



3 React 기초

리액트 작업 환경 설정

3주차 세션 강의록

1. React?



특히 SPA(Single Page Application)을 위한 사용자 인터페이스(UI)를 구축하는 데 사용되는 오픈 소스 JavaScript 라이브러리

▼ SPA 장점?

SPA는 웹 앱에 필요한 모든 정적 리소스를 처음에 한번 다운로드

페이지 간 이동 시, 페이지 간신에 필요한 데이터만을 JSON으로 전달받아 페이지를 간신하므로 전체적인 트래픽을 감소

전체 페이지를 다시 렌더링하지 않고 변경되는 부분만을 간신하므로 새로고침이 발생하지 않음.

▼ SEO?

SEO (검색 엔진 최적화)

검색자의 의도를 이해하고 이에 맞춰 웹 페이지의 콘텐츠를 제작, 검색 결과 페이지에서 잘 노출 되도록 웹페이지의 태그와 링크 구조를 개선하여 자연 유입 트래픽을 늘림

리액트는 프레임 워크가 아닌 라이브러리

▼ 차이점?

프레임워크는 **밀키트**라고 생각하면 된다.

→ 이미 주어진 재료를 가지고 정해진 레시피대로만 하면 요리의 완성! 요리의 완성도는 어느정도 보장되어 있지만 자유도는 떨어진다.

라이브러리는 **팬트리**라고 생각하면 된다.

→ 요리하는 데에 정해진 규칙은 없지만 재료는 준비되어 있다. 팬트리에 있는 식재료와

식기 등을 원하는 대로 가져가 원하는 대로 요리하면 된다. 그 요리가 어떤 결과를 가져 올지는 아무도 모르지만.

다른 웹 프레임워크가 Ajax, 데이터 모델링, 라우팅등과 같은 기능을 내장하고 있는 반면 리액트는 뷰만 신경쓰는 라이브러리로 기타기능은 직접 구현해야함

다른 개발자들이 만든 라이브러리 라우팅 ⇒ 리액트 라우터, Ajax ⇒ Axios, 상태관리 ⇒ 리덕스등을 사용하여 빈자리를 채우면 됨

자신의 취향대로 스택을 설정할 수 있다는 장점이 있지만 여러 라이브러리를 접해야한다는 단점도 존재

데이터가 변할때마다 어떤 변화를 줄지 고민하는 것이 아니라 기존의 뷰를 날려버리고 처음부터 새로 렌더링 하는 방식

- 애플리케이션 구조 간단
- 작성해야할 코드양 줄어듦
- 어떻게 변화를 줄지 신경 X → 뷰가 어떻게 생길지만 선언하여 데이터의 변화가 있으면 기존에 있던 것은 버리고 정해진 규칙에 따라 새로 렌더링

최대한 성능을 아끼고 편안한 사용자 경험을 제공하면서 구현하고자 개발한 것 “[React](#)”

2. 리액트의 특징

Virtual DOM



DOM(Document Object Model)?

객체로 문서 구조를 표현하는 방법, HTML or XML

웹페이지가 브라우저 안에서 화면에 나타나고, 이벤트에 반응하며 값을 입력받는 등 기능들을 수행할 객체들로 실체화된 형태

▼ xml?



HTML은 웹 페이지 및 웹 응용 프로그램의 구조를 만들기 위한 표준 마크업 언어이다.



XML은 사람과 기계가 읽을 수 있는 형식으로 문서를 인코딩하기 위한 규칙 집합을 정의하는 마크업 언어이다.

사람과 기계가 읽을 수 있는 형식으로 **문서를 인코딩하기 위한 규칙 집합을 정의하는 마크업 언어**

데이터베이스, 프로그램 및 응용 프로그램, 모바일 응용 프로그램 등에서 **다른 플랫폼간에 데이터를 교환**

하기 위해 사용

HTML은 미리 정의된 태그가 있다.

