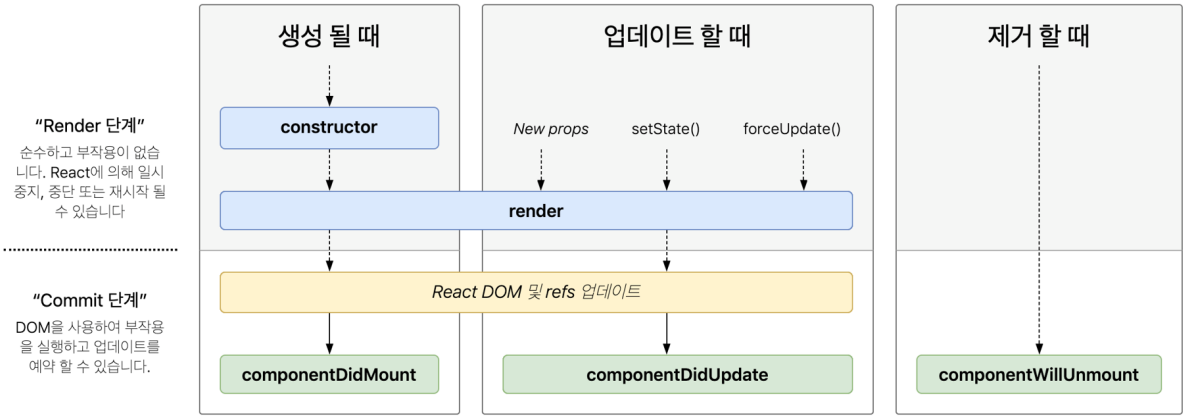
학습목표

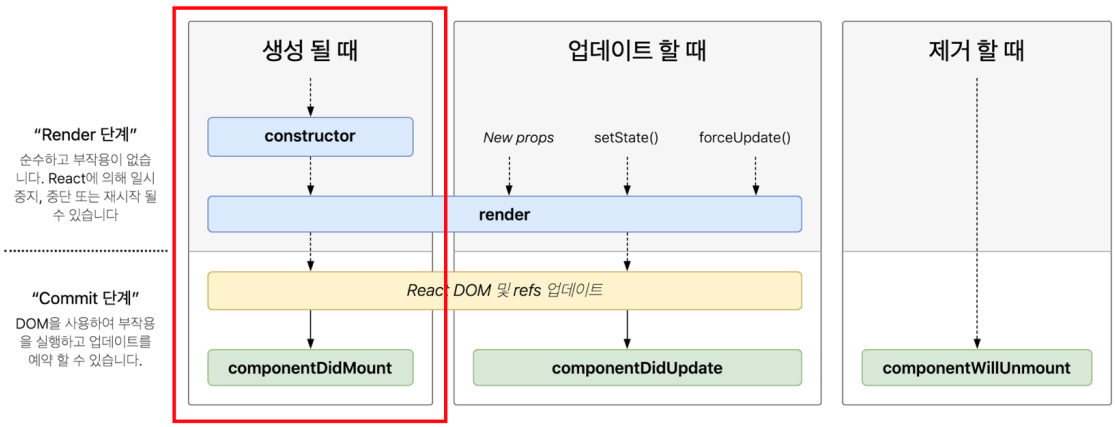
- 리액트 라이프사이클 (class vs function)
- 리액트 훅 (useState, useEffect, useRef)
- CRA (Create React App) 시작하기

