



# Reactjs

✓ 원리를 알면 IT가 맛있다

**Reactjs for Beginners**

chapter 01.

---

# Reactjs개요

### ○ Reactjs 소개

가. Facebook에서 Reactjs 발표

나. UI상태 자동 관리

다. 가상 DOM을 이용한 빠른 DOM 조작 가능

라. 자바스크립트로 정의하는 화면

마. MVC의 'V' 담당

비주얼한 요소와 그 상태를 최신으로 유지하는데 중점을 둔 뷰 레이어에서 동작.

<https://github.com/facebook/react/wiki/Sites-Using-react>

## □ 1) Reactjs 환경 설정

Reactjs

### ○ Reactjs 개발을 위한 환경 설정

가. Node.js 설치

[https://nodejs.org/](https://nodejs.org/ko/)

New security releases now available for 15.x, 14.x and 12.x  
release lines

Download for Windows (x64)

14.15.1 LTS

Recommended For Most Users

15.2.1 Current

Latest Features

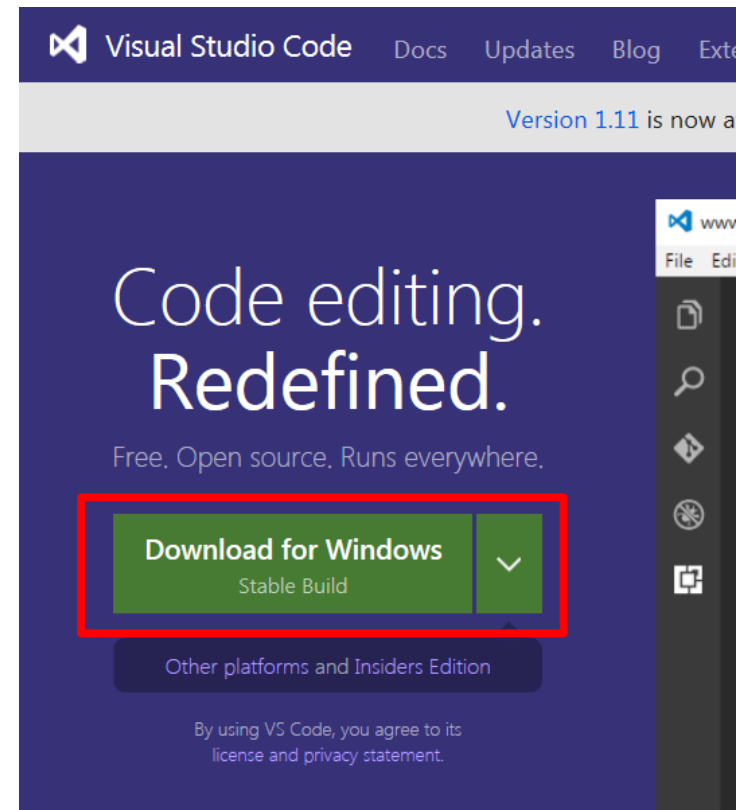
[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

```
C:\Users\ledzep>node -v  
v14.13.1
```

나. 개발 툴 설치 ( Visual Studio Code )

<https://code.visualstudio.com/>



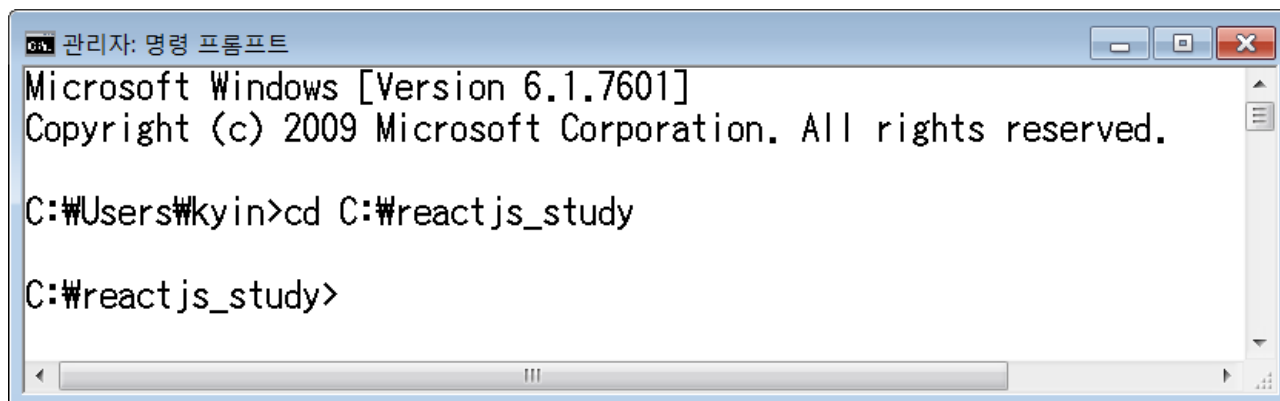
## □ 1) Reactjs 환경 설정

### 다. Reactjs 설치

<https://reactjs.org>

<https://reactjs.org/docs/create-a-new-react-app.html>

```
npx create-react-app my-app  
cd my-app  
npm start
```



```
관리자: 명령 프롬프트  
Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\kyin>cd C:\reactjs_study  
  
C:\reactjs_study>
```

```
C:\reactjs_study>npx create-react-app my-app  
npx: installed 1 in 13.788s
```

The "path" argument must be of type string

```
C:\Users\kyin\AppData\Roaming\npm\node_modules\create-react-app\index.js
```

Creating a new React app in **C:\reactjs\_study\my-app**.

## ❑ 1) Reactjs 환경 설정

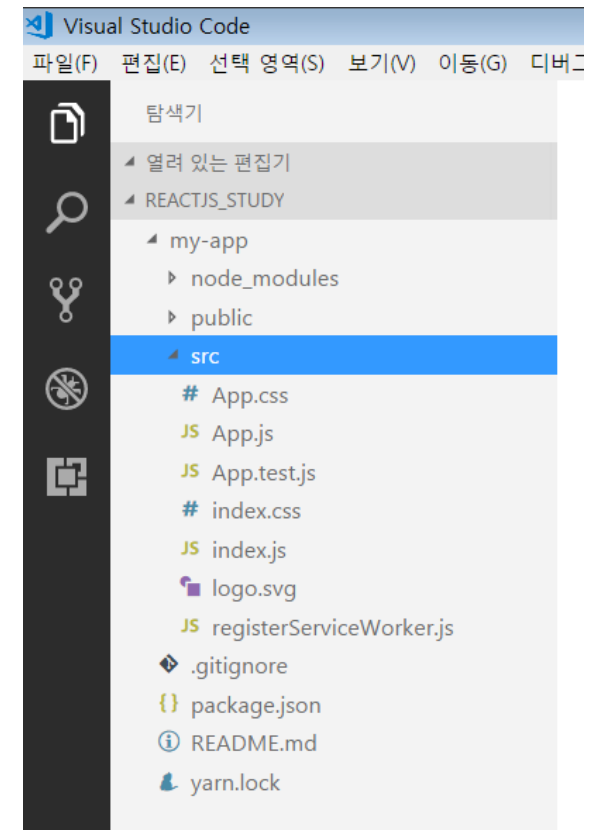
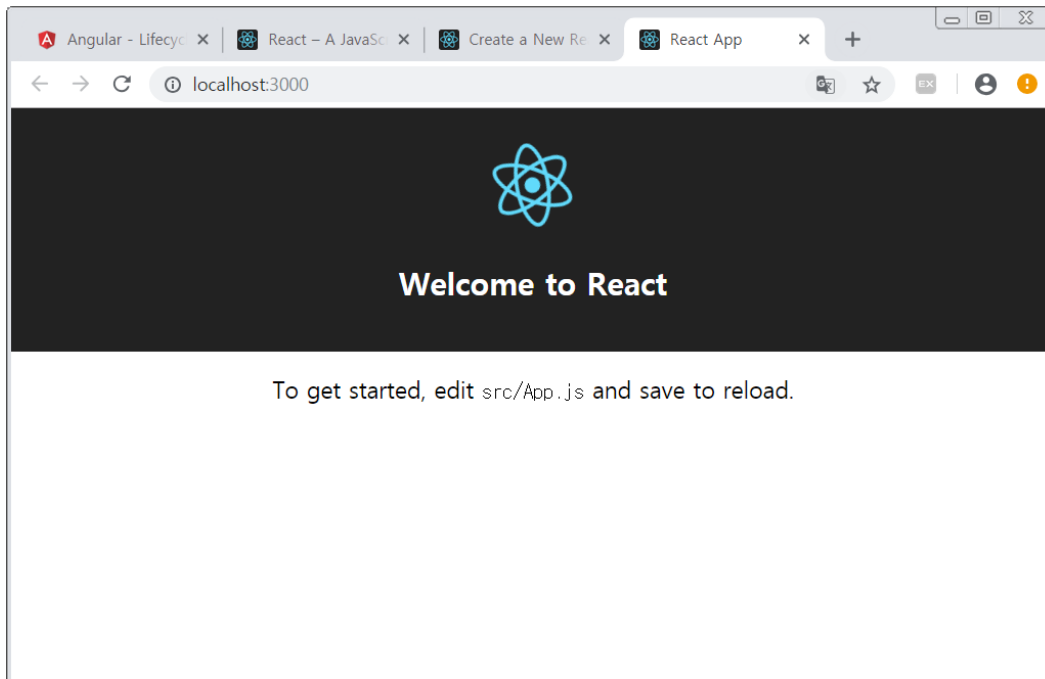
Reactjs

```
관리자: 명령 프롬프트
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

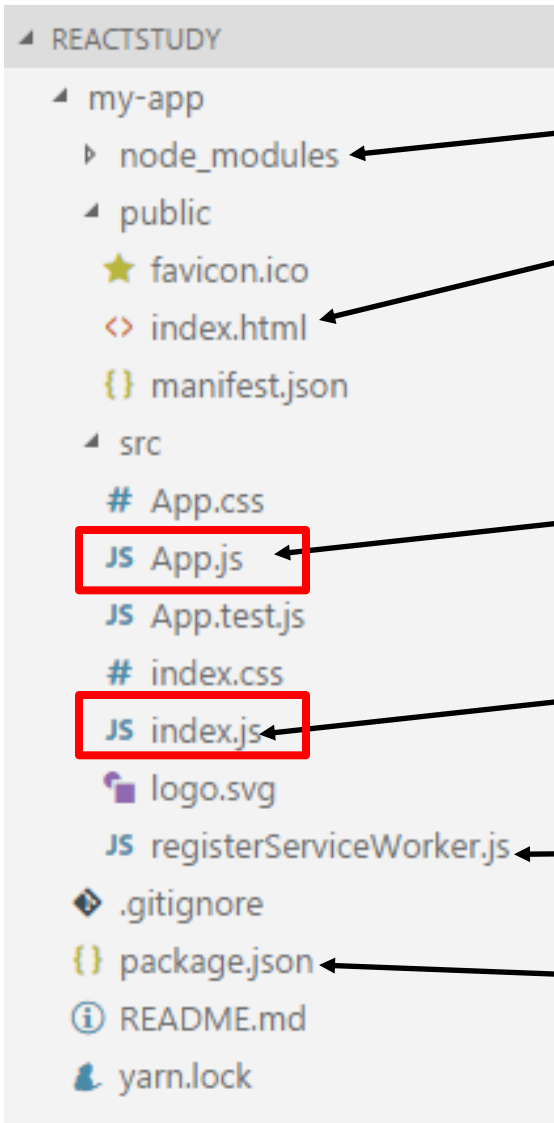
C:\Users\kyin>cd c:\reactjs_study

c:\reactjs_study>cd my-app

c:\reactjs_study\my-app>yarn start
```



## □ 1) Reactjs 환경 설정



Node.js는 node\_modules 폴더를 생성하고 이곳에 package.json 파일에서 명시된 외부 모듈을 다운로드 저장함.

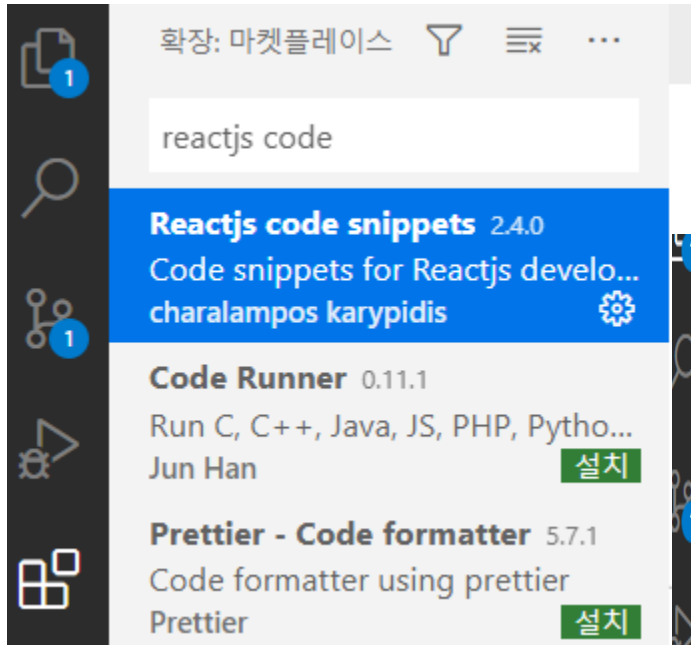
메인 홈페이지 파일. 편집할 필요 없다.  
어플리케이션을 빌드 할 때 자동으로 모든 js 및 css 파일을 동적으로 추가된다.

