차세대 챗봇 **Lab I**

Lecture-1 React Basic

Sungja Choi,
School of Computing, Gachon University
2023-1

가천대학교
Gachon University

학습내용

리액트의 기본내용에 대해 학습합니다.

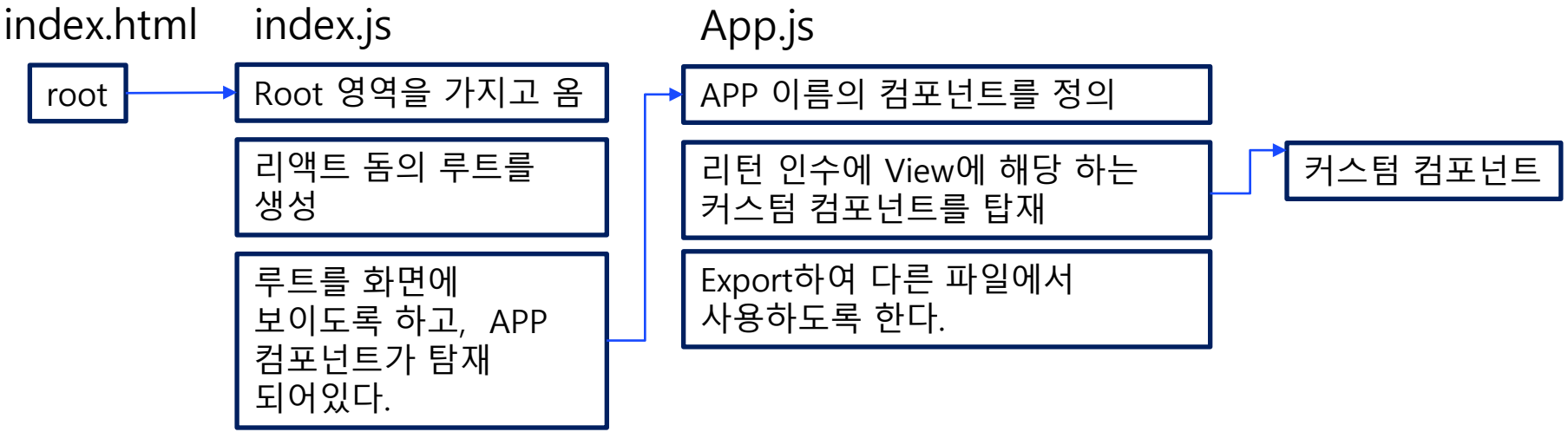
- 컴포넌트
- 자주 쓰는 문법 정리
- Props
- State
- 라이프사이클
- 이벤트
- 리액트 렌더링
- 리스트
- 라우터

Lecute-1. React Basic

리액트의 기본내용을 학습합니다.

[참조]
https://ko.reactjs.org/
https://www.w3schools.com/

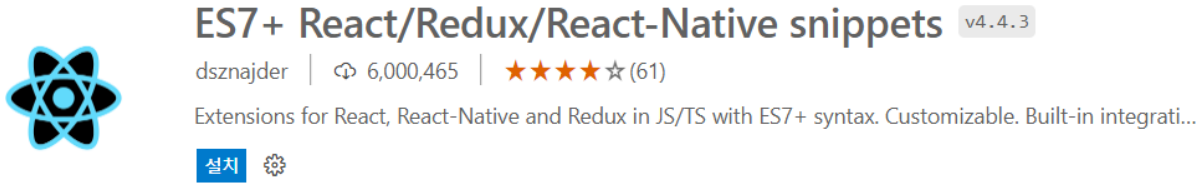
리액트의 문서 참조 패턴


index.html
index.js
App.js
root
Root 영역을 가지고 옴
리액트 돔의 루트를 생성
루트를 화면에 보이도록 하고, APP 컴포넌트가 탐재 되어있다.
APP 이름의 컴포넌트를 정의
리턴 인수에 View에 해당 하는 커스텀 컴포넌트를 탑재
Export하여 다른 파일에서 사용하도록 한다.
커스텀 컴포넌트