클래스형 컴포넌트 vs 함수형 컴포넌트

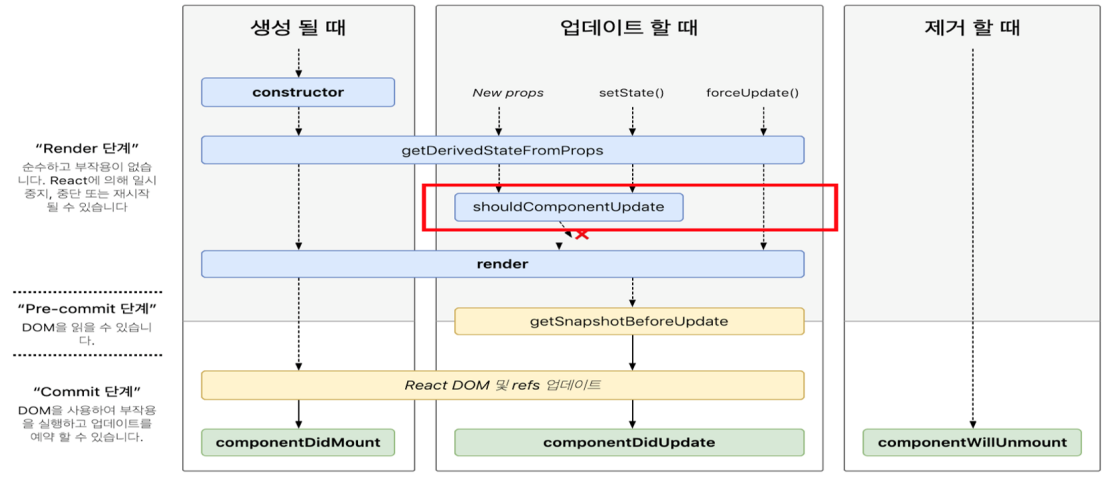
- 메타에서 함수형 컴포넌트 권장(메모리 사용 적음 등 효율적...)
- ~ will: 어떤 작업을 수행하기 전에 실행되는 메서드와 관련
- ~ did: 어떤 작업을 수행한 이후에 실행되는 메서드와 관련
- mount: 컴포넌트 내에서 DOM이 생성되고 웹 브라우저 상에 나타는 메서드와 관련
- unmount: 컴포넌트 내에서 DOM을 제거되고 웹 브라우저 상에 사라지는 메서드와 관련
- update: 컴포넌트 내에서 변화가 발생하였을때 수행하는 메서드와 관련
- 클래스형 라이프사이클

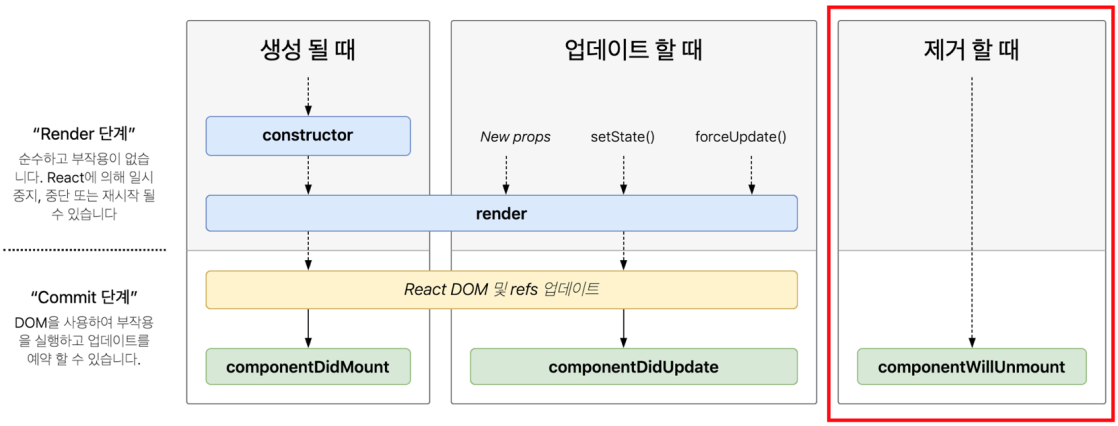


- 1단계: 생성 (Mounting)



  - constructor(초기화) -> componentWillMount -> render -> componentDidMount
  - constructor: state 초기화
  - componentWillMount: 렌더링전 세팅

        - render: 화면이 그려짐, JSX가 브라우저 이해할 수 있는 코드로 babel에 의해 변환후 DOM 처리됨(화면에 그려짐)
        - componentDidMount
            - ajax요청, DOM접근가능(이벤트 등록, 엘리먼트크기 알아내기)
            -
```
const container = document.querySelector('.sky-wrap');
const containerField = container.getBoundingClientRect();
console.log(containerField.height)
```
    - 2단계: 업데이트
        - 

        - shouldComponentUpdate
            - 컴포넌트를 다시 렌더링해야 할지를 정의
            - 변화된 부분만 변경
            - shouldComponentUpdate(nextProps,nextState){} return (JSON.stringify(nextProps)!=JSON.stringify(this.props)); }
        - render(rerender)
        - componentDidUpdate
    - 3단계: 제거 (unmounting)
        - 