Reactjs 컴포넌트 파일

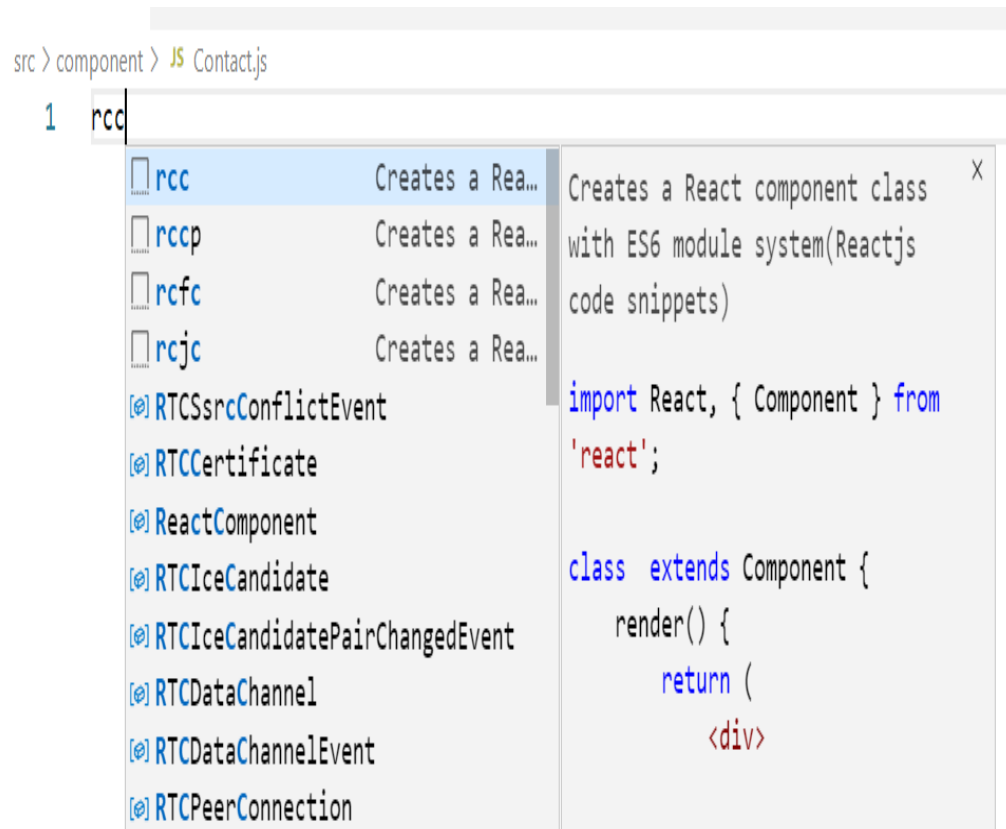
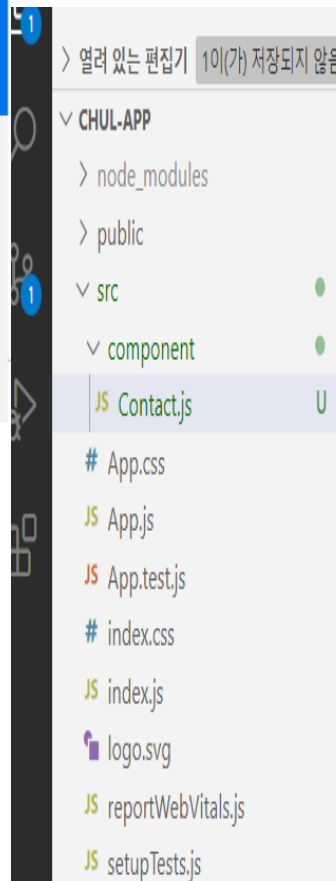
Starting point. 어플리케이션의 App 컴포넌트를 로딩하여 웹브라우저에서 실행시킴.

개발용, 캐시 향상을 위한 서비스 워커

Nodejs의 모듈 관리 설정 파일  
프로젝트에서 사용하는 third party용 패키지를 나열



Reactjs code snippets 사용  
Rcc를 입력하여 자동완성





## ❑ 2) Reactjs 컴포넌트

### \* Reactjs 컴포넌트 작성

가. Contact.js 생성 후 extends Component

나. render() 메서드 구현

다. return ( html 코드 );

Rcc사용

JS Contact.js x

```
1 import React, { Component } from 'react';
2
3 export class Contact extends Component {
4   render() {
5     return (
6       <h1>Contacts 홈페이지</h1>
7     );
8   }
9 }
```

JS App.js x

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

localhost:3000

Contacts 홈페이지

Rcc 이용 자동생성

Yarn start --port =3002

```
JS Contact.js X
src_02_Contact > component > JS Contact.js > Contact
1 import React, { Component } from 'react';
2
3 export class Contact extends Component {
4   render() {
5     return (
6       <h1>Contacts 홈페이지</h1>
7     );
8   }
9 }
10 export class Contact2 extends Component {
11
12   info(){
13     return <h1>Contact2 홈페이지</h1>;
14   }
15
16   render() {
17     return (this.info());
18   }
19 }
```

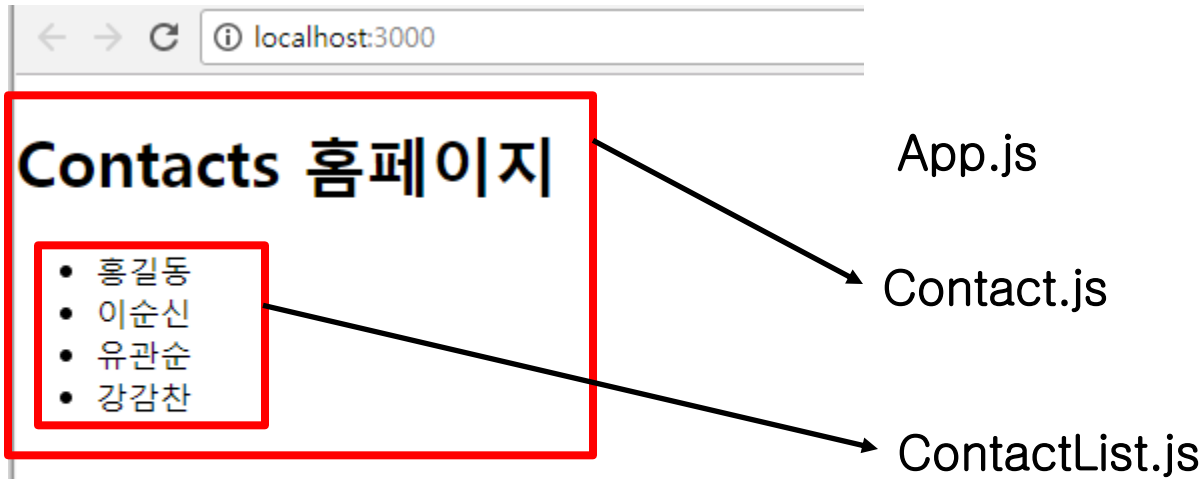
```
PS C:\react_study_chul_2\chul-app> yarn start
```

```
JS App.js X JS Contact.js # App.css
src > JS App.js > App > render
1 import React, { Component } from 'react';
2 import {Contact, Contact2} from './component/Contact';
3 class App extends Component {
4   render() {
5     return (
6       <div>
7         <Contact/>
8         <Contact2/>
9       </div>
10    );
11  }
12 }
13
14 export default App;
```

## □ 2) Reactjs 컴포넌트

Reactjs

### \* 실습



```
src > component > JS Concat.js > ...
1 import React, { Component } from 'react';
2
3 export class Concat extends Component {
4   render() {
5     return (
6       <div>
7         <h1>Concats 홈페이지 </h1>
8       </div>
9     );
10   }
11 }
12
```

```
src > component > JS ConcatList.js > ...
1 import React, { Component } from 'react';
2 export class ConcatList extends Component {
3   render() {
4     return (
5       <div>
6         <ul>
7           <li>홍페이지</li>
8           <li>이순신</li>
9           <li>유관순</li>
10          <li>강감찬</li>
11        </ul>
12      </div>
13    );
14  }
15 }
```

The screenshot shows a VS Code editor window with the following structure:

- File Explorer (Left):**
  - CHUL-APP
    - node\_modules
    - public
    - src
      - component
        - Concat.js (U)
        - ConcatList.js (U)
      - App.css (#)
      - App.js (M) - Selected
      - App.test.js (JS)
      - index.css (#)
      - index.js (JS)
      - logo.svg
      - reportWebVitals.js (JS)

- Editor (Right):**
- Tab: JS App.js
- Content:
 

```

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

Yarn start --port =3002

### ○ JSX소개

- 일반 자바스크립트 문법이 아닌 JSX 문법을 사용하여 UI를 구현한다.  
(JSX를 자바스크립트로 변환해주는 트랜스파일 동작: babel 프로젝트)
- JSX는 컴파일링 되면서 최적화되기 때문에 빠르다.
- 컴파일 단계에서 에러를 확인할 수 있다.
- HTML 문법과 비슷하여 더 쉽고 빠르게 UI 템플릿 작성 가능하다.

```
export class Contact extends Component {  
  render() {  
    return (  
      <h1>Contacts 홈페이지</h1>  
    );  
  }  
}
```

- “”로 감싸지 않는다.
- 반드시 root태그 존재

## 1) JSX에서 변수값 사용시 {} 사용

//1. jsx에서 변수값 사용시 {} 이용하고 반드시 root 태그가 존재야하며 "" 사용안됨

```
export default class Contact extends Component {
  render() {
    let mesg="world";
    return (
      <h1>{mesg}&nbsp; {100}:{"hello"}</h1>
    );
  }
}
```

//2. JSX에서 자바스크립트 사용시 {} 이용

```
export class Contact2 extends Component {
  render(){
    var myArr = ["A","B","C"];
    return ( <ul>
      { //자바스크립트 사용하기 위해서는 반드시 {
        myArr.map(function(row,idx){
          console.log(row, idx );
          return <li key={idx}>{row}</li>;
        })
      }
    </ul>);
  }
}
```

JS App.js

# Contact.css

JS Contact.js

# App.css

src > JS App.js > App > render

```
1 import React, { Component } from 'react';
2 import Contact from './component/Contact'; //export default
3 import {Contact2} from './component/Contact';
4 class App extends Component {
5   render() {
6     return (
7       <div>
8         <Contact/>
9         <Contact2/>
10      </div>
11    );
12  }
13 }
14 export default App;
```

## ○ 외부변수와 함수 실행

```

37 //////////////자바스크립트 render()메소드 밖에 있어서 다양한 작업 가능 ///
38 var myArr=["A", "B", "C"];
39 function my() {
40     return myArr.map(function (row, idx) {
41         console.log(row, idx);
42         return <li key={idx}>{row}</li>;
43     })
44 }
45 ///////////////////////////////////////////////////
46 export class Contact21 extends Component{
47     render(){
48         return(
49             <ul>
50                 { //자바스크립트 사용시 반드시 {} 사용
51                     my()
52                 }
53             </ul>
54         );
55     }
56 }
57 ///////////////////////////////////////////////////

```

## 3 ) JSX에서 spread 연산자 사용

```
//3. JSX에서 spread 연산자 사용 예
export class Contact3 extends Component {
  render(){
    var attr={
      href:"http://www.daum.net",
      target:"_blank"
    };
    // return ( <a href={attr.href} target={attr.target}>daum
    return (<a {...attr}>daum:{attr.href}</a>);
  }
}
```

## 4 ) JSX에서 multi node 사용

```
//4. JSX에서 multi node 사용 예
export class Contact4 extends Component {
  render(){
    var multi=[
      <span>hello</span>,
      <span>world</span>
    ];
    return (<div>{multi}</div>) ;
  }
}
```



## 5) JSX에서 camel 표기법 필수

```
//5. JSX에서는 camel 표기법 필수
function myok(){
  console.log("myOK");
}
export class Contact5 extends Component {
  render(){
    return (<button onClick={myok}>OK</button>) ;
  }
}
```

```
Warning: Invalid event handler property `onclick`. Did you mean `onClick`?
    at button
    at Contact5 (http://localhost:3000/static/js/main.chunk.js:438:1)
```

카멜표기법사용

## 6) JSX에서 className, htmlFor 형식으로 사용

```
//6. JSX에서는 className , htmlFor 형식으로 사용
export class Contact6 extends Component {
```

```
  render(){
    return (
      <div>
        <p className="xyz">hello</p>
        <label htmlFor="myInput">hello2</label>
        <input type="text" name="myInput" />
      </div>
    ) ;
  }
}
```

```
1 /* Contact.css */
2 .xyz {
3   font-size: large;
4   color: red;
5 }
```

JSX에서 사용 가능한 태그 및 속성 정보  
<https://reactjs.org/docs/dom-elements.html>

<https://magic.reactjs.net/htmltojsx.html>

## 7) JSX에서 style은 객체로 사용 ( 인라인 스타일 권장 )

//7.JSX에서 style은 객체로 처리되어 인라인스타일지정을 권장함  
 //backgroundColor, fontSize, textAlign의 카멜표기법 사용//px만 생략가능  
 export class Contact7 extends Component{

```
    render(){
      var myStyle={fontSize:'50px', backgroundColor:'red'};
      return(
        <div>
          <p style={myStyle}>Happy</p>
          <p style={{fontSize:'20', backgroundColor:'yellow'}}>world</p>
        </div>
      )
    }
  }
```

backgroundColor  
 fontSize  
 textAlign  
 과 같은 camel 표기법 사용

px 만 생략 가능

## 8) JSX에서 주석은 {/\* \*/} 형식, 반드시 root태그 내에서 사용

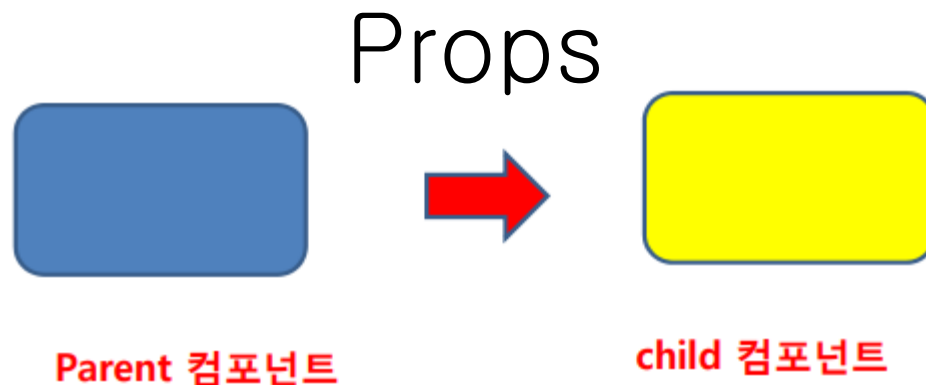
```
return (
  <div>
    {/* 이것은 주석입니다*/}
    <p style={{fontSize:'20px', background:'red'}}>world</p>
    <p style={myStyle}>happy</p>
  </div>
) ;
```

```
JS App.js  X  # Contact.css  JS Contact.js  # App.css
src > JS App.js > App > render
1  import React, { Component } from 'react';
2  import Contact from "../component/Contact"; //export default
3  import {Contact2, Contact21, Contact3, Contact4} from './component/Contact';
4  import {Contact5,Contact6,Contact7,Contact8} from './component/Contact'
5  class App extends Component {
6    render() {
7      var xxx= <Contact8/>;
8      // 외부변수에 component를 저장해서 사용
9      return (
10       <div>
11         <Contact/>
12         <Contact2/>
13         <Contact21/>
14         <Contact3 />
15         <Contact4/>
16         <Contact5/>
17         <Contact6/>
18         <Contact7/>
19         {xxx}
20         {/* 외부변수에 저장된 class사용 */}
21       </div>
22     );
23   }
24 }
```

### ○ Props 소개

- 컴포넌트에서 사용할 데이터 중에서 **변동되지 않는 데이터 (immutable)**를 다룰 때 사용.

- 부모(parent)컴포넌트에서 자식(child) 컴포넌트로 데이터를 전달 할 때 사용됨.



## □ 4) Props

### ○ Props 사용 1

가. 부모(parent)에서 자식(child)태그의 속성에 값을 설정하여 전달.

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Contact mesg="홍길동" mesg2="10" />  
      </div>  
    );  
  }  
}
```

***propName="value"***

Props는 필수가 아니다.

