React.js

강경미 (carami@nate.com)

프론트엔드 라이브러리란

- 요즘의 웹은 웹 애플리케이션이라 할 수 있다.
- 웹페이지를 동적으로 보여주기 위해서는 수많은 작업이 필요하다.
- 수많은 돔요소를 직접 관리하고 코드들을 관리하는 것이 어렵다.
- 돔관리 상태값 업데이트 관리를 최소한으로 하고 기능개발 , 사용자 인터페이스 개발에 집중하도록 도와준다.
- 개발시의 생산성 유지보수성을 좋게 도와준다.
- AngularJS, Backbone.js, Derby.js, Ember.js, Exr.js, Knockback.js, Sammy.js, PureMVC, Vue.js

Angular vs React vs Vue

- Angular
  - http클라이언트, 라우터, 다국어 지원등 다양한 기능들을 가지고 있음.
  - Type script 사용이 기본임
- React
  - 컴포넌트 개념에 집중된 라이브러리.
- Vue
  - 쉽다. 공식 라우터, 공식 상태관리 라이브러리가 존재.
  - Angular 와 React 에서 좋은 것들을 가져 다가 사용

1. 리액트란?

- 유저 인터페이스를 만드는데 사용하는 기술
- 페이스북 개발팀에서 만듦.
  - 데이터가 변할 때마다 어떤 변화를 줄지 고민하는 것이 아니라, 기존 뷰를 날리고 새로 랜더링하는 방식으로 만든 기술
  - 지속적으로 데이터가 변화하는 대규모 애플리케이션 구축하기가 목적
- 오직(View)만 신경 쓰는 라이브러리
- 컴포넌트 방식이다.
  - 컴포넌트 하나에서 해당 컴포넌트의 생김새와 작동 방식을 정의한다.

1-1. 초기 랜더링

- UI관련 프레임워크, 라이브러리 어떤 것을 사용하든지 가장 처음 어떻게 보일지를 정하는 초기 랜더링이 필요하다.
- 리액트는 render 함수가 담당
  - render 함수는 컴포넌트가 어떻게 생겼는지 정의하는 역할을 담당한다.
  - render 함수는 html 문자열을 반환하는 것이 아니라, 뷰가 어떻게 생겼고 어떻게 동작할지에 대한 정보를 가지고 있는 객체를 반환한다.
  - render함수가 호출되면 그 내부에 있는 컴포넌트들도 재귀적으로 랜더링 된다.
- 최상위 컴포넌트의 렌더링 작업이 끝나면 HTML마크업이 만들어지고 실제 페이지의 DOM요소 안에 주입된다.

1-2. 조화과정

- 리액트에서 뷰를 업데이트할 때 "업데이트 과정을 거친다 " 라고 말하기 보다는 "조화과정 " 을 거친다고 말하는 것이 좀더 정확한 표현이다.
- 컴포넌트에서 데이터의 변화가 있을 때 보기에는 변화에 따라 뷰가 변형되는 것처럼 보이지만, 사실은 새로운 요소로 갈아 끼우기 때문이다.
- 이 작업 역시 render함수가 맡아서 처리한다. 데이터가 업데이트 되었을 경우 render함수가 또 다시 호출되게 된다.
- 이 때 render함수가 반환하는 결과를 DOM에 반영하지 않고, 이전에 render함수가 만들었던 컴포넌트 정보와 현재 render함수가 만든 컴포넌트 정보를 비교하여, 변화된 부분만 DOM에 업데이트를 하게 된다.
- 결국 방식 자체는 루트 노드부터 시작하여 전체 컴포넌트를 처음부터 다시 랜더링하는 것처럼 보이지만, 사실 최적의 자원을 사용하여 이를 수행하게 된다.

2. 리액트 특징

- Virtual DOM
  - Virtual DOM을 사용하면 실제 DOM에 접근하여 조작하는 대신, 이를 추상화한 자바스크립트 객체를 구성하여 사용한다. 가벼운 DOM의 사본으로 이해하면 된다.
  - 리액트에서 데이터가 변하여 웹 브라우저에 실제 DOM을 업데이트할 때는 다음과 같은 세가지 절차를 밟는다.

1. 데이터를 업데이트 하면 전체 UI를 Virtual DOM에 리렌더링한다.
2. 이전 Virtual DOM에 있던 내용과 현재 내용을 비교한다.
3. 바뀐 부분만 실제 DOM에 적용한다.

2. 리액트 특징

- 리액트는 View만 담당한다.
- 리액트는 프레임워크가 아니라 라이브러리다.
- 다른 개발자들이 만든 리액트 라우터, Ajax처리에서는 axios 나 fetch, 상태 관리에는 리덕스(redux)나 MobX를 사용한다.

3. 작업환경설정

- node.js / npm, yarn(패키지관리자) 설치
- 코드 에디터 설치
- create-react-app으로 프로젝트 생성하기

https://nodejs.org/ko/
https://yarnpkg.com/lang/en/docs/install/#windows-stable
https://code.visualstudio.com/
https://git-scm.com/download/win

3. 작업환경설정 확인

버전확인

- node –v
- yarn --version

create-react-app 설치

- yarn global add create-react-app (yarn 이용시)
- npm install –g create-react-app (npm 이용시)

