

iOS개발실무

3주차

담당: 김희숙

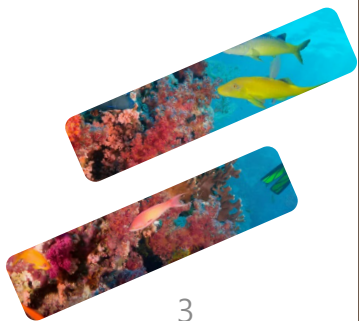
(jasmin11@hanmail.net)

리액트(React)

담당: 김희숙
(jasmin11@hanmail.net)

JSX

담당: 김희숙
(jasmin11@hanmail.net)

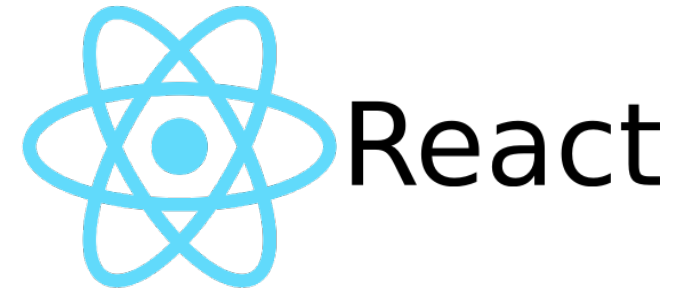


[요약] React.js

컴포넌트

❖ 리액트(React.js)

- 페이스북(Facebook)에서 개발
- View 전용 라이브러리
- 가상 돔(Virtual DOM) 사용
- JSX
 - return 안에 태그를 사용
 - JSX 통해 DOM 을 생성해 준다
- 컴포넌트
 - 장점) 코드 재사용, 직관적인 코드 구조
- 컴포넌트 + 재사용 + 백엔드(Back-end) 와의 분리




[요약] JSX

JSX

- ❖ JSX: 자바스크립트 확장문법
 - ❖ XML 과 매우 비슷
 - ❖ 브라우저에서 실행되기 전에 코드가 번들링되는 과정에서
바벨(Babel) 사용하여 일반 자바스크립트 형태의 코드로 변환

```
function App() {  
  return (  
    <div>  
      Hello <b>react</b>  
    </div>  
  );  
}
```

// JSX
// 만약 컴포넌트를 렌더링할 때마다 JSX 작성하는 것이 아니라
// 매번 React.createElement 함수를 사용해야 한다면 매우 불편
즉, **JSX** 사용하여 **간편하게 UI 렌더링** 가능



```
function App() {  
  return React.createElement("div", null, "Hello ", React.createElement("b", null, "react"));  
}
```

// 브라우저에서 실행되기 전에 코드가 번들링 되는 과정에서
// 바벨을 사용하여 일반 자바스크립트 형태의 코드로 변환된다

[문법] JSX (감싸인 요소)

JSX

JSX 예제

`npx create-react-app react-ex02`

`cd react-ex`

`npm start`

-- 오류

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

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

`export default App;`



-- 성공

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

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

`export default App;`

리액트 안녕!

잘 작동하니?

[문법] JSX (감싸인 요소) Fragment

JSX

```
-- 성공
// ./src/App.js
import React from 'react';
import './App.css';

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

export default App;
```



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

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

export default App;
```

// Fragment

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

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

export default App;
```

// Fragment

리액트 안녕!

잘 작동하니?

// 리액트 컴포넌트는 루트 요소를 무조건 하나 가져야 한다

// <Fragment>로 묶어주면
// 실제로 렌더링 될 때 추가 태그가 생성되지 않는다
// 줄여서 <></>로 사용 권장

[문법] JSX (자바스크립트 표현식)