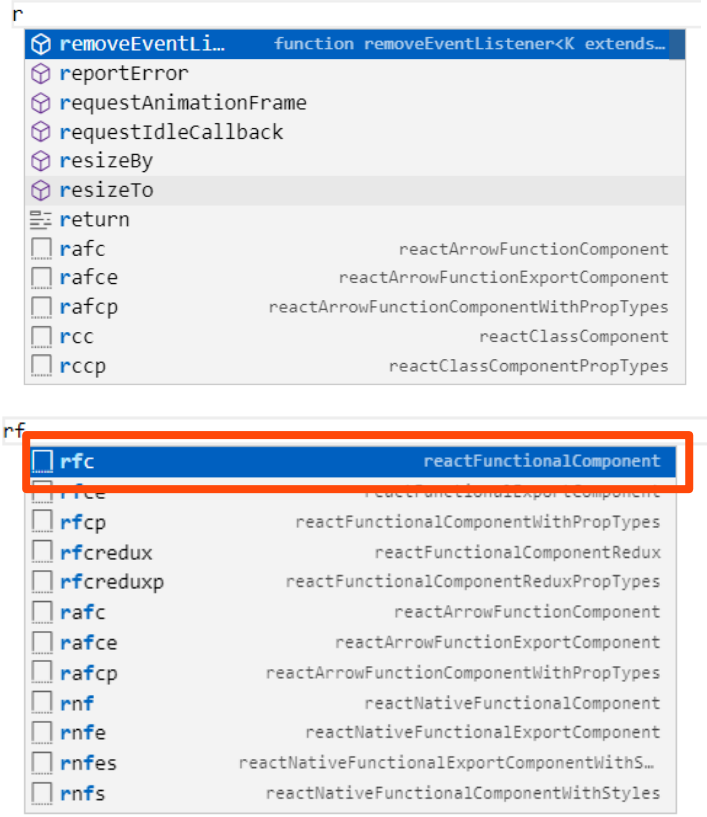
컴포넌트

- 화살형 컴포넌트
- 함수형 컴포넌트
- 클래스형 컴포넌트

[REF.] VSCode의 익스텐션을 활용하여 패키지를 설치하고 자동코드완성 기능을 사용해 보세요.



React Basic 컴포넌트



☞ 화살형 컴포넌트

```
import React from 'react'

export const Student = ( ) => {
  return (
    <div>Student</div>
  )
}
```

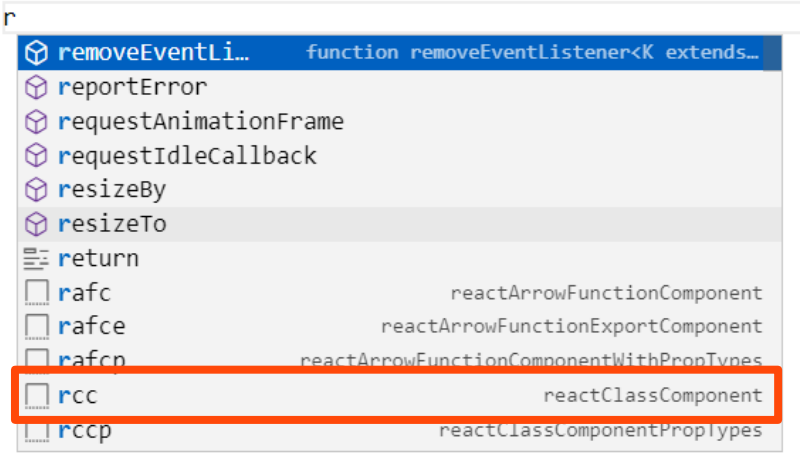
r을 입력하면 화살형 컴포넌트 패턴 생성

☞ 함수형 컴포넌트

```
import React from 'react'

export default function Student() {
  return (
    <div>Student</div>
  )
}
```