3. 프로젝트 생성 & 서버 실행

- 프로젝트 생성

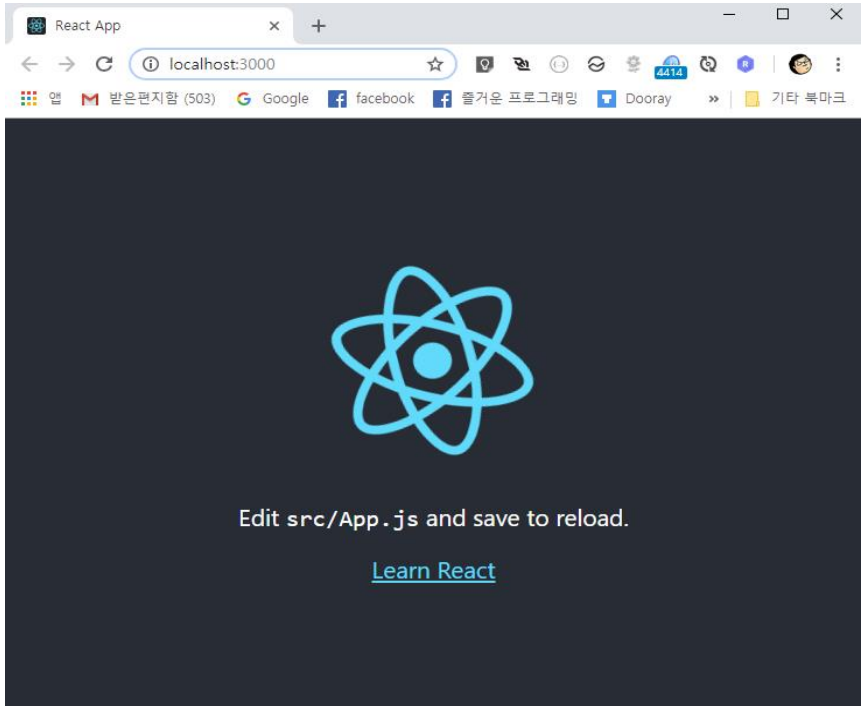
  create-react-app hello-react

- 서버 실행

  cd hello-react

  yarn start 또는 npm start

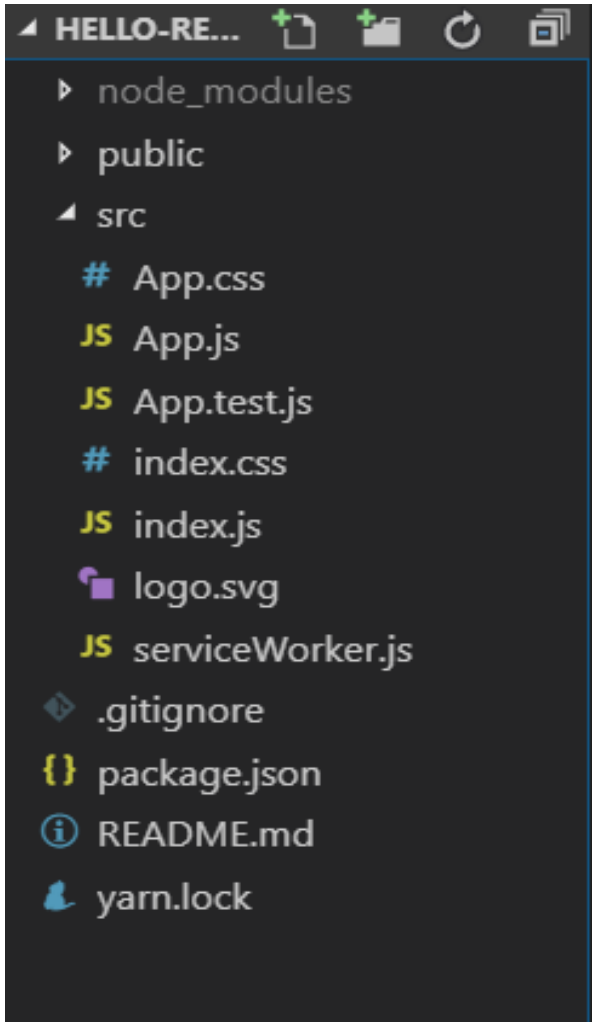
http://localhost:3000 브라우저가 자동으로 실행



- (참고 **'react-scripts'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는배치 파일이 아닙니다.** 라는 메시지가 나오고 실행이 안되면 npm update 명령을 한번 입력한다.)

Hello-react 프로젝트 살펴보기

- 디렉토리 구조



Hello-react 프로젝트 살펴보기

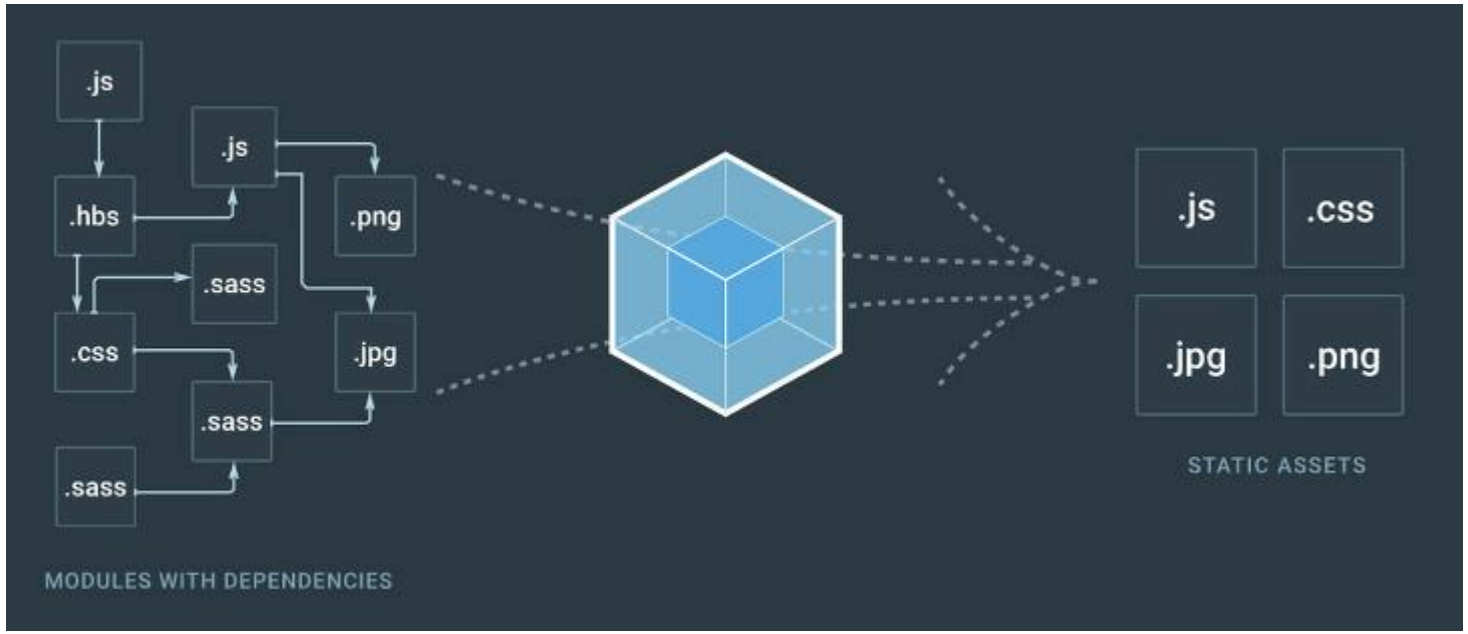
- App.js

App.js

```
1   import React, { Component } from 'react';
2   import logo from './logo.svg';
3   import './App.css';
4
5   class App extends Component {
6     render() {
7       return (
8         <div className="App">
9           <header className="App-header">
10            <img src={logo} className="App-logo" alt="logo" />
11            <p>
12              Edit <code>src/App.js</code> and save to reload.
13            </p>
14            <a
15              className="App-link"
16              href="https://reactjs.org"
17              target="_blank"
18              rel="noopener noreferrer"
19            >
20              Learn React
21            </a>
22          </header>
23        </div>
24      );
25    }
26  }
27
28  export default App;
29
```

Webpack

자바스크립트 모듈 번들러(Bundler)로 자바스크립트로 작성된 모듈을 번들링 하여 하나 또는 여러 개의 번들로 만들어 주는 도구 입니다.



