수 자리에 obj이름으로 컴포넌트에서 전달한 props를 받도록 했다.

console.log로 확인해보니 React 컴포넌트에서 보낸 props가 객체 형태로 들어오는 걸 확인했다.

ReactDom에서 element 입력 시 name이라는 props를 적어주면 해당 props가 객체 형태로 컴포넌트에 전달된다.

그리코 컴포넌트 내에서 받은 props를 컴포넌트 내에서 이용하려면 {} 사이에 입력해주면 된다.

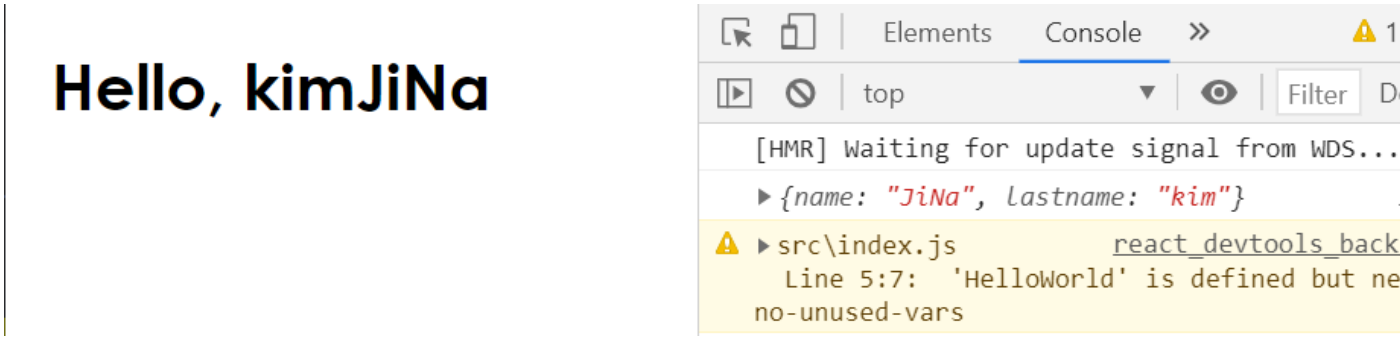
해당 코드를 실행시킨 화면이다.



props는 컴포넌트에 객체 형태로 전달된다.

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

function FunctionHelloWorld(obj) {
    console.log(obj);
    return <h1>Hello, {obj.lastname}{obj.name}</h1>;
}

ReactDOM.render(
    <FunctionHelloWorld name="JiNa" lastname="kim" />,
  document.getElementById('root')
);
```

객체 형태로 전달되기 때문에 컴포넌트에 여러 개의 props를 보내는 것도 가능하다.



props에 name 외에 lastname 도 추가했다.

이렇게 우리가 만든 컴포넌트에 props 기능을 이용하면 컴포넌트에서 각각 다른 데이터를 가지고

그 조건에 맞는 결과를 출력하는 게 가능하다는 것을 알 수 있다.

react 에서 State 사용하기

지난 포스팅에 이어서 오늘은 React 에서 State 를

공부해보도록 하겠다.

단순히 컴포넌트에서 데이터를 받아 올 경우 props 를 이용했다면,

우린 State 는 함수 내부에서 선언하는 변수처럼 해당 값을 우리가 원하는 대로 변경하여 사용하는 것이 가능해진다.

기존 코드 복습하기

전 포스팅의 코드이다. 성과 이름을 보내서 화면에서 특정 사용자 이름이 들어올 경우

Hello, 뒤에 특정 사용자의 이름을 넣어서 보여주었다.

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';



class HelloWorld extends React.Component {

    render() {

        return <h1>Hello, {obj.lastname}{obj.name}</h1>;

    }

}

ReactDOM.render(

    <HelloWorld name="JiNa" lastname="kim" />,

    document.getElementById('root')

);
```

이처럼 props 를 이용하면 함수의 매개변수처럼 데이터 전달을 할 수 있었다.

하지만, 우리가 늘 필요한 코드는 고정적인 값이 아니라 계속해서 변하는 데이터일 수도 있다

이럴 경우에 이용해야 하는 것은 Props 가 아닌 State 이다.