rfc를 입력하면 함수형 컴포넌트 패턴 생성

React Basic 컴포넌트



rcc를 체크하면 클래스형 컴포넌트 패턴 생성

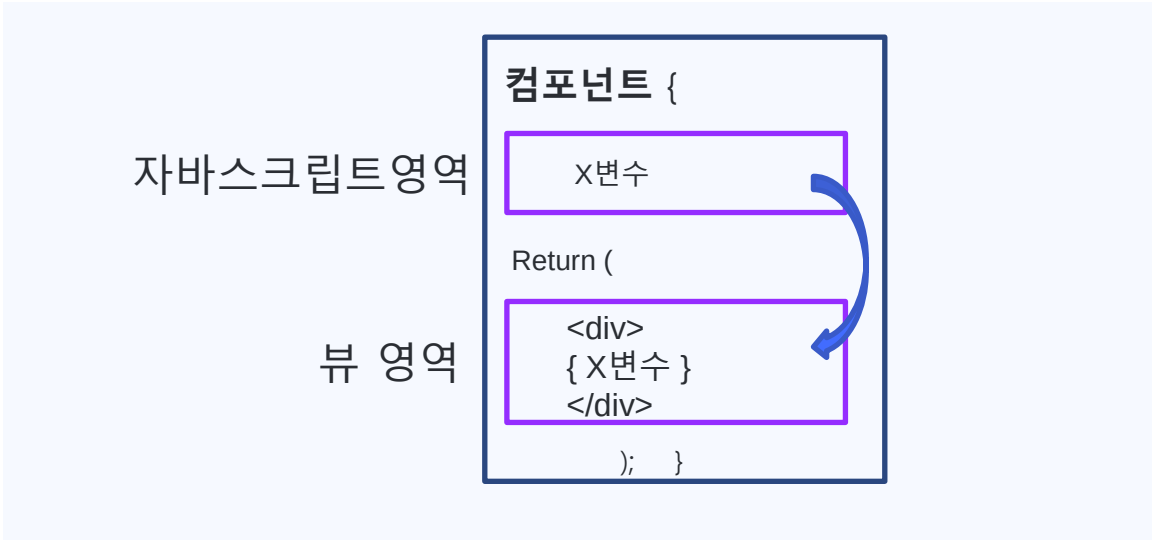
☞ 클래스형 컴포넌트

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

export default class Student extends
Component {
  render() {
    return (
      <div>Student</div>
    )
  }
}
```

문서 구조

☞ { } 중괄호 사용

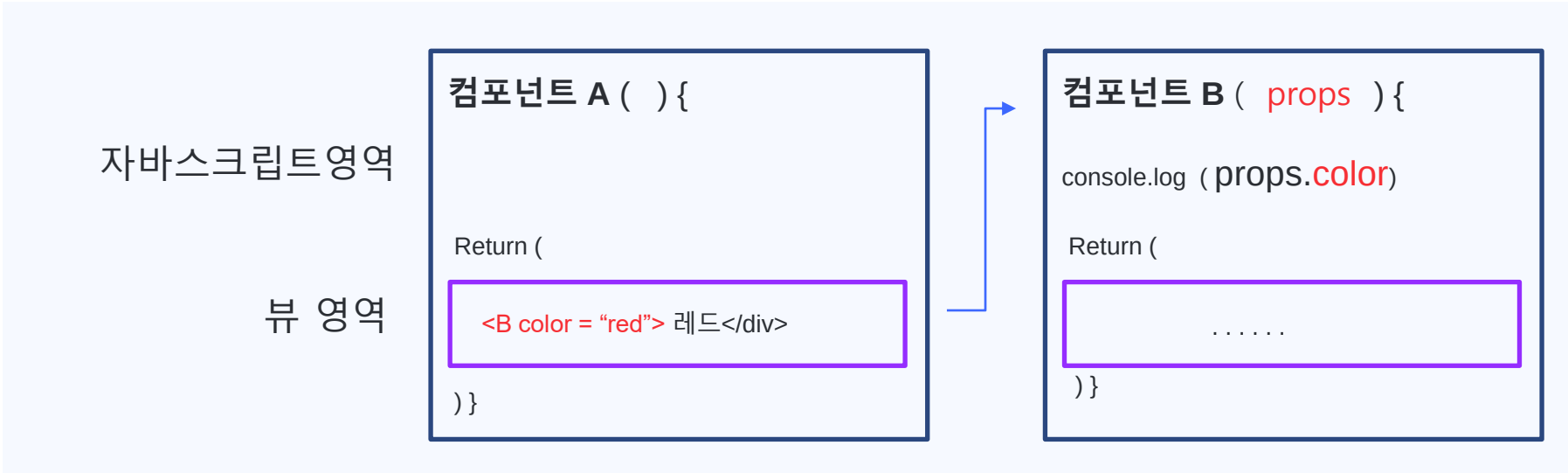
(예) 자바스크립트 영역의 변수를 JSX 태그 내에서 사용하고자 할 때
{변수명} 형태로 사용



문서 구조

☞ 엘리멘트의 속성 값을 컴포넌트로 전달

컴포넌트 A의 뷰 영역의 color 속성을 컴포넌트 B로 props를 사용해서 전달



기타 표기

☞ 엘리멘트의 속성 값을 전달 컴포넌트로 전달

- === 같은지를 체크
- ... 복사
- $ 백틱 `` 내에서 변수 표현할 때 사용
- => 값을 대입, 주로 함수의 인수에서 사용
- 속성명은 Camel 표현 사용, class => className

자주 쓰는 문법 정리

❶ 컴포넌트 안에서 쓰는 if/else

```
function Component() {
  if ( true ) {
    return <p>Gachon</p>;
  } else {
    return null;
  }
}
```

```
function Component() {
  if ( true ) {
    return <p>Gachon</p>;
  }
}
```

삼항 연산자

❷ JSX안에서 쓰는 삼항연산자

```
function Component() {
  return (
    <div>
      {
        1 === 1
        ? <p> Gachon </p>
        : null
      }
    </div>
  )
}
```

```
function Component() {
  return (
    <div>
      {
        1 === 1
        ? <p> Gachon </p>
        : ( 2 === 2
            ? <p> Cacao </p>
            : <p> University</p>
          )
      }
    </div>
  )}
```

삼항 연산자

❸ && 연산자로 if 역할 대신하기

```
function Component() {
  return (
    <div>
      {
        1 === 1
        ? <p> Gachon </p>
        : null
      }
    </div>
  )
}
```

```
function Component() {
  return (
    <div>
      { 1 === 1 && <p> Gachon </p> }
    </div>
  )
}
```

조건문

❹ switch / case 조건문

```
function Component(){
  var user = ‘cacao';
  switch (user){
    case ‘cacao' :
      return <h4>과정수료</h4>
    case  ‘gachon' :
      return <h4>졸업신청</h4>
    default :
      return <h4>수강신청</h4>
  }
}
```

조건문

❺ object/array 자료형 응용

```
function Component() {
  var 현재상태 = 'info';
  return (
    <div>
      {
        {
          info : <p>수강정보</p>,
          student : <p>학사관리</p>,
          grade : <p>성적관리</p>

        } [현재상태]
      }

    </div>
  ) }
```

```
var 탭UI = {
  info : <p>수강정보</p>,
  student : <p>학사정보</p>,
  grade : <p>성적정보</p>
}

function Component() {
  var 현재상태 = 'info';
  return (
     <div>
      {
        탭UI[현재상태]
      }
    </div>
  ) }
```

Props

☞ HTML 속성을 통해 리액트 컴포넌트로 전달되는 값