나. 자식(child)에서 속성값 얻기. (render() 메서드 안에서 )

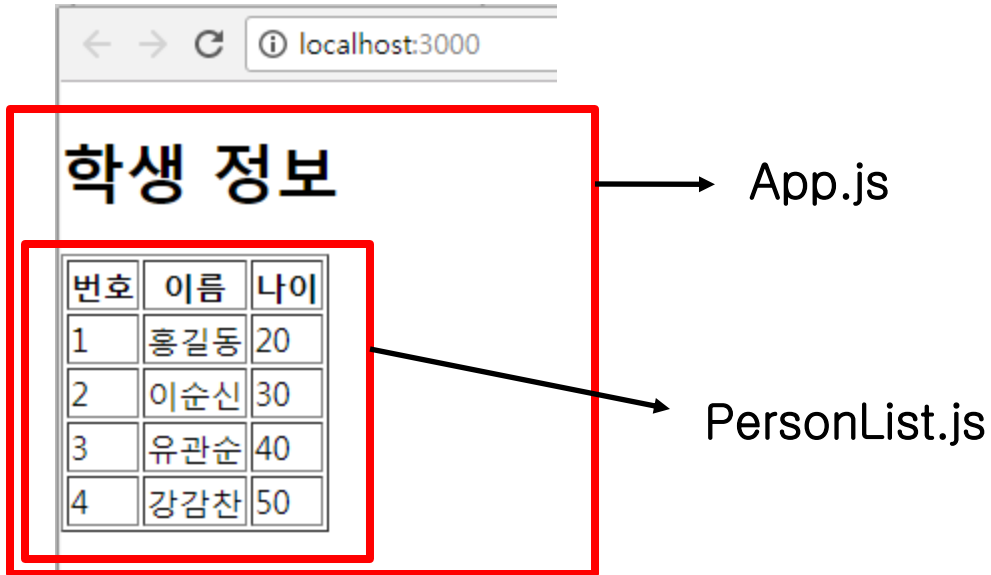
```
export class Contact extends Component {  
  render(){  
    let mesg = this.props.mesg;  
    return (  
      <div>  
        <h1>{mesg}</h1>  
        <h1>{this.props.mesg2}</h1>  
      </div>  
    );  
  }  
}
```

***{ this.props.propName }***

```
let {mesg,mesg2} = this.props;
```

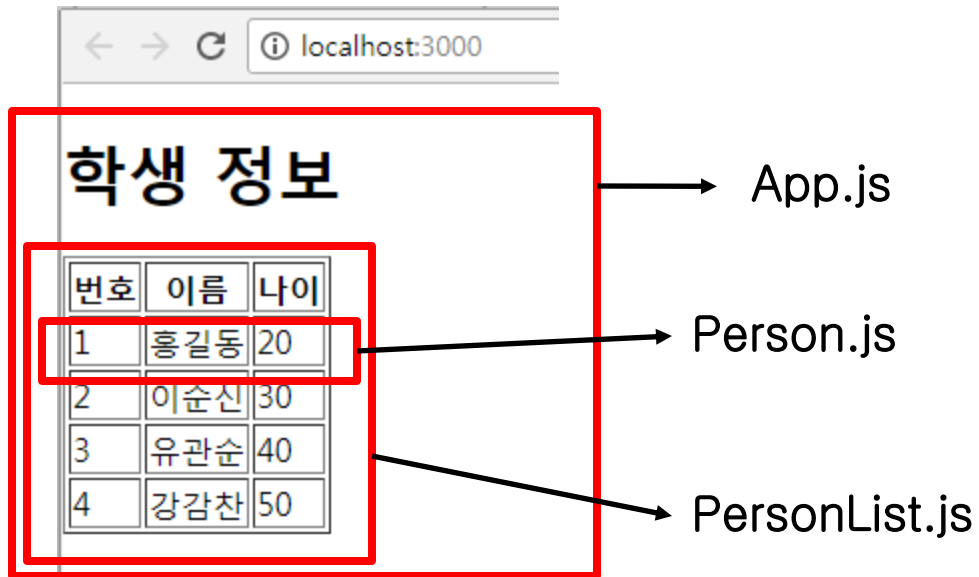
## □ 4) Props

### ○ 실습 1



```
class App extends Component {  
  
  persons=[  
    {name: "홍길동", age: 20},  
    {name: "이순신", age: 30},  
    {name: "유관순", age: 40},  
    {name: "강감찬", age: 50}  
  ];  
  
  render() {  
    return (  
      <div>  
        <h1>학생 정보</h1>  
      </div>  
    );  
  }  
}
```

### ○ 실습 2



```

JS Person.js  JS App.js  X  JS PersonList.js

src > JS App.js > App > render
1  import React, { Component } from 'react';
2  import {PersonList} from './component/PersonList';
3  class App extends Component {
4      persons=[
5          {name:"홍길동", age:20},
6          {name:"이순신", age:30},
7          {name:"유관순", age:40},
8      ];
9      render() {
10         return (
11             <div>
12                 <h1>학생정보</h1>
13                 <table border="1">
14                     <thead>
15                         <tr>
16                             <th>번호</th>
17                             <th>이름</th>
18                             <th>나이</th>
19                         </tr>
20                     </thead>
21                     <PersonList xxx={this.persons} />
22                 </table>
23             </div>

```

```

JS PersonList.js X

src > component > JS PersonList.js > PersonList > render > persons.map() callback
1  import React, { Component } from 'react';
2
3  export class PersonList extends Component {
4      persons;
5      render() {
6          this.persons= this.props.xxx;
7          console.log(this.persons);
8          return (
9              <tbody>
10                 {
11                     this.persons.map(function(row, idx){
12                         return (
13                             <tr key="idx">
14                                 <td>{idx+1}</td>
15                                 <td>{row.name}</td>
16                                 <td>{row.age}</td>
17                             </tr>
18                         )
19                     })
20                 }
21             </tbody>

```



## □ PersonList->Person으로 pros전달

Reactjs

```
PersonList.js  JS App.js  X  JS Person.js
rc > JS App.js > App > persons
8      {name: '유관순', age: 40},
9      {name: '강감찬', age: 50},
10     ];
11     render() {
12       return (
13         <div>
14           <h1>학생정보</h1>
15           <table border="1">
16             <thead>
17               <tr>
18                 <td>번호</td>
19                 <td>이름</td>
20                 <td>나이</td>
21               </tr>
22             </thead>
23             <PersonList xxx={this.persons}/>
24           </div>
25         </div>
26       );
27     }
28   }
29   export default App;
```

```
export class PersonList extends Component {
  persons;
  render() {
    this.persons= this.props.xxx;
    return(
      <tbody>
        {
          this.persons.map(function(row, idx){
            return (
              <Person key={idx} yyy={row} yyy2={idx}/>
            )
          })
        }
      </tbody>
    )
  }
}
```

```
PS C:\react_study_chul_2\chul-app> yarn start
```

## ❑ 실습(PersonList->Person으로 props전달

```
JS Person.js × JS App.js JS PersonList.js
src > component > JS Person.js > Person > render
1  import React, { Component } from 'react';
2
3  export class Person extends Component {
4    person;
5    render() {
6      this.person= this.props.yyy;
7      return (
8        <tr>
9          <td>{this.props.yyy2+1}</td>
10         <td>{this.person.name}</td>
11         <td>{this.person.age}</td>
12       </tr>
13     )
14   }
15 }
16
```

## □ 4) Props

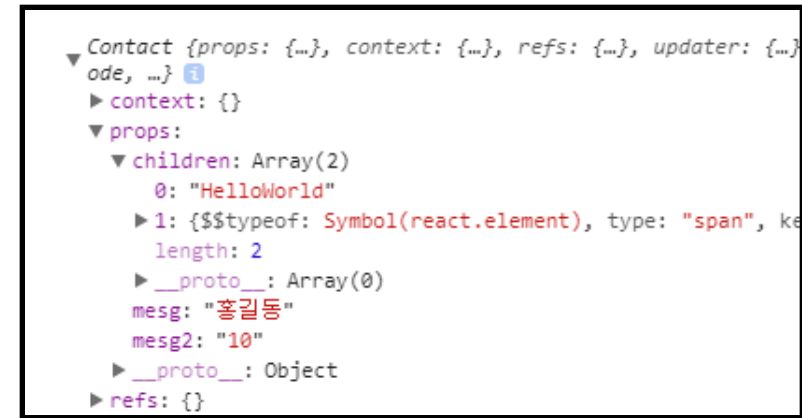
### ○ Props 사용 2

가. 부모(parent)에서 자식(child)태그의 **몸체(body)**에 **값을 설정**하여 전달.

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        { /* body가 존재*/ }  
        <Contact msg="홍길동" msg2="10">  
          HelloWorld  
          <span>Happy</span>  
        </Contact>  
      </div>  
    );  
  }  
}
```

속성

몸체(body)



나. 자식(child)에 서 속성값 얻기.  
(render() 메서드 안 )

**{ this.props.children }**

```
export class Contact extends Component {  
  render(){  
    let msg = this.props.msg;  
    let body = this.props.children;  
    return (  
      <div>  
        <h1>{msg}</h1>  
        <h1>{this.props.msg2}</h1>  
        <h1>{body}</h1>  
      </div>  
    );  
  }  
}
```

### ○ 기본 Props 속성값 지정

- 기본적으로 Props 속성은 필수가 아니다.
- 기본값을 설정하기 위해서는 컴포넌트 클래스 하단에

`className.defaultProps = { propName: value }` 를 삽입;

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Contact msg="홍길동" msg2="10" />  
        <Contact />  
      </div>  
    );  
  }  
}
```

기본값으로 설정

```
export class Contact extends Component {  
  render(){  
    let msg = this.props.msg;  
    return (  
      <div>  
        <h1>{msg}</h1>  
        <h1>{this.props.msg2}</h1>  
      </div>  
    );  
  }  
}  
  
//기본값 설정  
Contact.defaultProps={  
  msg:"유관순",  
  msg2:"200"  
}
```

localhost:3000

홍길동

10

유관순

200

```

JS App.js JS Contact.js X
src > component > JS Contact.js > ...
1  import React, { Component } from 'react';
2
3  export class Contact extends Component {
4
5      render() {
6          let {mesg:a, mesg2:b}= this.props;
7          return (
8              <div>
9                  <h1>{a}</h1>
10                 <h1>{b}</h1>
11             </div>
12         );
13     }
14 }
15 //기본값 설정 class외부에서 선언
16 Contact.defaultProps={
17     mesg:"유관순",
18     mesg2:"20"
19 }
20

```

클래스명.defaultProps={}사용  
또는 클래스내 static  
defaultProps={}사용

```

component / JS Contact.js / Contact
1  import React, { Component } from 'react';
2  export class Contact extends Component {
3      static defaultProps={
4          mesg:"유관순",
5          mesg2:"20"
6      }
7
8      render() {
9          let {mesg:a, mesg2:b}= this.props;
10         return (
11             <div>
12                 <h1>{a}</h1>
13                 <h1>{b}</h1>
14             </div>
15         );
16     }
17 }
18

```

### ○ Props 의 유효성 검사 ( validate )

- 컴포넌트에서 원하는 Props의 type과 전달된 type 일치 여부 검증.
- 추가로 필수(required) Props 설정도 가능.

```
import PropTypes from 'prop-types';
```

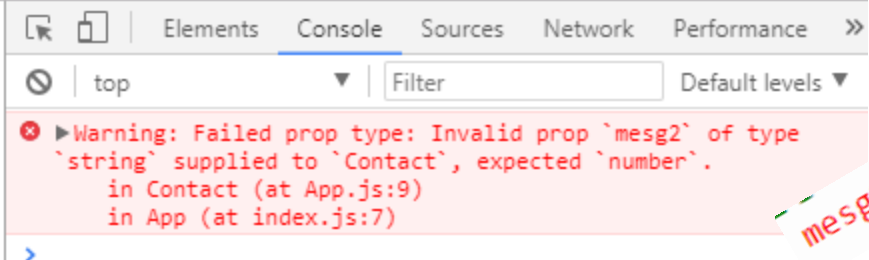
```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Contact msg="홍길동" msg2="10" />  
        <Contact />  
      </div>  
    );  
  }  
}
```

```
//validate  
Contact.propTypes={  
  msg: PropTypes.string,  
  msg2: PropTypes.number  
}
```

```
//validate  
Contact.propTypes={  
  name: PropTypes.string,  
  age: PropTypes.number,  
  phones: PropTypes.array,  
  my: PropTypes.func,  
  isMarried: PropTypes.bool  
}
```