State 초기 값 설정하기

State 는 컴포넌트 내 초기값 설정이 가능하다.

컴포넌트에서 State 초기값을 설정하기 위해서는 constructor() 이용한다.

constructor 는 해당 컴포넌트가 실행되면 가장 먼저 호출이 된다.

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

class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            user : 'visitor'
        }
    }
    render() {
        return <h1>Hello, {this.state.user}!</h1>;
    }
}

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

super(props)의 필요성

constructor 내부 코드 중 props 변수를 받아 super(props)를 선언해주는 걸 볼 수 있다.

class 내 super 라는 메서드는 기존의 정의된 class 의 값을 가져와 해당 class 에서도 사용할 수 있게 해주는 method 다

컴포넌트 실행 시 this.props 생성자가 필수로 필요하기 때문에 constructor 를 이용할 경우에는

반드시 super 메서드를 이용해 props 생성자를 호출해주자.

해당 코드는 반드시 입력해줘야 오류 없이 컴포넌트가 실행된다.

render() 내 state 호출 방법

설정한 초기 값을 render()에서 호출할 때는 {} 중괄호 안에 넣어줘서 사용할 수 있다.

위 코드 중 {this.state.user}라는 형식으로 만들어준 초기 값을 가져왔다.

해당 코드 실행 시 화면 내 Hello Visitor! 가 뜨는 것을 확인할 수 있다.

State Update 해보기

초기값으로 만들어진 State 값은 setState()라는 컴포넌트를 이용하면 업데이트가 가능하다.

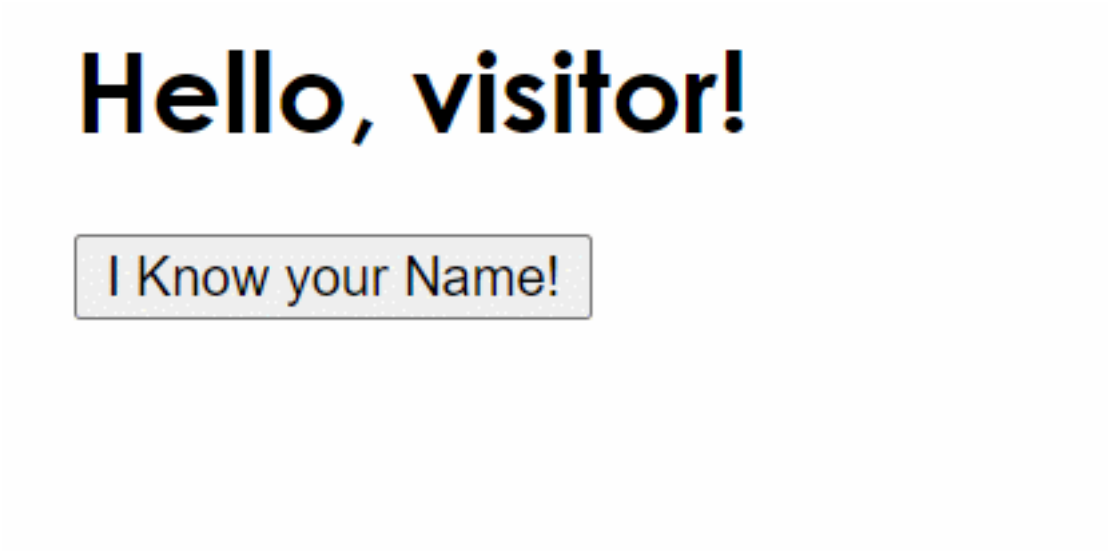
react 내에서는 해당 코드가 실행되어 State 값이 변경되면 컴포넌트를 리 랜더링 시킨다.

우리는 버튼을 하나 추가해서 이름을 setState()를 이용해서 이름을 바꿔줘 보도록 하겠다.

