Webpack

코드들을 의존하는 순서대로 합쳐서 하나의 또는 여러 개의 결과물을 만들어낸다.

일일이 HTML을 불러오는게 아니라 자바스크립트 파일에서 Import를 통해서 번들링 작업을 통해 불러와준다.

-원한다면 규

ES6 모던 자바스크립트

구형브라우저에서 일부가 지원이 안된다.

바벨이라는 것을 사용.하여 구형 자바스크립트도 지원하게된다.

Sass 사용 할 수 도 있다.

바벨-자바스크립트 변환도구 이다.

-https://babeljs.io/

https://codesandbox

https://bit.ly/beginreact

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

class App extends Component {
  render() {
    return (
      <div>
        <h1>안녕하세요 리액트</h1>
      </div>
```

```
    );
  }
}

export default App;
```

사용 할 때 import React를 꼭 해줘야한다.

컴포넌트를 만드는 두가지 방법

1. 클래스를 통해서 만드는 방법이 가장 일반적인 방법

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

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

JSX 기반의 형태로 함수를 리턴 해줘야한다.

2. 함수를 통해서 만드는 것.

JSX는

태그가 꼭 닫혀서 사용되어야 한다.

<input type="button"/>닫아줘야 오류가 안난다.

위에처럼 닫아주는 마지막 표시 기호를 셀프클로징태그라고한다.

두개 이상의 엘리먼트는 무조건 하나의 엘리먼트로 감싸져있어야한다.

이렇게 두개가 쓰는게 귀찮거나 불편하게 된다면 Fragment를 사용해서 간단하게 할 수 도 있다.

16.2v에서 사용가능

H

<dive>S

위의 코드처럼 작성하면 에러가 뜬다 하지만. 밑의 코드처럼 하면 에러가 안뜬다.

H

<dive>S

FRAGMENT 사용

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

class App extends Component {
render() {
return (
<Fragment>
<h1>안녕하세요 리액트</h1>
<input type="text"/>
</Fragment>
);
}
}
```

```
export default App;
```

JSX 안에 자바스크립트 값 사용하기

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

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

export default App;
```

const

let

스코프가 블록 단위이다. 둘다.

Var vs const vs let

Var는 잘 사용하지 않는다.

Const는 한번 선언 후 고정적인 값에 사용

Let은 유동적인 값을 사용할 때 선언

조건부 렌더링

JSX내부에서 조건부 렌더링을 할 때는 보통 삼항 연산자를 사용하거나, AND 연산자를 사용합니다. 반면에 if문을 사용 할 수 는 없습니다. 사용하려면 IIFE(즉시실행함수 표현)을 사용해야합니다.

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

class App extends Component {
render() {
const name='veloopoart';
return (
<div>{
1+1===3
?'맞다'
:'틀리다!'
}
</div>
);
}
}

export default App;
```

조건부 if를 엔드를 사용해서 나타낸 경우 조건부 렌더링

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

class App extends Component {
render() {
const name ='veloport!';
return (
<div>{
name==='veloport!'&& <div>벨로퍼티</div>
}
</div>
);
}
}

export default App;
```

일치 하지 않으면 화면에 표시 되지 않는다.

조건이 여러 개인 경우에는 JSX 밖에 부분에서 하는 것이 일반적이다.

JSX에서 내부에서 여러가지 조건을 적용하려면 IIFE를 함수 선언하고 바로 호출해서 사용 할 수 있다.

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

class App extends Component {
render() {
const value =1;
return (
<div>{
function(){
if(value===1)return <div>1</div>
if (value ===2) return <div>2</div>
if (value ===3) return <div>3</div>
return <div>없다.</div>
}
}
</div>
);
}
}

export default App;
```

함수를 호출하기 전의 코드 함수 호출후의 코드는 아래와 같다.

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

class App extends Component {
render() {
const value =1;
return (
<div>{
(function(){
if(value===1)return <div>1</div>
if (value ===2) return <div>2</div>
if (value ===3) return <div>3</div>
```

```
return <div>없다.</div>
})() 호출부분
}
</div>
);
}
}
```

리액트에서 CSS를 삽입 할 때는 객체 상태로 입력해준다.

Const style=등등

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

class App extends Component {
render() {
const style={
backgroundColor:'black',
padding:'16px',
color:'white',
fontSize:'36px'
};
return (
<div style={style}>
<h1>안녕하세요 리액트</h1>
</div>
);
}
}

export default App;
```

위의 형식처럼 스타일을 적용해서 사용 할 수 있다.