홍길동

10



msg2={10}

## □ 4) Props

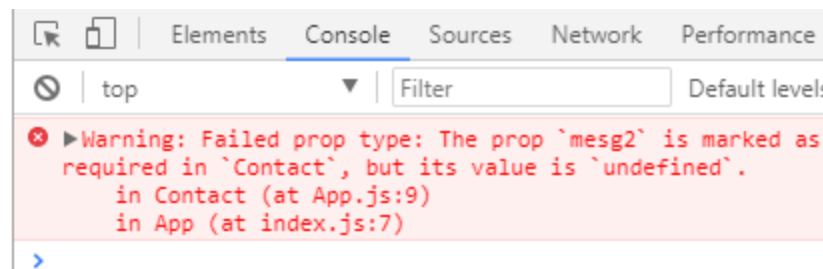
Reactjs

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Contact mesg="홍길동" />  
      </div>  
    );  
  }  
}
```

```
// 기본값 설정  
// Contact.defaultProps={  
//   mesg:"유관순",  
//   mesg2:200  
// }  
  
//validate  
Contact.propTypes={  
  mesg: PropTypes.string,  
  mesg2: PropTypes.number.isRequired  
}
```

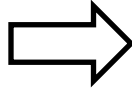
Required 예외 발생

홍길동



## ❑ 4) Props [ <https://reactjs.org/docs/typechecking-with-proptypes.html> ]

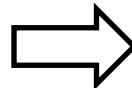
```
// 기본값 설정
// Contact.defaultProps={
//   msg:"유관순",
//   msg2:200
// }
```



```
export class Contact extends Component {
  /*
  If you are using a Babel transform like transform-class-properties,
  you can also declare defaultProps as static property
  */
  static defaultProps={
    msg:"유관순",
    msg2:200
  }

  render(){
```

```
//validate
// Contact.propTypes={
//   msg: PropTypes.string,
//   msg2: PropTypes.number
//   //msg2: PropTypes.number.isRequired
// }
```



```
export class Contact extends Component {
  // 기본값 설정
  static defaultProps={
    msg:"유관순",
    msg2:200
  }

  //validate
  static propTypes={
    msg: PropTypes.string,
    msg2: PropTypes.number
    //msg2: PropTypes.number.isRequired
  }
}
```



## □ Spread연산자의 사용

```
미널(T) 도움말(H)
JS First.js JS App.js X
src > JS App.js > App
1 import React, { Component } from 'react';
2 import './App.css';
3 import { First } from './component/First';
4
5 class App extends Component {
6
7   render() {
8     return (
9       <div>
10         <First id="userid" pw="1234" />
11       </div>
12     );
13   }
14 }
15
16 export default App;
```

```
PS C:\react_study_chul_2\chul-app> yarn start
```

```
JS First.js X
src > component > JS First.js > First
1 import React, { Component } from 'react';
2 import { Second } from './Second'
3
4 export class First extends Component {
5
6   render() {
7     return (
8       <div>
9         /* <Second id={this.props.id} pw={this.props.pw}/> */
10        /* spread연산자 사용 */
11        <Second {...this.props}/>
12      </div>
13    );
14  }
15 }
16
17
```

```
JS Third.js X
src > component > JS Third.js > Third > render
1 import React, { Component } from 'react';
2
3 export class Third extends Component {
4   render() {
5     return (
6       <div>
7         /* <p>{this.props.id}</p>
8         <p>{this.props.pw}</p> */
9         {this.props.id}<br/>
10        {this.props.pw}<br/>
11      </div>
12    );
13  }
14 }
15
```

## ❑ 4) handling Events

### ○ 이벤트 처리 (<https://reactjs.org/docs/events.html>)

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

is slightly different in React:

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

```
// This binding is necessary to make `this` work in the callback
this.handleClick = this.handleClick.bind(this);
```

c > child > JS Child.js > Child > render

```
1  import React, { Component } from 'react';
2
3  class Child extends Component {
4    render() {
5      return (
6        <div>
7          <button onClick={this.handleEvent}>handleEvent</button><br></br>
8          <button onClick={this.handleEvent2}>handleEvent2</button><br></br>
9          <a href="http://www.daum.net" onClick={this.handleEvent3}>daum-eventPrevent</a><br></br>
10         <button onClick={()=>this.a()}>this.a()</button><br></br>
11         <button onClick={(e)=>this.b(e, 100)}>b(x,y)</button><br></br>
12       </div>
13     );
14   } //end render
15   a(){
16     console.log("a=====");
17   }
18   b(x,y){
19     console.log("b=====", x, y);
20   }
21   handleEvent(){
22     console.log("Ok=====");
23   }
24   handleEvent2(e){
25     console.log("OK2====", e);
26   }
27   handleEvent3(e){
28     console.log("eventPreventDefault()");
29     e.preventDefault(); //return false 지원안됨
30   }
31
32 }
```

## □ 4) ref

```
constructor(){  
  super();  
  this.submit = this.submit.bind(this);  
}
```

Reactjs

### ○ 참조 ( ref )

- 사용자 입력을 받는 UI 요소와 상호작용할 때 가장 많이 사용된다.(직접 DOM 접근)

```
render(){  
  return(  
    <form onSubmit={this.submit}>  
      아이디<input type="text" ref="userid" />  
      비밀번호<input type="text" ref="passwd" />  
      <button>로그인</button>  
    </form>  
  );  
} //end render()  
  
//  
submit(e){  
  e.preventDefault(); // return false는 지원안함.  
  console.log("submit");  
  // input 태그의 ref 값을 this.refs로 참조한다.  
  const {userid, passwd} = this.refs;  
  console.log(userid.value , passwd.value);  
}
```

## □ 4) ref

### - ref 값으로 콜백함수 지정

```
render(){  
  return(  
    <form onSubmit={this.submit}>  
      아이디<input type="text" ref="userid" />  
      /* ref 값으로 콜백함수 지정 가능 */  
      비밀번호<input type="text" ref={(x)=>{console.log("x",x);this.passwd=x;}} />  
      <button>로그인</button>  
    </form>  
  );  
} //end render()
```

```
export class ChildComponent extends Component {  
  constructor(){  
    super();  
    this.submit = this.submit.bind(this); //함수 생성
```

```
//  
submit(e){  
  e.preventDefault(); //return false 지원 안함  
  console.log("submit", this.refs);  
  console.log("this.passwd", this.passwd);  
  //input태그의 ref 값은 this.refs로 참조함  
  const {userid} = this.refs;  
  userid.value = "AAAAAAA";  
  //userid.focus();  
  console.log(userid.value, "\t", this.passwd.value);  
}
```

## □ 4) ref

- 컴포넌트에 ref 속성값 지정하여 부모 컴포넌트에서 자식 컴포넌트 참조 가능

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <ChildComponent ref={(ref)=>{console.log(ref);this.AppComponent=ref;}} />  
        <button onClick={e=>this.AppComponent.info()}>OK</button>  
      </div>  
    );  
  }  
}
```

```
App.js JS ChildComponent.js ×  
> component > JS ChildComponent.js > ChildComponent > info  
1 import React, { Component } from 'react';  
2  
3 export class ChildComponent extends Component {  
4  
5   info(){  
6     console.log("ChildComponent info()");  
7   }  
8   render() {  
9     return (  
10      <div>  
11        <h1>컴포넌트 ref 실습</h1>  
12      </div>  
13    );  
14  }  
15 }  
16
```

## □ 4) ref state의 사용

```
JS App.js  X  JS ChildComponent.js
src > JS App.js > App > render
1 | import React, { Component } from 'react';
2 | import './App.css';
3 | import {ChildComponent} from './component/ChildComponent'
4 |
5 | class App extends Component {
6 |   render() {
7 |     return (
8 |       <div>
9 |         <ChildComponent ref={(ref)=>{
10 |           console.log(ref); //ChildComponent
11 |           this.AppComponent= ref; //this.AppComponent에 ChildComponent설정
12 |         }}/>
13 |         <button onClick={e=>this.AppComponent.info('red')}>OK</button>
14 |         <button onClick={e=>this.AppComponent.info('blue')}>OK</button>
15 |         /* event를 전달하며 자식컴포넌트 함수 호출 */
16 |       </div>
17 |     );

```

```
JS App.js    JS ChildComponent.js X
src > component > JS ChildComponent.js > ChildComponent > info
1  import React, { Component } from 'react';
2
3  export class ChildComponent extends Component {
4      constructor(){
5          super();
6          this.state={ //상태값 저장하기 위한 초기화 this.state={key:value}
7              colorName:'White'
8          }
9      }
10     info(x){
11         console.log("ChildComponent info()",x);
12         this.setState({ //상태값 변경을 위해서는 this.setState함수에서 상태값 변경
13             colorName:x //상태값 변경 this.state.colorName=x 안됨
14         });
15     }
16     render() {
17         return (
18             <div>
19                 <h1 style={{background:this.state.colorName}}>컴포넌트 ref 실습</h1>
20                 /* {this.state.key}로 사용 */
21             </div>
22         );
23     }
24 }
```



### ○ State 소개 <https://reactjs.org/docs/state-and-lifecycle.html>

- 컴포넌트에서 사용할 데이터 중에서 **유동적인 데이터(mutable)**를 다룰 때 사용.

### ○ State 사용법

- state 의 초기화는 constructor(생성자)에서 **this.state={key:value}** 형식으로 설정.

- state 를 랜더링하기 위해서는 **{ this.state.key }** 사용.

- state를 업데이트 할 때는 **this.setState({})** 메서드 사용  
**this.state= 값** 형식으로 직접 수정 불가

주의할 점은 **this.setState({})** 메서드를 사용하게 될 메서드를 반드시 bind 시켜 주어야 된다.

### 가) 생성자에서 state 초기화

```
constructor(){  
  super();  
  this.state={  
    xyz:0,  
    number:100  
  };  
  this.handleClick = this.handleClick.bind(this);  
} //end constructor
```

### 나) state 값 렌더링

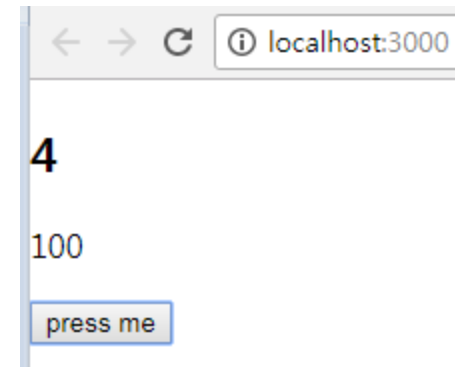
```
render(){  
  return (  
    <div>  
      <h2>{this.state.xyz}</h2>  
      <p>{this.state.number}</p>  
      <button onClick={this.handleClick}>press me</button>  
    </div>  
  );  
}
```

### 다) state 변경 ( 자동 re-rendering )

```
handleClick(){  
  console.log(this);  
  this.setState({  
    xyz: this.state.xyz + 1  
  });  
}
```

### 라) 메서드 this 바인딩

```
this.handleClick = this.handleClick.bind(this);
```



## □ 5) State

JS Counter.js X

src > component > JS Counter.js > Counter > state

```
1 import React, { Component } from 'react';
2
3 export class Counter extends Component {
4   //아래 생성자와 동일한 기능
5   state={
6     num:0
7   }
8   // constructor(){
9   //   super();
10  //   this.state={
11  //     num:0
12  //   };
13  //   this.handleClick= this.handleClick.bind(this);
14  //   //바인딩
15  // };
```

```
PS C:\react_study_chul_2\chul-app> yarn start
PS C:\react_study_chul_2\chul-app> yarn start
yarn run v1.22.5
```

### Arrow의 사용

```
7   handleClick= (e)=>{
8     console.log("click이벤트 동작")
9     this.setState(
10       {
11         num: this.state.num+1
12       }
13     );
14   }
```

```
render() {
  return (
    <div>
      <h2>{this.state.num}</h2>
      <button onClick={this.handleClick}>Ok</button>
    </div>
  );
}
```

### ○ State 내의 배열 관리 주의할 점.

```
constructor(){  
  super();  
  this.state={  
    mem:[],  
    inputValue:''  
  };  
  this.handleClick = this.handleClick.bind(this);  
  this.handleChange = this.handleChange.bind(this);  
} //end constructor
```



직접 state의 mem배열에 push하면 안됨.

this.state가 변경된다고 해서 컴포넌트가 업데이트되지 않기 때문이다.

this.state 수정은 this.setState()를 사용하여 수정 할것.

this.setState() 메서드가 실행되면 자동으로 re-rendering 이 진행된다.

```
handleClick(){  
  let addMem= this.state.mem;  
  addMem.push(this.state.inputValue);  
  this.setState({  
    mem: addMem  
  });  
}
```

## □ 5] State

\* arrow 방식

```
export class Counter extends Component{  
  
  // 아래 생성자와 동일한 기능  
  state={  
    |   num:0  
  }  
  
  // constructor(){  
  //   super();  
  //   this.state={  
  //     num:0  
  //   };  
  // }  
  
  //arrow 함수 사용하면 bind 필요없다.  
  handleClick=(e)=>{  
    |   this.setState({  
    |     num: this.state.num + 1  
    |   });  
  }  
}
```

## ○ State 내의 Input 태그 사용시 주의할 점.

```
<input type="text" name="username" value={this.state.inputValue}
<button onClick={this.handleClick}>저장</button>
```

### 문제점



Input 태그에 값을 입력하려고 해도  
값이 고정되어 변경 안됨.

```
constructor(){
  super();
  this.state={
    mem:[],
    inputValue:''
  };
  this.handleClick = this.handleClick.bind(this);
```

```
handleChange(e){
  // let nextState = {};
  // nextState[e.target.name] = e.target.value;
  // this.setState(nextState);
  this.setState({inputValue: e.target.value});
}
```

### 해결방법



onChange 이벤트를 이용하여  
텍스트 입력시 state를  
업데이트하도록 설정한다.

```
<input type="text" name="username" value={this.state.inputValue} onChange={this.handleChange}/>
```