Babel

- https://babeljs.io/
- **Babel is a JavaScript compiler.**
- 자바스크립트 변환 도구
  - Node.js 나 브라우저에서 새로운 JacaScript 문법이 바로 반영되지 않을 수 있으므로 알맞게 변환해 주는 일을 해준다.

코드이해

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

ES6 이전 문법으로 변환하면

```
Var React = require('react');
var Component = React.Component;
```

- 파일을 모듈화 해서 사용하는 기능은 node.js기능
- 웹브라우저에서는 사용할 수 없지만 번들링 기능으로 가능하다.
- 번들링 도구엔 Browserify, RequireJS, webpack등이 있다.

코드 이해

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

- SVG, CSS등도 webpack이 불러올 수 있다.
- 이렇게 파일들을 불러오는 것을 webpack의 로더(loader)가 담당한다.
- 로더 종류는 여러가지로 css-loader는 CSS파일을 불러오고, file-loader는 웹 폰트나 미디어 파일등을 불러올 수 있다. babel-loader는 js파일들을 불러오면서 ES6로 작성된 코드를 ES5로 변환해 준다.
- 로더는 직접 설치해야하지만, create-react-app이 이러한 작업을 대신해준다.

코드 이해

class App extends Component

- 컴포넌트를 만드는 방법
  - 함수를 이용해서 만드는 방법
  - Class를 이용해서 만드는 방법
- 클래스를 이용해서 컴포넌트를 만드는 방법
  - Component클래스를 상속받는다.
  - JSX를 리턴하는 render함수를 반드시 가져야한다.
- ES5에는 prototype문법을 사용했는데, ES6에서는 class문법을 제공한다.
- render()함수는 컴포넌트가 어떻게 보일지에 대한 정보를 리턴한다. 리턴하는 값이 HTML 코드 같지만, 단순한 문자열이나 템플릿이 아니다. 이런 코드를 JSX라고 한다.

코드이해

export default App;

- App 이라는 컴포넌트를 다른 곳에서 불러서 사용할 수 있도록 내보내기 해준다.

index.js

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(<App />, document.getElementById('root'));
8
9   // If you want your app to work offline and load faster, you can change
10  // unregister() to register() below. Note this comes with some pitfalls.
11  // Learn more about service workers: http://bit.ly/CRA-PWA
12  serviceWorker.unregister();
13
```

JSX

JSX

- JSX는 자바스크립트의 확장 문법으로 XML과 매우 비슷하게 생겼다. 이런 형식으로 작성한 코드는 나중에 코드가 번들링되면서 babel-loader를 사용하여 자바스크립트로 변환되게 된다.
- JSX는 리액트용으로 공식 자바스크립트 문법이 아니다. 바벨에서 여러 문법을 지원할 수 있도록 prest을 설정하는데 ES6에서는 babel-preset-es2015를 적용하여 JSX를 지원하려고 babel-preset-react를 적용하고 있다.

JSX 장점

- 보기 쉽고 익숙하다.
- 오류 검사
  - JSX에 오류가 있다면, 바벨이 코드를 변환하는 과정에서 이를 감지한다.
- 더욱 높은 활용도
  - JSX에서는 우리가 알고 있는 div와 span같은 HTML코드 뿐만이 아니라, 앞으로 만들 컴포넌트도 JSX안에서 작성할 수 있다.

JSX 문법

- 감싸인 코드
- Fragment
- 자바스크립트 표현
- if문대신 조건부 연산자
- &&를 사용한 조건부 렌더링
- 인라인 스타일링
- class대신 className
- 꼭 닫아야 하는 태그
- 주석

감싸인 코드

- 부모 요소로 감싸져 있어야 한다.
  - Virtual DOM에서 컴포넌트 변화를 감지해 낼 때 효율적으로 비교하기 위해 컴포넌트 내부는 DOM트리 구조여야 한다는 규칙이 있다.

```
class App extends Component {
  render() {
    return (
      <h1>hello react</h1>
      <p>리액트가 무엇인지 알아봅시다.</p>
    );
  }
}
```

오류 발생

```
class App extends Component {
  render() {
    return (
      <div>
        <h1>hello react</h1>
        <p>리액트가 무엇인지 알아봅시다.</p>
      </div>
    );
  }
}
```

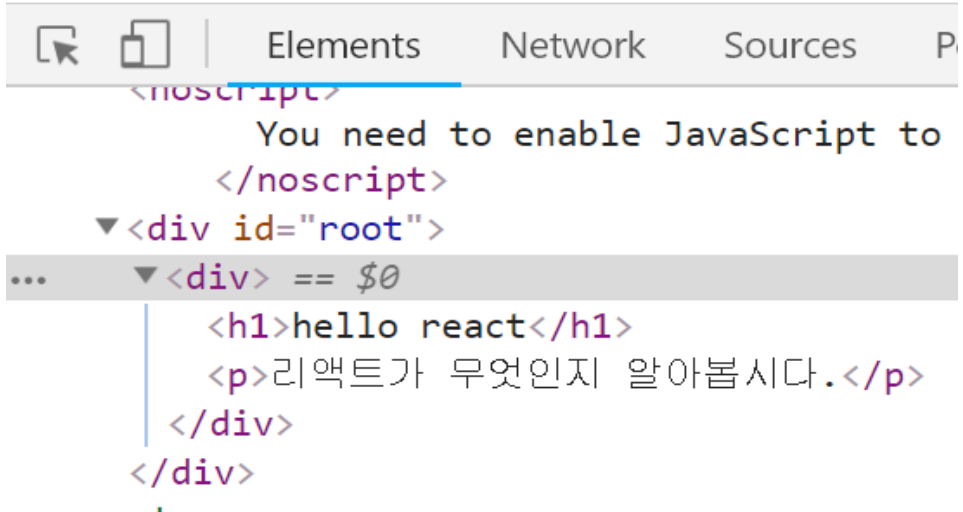
올바른 코드

Fragment

- 리액트 v16.2 에 도입.

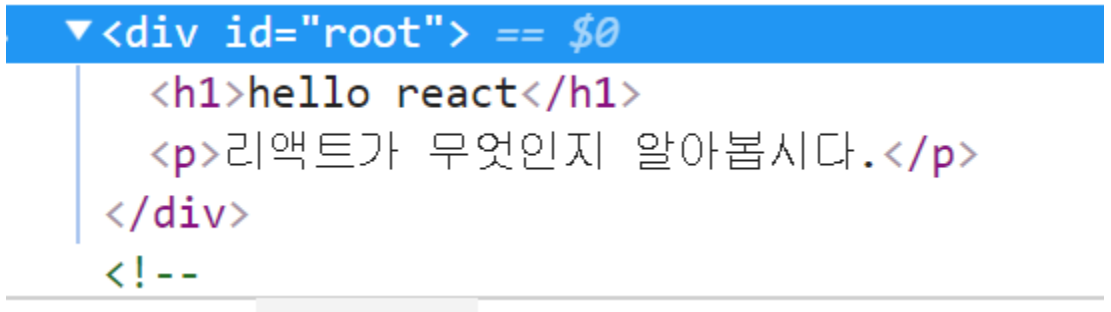
hello react

리액트가 무엇인지 알아봅시다.