JSX에서 클래스를 사용 하는 방법.

1. import를 해서 적용시키기
2. div className="App(파일명)"를 통해서 적용이가능하다.

```
import React, { Component } from 'react';
import './App.css';
class App extends Component {
render() {
return (
<div className="App">
<h1>안녕하세요 리액트</h1>
</div>
);
}
}

export default App;
```

-------App.css파일을 만들어서 위의 소스에 적용

```
.App{

background:black;
color: aqua;
font-size: 36px;
padding: 1rem;
font-weight:600;

}
```

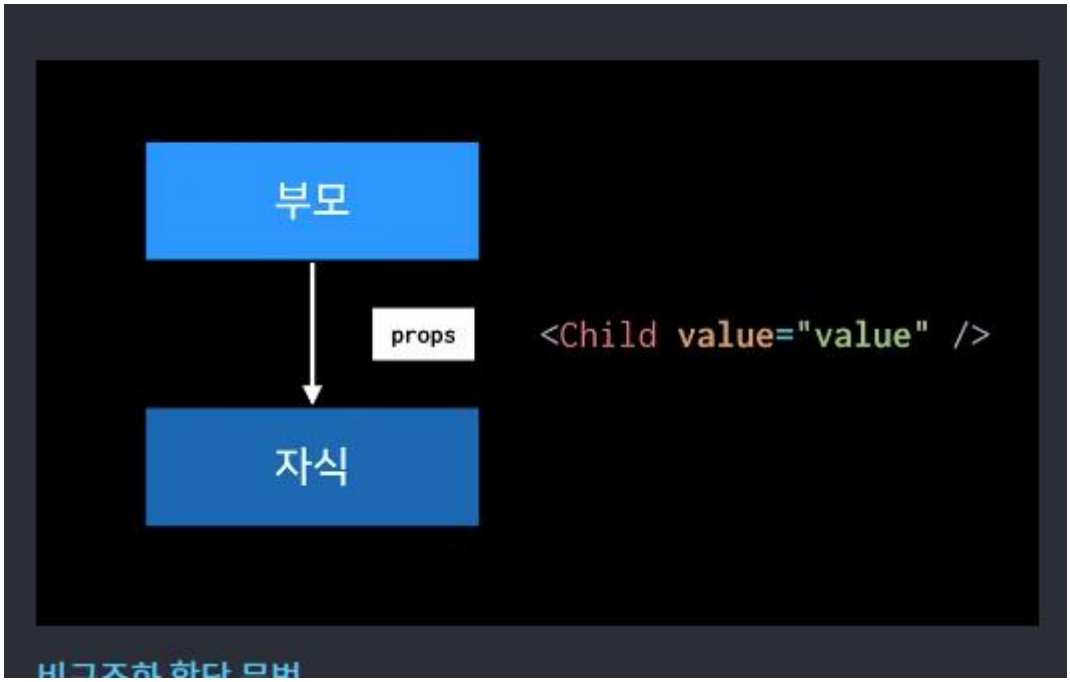
--리액트 주석처리방법

```
import React, { Component } from 'react';
import './App.css';
class App extends Component {
render() {
return(
<div>
{/*주석처리 방법이다.*/}
<h1
//내가여기에 주석을 달거야
>리액트</h1>
```

```
</div>
);
}
}

export default App;
```

Props와 State

리액트에서 데이터를 다룰 때 사용하는 것이다.



부모 컴포넌트가 자식 컴포넌트한테 데이터를 전달할 때 props를 사용한다.

사용예시 세련된코드 최신형

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

class MyName extends Component {
static defaultProps = {
name: '기본이름'
};
render() {
return (
<div>
```

```
안녕하세요!제이름은<b>{this.props.name}</b>에요.
</div>
);
}
}

export default MyName;

------------파일을 두개 생성해서 상속 받을 수 있도록 하는 예제

import React, { Component } from 'react';
import MyName from './MyName';
class App extends Component {
render() {
return<MyName name="리액트"/>;
}
}

export default App;
```

별다른 기능없이 값을 받아와서 보여주는 경우에는 보통 함수형 자바스립트를 사용한다.