JSX

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

function App() {
  const name = "React";
  return (
    <>
      <h1>{name} 안녕!</h1>
      <h2>잘 작동하니?</h2>
    </>
  );
}
```

React 안녕!
잘 작동하니?

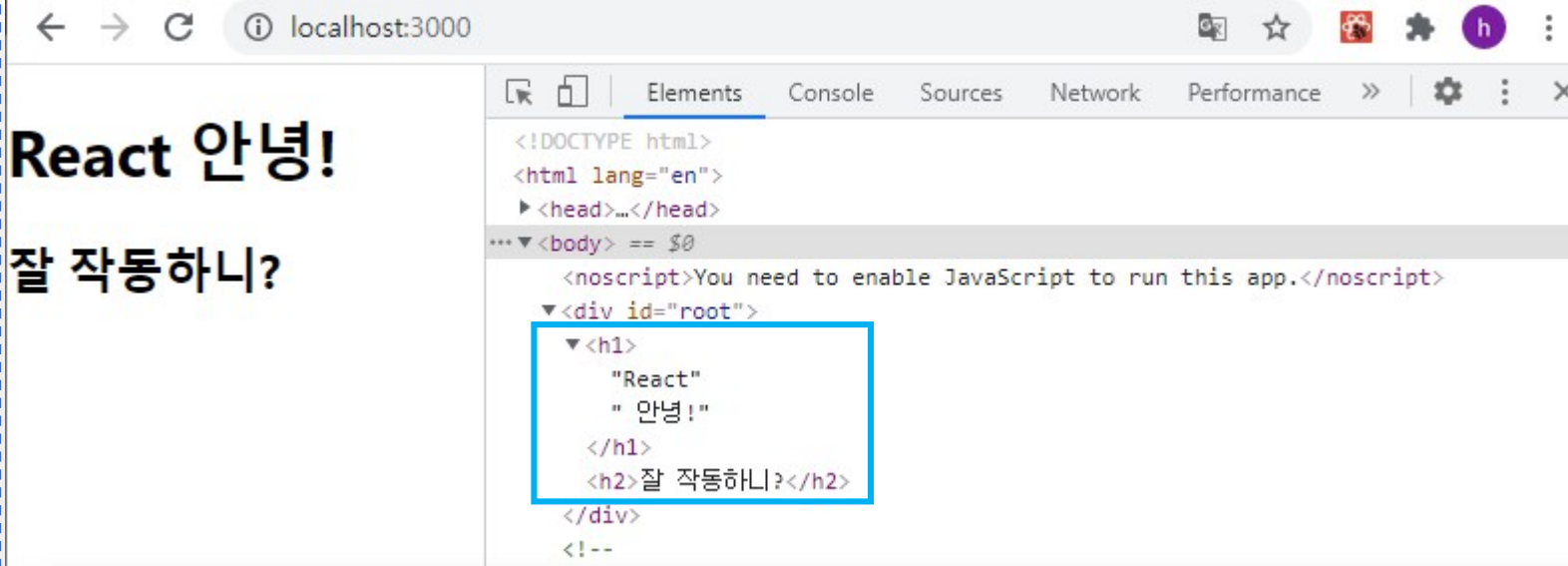
// JSX는 단순히 DOM 요소를 렌더링하는 기능
// 추가로 자바스크립트 표현식 사용 가능

[문법] JSX (자바스크립트 표현식)

JSX

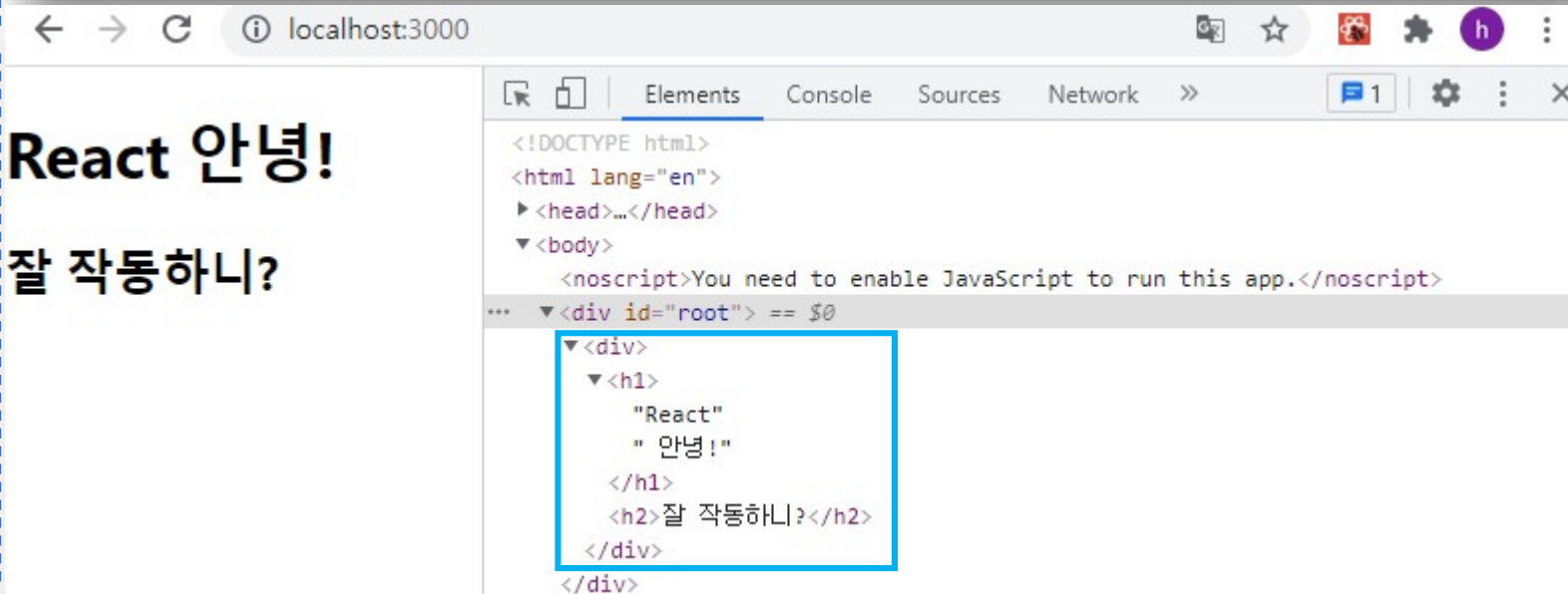
```
// ./src/App.js
import React from 'react';
import './App.css';

function App() {
  const name = "React";
  return (
    <>
      <h1>{name} 안녕!</h1>
      <h2>잘 작동하니?</h2>
    </>
  );
}
```



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

function App() {
  const name = "React";
  return (
    <div>
      <h1>{name} 안녕!</h1>
      <h2>잘 작동하니?</h2>
    </div>
  );
}
```