        - componentWillUnmount
        - 제거후 다시 그리기는 생성단계부터
    - 17버전이후 deprecated 메소드

        - 사용하려면 UNSAFE_ 접두사 붙여서 사용(미권장)
        - componentWillMount() -> UNSAFE_componentWillMount()
        - componentWillUpdate() -> UNSAFE_componentWillUpdate()
        - componentWillReceiveProps() -> UNSAFE_componentWillReceiveProps()
    - 코드 예시
        -

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count:
this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

- 함수형 라이프사이클: Hooks로 처리
    - 코드 예시

```jsx
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

    - componentDidMount, componentDidUpdate
        - useEffect Hook으로 구현 가능

React Hooks

- 함수형 컴포넌트 기능 보완을 위해 등장
- 버전 16.8부터 React 요소로 새로 추가됨
- Hook을 이용하여 기존 Class 바탕의 코드를 작성할 필요 없이 상태 값과 여러 React의 기능을 사용할 수 있음
- 기본
    - useState: 학습
    - useEffect: 학습
    - useContext
- 추가
    - useReducer
    - useCallback
    - useMemo
    - useRef: 학습
    - useImperativeHandle
    - useLayoutEffect
    - useDebugValue

CRA (Create React App) 사용: 2025년2월 공식지원중단

- nodeJS 설치
  - nodejs.org
  - 확인: 터미널창, 명령프롬프트, PowerShell
  - node -v, npm -v
- CRA: https://ko.reactjs.org/docs/create-a-new-react-app.html
  - 개발폴더 생성
  - npx create-react-app 프로젝트명
    - npm init react-app 프로젝트명
  - cd 프로젝트명
  - npm start
- package.json 확인
- Readme.md: 내용 지워도 됨
- .gitignore
- public 폴더
  - 정적 컨텐츠
  - index.html
    - `<div id="root"></div>` 외에 삭제가능
- src 폴더
  - 동적 컨텐츠
  - index.js
    - `<App />` 외에 삭제 가능
  - App.js의 템플릿코드

```
import './App.css';
function App() {
  return(
    <div className="App">
      <h1>안녕하세요.</h1>
    </div>

  );
}
export default App;
```

- App.test.js, index.css, reportWebVitals.js, setupTests.js, logo.svg, logo192.png, logo512.png, robo ts.txt 삭제

Vite 사용 설치

- 실행: npm create vite@latest
  - npm create vite@latest 프로젝트명 -- --template react
  - 처음 실행: 다음 패키지 설치 - y입력으로 설치
    - create-vite@버전

        - create-vite@6.4.1
- 프로젝트명 입력: react_vite
- 프레임워크 선택(방향키+엔터): React
- 언어선택(Select a variant): JavaScript
- package.json파일의 폴더로 이동: cd 프로젝트명 (cd react_vite)
- 패키지 설치: npm i (node_modules 생성확인)
- 서버 실행: npm run dev
- 브라우저 확인: http://localhost:5173/

리액트 작동방식

- Component ?
    - jsx 코드를 반환하는 함수 컴포넌트를 적는 것 => 함수를 호출하는 것
- 리액트는 이 컴포넌트들을 통과할 때마다 함수를 호출하고 더이상 불러올 컴포넌트(함수) 가 없을 때까지 통과해서 DOM으로 변환하여 화면에 그려줌 => 이런 과정을 처음 렌더링 되었을 때만 한번 실행하고 반복하지 않음. => 렌더링 이후 변경되는 부분을 업데이트 시켜주려면 방법은? state state변경시 다시 화면 업데이트
- 클래스형 컴포넌트의 state예시

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- 함수형 컴포넌트의 state예시

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

코드실습

- 1. components폴더 생성
- 2. App.js 수정

```
import './App.css';
import MyComp from './components/MyComp'; // auto import 플러그인 사용고려
function App() {
  return(
    <MyComp />
  );
```

```
}
export default App;
```

- 3. components/MyComp.js 작성
  - jsx에서는 하나의 루트엘리먼트로 시작해야 함
  - 해결1: `<div></div>`
  - 해결2: `<></>`, `<React.Fragment></React.Fragment>`
    - `<React.Fragment>` 사용시 `import React from 'react';` 필요