```
<컴포넌트명 속성="전달 값" />
```

```
function 컴포넌트명(props){

        return {props.속성}
}
```

Props（Properties）

☞ HTML 속성이 리액트 컴포넌트로 전달할 때 사용

>> Garage 컴포넌트의 뷰 부분에서 Car 컴포넌트의 **brand** 속성값인 Ford로 지정

>> Car 컴포넌트에서 파라메터로 **props**를 사용해서 전달받음

>> Car 컴포넌트에서 **props.brand**를 사용해서 뷰로 리턴

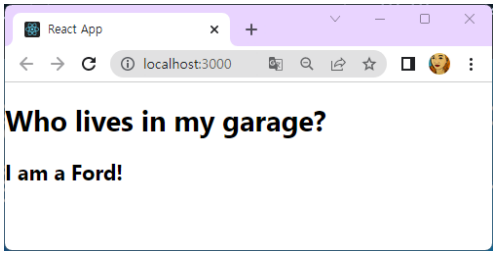
Garage.js

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
    </>
  );
}


export default Garage;
```

App.js

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

>> npm start



Props

>> Garage 컴포넌트의 자바스크립트 부분에서 **carName** 상수변수의 값을 Ford로 지정

>> Garage 컴포넌트의 뷰 부분에서 Car 컴포넌트의 **brand** 속성을 carName으로 지정

ex(2)

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}

function Garage() {
 const carName = "Ford";
 return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={carName} />
    </>
  );
}

export default Garage;
```

>> Car 컴포넌트에서 파라메터로 **props**를 사용해서 전달받음

>> Car 컴포넌트에서 **props.brand**를 사용해서 뷰로 리턴

Props

>> Garage 컴포넌트의 자바스크립트 부분에서 **carInfo** 리스트 지정

>> Garage 컴포넌트의 뷰 부분에서 Car 컴포넌트의 **brand** 속성을 carInfo로 지정

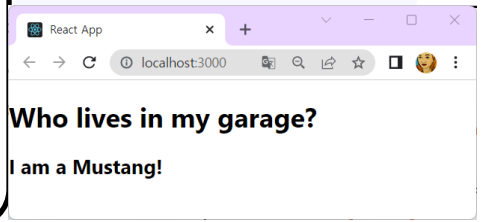
ex(3)

```
function Car(props) {
  return <h2>I am a { props.brand.model }!</h2>;
}

function Garage() {
 const carInfo = { name: "Ford", model: "Mustang" };
 return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={carInfo} />
    </>
  );
}

export default Garage;
```

>> Car 컴포넌트에서 파라메터로 **props**를 사용해서 전달받음

>> Car 컴포넌트에서 **props.brand.model**로 리스트 멤버 접근하고 뷰로 리턴

>> npm start



State

☞ 현재 컴포넌트에서 생성, 변할 수 있는 데이터를 지정

생성된 컴포넌트 내에서만 state 변경이 가능하고, 반드시 객체 형태로 생성되거나 null (state를 정의하지 않음) 타입으로 생성

this.state = { 업데이트할 state property: 값 } ;

setState({ 업데이트할 state property: 값 })

State

>> 클래스 컴포넌트는 state 변수를 가지고 있기 때문에 멤버를 지정할 수 있음

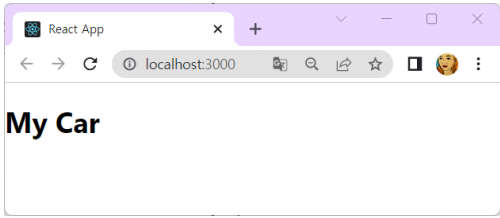
Car.js

```
import React from "react";
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    }; }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }}
export default Car;
```

App.js

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

>> npm start



React Basic **State**

>> 뷰 부분에서 state 멤버를 접근

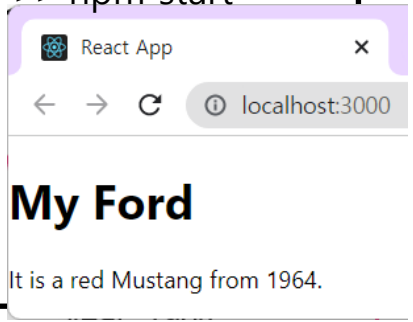
```
import React from "react";
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
```

Car.js

```
render() {
  return (
    <div>
      <h1>My {this.state.brand}</h1>
      <p>
        It is a {this.state.color}
        {this.state.model}
        from {this.state.year}.
      </p>
    </div>
  );
 }
}
export default Car;
```