```
import React, { Component ,Fragment} from 'react';
import './App.css';

class App extends Component {
    render() {
        return (
            <Fragment>
                <h1>hello react</h1>
                <p>리액트가 무엇인지 알아봅시다.</p>
            </Fragment>
        );
    }
}
```



자바스크립트 표현

- JSX안에는 자바스크립트 표현식을 사용할 수 있다. { } 사용
-

```
render() {
    const name = "kang kyung mi";
    return (
            <Fragment>
                <h1>hello {name}</h1>
                <p>리액트가 무엇인지 알아봅시다.</p>
            </Fragment>
      );
}
```

if문 대신 조건부 연산자, &&를 사용한 조건부 렌더링

- JSX 내부의 자바스크립트 표현식에는 if문을 사용할 수 없다.
  - 조건부(삼항) 연산자는 사용 가능하다.
  - {condition ? expr1:expr2}

  - {condition ? expr1 : null } == { condition && expr1 }

인라인 스타일링

```jsx
import React, { Component ,Fragment} from 'react';
import './App.css';

class App extends Component {
  render() {
    const name = "kang kyung mi";
    const style ={
      backgroundColor : 'blue',
      border : '1px solid black',
      height : '200px',
      width : '300px'
    };
    return (
      <Fragment>
        <h1>hello {name}</h1>
        <p>리액트가 무엇인지 알아봅시다.</p>
        <div style={style}></div>
      </Fragment>
    );
  }
}

export default App;
```

JSX 문법

- class대신 className
  -
- 꼭 닫아야 하는 태그
  - HTML에서 닫지않아도 문제 없던 태그들도 반드시 닫아 주어야한다.
  - JSX에서는 닫지않으면 Virtual DOM에서 트리 형태 구조를 만들지 못한다.
- 주석
  - {/* */}
  - 요소안에서는 {}없이 사용가능

```
<div //주석 /*주석 /> 닫는태크는 반드시 다음줄에 */
/>
```

컴포넌트

컴포넌트 생성

파일 만들기

↓

초기 코드 작성하기

↓

모듈 내보내고 불러오기

MyComponent.js

```
1  import React, { Component } from 'react';
2
3  class MyComponent extends Component {
4      render() {
5          return (
6              <div>
7                  나의 첫 컴포넌트^^
8              </div>
9          );
10     }
11 }
12
13 export default MyComponent;
```

App.js

```
1  import React, { Component } from 'react';
2  import MyComponent from './MyComponent';
3
4  class App extends Component {
5    render() {
6      return (
7        <MyComponent />
8      );
9    }
10 }
11
12 export default App;
13
```

props

- properties의 줄임말로 컴포넌트의 속성을 설정 할 때 사용하는 요소.
- 해당 컴포넌트를 불러와 사용하는 부모 컴포넌트에서만 설정가능
- 컴포넌트 자신은 해당 props를 읽기 전용으로만 사용 함.

```
import React, { Component } from 'react';
import MyComponent from './MyComponent';

class App extends Component {
  render() {
    return (
      <MyComponent name = 'kang kyung mi'/>
    );
  }
}

export default App;
```

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

class MyComponent extends Component {
    render() {
        return (
            <div>
                안녕하세요 제 이름은 {this.props.name} 입니다.
            </div>
        );
    }
}

export default MyComponent;
```

props 기본값 설정

- 부모 컴포넌트가 props값을 설정 하지 않았을 때 기본값 설정
  - defaultProps 설정
    - 방법 1 - 클래스 설정 바깥쪽에 아래처럼 설정

```
MyComponte.defaultProps = {
        name : '기본값'
}
```

    - 방법 2 – 클래스 내부에 정의 (transform-class-properties 문법 – ES6 stage-2)

```
static defaultsProps = {
    name : '기본값'
}
```

props 검증 : propTypes

- 필수 props를 지정하거나 props타입을 지정 할 때 사용
- 사용하기 위해서는 propTypes를 불러와야 함.
  - import PropTypes from 'prop-ypes';
    - 방법 1 - 클래스 설정 바깥쪽에 아래처럼 설정

```
MyComponte.propTypes = {
        name : propTypes.string
}
```

    - 방법 2 – 클래스 내부에 정의 (transform-class-properties 문법 – ES6 stage-2)

```
static propTypes = {
    name : propTypes.string
}
```

      - <MyComponet name={3}/> - Syntax error 발생

props 검증 : propTypes

- 필수 propTypes 설정
  - propTypes.isRequired
- propTypes 종류
  - array, bool, number, Object, string, symbol, node, element
  - instanceOf(MyClass) – 특정클래스의 인스턴스
  - one of(['a','b'] – 주어진 배열 요소 중 하나
  - oneOfType([React.PropTypes.string, React.PropTypes]) – 배열 안 종류 중 하나
  - arrayOf(React.PropTypes.number) – 주어진 종류의 값을 가진 배열
  - ObjectOf(React.PropTypes.number) – 주어진 종류의 값을 가진 객체
  - Shape({name: React.PropTypes.String, age: React.PropTypes.number}) – 주어진 스키마를 가진 객체
  - any – 아무 종류