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

class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            user : 'visitor'
        }
    }
    render() {
        return <div>
```

```
            <h1>Hello, {this.state.user}!</h1>
            <button onClick={()=>
this.setState({user:"JinaKim"})}>I Know your
Name!</button>
            </div>;
    }
}

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

해당 코드 실행화면이다.



버튼 클릭 시 사용자 이름이 변경하는 걸 볼 수 있다.

button 에 onclick 이벤트에 this.setState()를 이용해 state 내 user 이름에 JinaKim 으로

업데이트를 진행한 것이다.

React Event 사용하기

HTML을 이용한 홈페이지를 만들었을 경우, 우리는 해당 홈페이지에서 동적인 역할을 만들기 위해
JavaScript를 이용하여 동적인 이벤트를 만들어낸다.

가장 간단한 alert창을 HTML과 Javascript 문법으로 띄운다고 가정했을 경우
아래와 같이 입력할 것이다.

```
<button onclick="alert('확인')">
 Click Me!
 </button>
```

React Event 기본 문법

이와 같은 동작을 React에서 발생시키려면 아래와 같은 문법으로 작성한다.

- 이벤트 발생 문은 소문자가 아닌 camelCase(카멜 케이스)로 입력한다.
- JSX 를 사용하여 문자열이 아닌 함수로 이벤트를 전달한다.
- 기본 이벤트가 있는 태그 동작을 방지하려면, preventDefault 를 명시적으로 호출한다. (return false)

preventDefault 란?

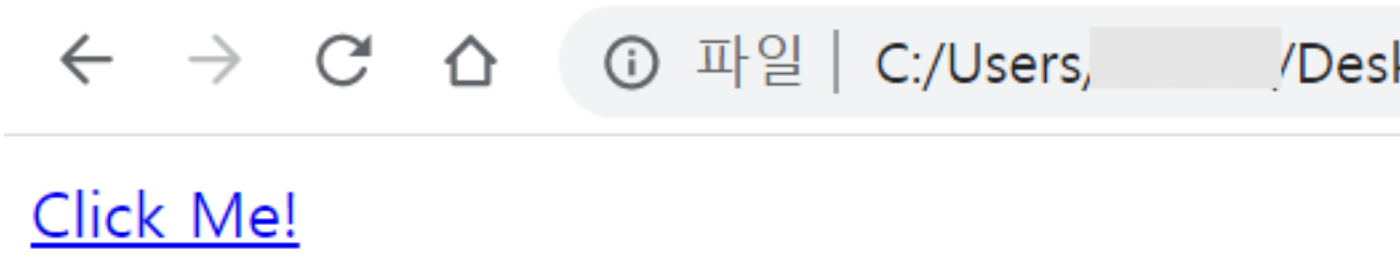
event.preventdefault는 이벤트를 취소할 때 호출합니다.
javascript 에서는 return false를 이용하여 기본 동작을 방지했지만, react의 경우는 return false 사용이 불가하다.
그렇기 때문에 기본 이벤트 동작을 방지하기 위해서는 preventdefault를 명시적으로 호출해 줘야 한다.

javascript Return false 사용 예시

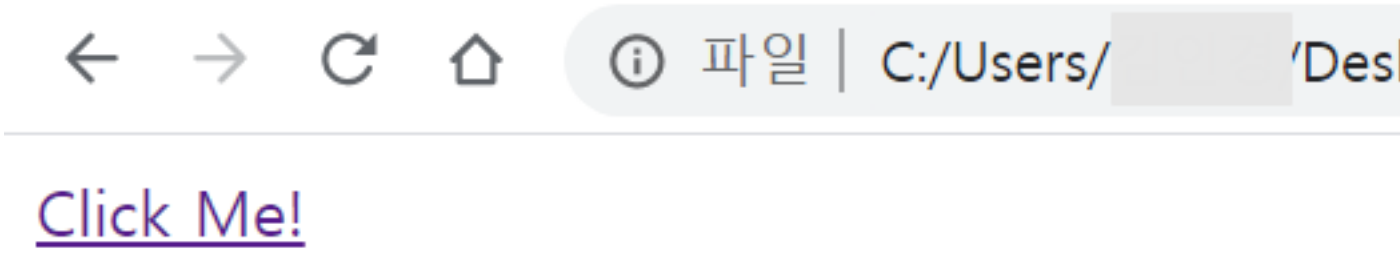
기존 HTML A태그와 Submit 같은 태그는 기본적인 이벤트가 발생한다.
예를 들어보면, A태그에 href="#"와 같은 속성을 줄 경우 주소창 뒤에 # 이 붙으며 가장 위로 올라가는
이벤트가 발생한다.