XML에서는 프로그래머가 자신만의 태그 집합을 정의한다.

DOM을 수많은 플랫폼과 웹브라우저에서 사용하는데 취약한 단점 존재

“동적 UI에 최적화 되어 있지 않다”

HTML은 자체적으로 정적 → 자바스크립트를 사용하여 동적으로 바꿈

ex) 트위터, 페이스북 : 스크롤바를 내릴 수록 수많은 데이터 로딩

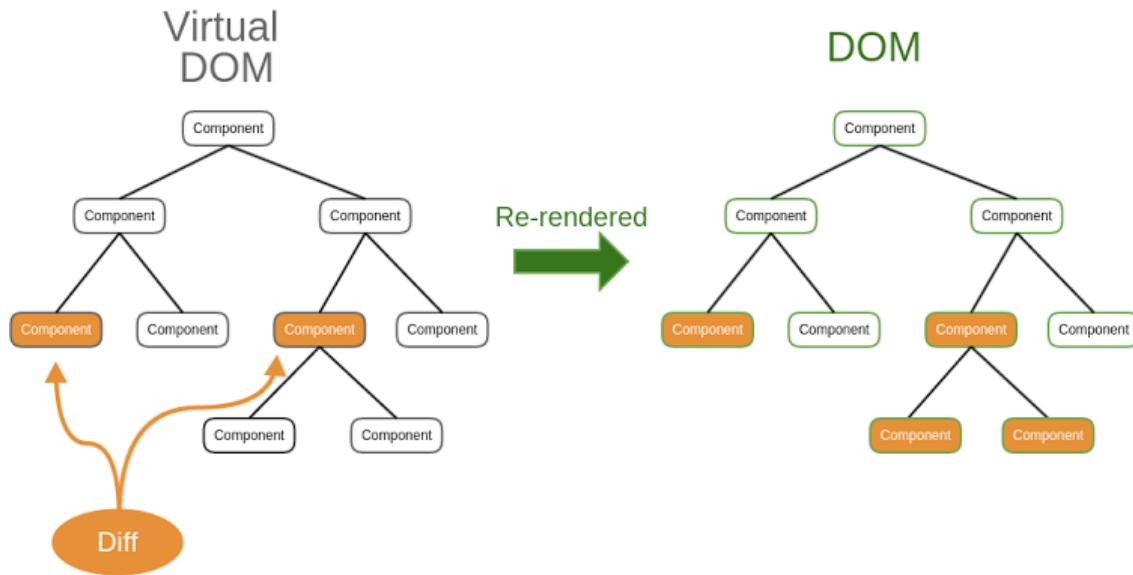
ex) 유저가 어떤 포스트에 좋아요를 누르거나 담아둔 장바구니 목록에서 상품을 하나 삭제하거나 어딘가에 댓글을 남기면 전체 노드들이 처음부터 다시 그려짐

이렇게 규모가 큰 웹 애플리케이션에서 DOM에 직접 접근하여 변화를 주면 성능 이슈 존재
DOM 자체는 빠르지만 웹 브라우저 단에서 DOM에 변화가 일어나면 웹 브라우저가 CSS를 다시 연산하고 레이아웃을 구성하고 페인트를 리페인트 → 여기서 느려짐 (성능저하의 문제)

SPA(Single Page Application)을 많이 사용하면서 DOM tree를 즉각적으로 변경할 일이 생겼다.

전체 페이지를 서버에서 매번 보내주는 것이 아니라, 브라우저 단에서 자바스크립트가 관리하기 때문에, DOM조작을 더욱 더 효율적으로 할 수 있게 최적화가 필요하게 됨

해결법



▼ Diff 알고리즘?