state

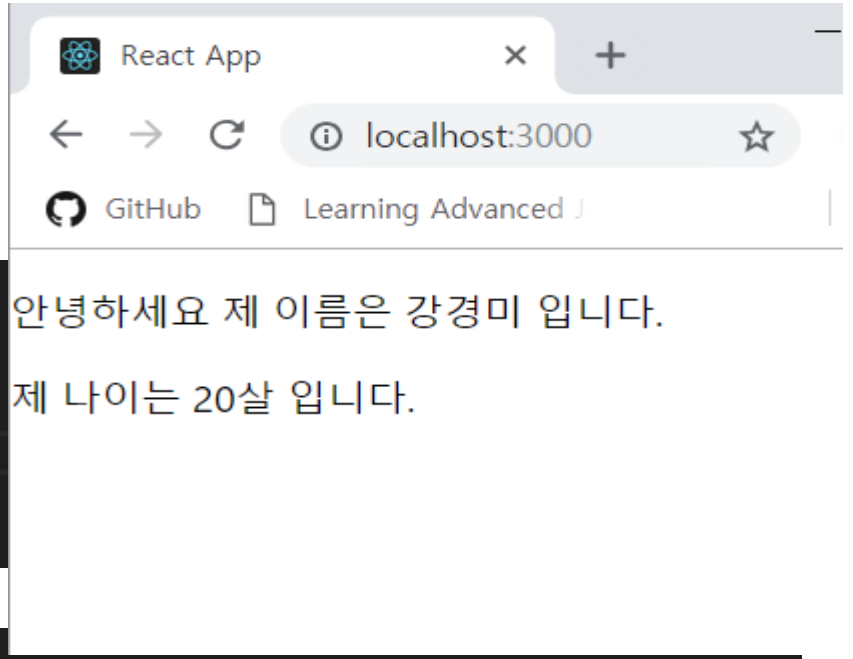
- 컴포넌트 내부에서 읽고, 업데이트 할 수 있는 값.
- 기본값을 미리 설정 할 수 있고, this.setState()메서드로만 값을 업데이트 해야함.
- 컴포넌트의 생성자 메소드인 constructor 내부에서 설정.

```
class App extends Component {
  //정의하지 않으면 부모(Component)의 생성자메서드가 그대로 사용됨.
  constructor(props){
    //부모의 constructor 메소드를 먼저 호출 해야함.
    super(props);
    //여기에 state의 초기값 설정.
  }
  render() {
    return (
```

state 사용

- state 초기값 설정

```
class MyComponent extends Component {
    constructor(props){
        super(props);
        this.state ={
          age : 20
        }
    }
```



- JSX 내부에서 state 렌더링

```
render() {
    return (
        <div>
            <p>안녕하세요 제 이름은 {this.props.name} 입니다.</p>
            <p>제 나이는 {this.state.age}살 입니다.</p>
        </div>
    );
}
```

state 값 업데이트 : setState()

```
this.setState({
   수정할 필드 이름 : 값,
   수정할 필드 이름 : 값
 )};
```

```
<div>
    <p>안녕하세요 제 이름은 {this.props.name} 입니다.</p>
    <p>제 나이는 {this.state.age}살 입니다.</p>
    <button onClick = {function(){
        this.setState({
            age : this.state.age + 1
        });
    }.bind(this)}>한 살 더하기 ㅠㅠ</button>
</div>
```

```
<div>
    <p>안녕하세요 제 이름은 {this.props.name} 입니다.</p>
    <p>제 나이는 {this.state.age}살 입니다.</p>
    <button onClick = {() => {
        this.setState({
            age : this.state.age + 1
        });
    }}>한 살 더하기 ㅠㅠ</button>
</div>
```

참고 : 화살표 함수를 사용했을 때와 일반 함수를 이용했을 때 차이를 찾아 보세요.^^

함수 안에서 사용되는 this 는 누구를 가리킬까요?

Arrows(활살표) 함수를 다시 정리해 보세요.

state 주의사항

- state는 원래 constructor 메소드에서 정의해야 하지만 transform-class-properties 문법으로 바깥에서도 정의 가능

```
state = {
        age : 20
}
```

- state값 업데이트할 때 주의 사항
  - 언제나 .setState로만 해야 한다.
    - this.state.age = this.state.number + 1; (잘못된 사용법)
    - setState() 메서드는 파라미터로 전달받은 필드를 업데이트 한 후 컴포넌트가 리렌더링 하도록 트리거 하는 역할.
- this.forceUpdate() 강제로 리렌더링하게 하는 메서드 (권장하지않음)

이벤트 핸들링

- 이벤트란 유저가 웹브라우저에서 DOM요소들과 상호 작용하는 것.
- 리액트 이벤트 시스템
  - HTML 이벤트와 인터페이스가 동일 해서 사용법이 비슷하다.
  - 이벤트 사용 할 때 주의 사항
    - 이벤트 이름은 camelCase로 작성한다.
    - 이벤트에 실행할 자바스크립트 코드를 전달하는 것이 아니라 함수형태로 전달 한다.
    - DOM요소에만 이벤트 설정이 가능하다. (직접 만든 컴포넌트에는 이벤트 설정 불가능)
  - 리액트에서 지원하는 이벤트
    - Clipboard, Form, Composition, Mouse, Keyboard, Selection, Focus, Touch, UI, Image, Wheel, Animation, Media, Transition (참고 : https://reactjs.org/docs/events.html)

이벤트 핸들링 실습

- 컴포넌트 생성 – EventExam.js 파일생성 후 초기 코드 작성
  - <h1>이벤트 실습</h1>
- App.js에서 EventExma 불러오기
  - import EventExam from './EventExam';
  - <EventExam />
- onChange 이벤트 설정

```
<div>
    <h1>이벤트 실습</h1>
    <input
        type="text"
        name="massage"
        placeholder="입력해보세요^^"
        onChange={
            (e)=>{
                console.log("이벤트 발생!!");
            }
        }
    />
```

state에 input 값 담기 (실습)

1. constructor 또는 transform-class-properties 문법으로 state 초기 값을 설정해 주세요.
2. 이벤트 핸들링 함수 내부에서 this.setState 메소드를 호출해서 state 값을 업데이트 해주세요.
3. 버튼을 하나 생성하고, 버튼에 onClick 이벤트 설정 하세요.
4. 버튼을 클릭하면 message 값을 alert()으로 띄우고 message 값은 공백으로 설정해 주세요.

state에 input 값 담기 (실습)

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

class EventExam extends Component {
    state = {
        message : ''
    }
    render() {
        return (
            <div>
                <h1>이벤트 실습</h1>
                <input
                    type="text"
                    name="massage"
                    placeholder="입력해보세요^^"
                    value={this.state.message}
                    onChange={
                        (e)=>{
                            this.setState({
                                message : e.target.value
                            })
                        }
                    }
                />
                <button onClick={
                    ()=>{
                        alert(this.state.message);
                        this.setState({
                            message : ''
                        });
                    }
                }>확인</button>
            </div>
        );
    }
}
export default EventExam;
```

임의 메서드 만들기

- 함수를 미리 준비하고 전달하기.

```
myChange(e){
    this.setState({
        message : e.target.value
    });
}
```

```
    placeholder="입력해보세요^^"
    value={this.state.message}
    onChange={this.myChange}
```

- 컴포넌트에 임의 메서드를 만들면 this에 접근 할 수 없다.
- 컴포넌트의 생성자 메서드인 constructor에 각 메서드를 this와 binding 해주어야 한다.