이러한 태그 자체에 있는 기본 이벤트를 없애기 위해서는 return false를 이용한다.

```
<a href="#" onclick="console.log('test');return false">
Click Me!
</a>
```



console.log('test');return false 입력시 href="#"은 동작하지 않음



console.log('test'); 만입력할 시 href="#"태그가 동작함

React 사용 예시

위와 같은 코드를 React의 이벤트 문법에 맞춰 작성한다고 하면 아래와 같이 바꿀 수 있다.

```
function clickEvent(event) {
    event.preventDefault();
    console.log("test");
}

ReactDOM.render(
    <a href="#" onClick={clickEvent}>Click Me!</a>,
    document.getElementById('root')
);
```

이벤트 기본 예시 만들기

react를 이벤트를 활용한 간단한 예시를 만들어보겠다.
0인 기본값에서 1씩 더하거나 빼는 간단한 예시이다.

Code

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

```
class Calc extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            value: 0
        };
        this.add = this.add.bind(this);
        this.minus=this.minus.bind(this);
    }

    add() {
        this.setState(state => ({
            value: state.value + 1
        }));
    }

    minus() {
        this.setState(state => ({
            value: state.value - 1
        }));
    }
```

```
    render() {
        return (
            <div>
                <p>현재 값 : {this.state.value}</p>
                <button onClick={this.add}>+</button>
                <button onClick={this.minus}>-</button>
            </div>
        )
    }
}

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

constructor에서 state에 vlaue 값을 0으로 기본값을 만들어준 뒤, onClick으로 호출할 add, minus function을 만들었다.

add와 minus에서 this.setState를 사용할 경우, constructor에서 해당 함수에 this를 바인딩해주어야 한다.
만약 this를 바인딩하지 않을 경우 아래와 같은 에러가 발생하니, 해당 코드는 반드시 넣어주어야 한다.

TypeError: Cannot read property 'setState' of undefined

add

C:/workspace/study-react/src/index.js:14

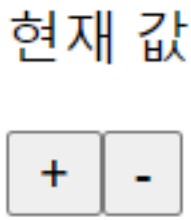
```
  11 |     }
  12 |
  13 |     add() {
> 14 |         this.setState(state => ({
  15 | ^           value: state.value + 1
  16 |         }));
  17 |     }
```

View compiled

▶ 19 stack frames were collapsed.

constructor에 this.add.bind(this)를 안넣을 경우 에러 발생

실제 실행 화면



react 계산기 예시

React JSX 개념 정리

오늘은 이제까지 미뤄두고 사용했던 JSX에 대해 좀 더 공부해보기로 했다.

기본적으로 JavaScript에서의 변수로 넣을 수 있는 값은 String, Number,

Array, Object 등이 주로 떠오른다.

하지만, 전 React 실습 중에서는 아래와 같은 코드들이 계속 보이게 된다.

아래와 같은 문법은 사실 문자열로 보기도 어렵고, HTML로 보기도 힘들다.

```
const sayHello = <h1>Hello, World!</h1>
```

JSX 란?

JSX는 JavaScript를 확장한 문법이다.

React에서 UI가 어떻게 생겼는지를 설명하기 위해 사용하고 있다.

해당 문법은 공식적인 문법은 아니며, React에서도 필수적으로 사용할 필요는 없다.

하지만, UI 관련 작업 시 가독성이 좋으며,

React 시 에러와 경고 메시지 등이 표시되어 디버깅 시 유리하다.

그리고, HTML을 다뤄본 개발자들에겐 너무 익숙한 문법이라 러닝타임도 없기 때문에 사용 시 매우 유용하다.

JSX 사용 방법

1️⃣ JSX 하나의 태그로 만들어준다.

JSX 사용할 경우 꼭 모든 Element를 감싸는 하나의 태그로 만들어줘야 한다.

같은 Level선에서 끝나는 태그가 2개일 경우 Filed to compile 오류가 발생하니,

반드시 가장 최상위 태그는 한 개가 되도록 감싸준다.

잘못된 사용법

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';


class HelloWorld extends React.Component {

    render() {

        return <h1>Hello, world!</h1><h4>This is React APP!</h4>;

    }

}
```