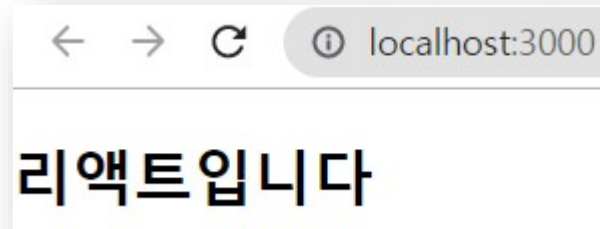
[문법] JSX (조건부 연산자)

JSX

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

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

export default App;
```



```
// JSX 내부 자바스크립트 표현식에 if문 사용할 수 없다
// 조건에 따라 다른 내용 렌더링할 때
// JSX 밖에서 if문 사용하여 미리 값을 설정하거나
// { } 안에 조건부 연산자를 사용한다
```

```
// 삼항연산자 (조건식) ? true : false
```

[문법] JSX (AND 연산자 &&)

JSX

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

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

export default App;
```

// 조건부 렌더링

// 특정 조건을 만족할 때 내용을 보여주고,
// 만족하지 않을 때 아무것도 렌더링하지 않는다

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

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

export default App;
```

// 주의) falsy한 값인 0은 예외적으로 화면에 나타난다
const number = 0;
return number && <div>내용</div>

← → ↻ ⓘ localhost:3000

리액트입니다

[문법] JSX (인라인 스타일링)

JSX

```
// ./src/App.js
import React, from 'react';
import './App.css';

function App() {
  const name = "리액트";
  const style = {
    backgroundColor: 'black',
    color: 'aqua',
    fontSize: '48px',
    fontWeight: 'bold',
    padding: 16
  };

  return (
    <div style = {style}>
      {name}
    </div>
  );
}

export default App;
```

// 인라인 스타일링



// 리액트에서 DOM 요소에 스타일 적용할 때
// 문자열 형태로 넣는 것이 아니라 **객체 형태**로 넣어준다
// background-color 처럼 - 문자가 포함되는 경우 **카멜 표기법**으로 작성

[문법] JSX (class 대신 className)

JSX

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

function App() {
  const name = "리액트";

  return (
    <div className="react">
      {name}
    </div>
  );
}

export default App;
```

// class 대신 className

```
// ./src/App.css
/* JSX 스타일 추가 */
.react {
  background: aqua;
  color: black;
  font-size: 48px;
  font-weight: 'bold';
  padding: 16px
};
```

[문법] JSX (class 대신 className)

JSX

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

function App() {
  const name = "리액트";

  return (
    <div className="react">
      {name}
    </div>
  );
}

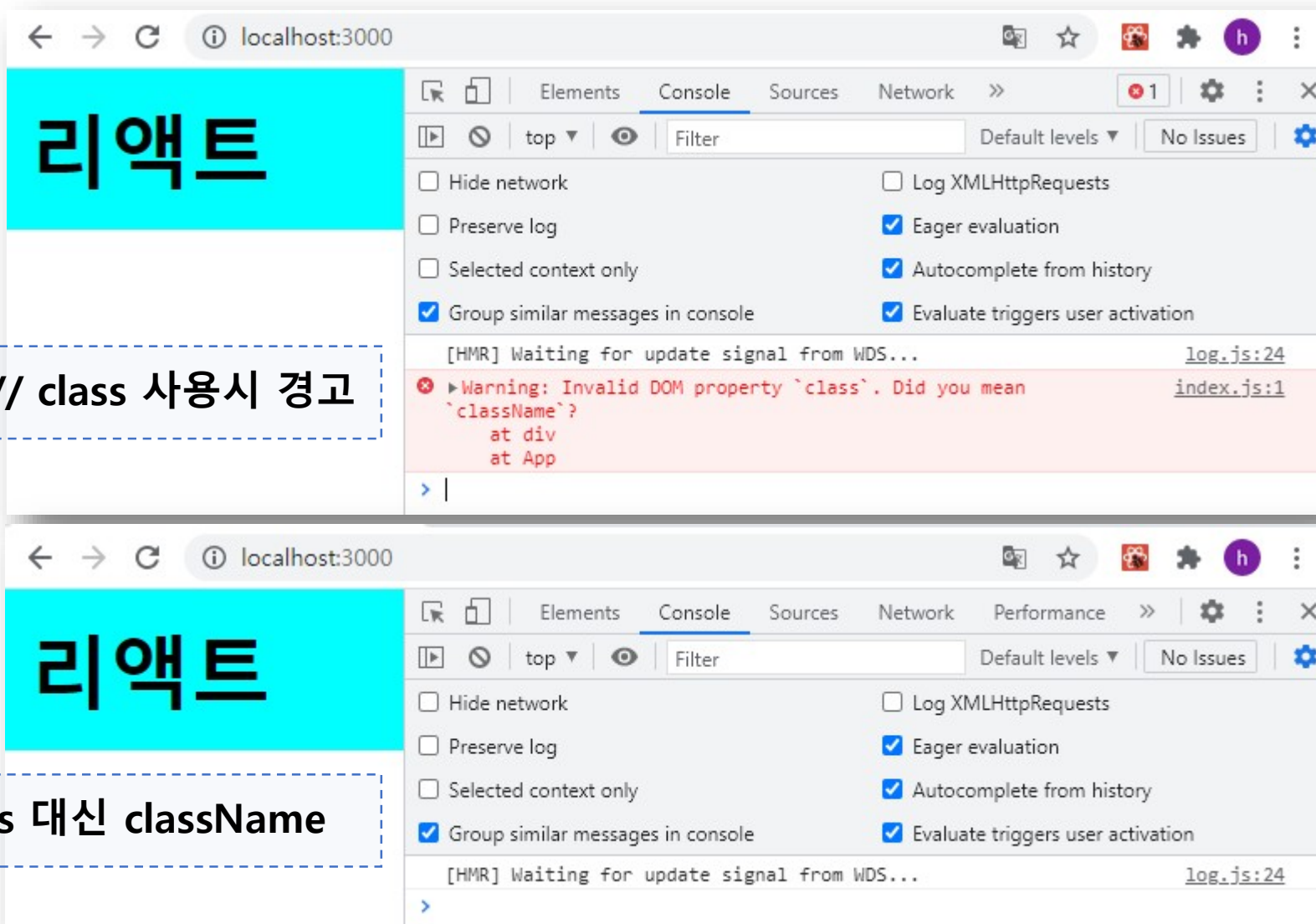
export default App;
```

리액트

// class 사용시 경고

리액트

// class 대신 className



[문법] JSX (꼭 달아야 하는 태그)

JSX

// 오류

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