```
<span ></span>
<span ></span>
```

>> npm start



State

>> 자바스크립트 부분에서 이벤트 컴포넌트(change Color)를 정의

>> setState메소드로 color를 blue로 지정

Car.js

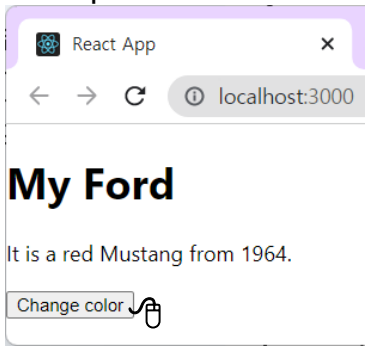
```
import React from "react";
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
```

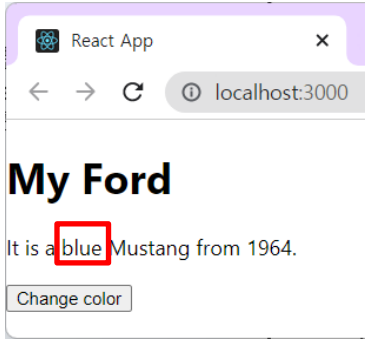
State

>> 뷰 부분에서 버튼이 클릭되면 changeColor 컴포넌트 호출

```
render() {
  return (
    <div>
      <h1>My {this.state.brand}</h1>
      <p>
        It is a {this.state.color}
        {this.state.model}
        from {this.state.year}.
      </p>
      <button
        type="button"
        onClick={this.changeColor} >
       Change color</button>
    </div>
  );  }}
export default Car;
```

>> npm start

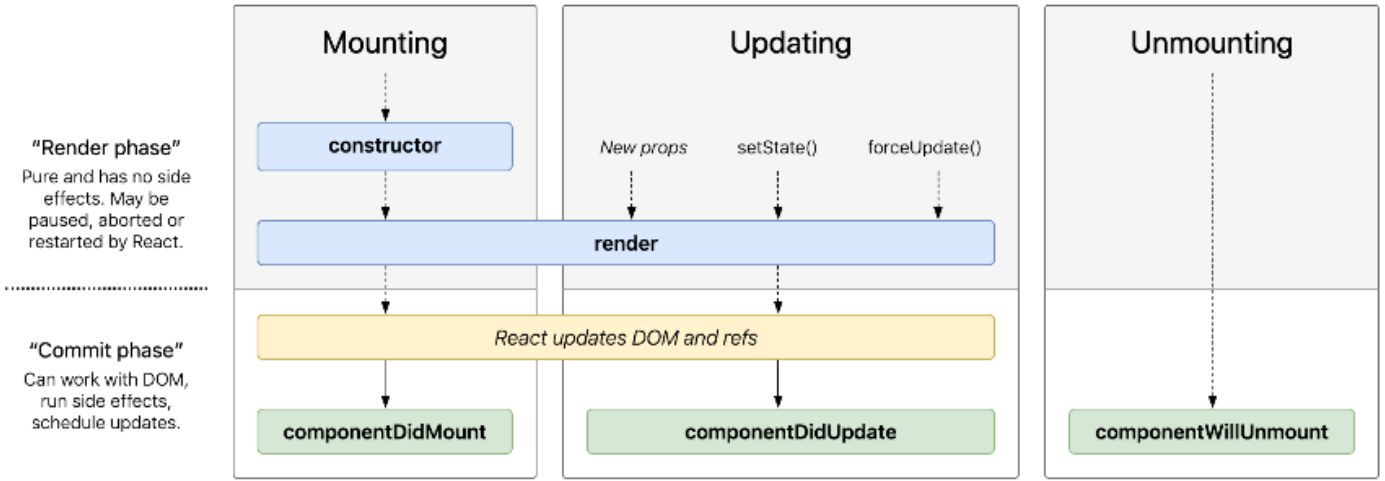




Life Cycle

라이프 사이클

☞ 리액트 컴포넌트를 감독 및 관리
Mounting, Updating, Unmounting 단계가 있음



Mounting

☞ 아래 내장 함수를 사용하여 DOM에 엘리먼트를 삽입

- Constructor()
- getDerivedStateFromProps()
- **render()**
- componentDidMount()

Header.js

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );  } }
```

Life Cycle

Constructor

- 컴포넌트의 초기화로써 초기값과 초기상태 정의
- 생성자 메소드는 props 를 인수로 사용하여 호출
- super(props) 를 호출하여 부모 컴포넌트 (React.Component) 생성자 메서드 시작으로 상속 진행

Header.js

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );   }  }
```

Life Cycle **getDerivedStateFromProps()**

- 렌더링 직전에 불려오는 메소드
- 초기 props에 기반을 둔 state 객체를 지정
- state를 인수로 받아서 상태가 변경된 객체를 반환

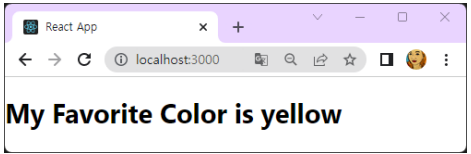
```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
```

Header.js

```
render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }}
export default Header;
```

App.js

```
<Header favcol="yellow"/>;
```



가천대학교 Gachon University

Life Cycle

render( )

☞ HTML을 DOM 에 출력하는 메소드

```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}
```

Life Cycle

componentDidMount()

☞ 컴포넌트가 렌더링 된 후 호출

DOM에 배치된 컴포넌트에게 요구되는 사항을 정의

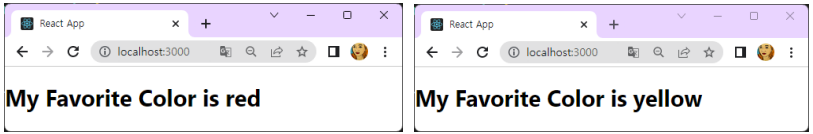
```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout( () => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}
export default Header;
```

Header.js