```
import React from 'react';

const MyName = ({ name }) => {
  return (
    <div>
      안녕하세요! 제 이름은 {name} 입니다.
    </div>
  );
};

export default MyName;
```

함수형 컴포넌트로 작성 프롭스만 단순히 받아서 보여주는 경우

상단에 REACT를 유지해야한다 왜냐하면 컴포너트 내에서 JSX를 운영하기 위해 필요하다.

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

const MyName=({name})=>{
return <div>안녕하세요 ! 제이름은 {name} 입니다.</div>;
```

```
};

MyName.defaultProps={
name: 'velopert'
};

export default MyName;
```

함수형 컴포넌트는 마운트 속도가 빠르다.

Props값은 부모가 자식한테 주는 값이다.

State는 처음에는 자기가 갖고있다가 변화가 필요하면 컴포넌트의 내장함수인 setState()를 통해 사용한다.

카운터에서 +,-가 있다여기서 해당 버튼을 클릭시에 값이 변화됨을 실시해보자.

State를 선언할 때에는 다른 문자열이나 , 다른 것들이면 안되고 객체여야 선언이 가능하다.

App.js

```
import React, { Component } from 'react';
import Counter from './Counter';
class App extends Component {
render() {
return <Counter/>;
}
}

export default App;
```

Counter.js

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

class Counter extends Component{

render(){
return(

<div>
<h1>카운터</h1>
<div>값: 0</div>
<button>+</button>
<button>-</button>
</div>
)

}
}

export default Counter;
```

값에 변화를 주기위해서는 커스텀 메소드를 만들어줘야한다.

```
handleIncrease=()=>{}

handleDecrese=()=>{}
```

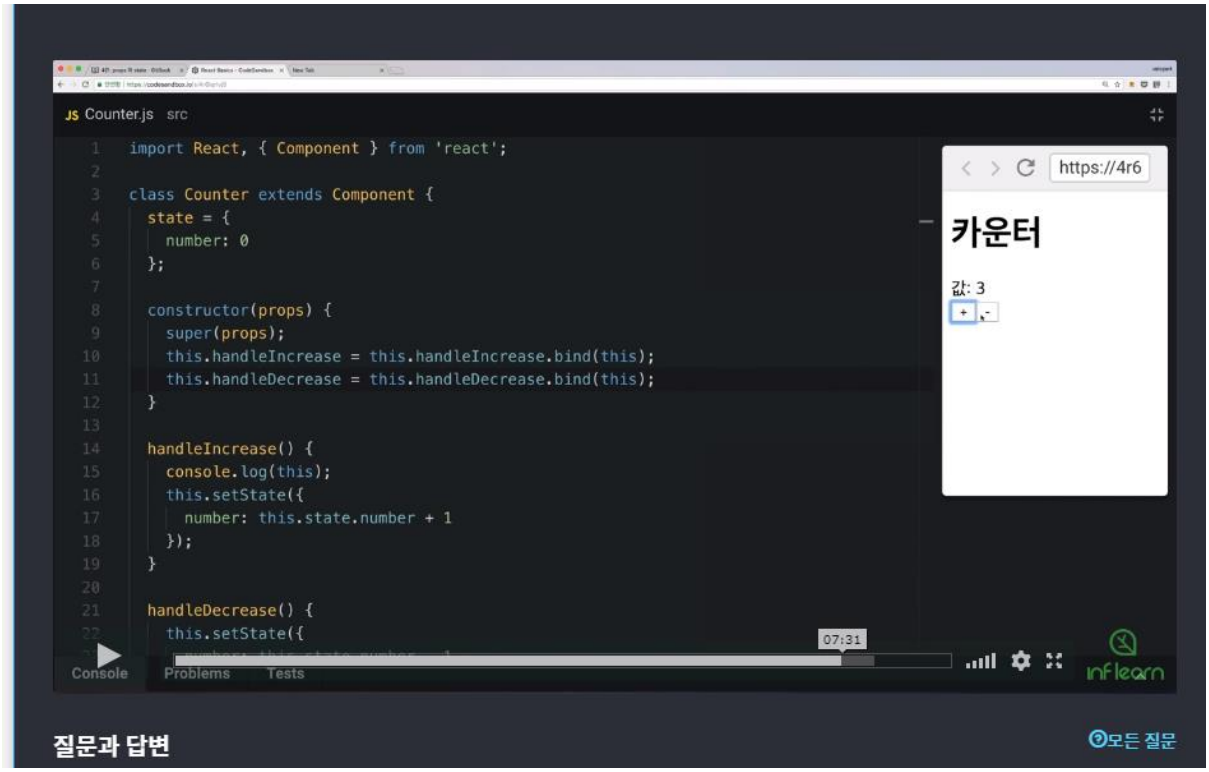
이런형식

업데이트를 할 때에는 항상 this.setState를 사용해서 한다.

증가/감소 버튼 만들어서 적용 핸들러 다루는방법

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

class Counter extends Component{
```