Virtual DOM이 업데이트되면, React는 Virtual DOM을 업데이트 이전의 Virtual DOM 스냅샷과 비교하여 정확히 어떤 Virtual DOM이 바뀌었는지 검사

해당 엘리먼트의 태그나 컴포넌트가 변경된 경우라면 해당 노드를 포함한 하위의 모든 노드들이 언마운트 즉, 제거한 뒤에 새로운 Virtual DOM으로 대체합니다. 이런 변경이나 업데이트가 모두 마무리된 이후에 딱 한번은 실제 DOM에 이 결과를 업데이트

DOM을 최소한으로 조작하여 작업을 처리하는 방식 “Virtual DOM”

- 데이터를 업데이트하면 전체 UI를 Virtual DOM에 리렌더링
- 이전 Virtual DOM에 있던 내용과 현재 내용 비교
- 바뀐 부분만 실제 DOM 적용

오해

VirtualDOM 을 이용한다고 해서 사용하지 않을때와 비교해 무조건 빠른 것은 아님
리액트 메뉴얼에는 해당 문장이 존재



우리는 다음 문제를 해결하기 위해 리액트를 만들었습니다.
지속적으로 데이터가 변화하는 대규모 애플리케이션 구축하기

작업이 매우 단순한 (단순 라우팅이 있는 정적 페이지)의 경우는 오히려 리액트를 사용하지 않는 것이 더 나은 성능을 보일 수 있음

리액트와 Virtual DOM이 제공할 수 있는 것은 바로 **업데이트 처리의 간결성**

UI를 업데이트 하는 과정에서 생기는 복잡성을 모두 해소하고 더욱 쉽게 업데이트 할 수 있음

3. react는 front-end인데 왜 node를 쓸까요?

리액트 프로젝트 만들때는 Node.js를 반드시 먼저 설치



Node.js

크롬 v8 자바스크립트 엔진으로 빌드한 자바스크립트 런타임

웹 브라우저 환경이 아닌 곳에서도 자바스크립트를 사용하여 연산 가능

▼ 정답

리액트 애플리케이션은 웹 브라우저에서 실행되는 코드이므로 Node.js와 직접적인 연관은 없지만, 프로젝트 개발하는데 필요한 주요 도구들이 Node.js를 사용하기 때문에 설치

Node.js를 설치하면 Node.js 패키지 매니저 도구인 npm이 설치됨.

→ npm으로 수많은 개발자가 만든 패키지 (재사용 가능한 코드)를 설치하고 설치한 패키지의 버전 관리 용이

4. node_modules가 뭔데?

설치된 파일

node_modules

- 해당 프로젝트에서 필요한 라이브러리들이 설치되어 있는 곳
- react 모듈이 설치
- import 구문을 통해 리액트 사용 가능

public

- 배포할때 외부적으로 보여지는 대표적인 것들이 들어가 있다

favicon : 로고 이미지

index.html : 뼈대가 되는 html 파일

manifest.json : 모바일에서 저장하는 웹 어플리케이션을 만들때 필요

robots.txt : 크롤링을 위해 이용되는 파일

`src` → 핵심 파일

- 최상위 루트 : **index.js**
- 코드의 대부분이 이곳으로 들어온다.(index.js, 리액트 컴포넌트 같은 js파일, css파일 등)