```
function App() {
  const name = "리액트";

  return (
    <>
      <div className="react">
        {name}
      </div>
      <input>
    </>
  );
}
```

```
export default App;
```

// 성공

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

```
function App() {
  const name = "리액트";

  return (
    <>
      <div className="react">
        {name}
      </div>
      <input />
    </>
  );
}
```

```
export default App;
```



// JSX는 반드시 닫기 태그도 작성해야 한다

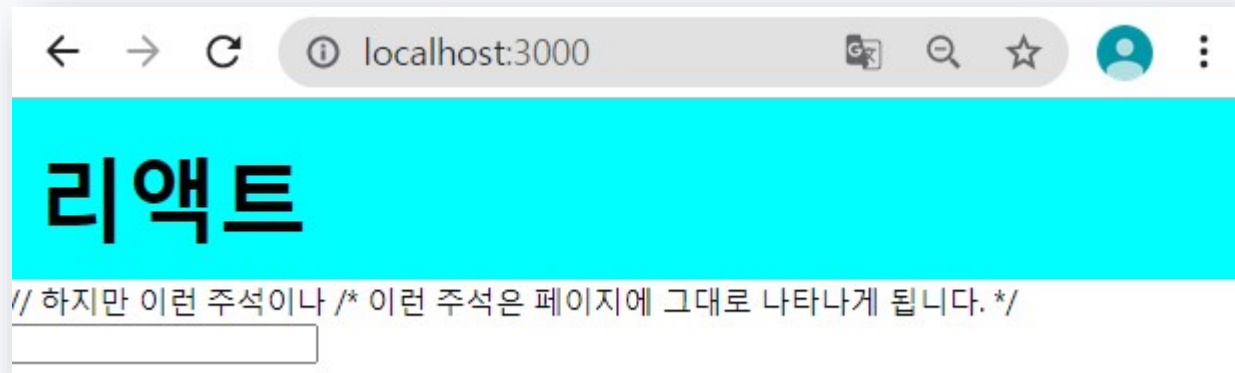
[문법] JSX (주석)

JSX

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

```
function App() {
  const name = '리액트';
  return (
    <>
      {/* 주석은 이렇게 작성합니다. */}
      <div
        className="react" // 시작 태그를 여러 줄로 작성하게 된다면 여기에 주석을 작성 할 수 있다
      >
        {name}
      </div>
      // 하지만 이런 주석이나 /* 이런 주석은 페이지에 그대로 나타나게 됩니다. */
      <input />
    </>
  );
}

export default App;
```



[요약] JSX

JSX

❖ JSX: 자바스크립트 확장문법

- ❖ 감싸인 요소: 여러 요소가 있다면 반드시 부모 요소 하나로 감싸야 한다
 - ⇒ 가상DOM에서 컴포넌트 변화를 감지해 낼 때 효율적으로 비교할 수 있도록 컴포넌트 내부는 하나의 DOM 트리구조로 작성
- ❖ 자바스크립트 표현: JSX 안에서 자바스크립트 표현식 가능
 - ⇒ { } 로 감싼다
- ❖ If 문 대신 조건부 연산자: JSX 내부의 자바스크립트 표현식에서 if 문 사용불가능
- ❖ 조건부 렌더링: AND 연산자(&&)
- ❖ class 대신 className: css 클래스 사용할 때는 className 으로 사용
- ❖ 꼭 달아야 하는 태그: 태그를 닫지 않으면 오류 발생
- ❖ JSX 주석

javascript: (React)

담당: 김희숙
(jasmin11@hanmail.net)



```
const singer = {  
  name: 'gdHong',  
  age: '23',  
  country: 'Korea'  
};
```

```
// 속성값을 각각 다른 변수에 저장  
const name = singer.name;  
const city = singer.city;  
const country = singer.country;
```