## □ onChange를 이용한 input값의 활용-1

Reactjs

```
JS App.js JS contact.js X
src > component > JS Contact.js > Contact > constructor
3 export class Contact extends Component {
4   constructor() {
5     super();
6     this.state = {
7       contactDate: [
8         {name: "Allen", phone: '000-000-0001'},
9         {name: "Bob", phone: '000-000-0022'},
10        {name: "Charlie", phone: '000-000-0333'},
11        {name: "David", phone: '000-000-444'},
12      ],
13      username: "" // 사용자 이름 추가함
14    };
15    //바인딩
16    this.handle = this.handle.bind(this);
17    this.handleChange = this.handleChange.bind(this);
18  } //end constructor
```

```

사용자 추가 이름 <input type="text" name="username" id="kkk"
value={this.state.username} onChange={this.handleChange}></input>
<button onClick={this.handle}>사용자 추가</button>
추가된 사용자 이름 :{this.state.username}<br/>
div>
```

```

} //end constructor
handleChange(e) { //추가됨
  this.setState(
    {
      username: e.target.value
    }
  )
}
handle() {
```

## □ onChange를 이용한 input값의 활용-2

Reactjs

```
render() {  
  return (  
    <div>  
      <form onSubmit={this.handleEvent}>  
        아이디<input type="text" value={this.state.userid} name="userid" onChange={this.handleChange}></input><br/>  
        비밀번호<input type="text" value={this.state.passwd} name="passwd" onChange={this.handleChange}></input><br/>  
        <button>로그인</button>  
      </form>  
      아이디:{this.state.userid}<br/>  
      패스워드:{this.state.passwd}<br/>  
    </div>  
  );  
}
```

```
handleEvent(e){  
  e.preventDefault();  
}  
  
handleChange(e){ //수정됨  
  console.log(">>>", e.target.name, e.target.value);  
  let nextState={};  
  nextState[e.target.name]= e.target.value;  
  this.setState(nextState);  
}
```

```
constructor(){  
  super();  
  this.state={  
    userid:'',  
    passwd:''  
  };  
  this.hadleEvent= this.handleEvent.bind(this);  
  this.handleChange= this.handleChange.bind(this);  
} //end constructor  
handleEvent(e){
```



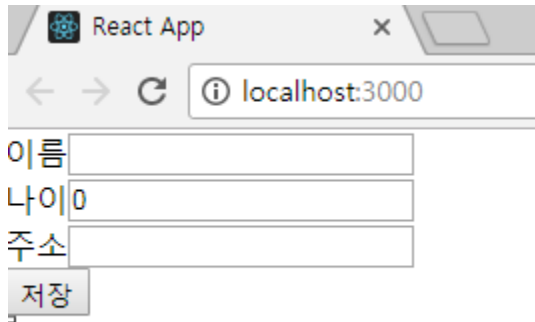
onChange를 사용하지 않음

```
render() {  
  return (  
    <div>  
      <form onSubmit={this.handleEvent}>  
        아이디<input type="text" ref="x" /><br />  
        비밀번호<input type="text" ref="y" /><br />  
        <button>로그인</button>  
      </form>  
      아이디:{this.state.userid}<br />  
      비번:{this.state.passwd}<br />  
    </div>  
  );  
}
```

```
3 export class Contact extends Component {  
4   constructor(){  
5     super();  
6     this.state={  
7       userid:"",  
8       passwd:""  
9     }  
10    this.handleEvent= this.handleEvent.bind(this);  
11  } //end constructor  
12  handleEvent(e){  
13    e.preventDefault();  
14    const {x, y} = this.refs;  
15    this.setState(  
16      {  
17        userid:x.value,  
18        passwd:y.value  
19      }  
20    )  
21  }
```

## □ 5] State

### ○ 실습 3



React App

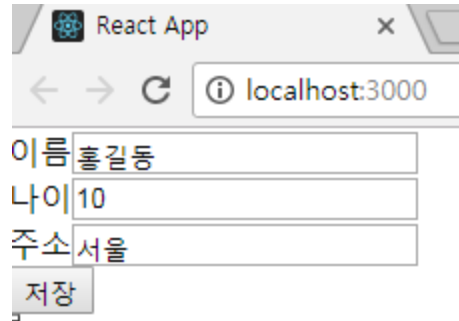
localhost:3000

이름

나이 0

주소

저장



React App

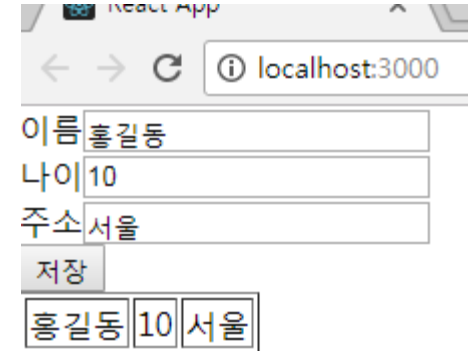
localhost:3000

이름 홍길동

나이 10

주소 서울

저장



React App

localhost:3000

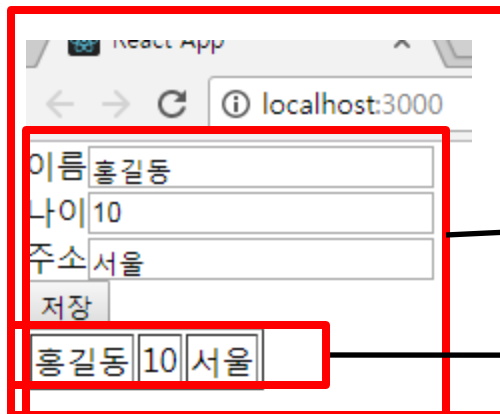
이름 홍길동

나이 10

주소 서울

저장

홍길동 10 서울



React App

localhost:3000

이름 홍길동

나이 10

주소 서울

저장

홍길동 10 서울

App.js

Member.js

MemberList.js

```
export class Member extend  
  
constructor(){  
  super();  
  this.state={  
    memberData:[],  
    username:"",  
    age:0,  
    address:""  
  }  
}
```

```

JS Member.js X JS App.js JS MemberList.js
src > component > JS Member.js > Member > constructor
15 }
16 render() {
17     return (
18         <div>
19             이름<input type="text" name="username" value={this.state.username}
20             onChange={this.handleChange}/><br></br>
21             나이<input type="text" name="age" value={this.state.age}
22             onChange={this.handleChange}/><br></br>
23             주소<input type="text" name="address" value={this.state.address}
24             onChange={this.handleChange}/><br></br>
25             <button onClick={this.addMember}>저장</button>
26             <button onClick={this.xyz.bind(this,"홍길동")}>xyz 함수 호출 </button>
27             <MemberList memberData={this.state.memberData} />
28             /* 데이터전송 */
29         </div>
30     );
31 } //end render

```

```
//input 태그상태 변경
handleChange(e){
  let nextState={};
  nextState[e.target.name]=e.target.value;
  this.setState(nextState);
}
//저장
addMember(e){
  let xxx= this.state.memberData;
  xxx.push({username:this.state.username, age:this.state.age, address:this.state.address});
  this.setState(
    {
      memberData:xxx
    }
  );
}
} //end handleChange
xyz(e){
  console.log("param===", e);
}
```

JS MemberList.js X

JS App.js

JS Member.js

src &gt; component &gt; JS MemberList.js &gt; MemberList &gt; render

```
1  import React, { Component } from 'react';
2
3  export class MemberList extends Component {
4    memberData;
5    render() {
6      this.memberData= this.props.memberData;
7      console.log(this.memberData);
8      return (
9        <table border="1">
10          <tbody>
11            <tr>
12              <td>이름</td>
13              <td>나이</td>
14              <td>주소</td>
15            </tr>
16            {
17              this.memberData.map(function(row, idx){
18                return (
19                  <tr key={idx}>
20                    <td>{row.username}</td>
21                    <td>{row.age}</td>
22                    <td>{row.address}</td>
23                  </tr>
24                )
25              })
26            }
27          </tbody>
28        </table>
29      );
30    }
31  }
```

## □ 7) 계층구조 이벤트 처리

- 자식에서 부모로 데이터 및 이벤트 전달
- (부모의 함수를 자식에게 전달, 자식에서 파라미터를 넘기며 부모의 함수를 호출함)

```
class App extends Component {  
  mesg = "홍길동";  
  
  constructor(){  
    super();  
    this.appHandleEvent = this.appHandleEvent.bind(this);  
  }  
  
  //이벤트  
  appHandleEvent(){  
    console.log("App.appHandleEvent");  
  }  
  
  render() {  
    return (  
      <div>  
        <Child mesg={this.mesg} />  
        <Child2 mesg={this.mesg} onEvent={this.appHandleEvent}/>  
      </div>  
    );  
  }  
}
```

```
class Child extends Component {  
  mesg;  
  constructor(){  
    super();  
    this.handleEvent = this.handleEvent.bind(this);  
  }  
  
  //이벤트  
  handleEvent(){  
    console.log("Child.handleEvent")  
  }  
  
  render() {  
    this.mesg = this.props.mesg;  
    return (  
      <div>  
        <h1>{this.mesg}</h1>  
        <button onClick={this.handleEvent}>OK</button>  
      </div>  
    );  
  }  
}
```

### \* 자식에서 부모로 데이터 및 이벤트 전달

```
class App extends Component {  
  mesg = "홍길동";  
  
  constructor(){  
    super();  
    this.appHandleEvent = this.appHandleEvent.bind(this);  
  }  
  
  //이벤트  
  appHandleEvent(){  
    console.log("App.appHandleEvent");  
  }  
  
  render() {  
    return (  
      <div>  
        <Child mesg={this.mesg} />  
        <Child2 mesg={this.mesg} onEvent={this.appHandleEvent}/>  
      </div>  
    );  
  }  
}
```

```
class Child2 extends Component {  
  mesg;  
  
  render() {  
    // 객체 destructuring  
    const { mesg, onEvent } = this.props;  
    return (  
      <div>  
        <h1>{mesg}</h1>  
        <button onClick={()=>{onEvent()}}>OK</button>  
        <button onClick={onEvent}>OK2</button>  
        /* <button onClick={onEvent()}>OK3</button> */  
      </div>  
    );  
  }  
}
```

```
C:\react_study_chul\chul-app>yarn info react
```

문제    출력    터미널    디버그 콘솔

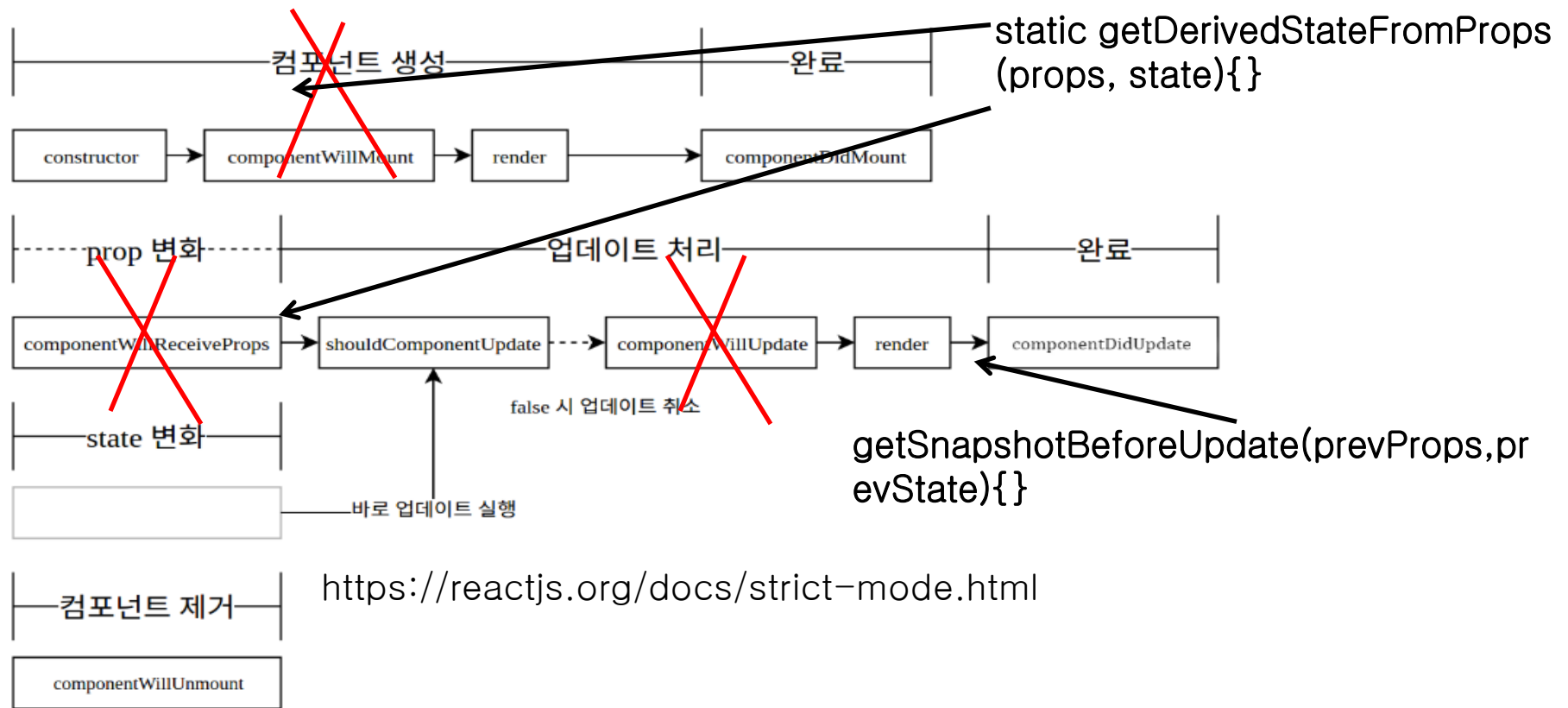
```
  andrialexandrou: true,
  zmzhoi: true,
  kc15155: true
},
license: 'MIT',
version: '17.0.1',
main: 'index.js',
engines: {
  node: '>=0.10.0'
},
dependencies: {
  'loose-envify': '^1.1.0',
  'object-assign': '^4.1.1'
```