`.gitignore`

- 해당 파일 안 명시되어 있는 폴더나 파일들은 git에 올라가지 않는다

`package.json`

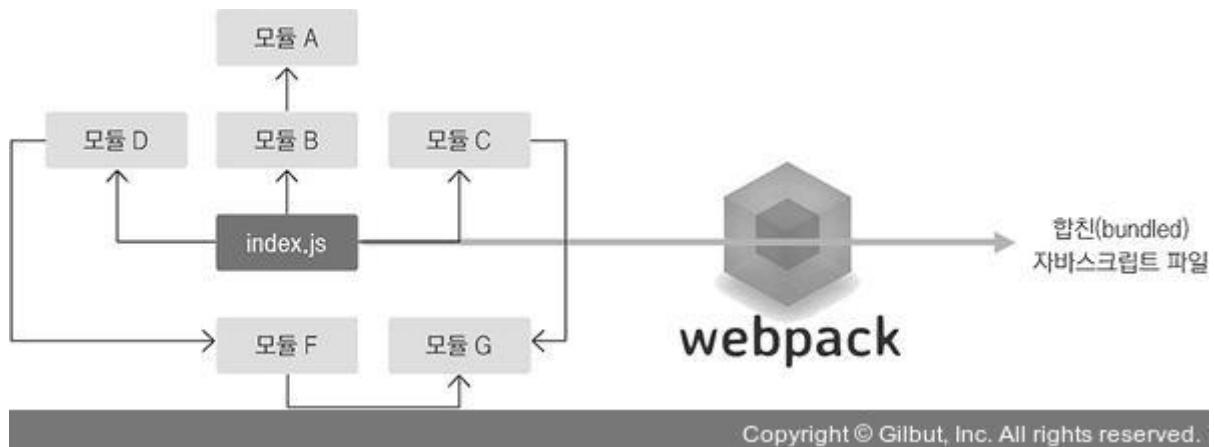
- 프로젝트의 정보를 담고 있는 파일
- 프로젝트에서 쓰이고 있는 라이브러리와 버전들이 명시되어 있다

`README.md`

- 프로젝트에 대해서 설명 하는 파일.

`yarn.lock`

- 프로젝트에 설치되어 있는 라이브러리들의 의존성을 나열한 파일



`node_modules` → 이렇게 모듈을 불러와서 사용하는 것은 Node.js에서 지원하는 기능 / 참고로 Node.js는 require 구문

이러한 기능을 브라우저에서 사용하기 위해 '번들러' 사용

2017년 이후로 브라우저에서도 import 구문을 사용할 수 있게 되지만 이는 단순히 다른 경로의 자바스크립트 불러오는 용도 이므로 프로젝트 번들링과는 다름.

```
import logo from './logo.svg';
```

웹팩을 사용하면 svg, css 파일도 불러올 수 있음



구버전 웹 브라우저와 호환하기 위해서 최신 자바스크립트 문법을 ES5 형태로
변환시킴

구버전 웹 브라우저에서는 실행되지 않기 때문에 사전에 꼭 변환

JSX라는 문법도 정식 자바스크립트 문법이 아니므로 ES5 형태의 코드로 변환
해야함

5. npm, yarn?

npm과 yarn은 자바스크립트 런타임 환경인 **노드(Node.js)**의 패키지 관리자

전 세계의 개발자들이 자바스크립트로 만든 **다양한 패키지를** npm 온라인 데이터베이스에
올리면 npm, yarn과 같은 패키지 관리자를 통해 **설치 및 삭제가 가능**

CLI를 통해 패키지 설치 및 삭제뿐 아니라 **패키지 버전 관리, 의존성 관리**도 편리

npm vs yarn

속도

npm과 yarn의 주요 차이점 중 하나는 패키지 설치 프로세스를 처리

npm : 패키지를 한 번에 하나씩 순차적으로 설치

yarn : 여러 패키지를 동시에 가져오고 설치하도록 최적화되어 있어

→ 패키지 설치 속도 측면에서 yarn이 npm보다 빠름

보안

yarn은 보안 측면에서 npm보다 더 안전한 것으로 알려져 있음

npm : 자동으로 패키지에 포함된 다른 패키지 코드를 실행 → 이 특징은 편리하지만 안정성을 위협

yarn : yarn.lock 또는 package.json 파일에 있는 파일만을 설치

보안은 yarn의 핵심 기능 중 하나이지만 최근 npm의 업데이트에서 npm의 보안 업데이트도 크게 향상

6. JSX란?

JSX는 자바스크립트 확장 문법으로 xml과 비슷한 형태

코드가 번들링 되는 과정에서 바벨을 사용하여 일반 자바스크립트 형태로 변환



JSX 도 자바스크립트 문법이라 할 수 있을까?