```
constructor(props){
    super(props);
    this.myChange = this.myChange.bind(this);
    this.myClick = this.myClick.bind(this);
}
```

Property Initializer Syntax를 사용한 메서드 작성

- 메서드 바인딩은 생성자에서 하는 것이 정석이지만, 간단하게 작성하는 방법
- transform-class-properties문법 사용

```
myChange = (e) =>{
        this.setState({
        message : e.target.value
        });
}
```

- 위와 같이 작성하면 생성자에서 바인딩 하지 않아도 됨.

- input 여러 개 핸들링하기.
  - e.target.name – 해당 input 의 name을 나타냄.
  - this.setState({ [e.target.name] : e.target.value})
- onKeyPress 이벤트 핸들링

ref

- HTML에서 id를 부여하는 것처럼 리액트에서 DOM에 이름을 부여하는 방법.
- HTML에서는 input 상자의 값을 얻어오거나 할때 DOM에 접근해서 값을 얻어옴.
- 리액트에서는 DOM에 접근하지 않고 state로 구현 가능.
  - 참고 https://jsbin.com/soyegiv/edit

ref

- 리액트에서 state 만으로 해결 할 수 없는 기능들.
  - 특정 input에 포커스 주기
  - 스크롤 박스 조작하기
  - Canvas 요소에 그림 그리기 등.

ref 사용

- 사용법
  - <input ref={(ref) => {this.input=ref}} />
  - ref 값으로는 콜백 함수를 전달한다.
  - 콜백함수는 ref를 파라미터로 가지고, 콜백함수 내부에서 컴포넌트의 멤버 변수에 ref를 담는 코드를 작성한다.
  - this.input은 input 요소의 DOM을 가리킨다.
  - ref 이름은 자유롭게 지정한다. Ex) this.carami = ref

input에 ref달기

- 버튼 클릭 후 포커스가 다시 input으로 이동
  - input에 ref 달기
    - <input ref={(ref)=>this.myInput = ref} ..... />
  - onClick 이벤트 수정
    - this.myInput.focus();

컴포넌트에 ref달기

- 컴포넌트 내부에 있는 DOM을 컴포넌트 외부에서 사용할 때 쓴다.
- 사용법
  - <MyComponent ref={(ref)=>{this.myComponent=ref}}
  - MyComponent내부의 메서드 및 멤버변수에도 접근가능
    - ex) myComponent.myClick, myComponent.myInput 등
- 정리
  - 컴포넌트 내부에서 DOM에 직접 접근할 때 만 사용한다.
  - Ref를 사용하지 않고도 원하는 기능을 구현 할 수 있는지를 꼭 고려한 후에 사용한다.

ScrollBox.js

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

class ScrollerBox extends Component {
    scrollToBottom = () =>{
        const {scrollHeight, clientHeight} = this.box;
        this.box.scrollTop = scrollHeight - clientHeight;
    }
    render() {
        const style={
            border : '1px solid black',
            height : '200px',
            width : '200px',
            overflow : 'auto',
            position : 'relative'
        }
        const innerStyle = {
            width : '100%',
            height : '500px',
            background :'linear-gradient(white, black)'
        }
        return (
            <div
                style = {style}
                ref = {(ref)=>{this.box=ref}}
            >
                <div style={innerStyle}></div>
            </div>
        );
    }
}

export default ScrollerBox;
```

App.js

```
import React, { Component } from 'react';
import ScrollBox from './ScrollBox';
class App extends Component {

  render() {
    return (
      <div>
        <ScrollBox ref={(ref)=>this.ScrollBox=ref}/>
        <button
          onClick = {()=>this.ScrollBox.scrollToBottom()}
          >맨밑으로 이동</button>
      </div>
    );
  }
}

export default App;

```

컴포넌트 반복

- 웹 애플리케이션을 만들다 보면 반복되는 코드를 작성하게 된다.

```
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

  - 위처럼 반복적인 내용을 효율적으로 보여주고 관리하는 방법들을 알아본다.

자바스크립트 배열의 map() 함수

- Map 함수는 파라미터로 전달된 함수를 사용해서 배열 내 각 요소를 가공한 후 그 결과로 새로운 배열을 생성한다.
- 문법
  - arr.map(callback, [thisArg])
    - callback : 새로운 배열의 요소를 생성하는 함수
      - - currentValue : 현재 처리하고 있는 요소
      - - index : 현재 처리하고 있는 요소의 index 값
      - - array : 현재 처리하고 있는 원본 배열
    - thisArg(선택항목) : callback 함수 내부에서 사용할 this 레퍼런스

Map 함수 사용 방법

```
const firstNames = ['kim','kang','hong','lee'];
let fullNames=firstNames.map(function(firstName){
        return firstName+" kyung mi";
});
console.log(fullNames);

//화살표함수 이용
const firstNames = ['kim','kang','hong','lee'];
let fullNames=firstNames.map((firstName)=>firstName+" kyung mi");
console.log(fullNames);
```

실습 : 배열 [12,3,4,5] 의 각 요소에 제곱해서 새로운 배열을 생성해 보세요.

데이터 배열을 컴포넌트 배열로 map하기

IterationExam.js

```
1  import React, { Component } from 'react';
2
3  class IterationExam extends Component {
4      render() {
5          const names = ['봄봄','삐삐','야옹이','멍멍이'];
6          const nameList = names.map(
7              (name) => (<li>{name}</li>)
8          )
9          return (
10             <ul>
11                 {nameList}
12             </ul>
13         );
14     }
15 }
16
17 export default IterationExam;
```

App.js