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

Failed to compile

```
./src/index.js
SyntaxError: C:\workspace\study-react\src\index.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (7:37)

   5 | class HelloWorld extends React.Component {
   6 |     render() {
>  7 |         return <h1>Hello, world!</h1><h4>This is React APP!</h4>;
     |                                      ^
   8 |     }
   9 | }
  10 |
```

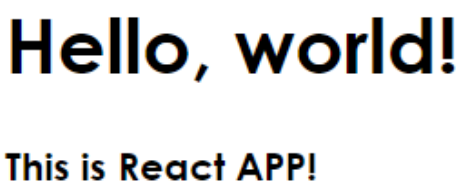
This error occurred during the build time and cannot be dismissed.

두가지 태그를 나열해서 작성할 경우 Compile Error가 생성된다.

올바른 사용법

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


class HelloWorld extends React.Component {
    render() {
        return <div>
                <h1>Hello, world!</h1>
                <h4>This is React APP!</h4>
```

```
                </div>;
        }
}

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



JSX 전체를 감싸는 태그를 추가하니 정상작동하는 것을 볼 수 있다.

2️⃣ JSX 내부 태그는 반드시 닫아주기

HTML 코드와 비슷하여 혼동할 수도 있으나, JSX 태그의 경우 반드시 닫아주지 않으면 정상 작동하지 않는다.

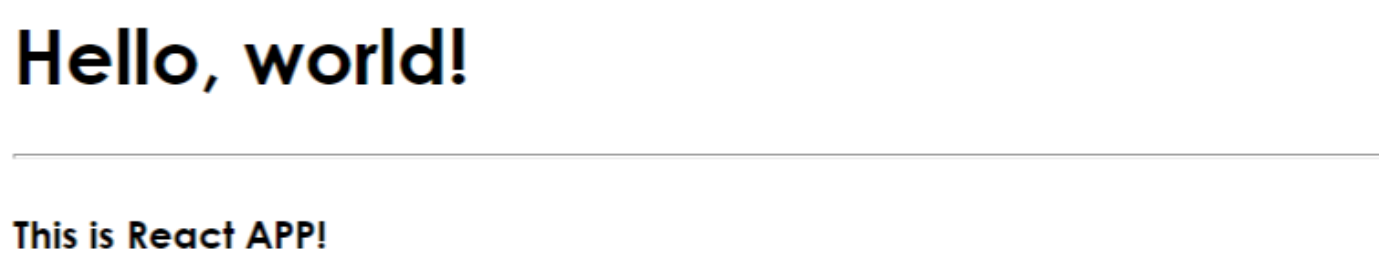
예를 들어
 <img> 등 안이 비어진 태그가 있다면, 반드시 닫아주어야 한다.

HTML에서는
, <hr> 태그 등은 닫기가 없어도 정상 작동하나, JSX 내부에서는 반드시 닫아주어야 한다.

태그를 닫지 않을 경우 마찬가지로 compile 오류가 난다.