```
<Header />
```

App.js

➢ red에서 1초 후 yellow로 바뀜



Life Cycle

Updating

☞ 컴포넌트의 state 혹은 props 가 변경될 때마다 컴포넌트 업데이트 발생 , 아래순서로 메소드 호출

- getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

Life Cycle （1）getDerivedStateFromProps()

☞ 컴포넌트가 **업데이트 될 때 처음 호출**되는 메소드
초기 props에 기반한 state 객체를 지정하기 위해 사용

Header.js

```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };        ❷
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }                                               ❸
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>                                      ❶
    );   } } export default Header;
```

```
<Header favcol="yellow"/>
```

실행결과

My Favorite Color is yellow

➢ 좋아하는 색상을 파란색으로 변경하는 버튼이 있지만 favcol 속성의 색상으로 상태를 업데이트하는 getDerivedStateFromProps() 메서드가 호출되기 때문에 좋아하는 색상은 여전히 노란색으로 렌더링

<Header **favcol**="**yellow**"/>; **App.js**

Life Cycle （2）shouldComponentUpdate()

☞ React가 **렌더링을 계속할지 여부**를 지정하는 부울 값을 반환
기본값은 true

```
import React from "react";
class Header extends React.Component
{
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
```

```
render() {
    return (
      <div>
      <h1>My Favorite Color is
                {this.state.favoritecolor}</h1>
      <button
                type="button“ onClick={this.changeColor}
>
                Change color</button>
      </div>
    );   } }

export default Header;
```

실행결과

My Favorite Color is red

Change color

- shouldComponentUpdate() 메소드를 false로 하여서 더 이상 렌더링을 하지 않기 때문에 red
- true로 변경하면 blue로 변경됩니다.

Life Cycle

실행결과

My Favorite Color is red

Change color

My Favorite Color is blue

Change color

Header.js

```
import React from "react";

class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  shouldComponentUpdate() {
    return true;
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}
export default Header;
```

가천대학교 Gachon University

Life Cycle

(3) render()

☞ 컴포넌트가 업데이트 할 때 호출
변경사항이 발생하면 DOM 에 HTML 을 다시 렌더링

Header.js

```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};   }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );   }} export default Header;
```

(4) getSnapshotBeforeUpdate()

☞ 업데이트 이전의 props 및 state 에 액세스

getSnapshotBeforeUpdate() 메서드가 있는 경우 componentDidUpdate() 메서드도 포함 해야 하며 , 그렇지 않으면 오류

Header.js

```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    document.getElementById("div1").innerHTML =
    "Before the update, the favorite was " + prevState.favoritecolor;
  }
```

Life Cycle

>> 마운트 시에 favoritecolor는 red로 렌더링 한 후 1초 후 업데이트 위해 yellow로 상태변경이 발생

>> getSnapshotBeforUpdate()메소드로 업데이트 전의 상태를 출력

>> componentDidiUpdate() 메소드로 변경사항을 렌더링합니다.

```
componentDidUpdate() {
    document.getElementById("div2").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
        <h1>My Favorite Color is {this.state.favoritecolor}</h1>
        <div id="div1"></div>
        <div id="div2"></div>
      </div>
    );
  }
} export default Header;
```

```
<Header />
```

App.js

실행결과

My Favorite Color is yellow

Before the update, the favorite was red
The updated favorite is yellow

Life Cycle

(5) componentDidUpdate()

Header.js

```
import React from "react";
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  componentDidUpdate() {
    document.getElementById("mydiv").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
```

Life Cycle

>> 클래스 생성시에 red 였지만 마운트 시 yellow이고 업데이트 발생으로 yellow를 유지합니다.

```
render() {
    return (
      <div>
      <h1>My Favorite Color is
                {this.state.favoritecolor}</h1>
      <div id="mydiv"></div>
      </div>
    );
  }
}


export default Header;
```

```
<Header />
```

App.js

실행결과

My Favorite Color is yellow

The updated favorite is yellow

Life Cycle

(6) Unmounting( )

☞ 컴포넌트가 DOM에서 제거되거나 마운트 해제

componentWillUnmount()

Child.js

```
import React from "react";
class Child extends React.Component {
  componentWillUnmount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
                <h1>Hello World!</h1>
    );
  }
}
export default Child;
```

Life Cycle

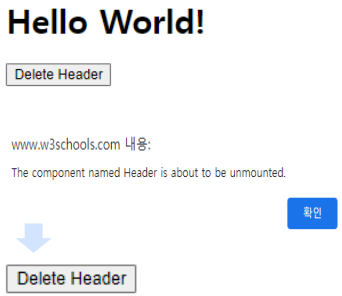
>> Delete Header 버튼이 클릭되면 state의 show가 false가 되어 Child 컴포넌트가 호출되지 않기 때문에 ComponetnWillUnmount ()메소드를 사용해서 Child 컴포넌트를 DOM에서 제거합니다.

<Container /> **App.js**

Container.js

