학습목표: 클래스컴포넌트 생성, 실행, props, state, 이벤트 (클래스를 먼저하는이유 : 예전부터 리액트하셨던분은 클래스형이 익숙하기 때문에 프로젝트를 클래스만 하는경우가 있음, 이미만들어져 있는 코드를 확인할경우, 클래스문법은 약간이라도 알아야하고 es6 사용과 컴포넌트 개념이 익숙하지 않기 때문에 1 일동안 만들면서 친숙해지려고함 , 함수형 컴포넌트와 유사하기 때문에 미리 해보고 3 일차부터는 본격적으로 함수형으로 진행합니다.)

vscode -> 보기 -> 터미널 (ctrl + ~) , cmd 하셔도 됩니다.

프로젝트 생성하기 (카페글 참고)
cd 프로젝트명 (프로젝트 폴더 안으로 들어가기) / cd.. 상위
npm start (실행) / yarn start

컴포넌트란
컴포넌트는 UI 를 구성하는 조각(piece)에 해당되며, 독립적으로 분리되어 재사용을 됨을 목적으로 사용됩니다. React 앱에서 컴포넌트는 개별적인 JavaScript 파일로 분리되어 관리합니다. (예: Header, HeaderTitle, Wrapper, List, ListItem 컴포넌트)

JSX 는 JavaScript 문법 확장으로 구문이 HTML 과 유사합니다. React 애플리케이션 제작 시 꼭 JSX 가 필요한 것은 아니지만, JavaScript 로 UI View 를 구성하는 마크업하는 것은 매우 까다로우므로 특별한 경우가 아니라면 JSX 사용을 권장합니다.
JSX 가 하는 일은 React 요소(Element)를 만드는 겁니다. React 요소는 실제 DOM 요소가 아니라, JavaScript 객체입니다.

실제 DOM 조작 대신 가상 DOM 을 사용하는 이유?
UI 는 사용자의 요구사항에 따라 수시로 변경(업데이트) 됩니다. 업데이트 과정에서 실제 DOM 이 변경되면 업데이트 된 요소와 그 자식 요소를 다시 렌더링 해야 하는데, 이러한 일련의 과정이 UI 속도를 느리게 만듭니다. UI 컴포넌트 개수가 많을 수록 렌더링 비용은 더욱 많이 들어 속도는 더더욱 느려지게 됩니다.

```
import React, { Component } from 'react';
class App extends Component {
  클래스영역
  render() {
    return (
      <div>
          jsx 영역
      </div>
    );
  }
}
export default App;
```

```
import React from 'react';
const App = () => {
  함수영역
  return (
    <div>
        jsx 영역
    </div>
  );
};
export default App;
```

```
├── README.md
├── node_modules/ # 개발 의존 모듈 집합 디렉토리
├── package.json
├── public/ # 정적 리소스 디렉토리
│   ├── favicon.ico
│   ├── index.html # 애플리케이션 기본 템플릿
│   └── manifest.json
├── src/ # React 애플리케이션 개발 디렉토리
│   ├── App.css
│   ├── App.js # 애플리케이션 파일
│   ├── App.test.js
│   ├── index.css
│   ├── index.js # 엔트리 파일
│   ├── logo.svg
│   └── serviceWorker.js
└── yarn.lock
```

public/index.html

```
    <div id="root"></div>
    <!--
      이 HTML 파일은 템플릿입니다.
      브라우저에서 직접 열면 빈 페이지가 나타납니다.
```

src/index.js

```
import React from 'react' // React 모듈 로드
import ReactDOM from 'react-dom'  // ReactDOM 모듈 로드
import './index.css'     // 메인(인덱스) 스타일 로드
import App from './App'  // 앱 컴포넌트 로드
import * as serviceWorker from './serviceWorker' // 서비스 워커 로드

// ReactDOM 모듈의 렌더 함수를 사용해 #root (src/index.html) 요소
// 내부에 동적으로 App 컴포넌트(React Element)를 렌더링 합니다.
ReactDOM.render(<App />, document.getElementById('root'))
serviceWorker.unregister()
```

App.js

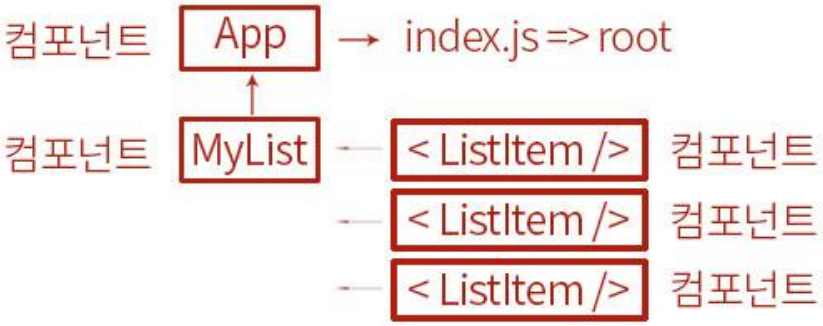
```
import React from 'react'   // React 모듈 로드
import logo from './logo.svg'   // 로고 이미지 로드
import './App.css'  // App 스타일 로드

// 함수형 컴포넌트(Functional Component)
function App() {
  // JSX(JavaScript 문법 확장) 반환
  return (
    <div>

    </div>
  )
}

export default App  // App 컴포넌트 모듈 내보내기
```

함수형컴포넌트와 클래스컴포넌트 JSX 문법, props

** component : ** : component 는 UI 를 독립적이고 재사용 가능한 조각으로 분리

컴포넌트 이름은 첫글자 반드시 대문자로 작성한다 App , List 등

< MyList />



1. class component

```
import React , { Component } from 'react';
class 컴포넌트이름 extends Component {
 render() {
    return (
        JSX 문법
    )
  }
}
export default 컴포넌트이름;
```