## ❑ 7) Component LifeCycle API ( react 17 버전에서 변경됨 )

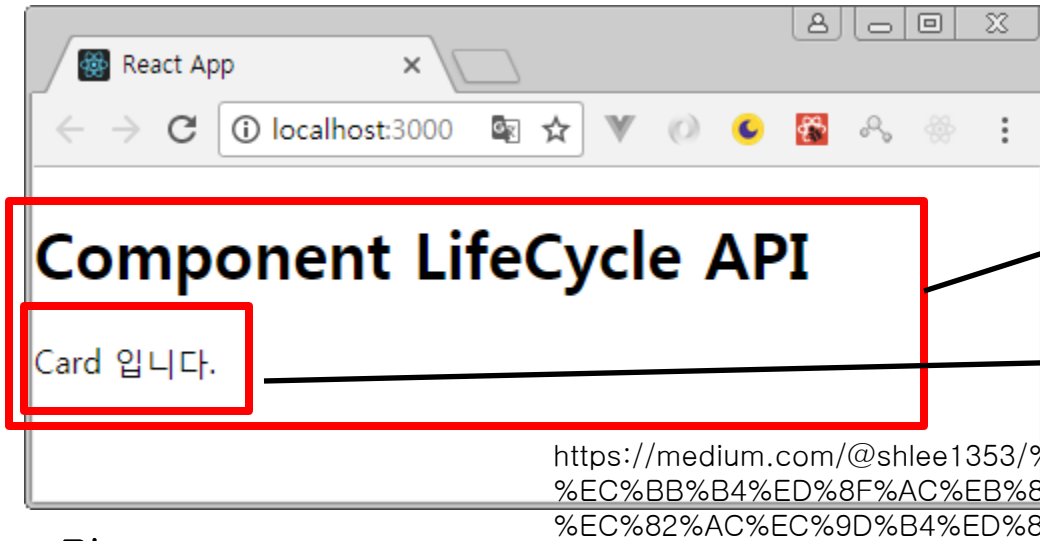
<https://reactjs.org/docs/react-component.html> 참조할 것

컴포넌트가 DOM 위에 생성되기 전/후, 데이터가 변경되어 상태를 업데이트하기 전/후로 실행되는 메서드를 의미한다.



These methods are called “lifecycle hooks”.

### 1. 실습 (step1 ) – 랜더링(rendering) 전/후



App.js

CardPanel.js

Card.js

Component LifeCycle API

Card 입니다.

컴포넌트 라이프사이클 초기 화면 로딩시:

- constructor
- getDerivedStateFromProps
- render
- componentDidMount

<https://medium.com/@shlee1353/%EB%A6%AC%EC%95%A1%ED%8A%B8-v16-3-%EC%BB%B4%ED%8F%AC%EB%84%8C%ED%8A%B8-%EB%9D%BC%EC%9D%B4%ED%94%84-%EC%82%AC%EC%9D%B4%ED%81%B4-%EC%A0%95%EB%A6%AC-a3a65de60beb>

#### 가. constructor

새로운 객체가 생성될 때마다 호출. super(props) 하여 this.props로 접근 가능, this.setState 또는 Ajax 불가

#### 나. static getDerivedStateFromProps()

컴포넌트 초기화 또는 새로운 props을 받았을 때, this 키워드 사용 불가.

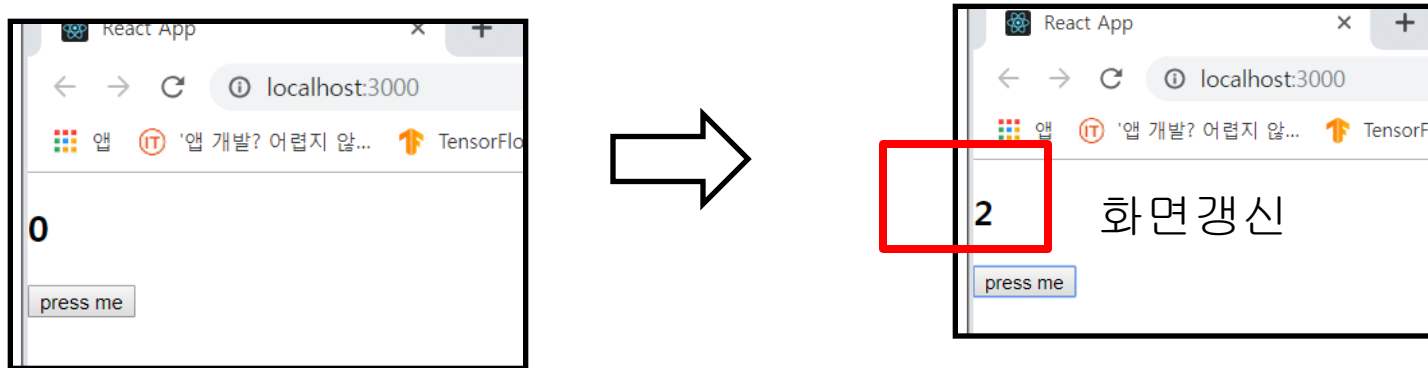
#### 다.render()

this.props와 this.state가 분석된다. state를 변경하면 안되고 브라우저와 직접 접근 불가

#### 라. componentDidMount()

Ajax 요청 같은 데이터 로직 처리에 적합.

### 1. 실습 (step2) - 화면 갱신 전/후: state 변경



```
shouldComponentUpdate(nextProps, nextState){}
```

state가 변경될 경우 shouldComponentUpdate 부터 lifecycle이 진행된다.

```
getSnapshotBeforeUpdate(prevProps, prevState){}
```

render() 메서드 호출후 DOM에 변화를 반영하기 바로 전에 호출  
여기에서 리턴된 값은 componentDidUpdate() 메서드의 세 번째 파라미터인 snapshot에서 받을 수 있다  
주로 업데이트하기 직전의 값을 참고할 때 사용된다..

```
componentDidUpdate (prevProps, prevState, snapshot){}
```

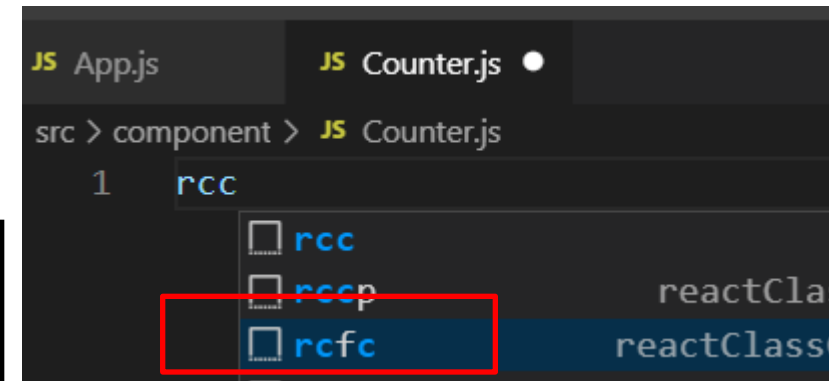
모든 re-rendering이 완료된 후에 호출된다. 따라서 DOM관련 처리 가능하다.  
prevProps와 prevState 값을 이용하여 컴포넌트가 이전에 가졌던 데이터에 접근할 수 있다.

## ❑ 7) Component Lifecycle API

```
export class Counter extends Component{  
  constructor(props){  
    console.log("Counter:constructor" );  
    super(props);  
    this.state={  
      num:0  
    };  
    this.handleClick = this.handleClick.bind(this);  
  } //end constructor
```

true: 화면 갱신  
false: 화면 갱신 안됨

```
// Lifecycle  
shouldComponentUpdate(nextProps, nextState){  
  console.log("Counter:shouldComponentUpdate" , "컴포넌트가 수정할지 결정중...");  
  console.log("Counter:nextProps" , nextProps);  
  console.log("Counter:nextState" , nextState);  
  
  if(nextState.num === 4){  
    return false;  
  }else{  
    return true;  
  }  
} //end shouldComponentUpdate  
  
componentDidMount(){  
  console.log("Counter:componentDidMount" , "랜더리 된 후");  
}
```



## ❑ 7) Component LifeCycle API

```
getSnapshotBeforeUpdate(prevProps, prevState){  
  console.log("Counter:getSnapshotBeforeUpdate" , "컴포넌트가 re-rendering된후 바로...");  
  console.log("Counter:prevProps" , prevProps);  
  console.log("Counter:prevState" , prevState);  
  
  return {"aa":300};  
}
```

```
componentDidUpdate (prevProps, prevState, snapshot){  
  console.log("Counter:componentDidUpdate" , "컴포넌트가 re-rendering 완료된 후.");  
  console.log("Counter:prevProps" , prevProps);  
  console.log("Counter:prevState" , prevState);  
  console.log("Counter:snapshot" , snapshot);  
}
```

### 1. 실습 (step3) – 새로운 props 받을 때

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Counter xxx={3}/>  
      </div>  
    );  
  }  
}
```

// LifeCycle

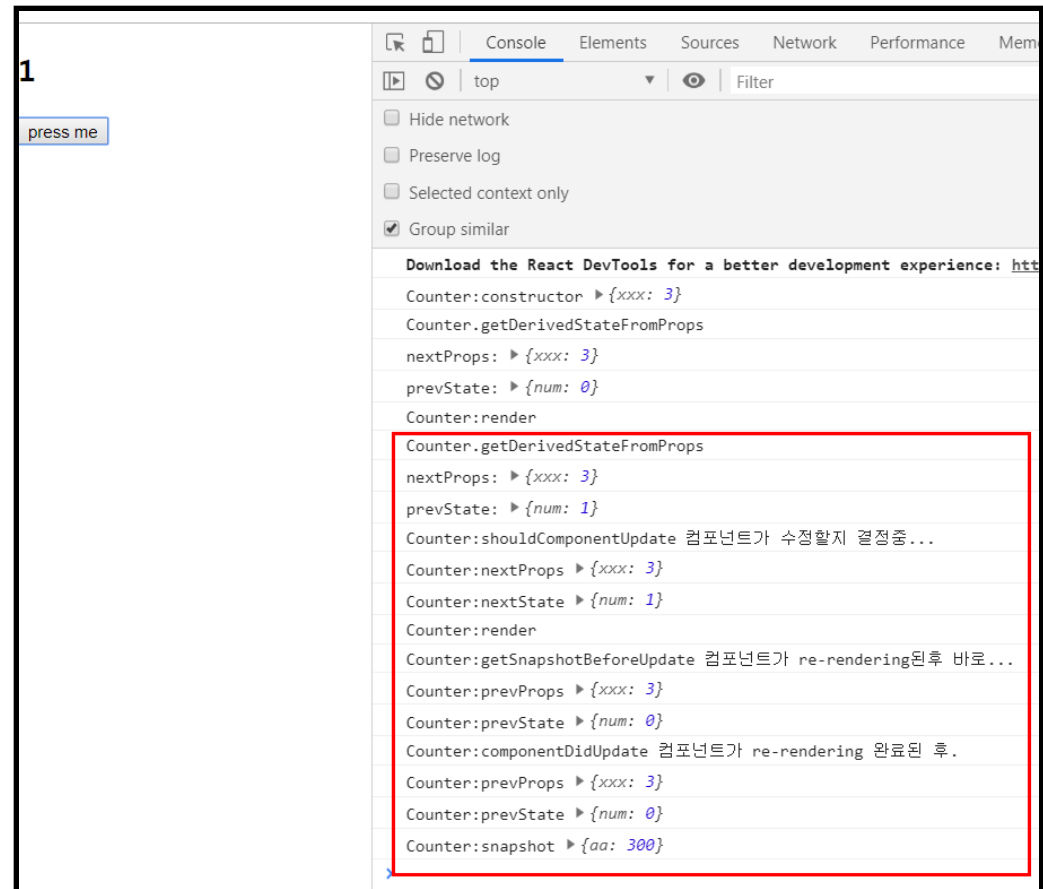
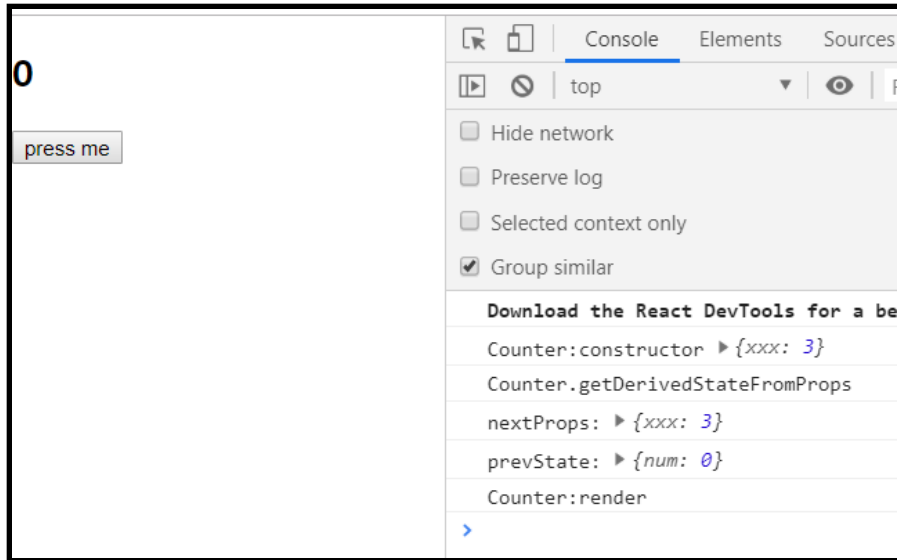
```
static getDerivedStateFromProps(nextProps, prevState){  
  console.log("Counter.getDerivedStateFromProps");  
  console.log("nextProps:" , nextProps);  
  console.log("prevState:" , prevState);  
  
  // props값과 state 값이 동일하면 state값 새로운 값 100으로 갱신(실)  
  if(nextProps.xxx === prevState.num){  
    return {num:100};  
  }else{  
    return null;  
  }  
}
```