```
import React from "react";
import Child from "./Child";
class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = {show: true};   }
  delHeader = () => {
                  this.setState({show: false});   }
  render() {
    let myheader;
    if (this.state.show) {
                  myheader = <Child />;
    };
    return (
      <div>
      {myheader}
      <button type="button" onClick={this.delHeader}>
            Delete Header </button>
      </div>  );   }}
export default Container;
```

실행결과



가천대학교 Gachon University

Event 개요

☞ 이벤트를 처리할 때 { }를 사용해서 호출

HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

React:

```
<button onClick={shoot}>Take the Shot!</button>
```

Event Passing Arguments

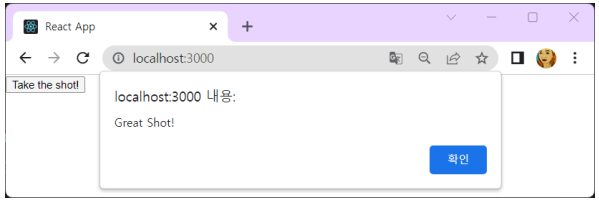
☞ 이벤트 컴포넌트로 파라메터를 전달
{()=>메소드(파라메터)} 형태로 사용

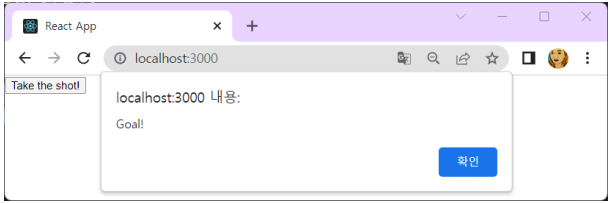
Football.js
>> No passing Arguments

```
export default function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}
```

Football.js
>> Goal! 문자열을 a 매개변수로 전달

```
export default function Football() {
  const shoot = (a) => {
    alert(a);
  }

  return (
    <button onClick={() =>
shoot("Goal!")}>Take the shot!</button>
  );
}
```



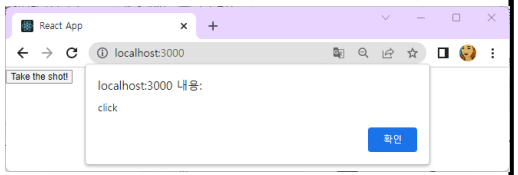


React Event Object

☞ 이벤트 오브젝트를 인수로 전달 받아서 컴포넌트에서 처리

Football.js

```
export default function Football() {
  const shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }

  return (
    <button onClick={(event) => shoot("Goal!", event)}>Take the
shot!</button>
  );
}
```



React Conditional Rendering

☞ 컴포넌트 렌더링을 결정하기 위해 if 사용

Goal.js

```
function MissedGoal() {
  return <h1>MISSED!</h1>;
}

function MadeGoal() {
  return <h1>Goal!</h1>;
}
```

```
export default function Goal(props) {
  const isGoal = props.isGoal;
  if (isGoal) {
    return <MadeGoal/>;
  }
  return <MissedGoal/>; //default
}
```

App.js

```
<Garage isGoal={false} />
```

```
<Garage />
```

App.js

```
<Garage isGoal={true} />
```

실행결과 MISSED!

실행결과 GOAL!

삼항연산자

React Conditional Rendering

☞ condition ? true : false

Goal.js

```
export default function Goal(props) {
  const isGoal = props.isGoal;
  return (
    <>
      { isGoal ? <MadeGoal/> : <MissedGoal/> }
    </>
  );
}
```

App.js

```
<Garage isGoal={false} />
```

실행결과 **MISSED!**

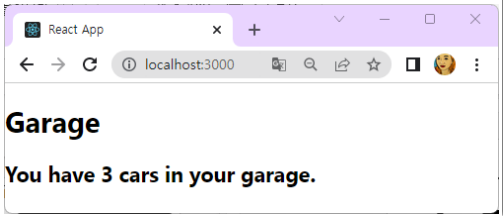
논리연산자 &&

Logical && Operator

☞ 컴포넌트의 조건적 렌더링 제공
중괄호를 사용하여 JSX에 JavaScript 표현식
참일 때만 실행하므로 삼항 연산자를 축소

Garage.js

```
export default function Garage(props) {
  const cars = props.cars;
  return (
    <>
      <h1>Garage</h1>
      {cars.length > 0 &&
        <h2>
          You have {cars.length} cars in your garage.
        </h2>
      }
    </>
  ); }
```



App.js

```
<Garage cars="BMW"   />
```

가천대학교 Gachon University

리스트 정의 및 사용

☞ [ ]를 사용하여 map() 메소드를 사용

Garage.js

```
function Car(props) {
  return <li>I am a { props.brand }</li>;
}
export default function Garage() {
  const cars = ['Ford', 'BMW', 'Audi'];
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <ul>
        {cars.map((car) => <Car brand={car} />)}
      </ul>
    </>
  ); }
```

App.js

```
<Garage  />
```

실행결과

Who lives in my garage?

- I am a Ford
- I am a BMW
- I am a Audi

List

Key 사용

☞ 키를 사용하여 React가 엘리먼트를 추적 가능하도록 함
항목이 업데이트 되거나 제거되면 전체목록 대신 해당 항목만 다시 렌더링

Car.js