2. function component

```
import React from 'react'
const 컴포넌트이름 = () => {
    return ( JSX 문법 )
}
export default 컴포넌트이름;
```

 # JSX 규칙 : javascript + XML

JSX 는 리액트 컴포넌트를 작성하면서 return 문에 사용하는 문법 얼핏보면 HTML 같지만, 실제로는 자바스크립트

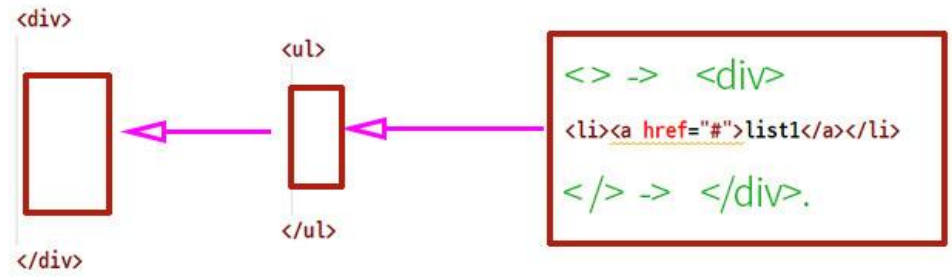
규칙

1. 태그는 반드시 닫아줘야 한다.
2. 최상단에서는 반드시 div 로 감싸주어야 한다. (Fragment 사용 , <> 상황에따라)
3. JSX 안에서 자바스크립트 값을 사용하고 싶을때는 {}를 사용한다.
   변수값 출력예시 참고 -> { name }
4. 조건부 렌더링을 하고싶으면 &&연산자나 삼항연산자를 사용한다.
5. 인라인 스타일링은 항상 객체형식으로 작성한다.
6. 별도의 스타일파일을 만들었으면 class 대신 className 을 사용한다.
7. 주석은 {/* */}을 사용해 작성한다.

1. 태그는 반드시 닫아줘야 한다.
2. 최상단에서는 반드시 div 로 감싸주어야 한다.

```
  return (
    <div>
      <p>hello react!</p>
      <input type="text" />
    </div>
  )
```

```
<div>
  <ul>
    <li><a href="#">list1</a></li>
    <li><a href="#">list2</a></li>
    <li><a href="#">list3</a></li>
    <li><a href="#">list4</a></li>
    <li><a href="#">list5</a></li>
  </ul>
</div>
```



3. JSX 안에서 자바스크립트 값을 사용하고 싶을때는 { }를 사용한다.

```
return (
    <div>
      hello {name}
    </div>
  )
```

4. 조건부 렌더링을 하고싶으면 &&연산자나 삼항연산자를 사용한다.

```
return (
        <div>
         { true ? console.log('참') : console.log('거짓')  }
        </div>
    )
```

5. 인라인 스타일링은 항상 객체형식으로 작성한다.

```
    스타일 작성시 - 빼고 첫글자는 대문자로 작성한다
    font-size : fontSize   , background-color : backgroundColor , line-height : lineHeight , text-indent : textIndent

   const style ={   bckgroundColor :'red' }
<div style= { style }>react</div>
   <div style={{ backgroundColor: 'red' }}>react</div>
```

6. 별도의 스타일파일을 만들었으면 class 대신 className 을 사용한다. (권장사항)

```
   <div className="App">   react   </div>
```

7. 주석은 {/* */}을 사용해 작성한다.

```
 <div>
        {/* 주석은 이렇게 */}
        <h1 // 태그안에서 주석작용 >
           react
        </h1>
  </div>
```

클래스 문법

```
class Counter extends Component {

  클래스 변수
  클래스 함수
  render() {
     변수 작성
     return (
          JSX 문법  클래스 함수나 클래스 변수를 출력할 경우에는 this 가 붙는다
        { 클래스변수 : this.변수   }
        { 클래스함수 : this.함수   }
     )
   }
}
export default Counter;
```

 # props 와 state
*** props : 부모 컴포넌트가 자식 컴포넌트에게 내려주는 값**
*** state : 컴포넌트 자기자신이 선언하는 값**

1. props (property 줄임말)
부모컴포넌트에서 자식컴포넌트에게 props 를 집어넣으면 자식컴포넌트는 **this.props** 로 해당값을 사용할 수 있음

2. state

리액트는 제이쿼리와 달리 dom 을 직접 가져와서 조작하는 것이 아니기 때문에,
유동적으로 변하는 값(mutation)은 모두 state 로 관리해 줌

```
-> 선언
  state = {
    변수: 값
  }

 -> 데이터 값을 변경 (유동적 변하는 값 )
 this.setState({변수:값})

 -> 출력
 {this.state.변수}

 class Test extends Component {
  state = {    변수: 값   }

  함수 = () => {
    this.setState({ 변수 : this.state.변수  })
  }
```

```
  render() {
    return (
      <div>
        <div>{this.state.변수}</div>
      </div>
    )
  }
}

**예전형식 **
import React, { Component } from 'react'
class Test extends Component {
   constructor(props) {
    super(props)
    this.state = {
      변수: 값
    }
  }  ...
}
```

이벤트 핸들링

```
1. 이벤트 이름은 카멜표기법로 작성
 onclick → onClick    onchange → onChange

 <p onClick={ this.함수명 }>

 this.함수명뒤에 () 붙이지 않는다  -> 이벤트하지않고 무조건 호출
```

비구조 할당

const { name, age } = this.props

const { name, age } = this.state

{ this.props.name } => { name }

{ this.state.name } =>{ name }