JSX는 프로젝트 개발할때 사용되므로 공식적인 자바스크립트 문법이 아님

바벨에서는 여러 문법을 지원 할 수 있도록 preset 및 plugin 설정

바벨을 통해 개발자들이 임의로 만든 문법, 혹은 차기 자바스크립트 문법을 사용 가능

1. 보기 쉽고 익숙하다

JSX를 사용하는 것이 가독성이 높고 작성하는데도 쉬움.

2. 더욱 높은 활용도

`div` 나 `span` 같은 html 태그를 사용할 수 있을 뿐만아니라 앞으로 만들 컴포넌트도 JSX 안에서 사용 가능

index.js

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

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



ReactDOM.render?

컴포넌트를 페이지에 렌더링하는 역할
react-dom 모듈을 불러와 사용할 수 있음
첫번째 파라미터에서는 페이지에 렌더링할 내용을 JSX 형태로 작성, 두번째 파라미터에서는 해당 JSX를 렌더링 할 document 내부 요소 설정
여기서는 id가 root인 요소 안에 렌더링하도록 설정



React.StrictMode?

React.StrictMode는 리액트 프로젝트에서 리액트의 레거시 기능을 사용하지 못하게 하는 기능
문자열 ref, componentWillMount 등 나중에는 완전히 사라지게될 옛날 기능을 사용했을 때 경고를 출력함.

7. JSX 간단 실습 - 오류 고치기!

```
import React from 'react';

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

export default App;
```

▼ 힌트!

하나의 DOM 트리 구조로 이루어 져야한다는 규칙

꼭 div가 아니라 리액트 v16 이상부터 도입된 Fragment 기능도 사용 가능 (<> </>)

▼ 정답

```
import React from 'react';

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

export default App;
```

```
        </div>
    );
}

export default App;
```

```
import React from 'react';

function App() {
  const name = '리액트';
  return (
    <>
      {name === '리액트' ? (
        <h1>리액트 입니다</h1>
      ) : (
        <h1>리액트가 아닙니다</h1>
      )}
    </>
  );
}

export default App;
```

▼ 힌트!

JSX 내부의 자바스크립트 표현식에서는 if 문 사용 불가능

▼ 정답

```
import React from 'react';

function App() {
  const name = '리액트';
  return (
    <>
      {name === '리액트' ? (
        <h1>리액트 입니다</h1>
      ) : (
        <h1>리액트가 아닙니다</h1>
      )}
    </>
  );
}

export default App;
```

특정 조건을 만족할 때 내용을 보여주고 만족하지 않을 때 아예 아무것도 렌더링 하지 않아야 할 상황

```
import React from 'react';

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

▼ 힌트!

&& 연산자를 이용하면 조건부 렌더링 가능

▼ 정답

```
import React from 'react';

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

export default App;
```

```
import React from 'react';

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

export default App;
```

▼ 힌트!

어떤 값이 undefined 일 수도 있다면 **OR (||)** 연산자를 사용하면 해당 값이 undefined 일 때 사용할 값을 지정할 수 있으므로 간단하게 오류 방지 가능

반면에 JSX 내부에서 undefined 렌더링하는 것은 괜찮음

```
import React from 'react';

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

export default App;
```

▼ 정답

```
import React from 'react';

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

export default App;
```

주석을 달고 싶다면?

```
import React from 'react';