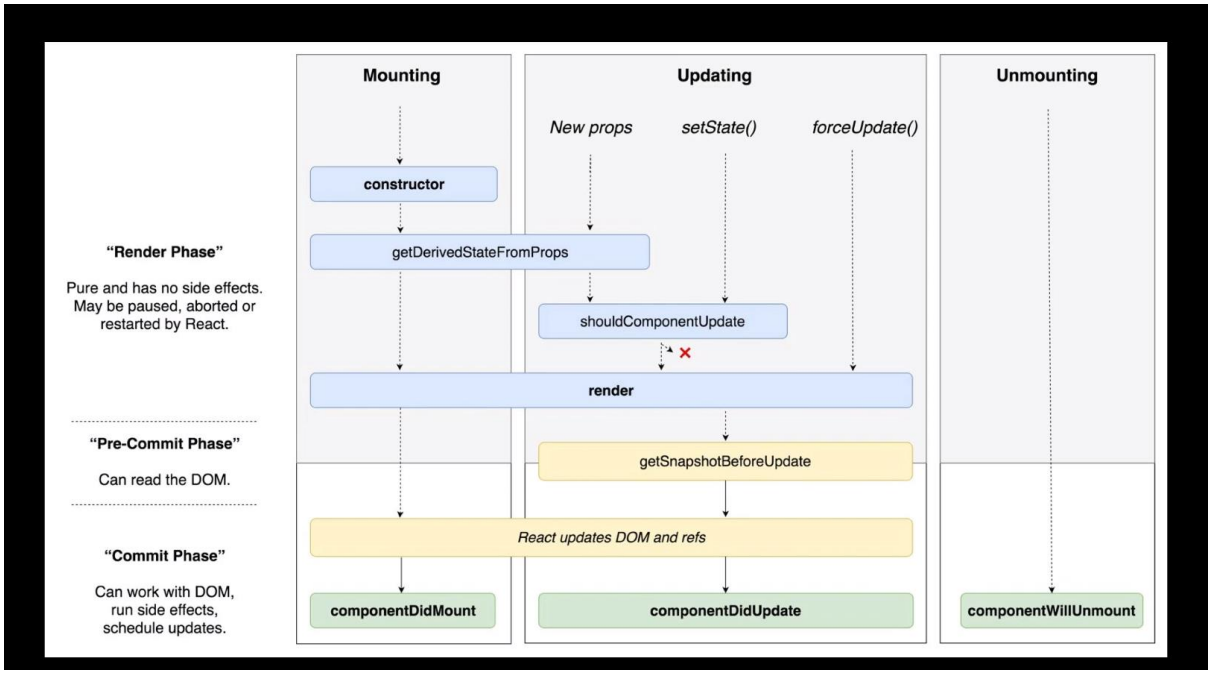
```
state={

number:1
}

handleIncrease=()=>{
this.setState(
{
number:this.state.number+1
}
)
}

handleDecrese=()=>{
this.setState(
{
number: this.state.number-1
})

}

render(){
return(

<div>
<h1>카운터</h1>
<div>값: {this.state.number}</div>
<button onClick={this.handleIncrease}>+</button>
<button onClick={this.handleDecrese}>-</button>
</div>
)

}
}

export default Counter;
```



이런 식으로도 사용이 가능하다.

<LifeCycle API>생명주기

컴포넌트가 우리 브라우저 상에서 나타날 때 업데이트 될 떄 사라 질 때 사용한다.



마운팅->컴포넌트가 우리 브라우저상에 나타나는 것을 말함

가장 먼저 실행되는 단계

초기설정, 미리 선작업을 여기서 처리한다.(컨스트럭터)

getDerivedStateFromProps->프롭스로 받은 값을 스테이트로 그래도 동기화 시키고 싶을 때 사용.

Render 내부에서 사용할 것들을 정의

업데이팅-> 상태가 변경될 때

언마운팅-> 데이터가 사라질떄

componentDidMount를 사용하면 API요청과 어떤 차트를 어디에 그려주세요 라든지 그런 부분을 형성하게끔 도와준다. 이벤트를 리스닝 API를 요청한다던지 등등

shouldComponentUpdate 컴포넌트의 성능을 최적화 할 때 사용된다.

getSnapshotBeforeUpdate -> 렌더링을 한 다음에 브라우저에 반영되기 전에 호출되는 함수

componentDidMount(){

//외부 라이브러리연동:D3,masonry,etc

//컴포넌트에서 필요한 데이터요청: Ajax,GraphQl,etc

//DOM에 관련된 작업: 스크롤설정 크기 읽어오기 등을 할 때 사용한다.