```
function Car(props) {
  return <li>I am a { props.brand }</li>; }
export default function Garage() {
  const cars = [
    {id:1, brand: 'Ford'},
    {id:2, brand: 'BMW'},
    {id:3, brand: 'Audi'} ];
 return (
    <>
      <h1>Who lives in my garage?</h1>
      <ul>
        {cars.map((car) => <Car key={car.id} brand={car.brand} />)}
      </ul>
    </>   ); }
```

실행결과

Who lives in my garage?

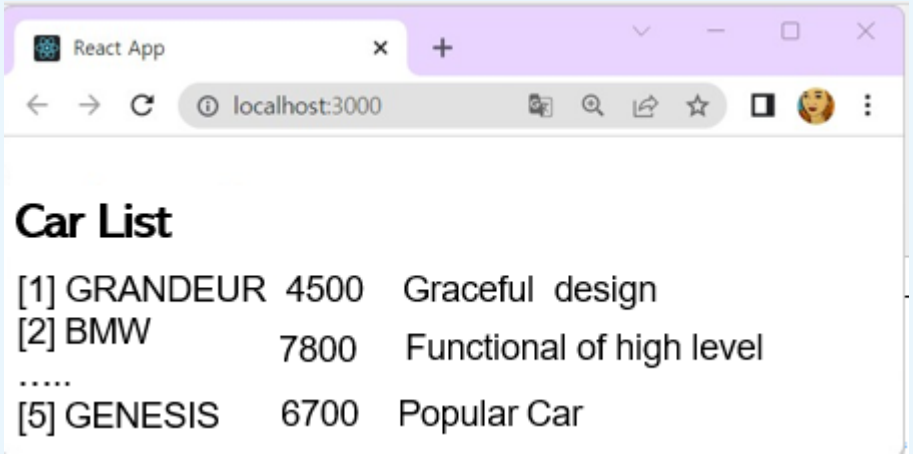
- I am a Ford
- I am a BMW
- I am a Audi

가천대학교 Gachon University

수업시간 내 진행하시기 바랍니다.

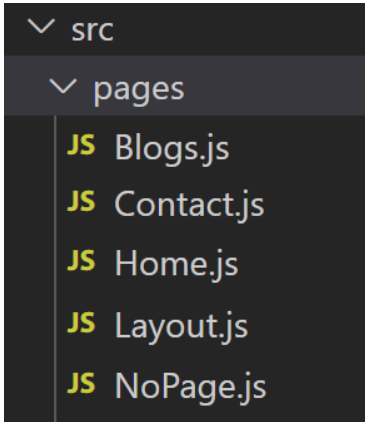
자동차 판매 페이지 제작

- 자동차 판매를 위해 정보를 소개하는 페이지
- 리액트의 CRA 패턴을 사용, List사용
- 수업시간 결과확인
- 평가 (10점)



Router

예제 폴더 구조



Navigation

☞ 리액트 라우터 돔을 설치

```
npm install react-router-dom@6
```

* V5와 V6는 차이가 있습니다. Switch->Routes로 변경

실행결과

라우터 분류

☞ 라우팅 될 주소와 이동하고자 하는 페이지, 렌더링 할 컴포넌트를 정의

- <BrouwerRouter> 혹은 <HashRouter> : 웹서비스에 사용
- <StaticRouter> : 서버렌더링(SSR) 웹 제작시 사용
- <NativeRouter> : React Native(스마트폰 어플리케이션)에 사용
- <MeomryRouter> : 테스트 시나리오 또는 다른 라우터 참조

App.js

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";
```

Router

App.js

```
export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Router

>> Link 엘리먼트를 사용하면 설정한 경로로 라우팅

>> HTML의 <a> 태그와 유사한 기능

>> Outlet은 해당 위치에 렌더링

```
//Layout.js
import { Outlet, Link } from "react-router-dom";

const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  ) };
export default Layout;
```

Router

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

export default Home;
```

```
//NoPage.js
const NoPage = () => {
  return <h1>404</h1>;
};

export default NoPage;
```

```
//Blogs.js
const Blogs = () => {
  return <h1>Blog Articles</h1>;
};

export default Blogs;
```

```
//Contact.js
const Contact = () => {
  return <h1>Contact Me</h1>;
};

export default Contact;
```

[REF.] Styling

UI/UX

CSS / SCSS(Sass)

Styled-components
https://styled-components.com/

MUI(MaterialUI)
https://mui.com/

* VSCode 확장 메뉴를 사용하여 설치하거나 npm install을 사용하여 설치

HOMEWORK

수업시간 내 진행하는 것을 원칙으로 합니다.

팀 소개 홈페이지 제작

- 팀을 나타낼 수 있는 케치프레이즈 및 로고 제작
- 홈 – 비전 – 팀원소개 – 커스텀 컨텐츠
- 팀별 메일 제출 (artchoi0g@gachon.ac.kr)
  - 결과화면 파워포인트로 작성
  - 실행파일 첨부 (node_modules 제외)
- 평가 (10점)

CONNECT.
SOLVE.
CREATE. + AI

가천대학교 Gachon University