```
// 비구조화 할당  
// name, age, country 각 변수에  
// singer 속성 name, age, country를 차례로 할당
```

```
const { name, age, country } = singer;
```

// ES6

// 객체 안에 있는 필드 값을
원하는 변수에 대입
(배열이나 객체 속성을 분해하여
그 값을 개별 변수에 저장)

구조분해 할당

```
var data1 = [1, 2, 3];  
var data2 = data1;
```

```
var array1 = [4, 5, 6];  
var newArray2 = [...array1];
```

// ES6

// ECMAScript 2015

spread

```
const animals=['개', '고양이', '참새'];  
const anotherAnimals=[...animals,'비둘기'];
```

spread

//배열, 문자열과 같이 반복가능한 문자
를 인수 목록으로 확장

// 배열, 객체를 결합, 배열에 항목추가,
배열을 함수의 인수로 확산

```
const animalsOne = ['개', '고양이', '참새'];  
const animalsTwo = ['병아리', '돌고래']
```

```
// 배열 animalsOne과 배열 animalsTwo 합친다  
const combinedOne = animalsOne.concat(animalsTwo);  
const combinedTwo = [...animalsOne, ...animalsTwo];
```

```
// combinedOne 과 combinedTwo 의 결과값에는 차이가 없다  
// combinedTwo 에는 다른 string 추가 가능
```

```
const combinedTwo = [...animalsOne, '비둘기', ...animalsTwo];
```

```
// 스프레드연산자(...)사용해 배열 가지고 오기  
// clone 이라는 변수에 animalsOne 의 배열을  
가져오기
```

```
const clone = [...animalsOne];  
console.log(clone);
```

spread

//배열, 문자열과 같이 반복가능한 문자
를 인수 목록으로 확장

// 배열, 객체를 결합, 배열에 항목추가,
배열을 함수의 인수로 확산

```
// 객체(Object) 에서 스프레드연산자(...) 사용
```

```
const singerName = {  
  name: '신아로미'  
}
```

```
const singerAge = {  
  age: '20'  
}
```

```
const combinedSinger = {  
  ...singerName, ...singerAge, birthplace: '대한민국'  
};
```

map()

```
// map()
const books = ['Harry Potter', 'Anne of green gables', 'Lord of the rings']
const booklist = books.map(book => `<li>${book}</li>`);
console.log(booklist);
```

```
// [설명]
// 요소 'Harry Potter', 'Anne of green gables', 'Lord of the rings' 를
// 차례대로 돌면서 순서대로 리스트를 나열
```

```
> const books = ['Harry Potter', 'Anne of green gables', 'Lord of the rings']
  const booklist = books.map(book => `<li>${book}</li>`);
  console.log(booklist);

react devtools backend.js:4049
▶ (3) ['<li>Harry Potter</li>', '<li>Anne of green gables</li>', '<li>Lord of
  the rings</li>']
< undefined
>
```

[실습] 구조 분해(destructuring)

javascript

➤ 구조분해: 객체 안에 있는 필드 값을 원하는 변수에 대입

❖ 구조 분해를 사용한 대입(객체)

```
var sandwich = {  
  bread: "더치크런치",  
  meat: "참치",  
  cheese: "스위스",  
  toppings: ["상추", "토마토", "머스타드"]  
};
```

```
var {bread, meat} = sandwich;  
console.log(bread, meat); // 더치크런치 참치
```

```
> var sandwich = {  
  bread: "더치크런치",  
  meat: "참치",  
  cheese: "스위스",  
  toppings: ["상추", "토마토", "머스타드"]  
};
```

```
var {bread, meat} = sandwich;  
console.log(bread, meat); // 더치크런치 참치
```

더치크런치 참치

VM13:9

< undefined

>

❖ 두 변수를 변경해도 원래 필드 값은 불변

```
var {bread, meat} = sandwich;
```

```
bread = "마늘";  
meat = "칠면조";
```

```
console.log(bread, meat); // 마늘 칠면조  
console.log(sandwich.bread, sandwich.meat);  
// 더치크런치, 참치
```

```
> var {bread, meat} = sandwich;
```

```
bread = "마늘";  
meat = "칠면조";
```

```
console.log(bread, meat); // 마늘 칠면조  
console.log(sandwich.bread, sandwich.meat);  
// 더치크런치, 참치
```

마늘 칠면조

VM23:6

더치크런치 참치

VM23:7

< undefined

>

[실습] 스프레드 연산자(spread)

javascript

➤ 스프레드 연산자: 점 3개(...)로 이루어진 연산자(배열의 내용을 조합)

❖ 스프레드 연산자

```
var [firstResort] = ["용평","평창","강촌"]
```

```
console.log(firstResort)
```

```
// 배열을 구조 분해해서 원소의 값을 변수에 대입  
// 배열의 첫 번째 원소를 변수에 대입
```

❖ 스프레드 연산자

```
var [,thirdResort] = ["용평","평창","강촌"]
```

```
console.log(thirdResort)
```

```
// 리스트 매칭(list matching): 불필요한 값을 콤마(,) 사용해 생략  
// 무시하고 싶은 원소 위치에 콤마를 넣으면 리스트 매칭
```