특정돔에 이벤트를 넣을 때도 사용한다.

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

class App extends Component{
constructor(props){
super(props);
console.log('constructor');
}

componentDidMount(){
console.log('componentDidMount');
console.log(this.myDiv.getBoundingClientRect().height);
}
render(){
return(
<div ref={ref=>this.myDiv=ref}>
안녕하세요.
</div>

);
}
}
export default App;
```

}

static getDerivedStateFromProps(nextProps,prevState){

nextProps는 다음에 받아 올 값을 나타내고 preState 업데이트 되기전에 상태

App.js

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

counter:1,
}
constructor(props){
super(props);
console.log('constructor');
}

componentDidMount(){
console.log('componentDidMount');
console.log(this.myDiv.getBoundingClientRect().height);
}

handleClick = () => {
this.setState({
counter:this.state.counter + 1
});
}
render(){
return(
<div ref={ref=>this.myDiv=ref}>
안녕하세요.
<MyComponent value={this.state.counter}/>
<button onClick={this.handleClick}>Click Me</button>
</div>

);
}
}
export default App;
```

MyComponent.js

```
import React, {Component} from 'react';
class MyComponent extends Component{
state={
value:0
};

static getDerivedStateFromProps(nextProps,prevState){
if(prevState.value !==nextProps.value){
return{
value:nextProps.value
}
}
return null;
}
render(){
return(
<div>
<p>props:{this.props.value}</p>
<p>state:{this.state.value}</p>
</div>

)
}
}
export default MyComponent;
```

shoulComponentUpdate

//return false 하면 업데이트를 안함

//return this.props.checked !==nextProps.checked

return true;

따로 설정하지 않으면 return true가 설정된다.

getSnapshotBeforeUpdate()

컴포넌트가 업데이트가 되서 브라우저 돔에 반영되기 직전에 바로 호출되는 함수

1render()

2getSnapshotBeforUpdate()

실제 DOM에 변화발생

componentDidUpdate를 통해 이 Api를 통해서, Dom변화가 일어나기 직전의 DOM상태를 가져오고, 여기서 리탄하는 값은 componentDidUpdate에서 3번째 파라미터로 받아올 수 있게 된다.

Dom 업데이트가 일어나기 직전의 시점입니다.

새 데이터가 상단ㅇ ㅔ추가되어도 스크롤바를 유지해보겠습니다.

scrollHeight는 전 후를 비교해서 스크롤 위치를 설정하기 위함이고,

scrollTop은 이기능이 크롬에 이미 구현되어 있는데,

componentDidCatch

componentDidCatch(error,info){

this.setState({

error: true});

}

18.07.05전체소스

```
import React, { Component } from 'react';
import MyComponent from './MyComponent';
class App extends Component {
state = {
counter: 1,
error: false
};

componentDidCatch(error, info) {
this.setState({
error: true,
});
}
constructor(props) {
super(props);
console.log('constructor');
}

componentDidMount() {
console.log('componentDidMount');
console.log(this.myDiv.getBoundingClientRect().height);
}

handleClick = () => {
this.setState({
counter: this.state.counter + 1
});
};
render() {
if (this.state.error) {
return <div>에러</div>;
}
return (
<div ref={ref => (this.myDiv = ref)}>
안녕하세요.
{this.state.counter < 10 && <MyComponent value={this.state.counter} />}
<button onClick={this.handleClick}>Click Me</button>
</div>
);
}
}
export default App;
```

18.07.05 전체소스

```
import React, {Component} from 'react';
class MyComponent extends Component{
state={
value:0
};

static getDerivedStateFromProps(nextProps,prevState){
if(prevState.value !==nextProps.value){
return{
value:nextProps.value
}
}
return null;
}

shouldComponentUpdate(nextProps,nextState){
if(nextProps.value===10) return false;
return true;
}

componentDidUpdate(prevProps,prevState){
if(this.props.value !==prevProps.value){
console.log('value 값이 바뀌었다!',this.props.value);
}
}

componenetWillUnmount(){
console.log('GoodBye');
}
render(){
return(
<div>
<p>props:{this.props.value}</p>
<p>state:{this.state.value}</p>
</div>

)
}
```

```
}
export default MyComponent;
```