```
const MyComp = ()=>{
  return(
    <>
      <h1>안녕</h1>
      <h1>일본IT</h1>
    </>
  );
};
export default MyComp;
```

- 4. 브라우저 실행 확인

React Hooks2

1. useState

- `const [상태명, set상태명] = useState(초기값);`
  - 첫 번째: 상태명 - 변수
  - 두 번째: 상태를 업데이트하는 함수
    - 관례: set+상태변수명(첫글자 대문자):camelCase
- state는 독립적으로 관리됨
- 자식 컴포넌트 여러개에 props로 부모의 state 값을 넘겨주면 확인해볼 수 있음
- set상태명()
  - 비동기적으로 작동 ⇒ setState로 변경한 state값이 즉시 적용되지 않는다는 말

코드실습

- App.js 수정1

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState} from 'react';

function App() {
  const [state, setState] = useState('초깃값');
  return(
    <h1>{state}</h1>
```

```
  );
}
export default App;
```

- App.js 수정2

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState} from 'react';

function App() {
  const [state, setState] = useState(0);
  const clickListener = (e)=>{
    setState(state+1);  // 비동기 호출됨
    console.log(state); // 콘솔 출력과 화면 출력값이 다름: setState() 비동기호출
  };
  return(
    <>
    <h1>{state}</h1>
    // <button onClick="(e)=>{setState(state+1);}">증가</button>
    <button onClick={clickListener}>증가</button>
    </>
  );
}
export default App;
```

- App.js 수정3

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState} from 'react';

function App() {
  const [state, setState] = useState(0);
  let score = 0;
  const clickListener = (e)=>{
    setState(state+1);
    console.log('state',state);
    score+=1;
    console.log('score',score);  // 스테이트변경으로 리렌더링, 변수 다시 초기화
  };
  return(
    <>
    <h1>{state}</h1>
    <h1>{score}</h1>  // 변수는 변화해도 화면 변경안됨
    // <button onClick="(e)=>{setState(state+1);}">증가</button>
    <button onClick={clickListener}>증가</button>
    </>
  );
```

```
}
export default App;
```

- App.js 수정4

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState} from 'react';

function App() {
  const [state, setState] = useState(0);
  let score = 0;
  const clickListener = (e)=>{
    // useState 사용시 기존 상태를 업데이트 해야하는 경우라면?
    // setState(state+1); // 코드가 복잡해지면 작동안할 수 있음
    // 대신 하기 방법으로 코딩하는 것이 바람직
    setState((preveState) => prevState + 1 );
    console.log('state',state);
    score+=1;
    console.log('score',score);  // 스테이트변경으로 리렌더링, 변수 다시 초기화
  };
  return(
    <>
    <h1>{state}</h1>
    <h1>{score}</h1>  // 변수는 변화해도 화면 변경안됨
    // <button onClick="(e)=>{setState(state+1);}">증가</button>
    <button onClick={clickListener}>증가</button>
    </>
  );
}
export default App;
```

- STATE 설정방법 1_ state를 개별적으로 관리하기

```
const [title, setTitle] = useState('');
const [content, setContent] = useState('');
const [date, setDate] = useState('');

setTitle(event.target.value);
setContent(event.target.value);
setDate(event.target.value);
```

- STATE 설정방법 2_ state를 통합적으로 관리하기

```
const [userInput, setUserInput] = useState({
    title: '',  content: '',  date: '',
});
```

```
// 이방법은 피하기
setUserInput({
  ...userInput,  title: event.target.value,
});