```
// 첫 두 원소를 콤마로 대체
```

[실습] 스프레드 연산자(spread)

javascript

➤ **스프레드 연산자**: 점 3개(...)로 이루어진 연산자(배열의 내용을 조합)

❖ 스프레드 연산자

```
var peaks = ["대청봉", "중청봉", "소청봉"]  
var canyons = ["천불동계곡", "가야동계곡"]  
var seoraksan = [...peaks, ...canyons]  
  
console.log(seoraksan.join(', '))  
  
// 대청봉,중청봉,소청봉,천불동계곡,가야동계곡
```

```
> var peaks = ["대청봉", "중청봉", "소청봉"]  
var canyons = ["천불동계곡", "가야동계곡"]  
var seoraksan = [...peaks, ...canyons]  
  
console.log(seoraksan.join(', '))  
대청봉, 중청봉, 소청봉, 천불동계곡, VM53:5  
가야동계곡  
< undefined  
>
```

❖ 스프레드 연산자 (원본 배열 변경)

```
// .reverse()가 peaks 배열을 변경함  
  
var peaks = ["대청봉", "중청봉", "소청봉"]  
var [last] = peaks.reverse()  
  
console.log(last) // 소청봉  
console.log(peaks.join(', ')) // 소청봉,중청봉,대청봉
```

❖ 스프레드 연산자 (배열 복사본 사용)

```
// peaks를 스프레드 연산자로 복사한 후 reverse 수행  
  
var peaks = ["대청봉", "중청봉", "소청봉"]  
var [last] = [...peaks].reverse()  
  
console.log(last) // 소청봉  
console.log(peaks.join(', ')) // 대청봉,중청봉,소청봉
```

[실습] 스프레드 연산자(spread)

javascript

➤ **스프레드 연산자**: 점 3개(...)로 이루어진 연산자(배열의 내용을 조합)

❖ 스프레드 연산자

// 객체에 대한 스프레드 연산자

```
var morning = {  
  breakfast: "미역국",  
  lunch: "삼치구이와 보리밥"  
}
```

```
var dinner = "스테이크 정식"
```

```
var backpackingMeals = {  
  ...morning,  
  dinner  
}
```

```
console.log(backpackingMeals)
```

❖ 스프레드 연산자

> // 객체에 대한 스프레드 연산자

```
var morning = {  
  breakfast: "미역국",  
  lunch: "삼치구이와 보리밥"  
}
```

```
var dinner = "스테이크 정식"
```

```
var backpackingMeals = {  
  ...morning,  
  dinner  
}
```

```
console.log(backpackingMeals)
```

react devtools backend.js:4049

▶ {breakfast: '미역국', lunch: '삼치구이와 보리밥', dinner: '스테이크 정식'}

< undefined

> |

key

담당: 김희숙
(jasmin11@hanmail.net)



[실습] 반복(IterationSample.js)

map()

❖ IterationSample.js (수정 전)

```
// ./src/Components/IterationSample.js
import React from 'react';

const IterationSample = () => {
  return (
    <ul>
      <li>눈사람</li>
      <li>얼음</li>
      <li>눈</li>
      <li>바람</li>
    </ul>
  );
};

export default IterationSample;
```

- 눈사람
- 얼음
- 눈
- 바람

❖ IterationSample.js (수정 후)

```
// ./src/Components/IterationSample.js
import React from 'react';

const IterationSample = () => {
  const names = ['눈사람', '얼음', '눈', '바람'];
  const nameList = names.map(name => <li>{name}</li>);
  return <ul>{nameList}</ul>;
};

export default IterationSample;
```

// 현재는 태그 간단한 구조이므로 문제가 안될 수 있지만
코드 복잡한 경우, 코드양이 늘어난다
유동적인 데이터인 경우, 다른 방법을 생각해야 한다

[실습] 반복(IterationSample.js)

map()

❖ App.js

```
// ./src/App.js
import React from 'react';
import './App.css';
import IterationSample from './Components/IterationSample';

function App() {
  return (
    <div>
      <IterationSample />
    </div>
  );
}