```
1  import React, { Component } from 'react';
2  import IterationExam from './IterationExam';
3  class App extends Component {
4
5    render() {
6      return (
7        <div>
8          <IterationExam />
9        </div>
10     );
11   }
12 }
13
14 export default App;
15
```

원하는대로 렌더링은 되었지만 오류가 발생함.

Warning: Each child in an array or iterator should have a unique "key" prop.

key

- 컴포넌트 배열을 렌더링 했을 때 어떤 원소에 변동이 있었는지 알아내기 위해서 사용한다.
- Key가 없 때는 virtual DOM의 비교과정에서 순차적으로 비교하면서 변화를 감지한다.
- Key를 이용해서 이 값을 사용하여 어떤 변화가 있는지를 더욱 빠르게 알아 낼 수 있다.

key 설정

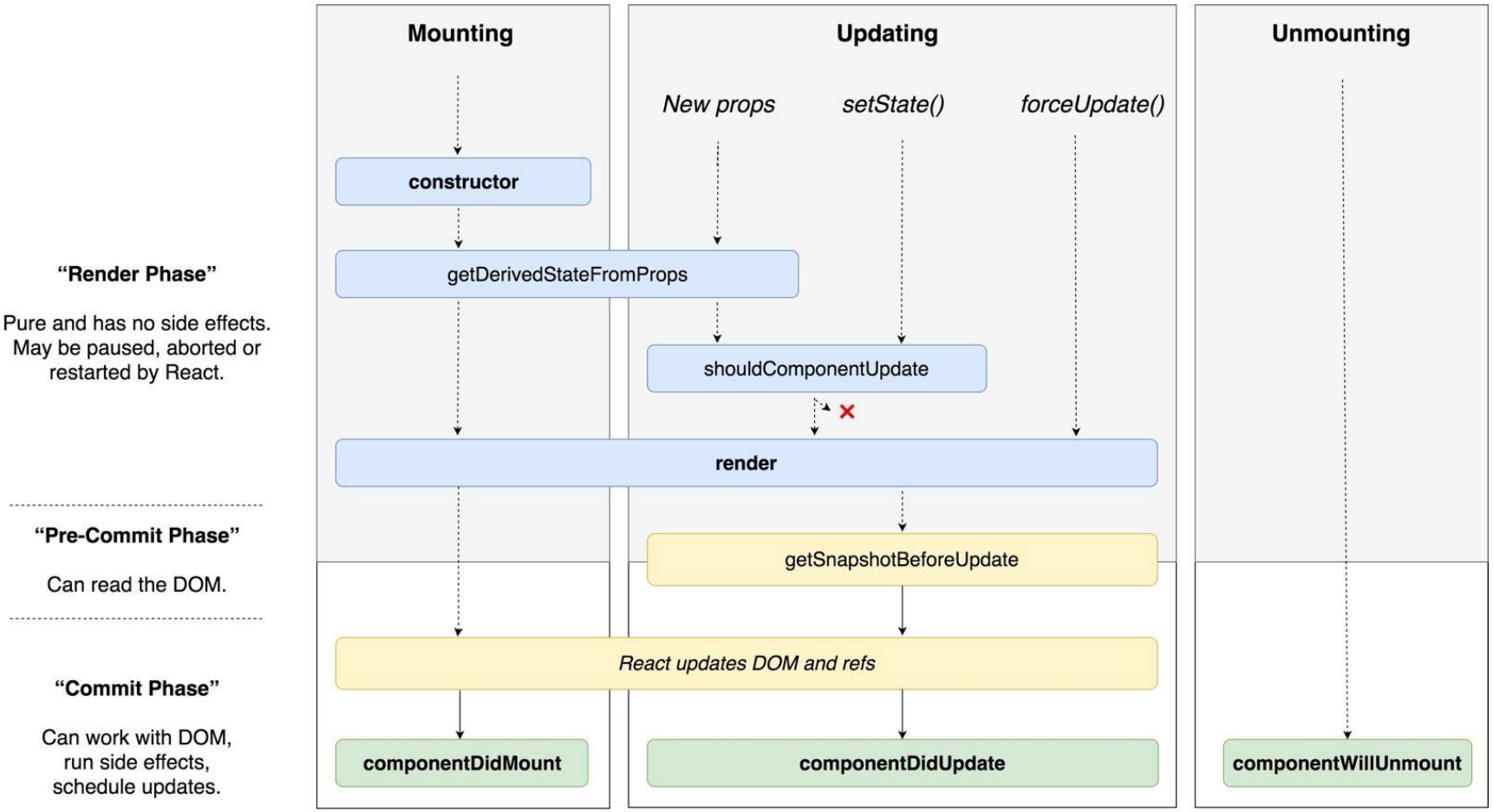
- map함수의 인자로 전달되는 함수 내부에서 설정한다.
- key값은 항상 유일해야 한다.
  - 데이터가진 고유한 값을 key로 설정한다. ex) 상품목록을 렌더링 한다면 상품코드로 설정한다.
  - 고유번호가 없을 때에는 map 함수에 전달되는 콜백 함수의 인수인 index 값을 사용한다.

```
const nameList = names.map(
        (name,index) => (<li key={index}>{name}</li>)
);
```

실습

- state에 초기 데이터 담기
  – 상수에 담았던 names 배열을 state에 담아 보세요.
- 데이터 추가 기능 구현
  - 리스트 위쪽에 input 과 button 을 렌더링하고, 이벤트 핸들러 메서드를 구현하세요.
- 데이터 제거 기능 구현
  - 아이템을 두번 클릭하면 제거 하도록 구현 해 보세요. (onDoubleClick)
-
  - – 상수에 담았던 names 배열을 state에 담아 보세요.

컴포넌트 생명주기


"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.
Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

컴포넌트 생명주기

- 모든 리액트 컴포넌트는 생명주기가 존재한다.
  - 컴포넌트는 페이지에 렌더링되기 전 준비과정에서 시작해서 페이지에서 사라질 때 끝난다.
- 라이프사이클 메서드의 이해
  - 총 12가지 메서드로 구성된다.
  - Will 접두사가 붙은 메서드는 어떤 작업을 작동하기 전에 실행된다.
  - Did 접두사가 붙은 메서드는 어떤 작업을 작동한 후에 실행 된다.
  - 이 메서드들은 우리가 컴포넌트 클래스에서 덮어써서 선언하여 사용한다.

컴포넌트 생명주기

- 총 세가지 카테고리로 나뉜다.
  - 마운트
    - DOM이 생성되고 웹 브라우저상에 나타나는 것을 의미한다.
    - 마운트 할 때 호출 되는 메서드
      - constructor : 컴포넌트를 새로 만들 때 마다 호출되는 클래스 생성자 메서드
      - getDerivedStateFromProps : props에 있는 값을 state에 동기화하는 메서드
      - render : 우리가 준비한 UI를 렌더링 하는 메서드
      - componentDidMount : 컴포넌트가 웹 브라우저상에 나타난 후 호출 되는 메서드

컴포넌트 생명주기

- 총 세가지 카테고리로 나뉜다.
  - 업데이트
    - 컴포넌트를 업데이트하는 경우
      - props가 바뀔 때
      - state가 바뀔 때
      - 부모 컴포넌트가 리렌더링될 때
      - This.forceUpdate로 강제로 렌더링을 트리거 할 때
    - 업데이트 할 때 호출 되는 메서드
      - getDerivedStateFromProps : props가 바뀌어서 업데이트 할 때 호출
      - shouldComponentUpdate : 컴포넌트가 리렌더링을 할지 말지 결정하는 메서드. False를 반환하면 아래 메서드들을 호출하지 않음.
      - Render : 컴포넌트를 리렌더링 합니다.
      - getSnapshotBeforeUpdate : 컴포넌트 변화를 DOM에 반영하기 바로 직전에 호출하는 메서드
      - componentDidUpdate : 컴포넌트의 업데이트 작업이 끝난 후 호출 되는 메서드

컴포넌트 생명주기

- 총 세가지 카테고리로 나뉜다.
  - 언마운트
    - 마운트의 반대 과정
    - 컴포넌트를 DOM에서 제거 하는 것을 의미 한다.
    - 언마운트 할 때 호출 되는 메서드
      - componentWillUnmount : 컴포넌트가 웹 브라우저상에서 사라지기 전에 호출 되는 메서드

컴포넌트 생명주기

- render()함수
  - 컴포넌트의 모양새를 정의한다.
  - 라이프 사이클 메서드 중 유일한 필수 메서드 이다.
  - 이 메서드 안에서 this.props와 this.state에 접근 할 수 있다.
  - 리액트 요소를 반환한다.
  - 아무것도 보여주고 싶지 않다면 null 이나 false를 반환 한다.
  - 주의
    - 이 메서드 안에서는 절대로 state를 변형해서는 안되고, 웹브라우저에 접근해도 안된다.
    - DOM정보에 변화를 줄 때는 componentDidMount에서 처리한다.

컴포넌트 생명주기

- constructor메서드
  - 컴포넌트의 생성자메서드로 컴포넌트를 만들 때 처음 실행된다.
  - 초기 state를 정할 수 있다.
- getDerivedStateFromProps 메서드
  - v16.3이후에 추가된 메서드 이다.
  - props로 받아 온 값을 state에 동기화 시키는 용도로 사용한다.
  - 컴포넌트를 마운트하거나 props를 변경할 때 호출된다.
  - 사용방법