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

export default App;
```

컨트롤 + / (맥은 커맨드)

8. 컴포넌트

리액트로 화면을 구성하게 되면, 사용자가 볼 수 있는 여러 가지 **컴포넌트**로 구성.

사용자에게 보여지는 UI 요소를 **컴포넌트** 단위로 구분하여 구현

React 공식 문서인 “[Components and Props](#)“를 보면



Conceptually, components are like JavaScript **functions**. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

→ 개념적으로 구성 요소는 자바스크립트 함수와 같다. 그들은 임의의 입력 (“props”라고 함)을 받아들이고 화면에 표시될 내용을 설명하는 React 요소를 반환한다.

컴포넌트를 잘 만드는 것이 왜 중요할까요?

리액트로 만들어진 앱을 이루는 가장 최소한의 단위가 컴포넌트이기 때문

앱을 리액트로 만든다는 것은 곧 작고 단단한 컴포넌트들을 만들고 이 컴포넌트들을 유기적으로 연결한다는 것

▼ Watcha Pedia

The screenshot shows the homepage of Watcha Pedia. At the top, there's a navigation bar with categories: 영화, TV, 책, 웹툰. Below it is a search bar and a login button. The main content area features a section titled "박스오피스 순위" (Box Office Ranking) displaying five movie posters with their titles and release dates:

- 1. 아바타: 물의 길 (Avatar: The Way of Water) - 2022 · 미국 - 평균 ★3.7 - 예매율 51% · 누적 관객 878만명
- 2. 더 퍼스트 슬램덩크 (The First Slam Dunk) - 2022 · 일본 - 평균 ★4.3 - 예매율 11% · 누적 관객 42만명
- 3. 유령 (The Witcher) - 2023 · 한국 - 평균 ★4.0 - 예매율 9.1%
- 4. 교섭 (The Negotiation) - 2022 · 한국 - 평균 ★2.9 - 예매율 8.2%
- 5. 영웅 (Hero) - 2022 · 한국 - 평균 ★2.9 - 예매율 6.4% · 누적 관객 222만명

Below this, there's a section titled "왓챠 Top 10 영화" (Watcha Top 10 Movies) showing a grid of five movie thumbnails.

react 컴포넌트 선언하는 두 가지 방식 중 하나. [클래스 컴포넌트 & 함수 컴포넌트]

클래스형 컴포넌트

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

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

- class 키워드
- Component로 상속
- render() 메소드

함수형 컴포넌트

함수형 컴포넌트는 function을 사용하지 않고 **화살표 함수**로 일반적으로 정의

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

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

export default App;
```

차이점

클래스형

- state, lifeCycle 관련 기능사용 가능하다.
- 메모리 자원을 함수형 컴포넌트보다 조금 더 사용한다.
- 임의 메서드를 정의할 수 있다.

함수형

- state, lifeCycle 관련 기능사용 불가능 [Hook을 통해 해결 됨]
- 메모리 자원을 함수형 컴포넌트보다 덜 사용한다.
- 컴포넌트 선언이 편하다.

클래스형 컴포넌트에서는 state(상태)를 사용할 수 있으며 각종 라이프사이클 및 메서드를 이용하여 컴포넌트가 마운트 혹은 언마운트 될 때 추가 작업을 수행시키는 등 조작을 할 수 있었음.

but, Hook이 등장한 이후부터는 위 기능들을 함수형 컴포넌트에서도 대부분 구현이 가능해졌으므로 일반적으로 사용되는 함수형 컴포넌트 + Hook를 중심으로 학습

props란?

부모 컴포넌트에서 자식 컴포넌트에 전달해 주는 값을 props

```
/* App.js */
import React from 'react';
import MyComponent from './MyComponent';

function App() {
  return (
    <MyComponent name="React"/>
  );
}
export default App;
```

<MyComponent name="React"/> 에서 name 부분이 props 를 설정하는 부분

```
/* MyComponent.js */
import React from 'react';

const MyComponent = (props) => {
  return <div>테스트 페이지, {props.name}</div>;
};

export default MyComponent;
```

▼ 결과

테스트 페이지, React

컴포넌트 태그 사이의 내용을 보여주려면 **children**

```
/* App.js */
import React from 'react';
import MyComponent from './MyComponent';
```

```

function App() {
  return (
    <MyComponent>사이에들어온값</MyComponent>
  );
}

export default App;