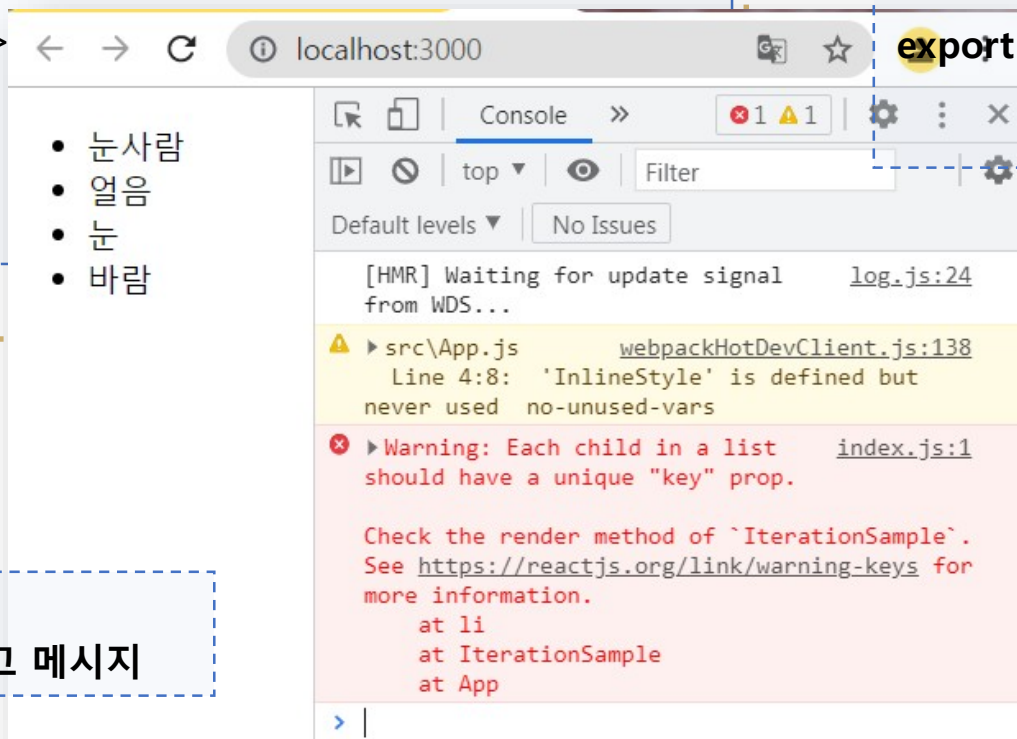
export default App;
```

// 개발자 도구
// key prop 없다는 경고 메시지

```
// ./src/Components/IterationSample.js
import React from 'react';
```

```
const IterationSample = () => {
  const names = ['눈사람', '얼음', '눈', '바람'];
  const nameList = names.map(name => <li>{name}</li>);
  return <ul>{nameList}</ul>;
};
```

```
export default IterationSample;
```



// 리엑트는 반복문 실행한 HTML 요소에는 반드시 **key={}**를 적으라고 권장

[요약] key props

key

배열 ['a', 'b', 'c', d'] → 배열 ['a', 'b', 'z', 'c', d'] // b 다음에 z 가 삽입되는 경우

// key 가 없는 경우

```
<div>a</div>
<div>b</div>
<div>c</div>
<div>d</div>
```

⇒

```
<div>a</div>
<div>b</div>
<div>z</div>
<div>c</div>
```

삽입 <div>d</div>

// key 가 있는 경우

```
<div key={0}>a</div>
<div key={1}>b</div>
<div key={2}>c</div>
<div key={3}>d</div>
```

⇒

```
<div key={0}>a</div>
<div key={1}>b</div>
<div key={4}>z</div>
<div key={2}>c</div>
<div key={3}>d</div>
```

[요약] key props

배열 ['a', 'b', 'c', 'd'] → 배열 ['b', 'c', 'd'] // a 가 삭제되는 경우

// key 가 없는 경우

1번째 <div>a</div>
2번째 <div>b</div>
3번째 <div>c</div>
4번째 <div>d</div>

⇒

삭제 <div>a</div>
1번째 <div>b</div>
2번째 <div>c</div>
3번째 <div>d</div>

// key 가 있는 경우

<div key={0}>a</div>
<div key={1}>b</div>
<div key={2}>c</div>
<div key={3}>d</div>

⇒

<div key={0}>a</div> 삭제
<div key={1}>b</div>
<div key={2}>c</div>
<div key={3}>d</div>

[요약] Key

Key 문법

❖ Key: (리액트)

❖ 컴포넌트 배열을 렌더링 했을 때 어떤 원소에 변동이 있었는지 알아내기 위해 사용

❖ 예) 유동 데이터 다룰 때(원소 새로 생성, 삭제, 수정)

Key가 없을 때 Virtual DOM을 비교하는 과정에서 리스트를 순차적으로 비교하면서 변화를 감지

❖ Key 값은 언제나 유일해야 한다(데이터가 가진 고유한 값을 key 값으로 설정)

```
// 게시판의 게시물을 렌더링한다면  
// 게시물 번호를 key 값으로 설정
```

```
const articleList = articles.map(article => {  
  <Article  
    title={article.title}  
    writer={article.writer}  
    key={article.id}  
  />  
});
```

```
// key 값을 설정할 때는 map 함수의 인자로 전달되는  
  함수 내부에서  
  컴포넌트 props를 설정하듯이 설정한다
```

[실습] 반복(IterationSample.js)

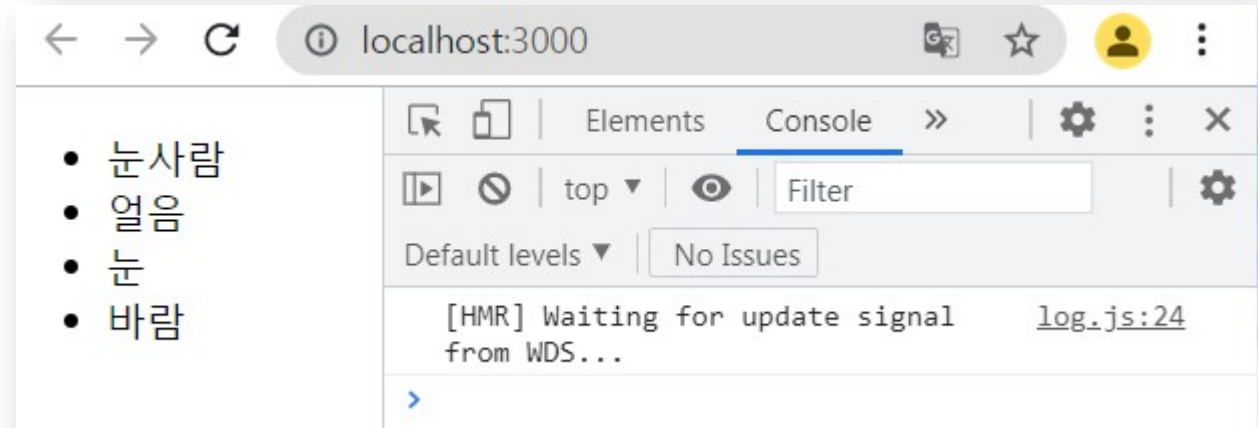
Key 문법

❖ Key 설정