```
static getDerivedStateFromProps(nextProps, prevState){
    if(nextProps.value !== prevState.value){
            return {value : nextProps.value};
     }
     return null;  //state가 변경할 필요가 없을 때 null 반환함.
}
```

컴포넌트 생명주기

- componentDidMount 메서드
  - componentDidMount(){ }
  - 컴포넌트를 만들고 첫 렌더링을 다 마친 후 실행된다.
  - 다른 자바스크립트 라이브러리 또는 프레임워크의 함수를 호출하거나 이벤트 등록, setTimeOut, setInterval, 네트워크 요청 같은 비동기 작업을 처리한다.
- shouldComponentUpdate 메서드
  - shouldComponentUpdate(nextProps, nextState){ }
  - props 또는 state를 변경 했을 때 리렌더링을 시작할지 여부를 결정하는 메서드이다.
  - 반드시 true 또는 false 를 반환 한다. 기본은 true를 반환 한다.
  - false 가 반환되면 업데이트 과정을 중지 한다.

컴포넌트 생명주기

- getSnapshotBeforeUpdate 메서드 (v16.3 이후 추가)
  - render 메서드를 호출 한 후 DOM에 변화를 반영하기 바로 직전에 호출 되는 메서드 이다.
  - 반환하는 값은 componentDidUpdate 에서 세번째 파라미터인 snapshot 값을 전달 받을 수 있다.
  - 주로 업데이트 직전의 값을 참고 할 일이 있을 때 활용된다.

```
getSnapshotBeforeUpdate(prevProps, prevState){
    if(prevState.arr !== this.state.array){
        const { scrollTop, scrollHeight } = this.list
        return { scrollTop, scrollHeight };
    }
}
```

컴포넌트 생명주기

- componentDidUpdate 메서드
  - componentDidUpdate(prevProps, prevState, snapshot){ }
  - 리렌더링이 완료된 후 실행된다.
  - 업데이트가 끝난 직후이므로 DOM관련 처리를 해도 된다.
  - prevProps, prevState를 사용하여 컴포넌트가 이전에 가졌던 데이터에 접근 가능하다.
  - getSnapshotBeforeUpdate 가 반환한 값이 있다면 snapshot 값을 전달받을 수 있다.
- componentWillUnMount 메서드
  - componentWillUnMount(){ }
  - 컴포넌트를 DOM에서 제거할 때 실행된다.
  - componentDidMount에서 등록한 이벤트, 타이머, 직접생성한 DOM등을 여기서 제거 한다.

컴포넌트 생명주기

- 실습

함수형 컴포넌트

- 라이프사이클, state등 불필요한 기능을 제거한 컴포넌트이다.
- 컴포넌트가 라이프사이클 API와 state를 사용할 필요가 없고, 오로지 props를 전달받아 뷰를 렌더링 하는 역할만 할 때 사용한다.
- 리액트 v16 이상에서는 함수형 컴포넌트 성능이 조금 더 좋다.
- state나 라이프사이클 API를 꼭 써야 할때 만 클래스 형태로 컴포넌트를 작성하는 것이 좋다.

함수형 컴포넌트 사용법

```
import React from 'react';
function Hello(props){
        return (
                <div>Hello {props.name}</div>
        )
}
export default Hello;
```

함수형 컴포넌트 사용법

- 화살표 함수 이용

```
import React from 'react';
const Hello = ({name}) =>{
        return (
            <div>Hello {props.name}</div>
        )
}
export default Hello;
```

- 위와 동일함.

```
const Hello = ({name}) => (<div>Hello {props.name}</div>)
```

컴포넌트 스타일링

CSS Module

- CSS를 모듈화 하여 사용하는 방식이다.
- CSS 클래스를 만들면 자동으로 고유한 클래스네임을 생성하여 스코프를 지역적으로 제한 한다.
- 모듈화된 CSS를 webpack으로 불러오면 사용자가 정의한 클래스 네임과 고유화된 클래스네임으로 구성된 객체를 반환 한다.
  - Ex) { box : 'src-App__box—mjrNr' }
- 클래스를 적용할 때는 className = {styles.box} 로 사용한다.

Sass

- Syntactically awesome style sheets 의 약어로 문법적으로 매우 멋진 스타일 시트를 의미한다.
- CSS에서 사용할 수 있는 문법을 확장하여 중복되는 코드를 줄여 더욱 보기 좋게 작성 할 수 있다.
- 참고 : htpps://sass-guidelines/ko/

Sass 적용

- 리액트 프로젝트에 Sass를 사용하려면 node-sass , sass-loader 설치 해야한다.
  - Yarn add node-sass sass-loader
  - sass-loader는 webpack에서 Sass 파일을 읽어온다.
  - node-sass 는 Sass로 작성된 코드들을 CSS로 변환한다.
  - sass-loader를 적용하려면 webpack 환경설정에서 css-loade에 설정한 내용들을 동일하게 복사하고 설정 아래쪽에 sass-loader 부분을 추가한다.