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


class HelloWorld extends React.Component {
    render() {
        return <div>
                    <h1>Hello, world!</h1>
                    <hr/>
                    <h4>This is React APP!</h4>
                </div>;
    }
}
```

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



<hr/>로 닫아서 실행 할 경우 정상작동하는 것을 볼 수 있다.

3⃣ JSX 내부에 JavaScript 표현식을 넣는 게 가능하다.

JSX에서는 정적인 String 외 변수를 넣어 사용이 가능하다.

중괄호 안에 Javascript에서 선언한 변수 혹은 Javscript내에서 사용 가능한 표현 식들은 넣어 사용한다.

숫자 계산, object 객체 value 가져오기 혹은 간단한 변수 등을 연결할 수 있으며,

더 넓게는 Function 호출 후 Return 값 까지도 표현이 가능하다.

다음은 JavaScript 표현식 중 object와 함수를 연결한 예시다.

user object의 key Vlaue를 가져오고 today() 함수를 통해 오늘 연월일을 가져와서 표시한다.

이렇듯 JSX 내부에 {}(중괄호) 안에 해당 javaScript 표현식을 적어 가져올 수 있다.

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


const user = {
    firstName :'jina',
    lastName: 'kim'
}
function today() {
    let year = new Date().getFullYear();
    let month = new Date().getMonth() + 1;
    let day = new Date().getDate();
    return year +'/' + month + '/' + day;


}
class HelloWorld extends React.Component {
    render() {
        return <div>
                    <h1>Hello, {user.firstName}{user.lastName}!</h1>
```

```
                    <h4>Today: {today()}</h4>
                </div>;
    }
}

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

함수와 객체에서 값을 가져와 JSX 안에 넣어서 호출한 결과

4️⃣ JSX javaScript 객체로 인식됩니다.

컴파일이 끝나면 JSX표현식은 정규 Javascirpt 객체로 인식되게 된다.

JSX는 return에서만 사용하는 것뿐 아니라,

Javascript변수에 할당도 가능하며,

for문 if문 내부에서 함수 return 형식으로도 반환이 가능하다.

밑의 예시를 보자.

기존 today의 경우 return에서 javascript 문법으로 넘겨주던 것을 JSX로 변경하여 return 시켰고

sayHello()라는 function을 새로 만들어 각각 if문에 맞게 JSX를 return 시켜주는 것으로 변경시켰고,

해당 코는 아래처럼 잘 작동한다.

```
import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';


const user = {

    firstName :'jina',

    lastName: 'kim'

}

function today() {

    let year = new Date().getFullYear();

    let month = new Date().getMonth() + 1;

    let day = new Date().getDate();


    const today = <h4>Today: {year} / {month} / {day}</h4>
```

```
    return today

}
function SayHello() {
    let time = new Date().getHours();
    if(time >= 17 || time < 6) {
        return <h1>Good Night!</h1>
    }else {
        return <h1>Good Morning</h1>
    }
}
class HelloWorld extends React.Component {
    render() {
        return <div>
                    <h1>Hello, {user.firstName}{user.lastName}!</h1>
                    {today()}
                    <h3>{SayHello()}</h3>
                </div>;
    }
}

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

```
);
```

JSX로 return 받은 결과값 실행결과

JSX 작동원리

마지막으로 조금 어려운 얘기이다.

그럼 기존에 JavaScript에 없던 JSX가 정상 작동하는 이유는 무엇인가?

JSX로 작성된 코드는 React 내부의 React.createElement()를 사용하여 웹에서 이해할 수 있는 형태로 변환된다.

물론 우리가 직접 React.createElement를 호출해서 똑같은 효과를 낼 수 있지만,

우린 좀 더 가독성이 좋은 JSX를 이용해서 작성하기 때문에, 아마 React를 하면서 우리가 직접 React.createElement를 호출하는 일은 없을 것이다.

React.createElement 사용법

React.createElement(component props,... children)

작동 예시

만약 우리가 class이름이 hi이며 안의 내용이 Say Hello! 내용을 h1태그 형식으로 뿌려주는 JSX를 만들었다고 하면,

```
const sayHello  = <h1 className="hi"> Say Hello! </h1>
```

내부적으로는 아래의 코드로 변경되어 컴파일된다.

```
React.createElement(
'h1',
{className : 'hi'},
'Say Hello!'
)
```

이런 식으로 JSX는 내부적으로 작동하나 우리는 이런 내부 사정은 잠시 미뤄두고

앞으로 JSX의 규칙에 맞춰 잘 사용하면 되겠다.