```
// ./src/Components/IterationSample.js
import React from 'react';

const IterationSample = () => {
  const names = ['눈사람', '얼음', '눈', '바람'];
  const nameList = names.map((name, index) =>
    <li key={index}>{name}</li>);
  return <ul>{nameList}</ul>;
};

export default IterationSample;
```



// index 를 key 로 사용
// 고유번호 없으므로 index 로 key 사용

⇒ 고유한 값이 없을 때만 index 값을 key 로 사용한다
// 배열이 변경될 때 효율적으로 리렌더링 되지 못한다

map() 함수

map()

❖ map() 함수

- 배열의 모든 원소마다 특정 작업을 하는 함수를 적용하고 그 함수가 반환한 결과를 모아서 배열로 반환한다
- 자바스크립트 배열 객체의 내장함수 map() 함수
반복되는 컴포넌트를 렌더링
- 파라미터로 전달된 함수를 사용해서
배열 내 각 요소를 원하는 규칙에 따라 변환한 후
그 결과로 새로운 배열을 생성

```
> const numbers = [1,2,3,4,5];  
const square = numbers.map(num => num *  
num);  
console.log(square);
```

```
▶ (5) [1, 4, 9, 16, 25] VM896:3
```

```
< undefined
```

```
> |
```

// map() 함수

```
var numbers = [1,2,3,4,5];  
var processed = numbers.map(function(num) {  
  return num * num  
});  
console.log(processed);
```

// map() 함수

```
const numbers = [1,2,3,4,5];  
const square = numbers.map(num => num * num);  
console.log(square);
```

// 배열의 값을 제공하여 새로운 배열 만든다

map() 함수

map()

❖ map() 함수

- 배열의 모든 원소마다 특정 작업을 하는 함수를 적용하고 그 함수가 반환한 결과를 모아서 배열로 반환한다

// map() 함수

// 배열의 값을 제공하여 새로운 배열 만든다

```
const numbers = [1,2,3,4,5];  
const square = numbers.map(num => num * num);  
console.log(square);
```

```
> const numbers = [1,2,3,4,5];  
   const square = numbers.map(num => num *  
   num);  
   console.log(square);  
  
▶ (5) [1, 4, 9, 16, 25] VM896:3  
  
< undefined  
> |
```

map() 함수

map()

❖ map() 함수

- 배열의 모든 원소마다 특정 작업을 하는 함수를 적용하고 그 함수가 반환한 결과를 모아서 배열로 반환한다

```
const friends = ["dal", "mark", "lynn"]  
friends
```

```
friends.map(current => {  
  console.log(current);  
  return 0;  
});
```

```
// 이름 뒤에 하트 붙이기  
friends.map(function(friend) { // 익명함수  
  return friend + " ♥";  
}) // 익명함수의 friend 에 friends 배열의 원소가  
    하나씩 넘어오고 그 원소에 하트를 붙여 반환
```

```
> const friends = ["dal", "mark", "lynn"]  
< undefined  
> friends  
< ▶ (3) ["dal", "mark", "lynn"]  
> friends.map(current => {  
  console.log(current);  
  return 0;  
});  
dal VM232:2  
mark VM232:2  
lynn VM232:2  
< ▶ (3) [0, 0, 0]  
>
```

// [0,0,0] friends.map() 이 최종적으로 반환한 값

// map() 함수 특징:

- 1) map() 함수 인자로 전달한 함수는 배열 friends 의 원소를 대상으로 실행 (즉, 원소 3개이므로 함수 3번 실행)
- 2) 그 함수가 반환한 값이 모여 배열이 되고 그 배열이 map() 함수의 반환값이 된다

[실습] [./src/App.js] map()

App.js

```
// ./src/App.js
import React from 'react';
// import './App.css';
import ReturnMap from './Components/ReturnMap'

function App() {
  return (
    <div>
      <h1>Start React!</h1>
      <p>CSS 적용하기</p>
      <ReturnMap />
    </div>
  );
}

export default App;
```

ReturnMap.js

```
// ./src/Components/ReturnMap.js
import React, { Component } from 'react';

class ReturnMap extends Component {
  render() {
    const element_Array = [
      <li>Start</li>
      , <li>react</li>
      , <li>Array map</li>
    ]
    return (
      <ul>
        {element_Array.map((array_val) => array_val)}
      </ul>
    )
  }
}

export default ReturnMap;
```

Map()