```

위처럼 `<MyComponent>` 태그 사이에 들어있는 값을 MyComponent에서 보여줄 수 있으려면 다음과 같이 `{props.children}` 이라고 작성

```

/* MyComponent.js */
import React from 'react';

const MyComponent = (props) =>{
  return (
    <div>테스트 페이지, {props.name}
    <br/>
    children 값은 {props.children}
    </div>
  );
};

MyComponent.defaultProps = {
  name: '기본'
}

export default MyComponent;

```

▼ 결과

테스트 페이지, 기본
children 값은 사이에들어온값

ES6의 비구조화 할당 문법

```

/* MyComponent.js */
import React from 'react';

const MyComponent = props =>{
  const { name, children } = props;
  return (
    <div>테스트 페이지, {name}
    <br/>
    children 값은 {children}
    </div>
  );
};

```

```
export default MyComponent;
```

state란?

| 컴포넌트 내부에서 변경될 수 있는 값.

props는 부모 컴포넌트가 설정하고, 컴포넌트 자신은 props를 바꾸지 못하는 특성이 있는 것과는 차이가 있다.

함수형 컴포넌트의 state - useState의 사용

리액트 16.8 이전 버전에서는 함수형 컴포넌트에 state 사용 불가

리액트 16.8 이후 버전부터 함수형 컴포넌트에서 Hooks 를 통해 사용 가능해짐

클래스 형

- constructor 안에서 this.state 초기 값 설정 가능
- constructor 없이 바로 state 초기값을 설정할 수 있다.
- 클래스형 컴포넌트의 state는 **객체 형식**
- this.setState 함수로 state의 값을 변경할 수 있다.

함수 형

- 함수형 컴포넌트에서는 useState 함수로 state를 사용한다.
 - useState 함수를 호출하면 배열이 반환되는데 첫 번째 원소는 현재 상태
- 두 번째 원소는 상태를 바꾸어 주는 함수

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

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

```

const Say = () => {
  const [message, setMessage] = useState('초기값');
  const onClickEnter = () => setMessage('안녕!');
  const onClickLeave = () => setMessage('잘가!');

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

  return (
    <div>
      <button onClick={onClickEnter}>입장</button>
      <button onClick={onClickLeave}>퇴장</button>
      <h1 style={message}></h1>
      <button style= onClick={()=>setColor('red')}>RED</button>
    </div>
  );
};

export default Say;

```

message는 현재 state의 상태가 저장되고, **setMessage** 는 state를 바꾸어 주는 setter 함수

message 에는 ‘초기값’이라는 문자열이 들어갈 것이며, setMessage는 그러한 state의 값을 변경하는 함수

props와 state의 차이

- props와 state 모두 컴포넌트에서 사용 혹은 렌더링할 데이터를 담고 있다.
- **props**는 부모 컴포넌트 쪽에서 설정해 주는 것이다. 컴포넌트 자신이 변경할 수 없다.
- 반면 **state**는 컴포넌트 자신이 직접 가지고 있는 값이고, 그 값의 변경도 내부에서 가능하다.

왜 컴포넌트를 분리?

리액트에서 컴포넌트는 다양한 역할을 합니다. 어떤 컴포넌트는 UI를 표현하고자 하고, 또 어떤 컴포넌트는 동작하는 로직을 담고 있을 수도 있음

컴포넌트는 재사용할 수 있는 **최소 UI 단위**이지만 웹에 따라 수행하려고 하는 역할이 복잡해지고 다양

이 때 생각해봐야 할 것이 바로 **관심사의 분리**

관심사의 분리란 복잡한 코드를 비슷한 기능을 하는 코드끼리 별도로 관리

컴포넌트 분리 기준?

컴포넌트를 분리하는 기준은 프로젝트마다 기준점을 어디에 잡느냐에 따라 다양한 분리 기준이 존재

1. view와 로직을 분리하는 경우도 있고
2. state에 따라서 분리하는 경우도 있는데 이외에도 컴포넌트를 분리하는 기준에 대한 다양한 설계 패턴들이 존재