```
export class Counter extends Component{  
  constructor(props){  
    super(props);  
    console.log("Counter:constructor" , props );  
    this.state={  
      num:0  
    };  
  }  
}
```

```
shouldComponentUpdate(nextProps, nextState){  
  console.log("Counter:shouldComponentUpdate" , "컴포넌트가  
  console.log("Counter:nextProps" , nextProps);  
  console.log("Counter:nextState" , nextState);  
  
  if(nextState.num === 4){  
    return false;  
  }else{  
    return true;  
  }  
} //end shouldComponentUpdate  
  
getSnapshotBeforeUpdate(prevProps, prevState){  
  console.log("Counter:getSnapshotBeforeUpdate" , "컴포넌트  
  console.log("Counter:prevProps" , prevProps);  
  console.log("Counter:prevState" , prevState);  
  
  return {"aa":300};  
}
```

## 7) Component LifeCycle API

Reactjs



### ○ 함수형 컴포넌트

- Lifecycle 메서드가 필요없고
- State도 사용하지 않고
- 그냥 **Props만 사용하는 경우에** 사용될 수 있는 컴포넌트 의미.

#### 1. 일반적인 Component 클래스 사용

```
class App extends Component {  
  render() {  
    return (  
      <div>  
        <ChildComponent msg="ChildComponent"/>  
        <ChildFunction msg="ChildFunction"/>  
        <ChildFunction2 msg="ChildFunction2"/>  
        <ChildDestructuring msg="ChildDestructuring"/>  
      </div>  
    );  
  }  
}
```

```
export class ChildComponent extends Component {  
  render(){  
    return(  
      <div>  
        <h1>함수형 컴포넌트(Functional Component)1</h1>  
        props:{this.props.msg}  
      </div>  
    );  
  }  
} //end Contact
```



### 2. 함수형 컴포넌트 사용

일반 함수

```
JS ChildFunction.js • JS App.js JS ChildFunction2.js JS ChildDestructuring.js
src > component > JS ChildFunction.js > ChildFunction
1 import React from 'react'; //주의
2 export default function ChildFunction (props) {
3     //render가 사라짐
4     return (
5         <div>
6             <h1>함수형컴포넌트(function Component) </h1>
7             props:{props.mesg}
8         </div>
9     );
10     //div 태그 리턴
11 }
12
13
```

Arrow 함수

```
const ChildFunction2=(props)=>{
    return(
        <div>
            <h1>함수형 컴포넌트(Functional Component)3</h1>
            props:{props.mesg}
        </div>
    );
};

export default ChildFunction2;
```

|          |                              |                                      |
|----------|------------------------------|--------------------------------------|
| rsc      | reactStateless               | Creates a stateless React componen X |
| rscm     | reactMemo                    | t without PropTypes and ES6 module   |
| rscp     | reactStatelessProps          | system(Reactjs code snippets)        |
| rscpm    | reactMemoProps               |                                      |
| rsf      | reactStatelessFunction       | import React from 'react';           |
| rsfp     | reactStatelessFunctionProps  |                                      |
| rsi      | reactStatelessImplicitReturn | const = () => {                      |
| resizeBy |                              | return (                             |

### 3. Object Destructuring

```
const ChildDestructuring=({mesg})=>{  
  return(  
    <div>  
      <h1>함수형 컴포넌트(Functional Component)3</h1>  
      props:{mesg}  
    </div>  
  );  
};  
  
export default ChildDestructuring;
```

```
class App extends Component {
  render() {
    return (
      <div>
        <ChildComment msg="ChildComponent" msg2="100"/>
        <ChildFunction msg="ChildFunction" msg2="100"/>
        <ChildFunction2 msg="ChildFunction2" msg2="100"/>
        <ChildDestructuring msg="ChildDestructuring" msg2="100"/>
      </div>
    );
  }
}
```

JS ChildComment.js

JS ChildFunction.js ×

JS ChildFunction2.js

JS

ponent > JS ChildFunction.js > ChildFunction

import React from 'react'; //주의

export default function ChildFunction (props) {

//render가 사라짐

const {msg, msg2}= props;

return (

<div>

<h1>함수형컴포넌트(function Component)2 </h1>

props:{msg}<br></br>

props:{msg2}<br></br>

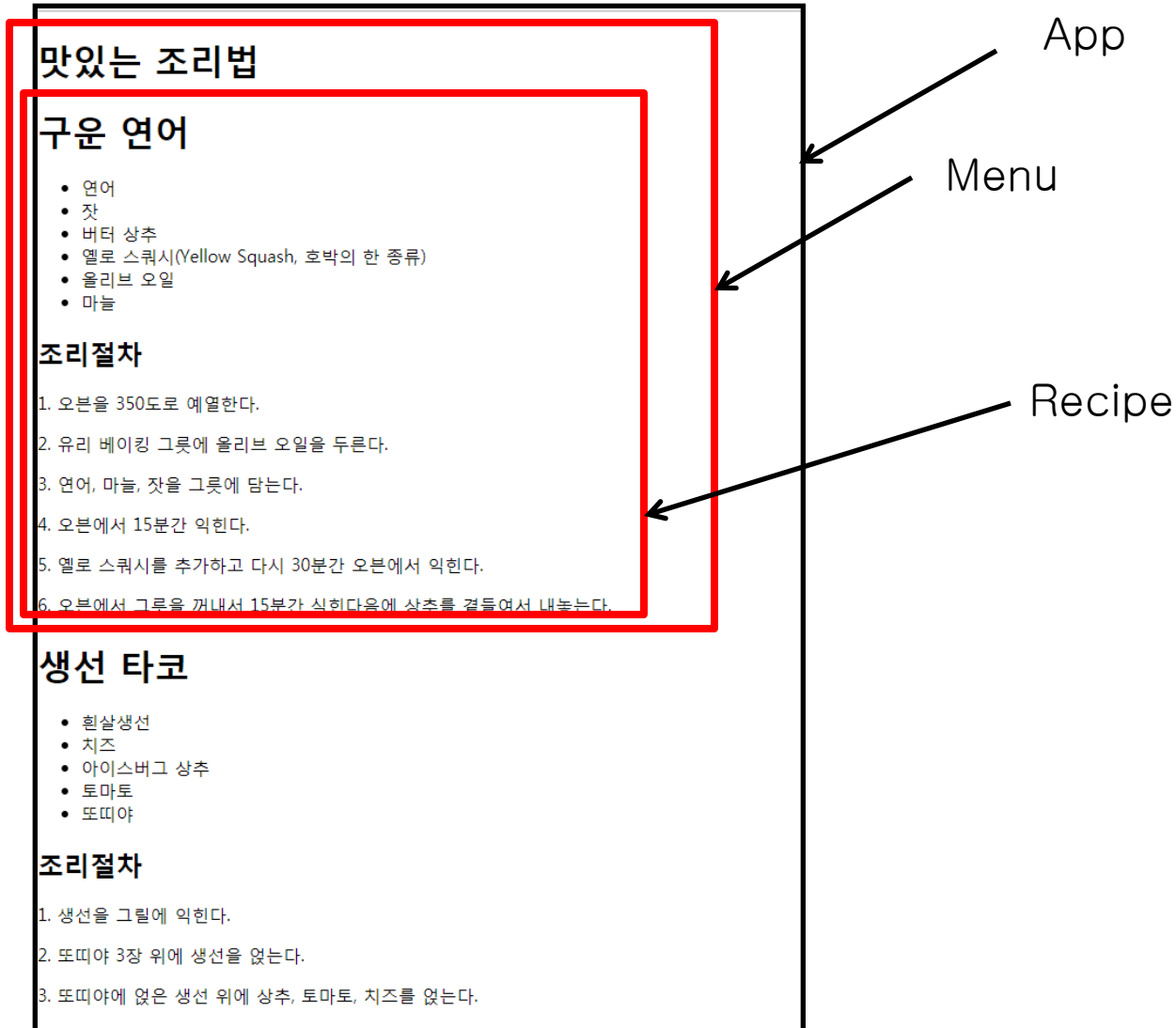
</div>

);

//div 태그 리턴

1

### \* 함수형 컴포넌트 샘플



### ○ Ajax 라이브러리

- jquery
- axios
- fetch
- promise 등

### ○ axios 라이브러리 (<https://reactjs.org/docs/faq-ajax.html>)

- <https://github.com/axios/axios> 참조

```
C:\W yarn add axios
```

```
C:\W yarn start
```

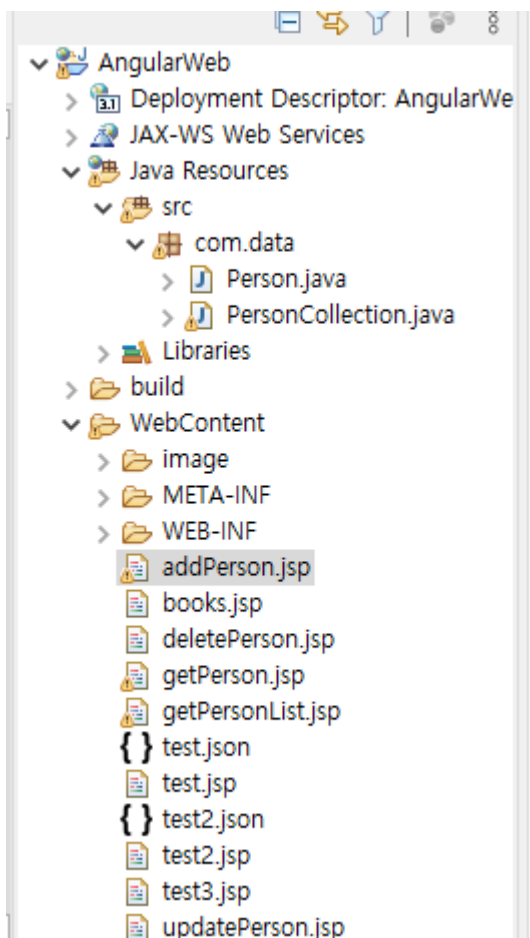
## □ 9) Ajax 연동(AngularWeb)

```
import axios from 'axios';
```

```
JS App.js x
src > JS App.js > App
3 class App extends Component {
4   msg="홍길동";
5   constructor(){
6     super();
7     this.handleClick= this.handleClick.bind(this);
8     this.handleClick2= this.handleClick2.bind(this);
9   }
10  //이벤트
11  handleClick(e){
12    axios.get("http://localhost:8090/AngularWeb/getPersonList.jsp").
13    then((responseData)=>{
14      console.log(responseData);
15    }).catch((error)=> {
16      console.log("error");
17    });
18  } //end handleClick
19  handleClick2(e){
20    const person={
21      id: 'a',
22      name: 'a',
23      age: '30'
24    };
25    axios.get('http://localhost:8090/AngularWeb/addPerson.jsp',
26      {params:person}).then((responseData)=>{
27      console.log(responseData);
28    }).catch((error)=>{console.log("error")});
29  } //end handleClick2
```

```
AngularWeb
  > Deployment Descriptor: AngularW
  > JAX-WS Web Services
  > Java Resources
    > src
      > com.data
        > Person.java
        > PersonCollection.java
    > Libraries
  > build
  > WebContent
    > image
    > META-INF
    > WEB-INF
    > addPerson.jsp
    > books.jsp
    > deletePerson.jsp
    > getPerson.jsp
    > getPersonList.jsp
    > test.json
    > test.jsp
    > test2.json
    > test2.jsp
    > test3.jsp
    > updatePerson.jsp
```

id, name, email 로 파싱



```

getPersonList.jsp
addPerson.jsp
5  <!-- page language= java contentType= text/html; charset=UTF-8
   pageEncoding=UTF-8 -->
6  <%
7  response.setHeader("Access-Control-Allow-Origin", "*");
8  PersonCollection collection = PersonCollection.getInstance();
9  ArrayList<Person> list = collection.getList();
10 StringBuffer buffer = new StringBuffer();
11 buffer.append("[");
12 for(int i=0;i<list.size();i++){
13     Person p = list.get(i);
14     String msg = "{";
15     msg+="\"id\": \""+p.getId()+"\", ";
16     msg+="\"name\": \""+p.getName()+"\", ";
17     msg+="\"age\": \""+p.getAge()+"\"";
18     msg+="}";
19     buffer.append(msg);
20     if(i!= list.size()-1)buffer.append(",");
21 }
22 buffer.append("]");
23 System.out.println(buffer.toString());
24 //[{ "id": "01", "name": "aaa", "age": "20" }]
25 out.print(buffer.toString().trim());
26 %>

```





```
list(){
  var url="http://localhost:8077/app/list";
  axios.get(url)
    .then((responseData)=>{
      console.log(responseData);
      this.setState(
        {
          deptData:responseData.data
        }
      )
    }).catch((error)=>{console.log("error")});
} //end list
```

```
//저장이벤트
onAppSave(x){
  var url="http://localhost:8077/app/add";
  axios.post(url, {deptno:x.deptno.value, dname:x.dname.value, loc:x.loc.value})
    .then((responseData)=>{
      var arr= this.state.deptData;
      arr.push({deptno:x.deptno.value, dname:x.dname.value, loc:x.loc.value});
      this.setState({
        deptData:arr
      });
    }).catch((error)=>{console.log("error")});
} //end onAppSave

//삭제이벤트
onAppDelete(x){
  var url="http://localhost:8077/app/del?deptno="+x.value;
  console.log("x>>>>" + x.value);
  axios.delete(url).then((responseData)=>{
    this.list();
  }).catch((error)=>{console.log("error")});
} //end onAppDelete

render() {
```

## ❑ 10] 라우팅 (<https://reacttraining.com/react-router/web/api/Route>)

Reactjs

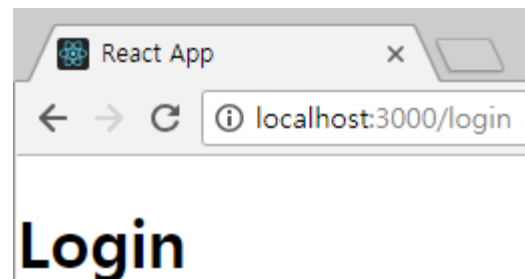
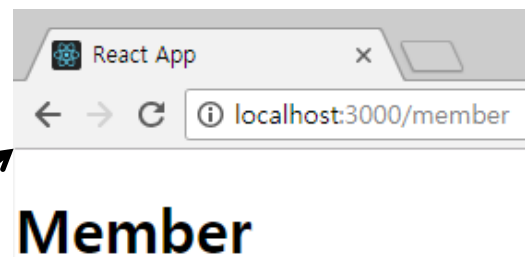
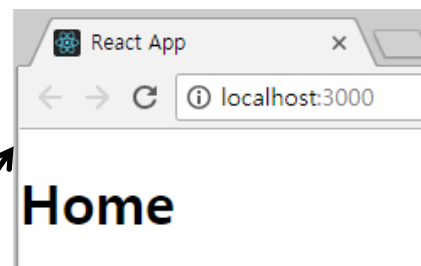
### ○ react-router-dom 라이브러리

```
c:\a_reactjs\94ReactJS_new\reactStudy\my-router>npm install react-router-dom
```

```
C:\react_study_chul\chul-app>yarn start  
yarn run v1.22.5
```

기본 1

```
import React, { Component } from 'react';  
import './App.css';  
  
import {BrowserRouter as Router, Route} from 'react-router-dom';  
import Home from './Home';  
import Member from './Member';  
import Login from './Login';  
  
class App extends Component {  
  render() {  
    return (  
      <div>  
        <Router>  
          <div>  
            <Route exact path="/" component={Home} />  
            <Route exact path="/member" component={Member} />  
            <Route exact path="/login" component={Login} />  
          </div>  
        </Router>  
      </div>  
    );  
  }  
}
```

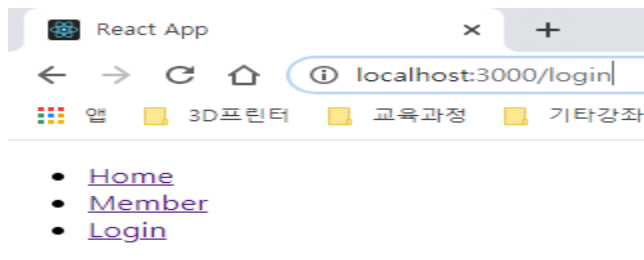


```
////////////////////////////////////////
import {BrowserRouter as Router, Route, Link} from 'react-router-dom';
////////////////////////////////////////
import {Home} from './component/Home';
import {Member} from './component/Member';
import {Login} from './component/Login';
class App extends Component {
  render() {
    return (
      <Router>
        { /* import Link추가 */ }
        <div>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/member">member</Link>
            </li>
            <li>
              <Link to="/login">login</Link>
            </li>
          </ul>
        </div>
      </Router>
    );
  }
}
```

```
</ul>
<hr/>
<div>
  <Route exact path="/" component={Home} />
  <Route exact path="/member" component={Member} />
  <Route exact path="/login" component={Login} />
</div>
</div>
```

## □ 라우팅(Switch의 사용)- 주소에 부합되지 않는 주소의 처리

```
JS App.js  X  JS Nav.js  JS NoMatch.js
src > JS App.js > App > render
1 | import React, { Component } from 'react';
2 | import './App.css';
3 | //////////////////////////////////////////////////
4 | import {BrowserRouter as Router, Route, Link, Switch} from 'react-router-dom';
5 | //////////////////////////////////////////////////
6 | import {Home} from './component/Home';
7 | import {Member} from './component/Member';
8 | import {Login} from './component/Login';
9 | import NoMatch from './component/NoMatch';
10 | class App extends Component {
11 |   render() {
```



Login

NoMathch

```
<div>
  <ul>
    <li>
      <Link to="/">Home</Link>
    </li>
    <li>
      <Link to="/member">member</Link>
    </li>
    <li>
      <Link to="/login">login</Link>
    </li>
  </ul>
  <hr/>
  {/* swithc가 없으면 NoMatch가 항상 출력됨,switch는 일체
  <Switch>
    <Route exact path="/" component={Home} />
    <Route exact path="/member" component={Member} />
    <Route exact path="/login" component={Login} />
    <Route component={NoMatch}/>
  </Switch>
</div>
```

## □ 10] 라우팅[Navi]의 사용

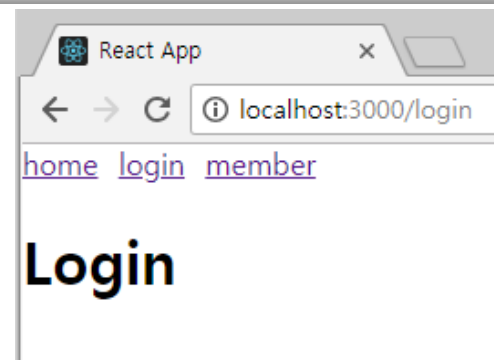
### 기본 2: 링크 설정

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

import { NavLink } from 'react-router-dom';

class Navi extends Component {
  render() {
    return (
      <div>
        <div>
          <NavLink to="/">home</NavLink>
          <NavLink to="/login">login</NavLink>
          <NavLink to="/member">member</NavLink>
        </div>
      </div>
    );
  }
}

export default Navi;
```



```
class App extends Component {
  render() {
    return (
      <div>
        <Router>
          <div>
            <Navi />
            <Route exact path="/" component=
            <Route exact path="/member" c
            <Route exact path="/login" co
          </div>
        </Router>
      </div>
    );
  }
}
```

## □ 10] 라우팅[Redirect의 사용]

```
class Navi extends Component {
  render() {
    return (
      <ul>
        <li>
          <NavLink to="/">Home</NavLink>
        </li>
        <li>
          <NavLink to="/member">member</NavLink>
        </li>
        <li>
          <NavLink to="/login">login</NavLink>
        </li>
        <li>
          <NavLink to="/users/login">users/login</NavLink>
        </li>
      </ul>
    );
  }
}
```

```
class App extends Component {
  render() {
    return (
      <Router>
        { /* import Switch */ }
        { /* Navi외부파일로 */ }
        <Navi/>
        <hr/>
        <div>
          { /* switch가 없으면 NoMatch가 항상 출력됨, switch는 일치하는 첫번째 */ }
          <Switch>
            <Route exact path="/" component={Home} />
            <Route exact path="/member" component={Member} />
            { /* <Route exact path="/login" component={Login} /> */ }
            //Redirect처리 시작
            <Redirect from='/users/login' to='/login'/>
            <Route exact path='/login' component={Login}/>
            //Redirect처리 끝
            <Route component={NoMatch}/>
          </Switch>
        </div>
      </Router>
    );
  }
}
```

### 기본 3: 클래스 대신에 함수 형식 사용

```
import React from 'react';
```

```
// class Home extends Component {  
//   render() {  
//     return (  
//       <div>  
//         <h1>Home</h1>  
//       </div>  
//     );  
//   }  
// }
```

```
const Home = ()=>{  
  return (  
    <div>  
      <h1>Home</h1>  
    </div>  
  );  
};
```

```
export default Home;
```

```
import React from 'react';
```

```
// class Login extends Component {  
//   render() {  
//     return (  
//       <div>  
//         <h1>Login</h1>  
//       </div>  
//     );  
//   }  
// }
```

```
const Login = ()=>{  
  return (  
    <div>  
      <h1>Login</h1>  
    </div>  
  );  
};
```

```
export default Login;
```

### 기본 4: 파라미터 처리

라우트로 설정한 컴포넌트는, 3가지의 props 를 전달받게 됩니다:

- `history` 이 객체를 통해 `push`, `replace` 를 통해 다른 경로로 이동하거나 앞 뒤 페이지로 전환 할 수 있습니다.
- `location` 이 객체는 현재 경로에 대한 정보를 지니고 있고 URL 쿼리 ( `/about?foo=bar` 형식) 정보도 가지고 있습니다.
- `match` 이 객체에는 어떤 라우트에 매칭이 되었는지에 대한 정보가 있고 `params` ( `/about/:name` 형식) 정보를 가지고 있습니다.

```
class Member extends Component {  
  render() {  
    console.log(this.props.match);  
    console.log(this.props.history);  
    console.log(this.props.location);  
    return (  
      <div>  
        <h1>Member</h1>  
      </div>  
    );  
  }  
}
```

```
const Member = ({match, history, location}) => {  
  console.log(match);  
  console.log(history);  
  console.log(location);  
  return (  
    <div>  
      <h1>Member</h1>  
    </div>  
  );  
};
```



React App

localhost:3000/member/홍길동

home login member

## Member

match

```

{path: "/member/:id", url: "/member/홍길동", isExact: true, params: {...}}
  isExact: true
  params: {id: "홍길동"}
  path: "/member/:id"
  url: "/member/홍길동"
  __proto__: Object

```

history

```

{length: 48, action: "PUSH", location: {...}, createHref: f, push: f, ...}
  action: "PUSH"
  block: f block()
  createHref: f createHref(location)
  go: f go(n)
  goBack: f goBack()
  goForward: f goForward()
  length: 48
  listen: f listen(listener)
  location: {pathname: "/member/홍길동", search: "", hash: "", state: undefined, key: "oqne9n"}
  push: f push(path, state)
  replace: f replace(path, state)
  __proto__: Object

```

location

```

{pathname: "/member/홍길동", search: "", hash: "", state: undefined, key: "oqne9n"}
  hash: ""
  key: "oqne9n"
  pathname: "/member/홍길동"
  search: ""
  state: undefined
  __proto__: Object

```

Npm install query-string

Import queryString from 'query-string'

```
3 // npm install query-string
```

```
4 import queryString from 'query-string';
```

Version 17 설치 필요없음



```
JS Home.js X
src > component > JS Home.js > ...
1 import React, { Component } from 'react';
2 //함수 구현
3 const Home=({history})=>{
4   return(
5     <div>
6       <h1>Home</h1>
7       <button onClick={()=>history.push('/login')}>login화면으로</button>
8     </div>
9   );
10 };
11 export default Home;
12
13
```

```

JS Home.js JS Member.js X JS MyPage.js JS Navi.js
src > component > JS Member.js > Member
1 import React, { Component } from 'react';
2 import queryString from 'query-string';
3 //함수로 구현
4 //npm install query-string , import queryString from
5 const Member=({match, history, location})=>{
6     console.log("match: ", match);
7     console.log("history : ", history);
8     console.log("location :", location);
9     const query=queryString.parse(location.search);
10    console.log("query>>>", query);
11    console.log("xxx>>>", query.xxx);
12    return(
13        <div>
14            <h1>Member</h1>
15            params:{match.params.id}입니다.<br></br>
16            queryString:{query.xxx}
17        </div>
18    );
19 };
20 export default Member;

```

```

JS MyPage.js X
src > component > JS MyPage.js > ...
1 import React, { Component } from 'react';
2 import {Redirect} from 'react-router-dom';
3
4 const isLogin=false;
5
6 const MyPage=()=>{
7     return(
8         <div>
9             <h1>Mypage</h1>
10             {(isLogin)?'마이페이지':<Redirect to="/login"/>}
11         </div>
12     )
13 };
14 export default MyPage;

```

## 기본 5: 중첩 라우터

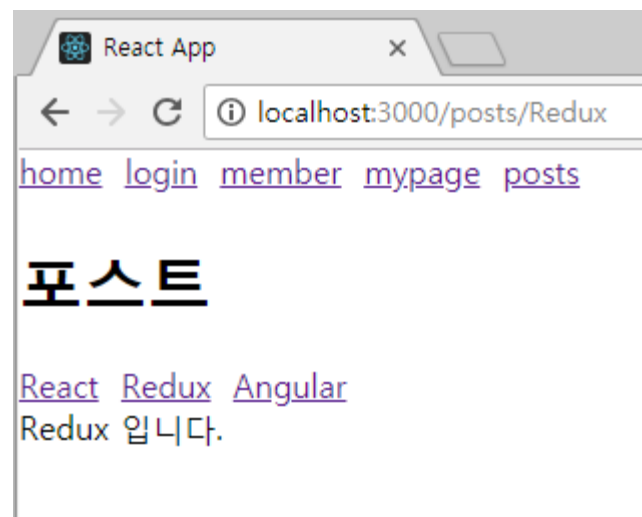
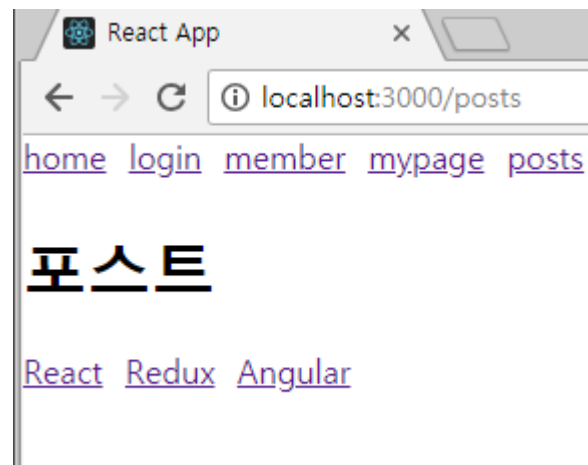
```
import React from 'react';

//중첩 라우터
import {Route,Link} from 'react-router-dom';

const Post = ({match}) => {
  return (
    <div>
      {match.params.title} 입니다.
    </div>
  );
};

const Posts = () => {
  return (
    <div>
      <h1>포스트</h1>
      <Link to="/posts/React">React</Link>
      <Link to="/posts/Redux">Redux</Link>
      <Link to="/posts/Angular">Angular</Link>
      <Route path="/posts/:title" component={Post} />
    </div>
  );
};

export default Posts;
```



```
1
2
3 class App extends Component {
4   render() {
5     return (
6       <Router>
7         <div>
8           <Navi/>
9           <Route exact path="/" component={Home} />
10          <Route path="/member/:id" component={Member} />
11          <Route path="/login" component={Login} />
12          <Route path="/myPage" component={MyPage} />
13          <Route path="/posts" component={Posts}/>
14        </div>
15      </Router>
16    ).
17  }
```

Exact의  
삭제



**Thank you**

---