// 대신 이방법으로
setUserInput((preveState) => {
    return { ...prevState, title: event.target.value}
});
```

- InputForm.js 예시

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

const InputForm = () => {
  // const [title, setTitle] = useState('');
  // const [content, setContent] = useState('');
  // const [date, setDate] = useState('');
  const [userInput, setUserInput] = useState({
    title: '',
    content: '',
    date: '',
  });

  const titleChangeHandler = (event) => {
    // setTitle(event.target.value);
    // setUserInput({
    //   ...userInput,
    //   title: event.target.value,
    // });
    setUserInput((prevState) => {
      return { ...prevState, title: event.target.value };
    });
  };

  const contentChangeHandler = (event) => {
    // setContent(event.target.value);
    setUserInput({
      ...userInput,
      content: event.target.value,
    });
  };

  const dateChangeHandler = (event) => {
    // setDate(event.target.value);
    setUserInput({
      ...userInput,
      date: event.target.value,
    });
  };
```

```
  return (
    <form>
      <div>
        <div>
          <label>제목</label>
          <input type='text' onChange={titleChangeHandler} />
        </div>
        <div>
          <label>내용</label>
          <input type='text' onChange={contentChangeHandler} />
        </div>
        <div>
          <label>날짜</label>
          <input
            type='date'
            min='2019-01-01'
            max='2022-12-31'
            onChange={dateChangeHandler}
          />
        </div>
      </div>
      <div>
        <button type='submit'>추가</button>
      </div>
    </form>
  );
};

export default InputForm;
```

2. useEffect

- componentDidMount, componentDidUpdate와 비슷
  - 처음 렌더링 될 때 + 리렌더링 될 때마다 실행

```
useEffect(() => {
    console.log('useEffect 함수실행')
});
```

=> onChange 이벤트에 setState걸어서 확인해보기

- 처음 렌더링 될 때만 실행

```
useEffect(() => {
    console.log('useEffect 함수실행')
}, []);
```

- 처음 렌더링 될 때 + 대괄호 안 값이 변할 때 실행

```
useEffect(() => {
    console.log('useEffect 함수실행')
}, [state]);
```

- clean up 하려면 return 안에 코드 작성하기

```
useEffect(() => {
    console.log('useEffect 함수실행')
    return() => {
        // clean up 코드 작성
        console.log('컴포넌트가 화면에서 사라질 때 실행 (자식컴포넌트가 부모컴포넌트에
서 사라질 때 )');
  }
}, []);

ex)
useEffect(()=> {
    const time = setInterval(() => {
        console.log('반복반복 setInterval')
    }, 1000)

    return() => {
        clearInterval(time)
    }
}, []);
```

- 클래스형 컴포넌트

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
```

```
          Click me
        </button>
      </div>
    );
  }
}
```

- 함수형 컴포넌트

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

코드실습

- App.js 수정5

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState, useEffect} from 'react';

function App() {
  const [state, setState] = useState(0);
  //const [count, setCount] = useState(10);

  // useEffect(() => {
    //   console.log('useEffect 함수실행')
  // });

  // useEffect(() => {
    //   console.log('useEffect 함수실행')
  // },[]);

  // useEffect(() => {
```

```
    //   console.log('useEffect 함수실행')
  // },
  // //[state]
  // //[count]
  // [count,state]
  // );

  const clickListener = (e)=>{
    //setState((preState)=>preState+1);
    setCount((preCount)=>preCount+1);
  };
  return(
    <>
    //<h1>{state}</h1>
    <h1>state: {state}</h1>
    <h1>count: {count}</h1>
    <button onClick={clickListener}>증가</button>
    </>
  );
}
export default App;
```

- App.js 수정6

```
import './App.css';
import MyComp from './components/MyComp';
import React, {useState} from 'react';

function App() {
  const [count, setCount] = useState(0);

  const clickListener = (e)=>{
    setCount((preCount)=>preCount+1);
  };
  return(
    <>
    {count === 2 && <MyComp />}
    <h1>count: {count}</h1>
    <button onClick={clickListener}>증가</button>
    </>
  );
}
export default App;
```

- MyComp.js 수정

```
import React, {useEffect} from 'react';
function MyComp() {
  useEffect(() => {
```

```
    console.log('my comp useEffect 함수실행');
    return() => {
      // clean up 코드 작성
      console.log('컴포넌트가 화면에서 사라질 때 실행 (자식컴포넌트가 부모컴포넌트에서 사라질 때 )');
    }
  }, []);
  return(
    <>
      <h1>MyComp</h1>
    </>
  );
}
export default MyComp;
```

- MyComp.js 수정2

```
import React, {useEffect} from 'react';
function MyComp() {
  useEffect(()=> {
    const time = setInterval(() => {
        console.log('반복반복 setInterval')
    }, 1000)

    return() => {
        clearInterval(time)
    }
  }, []);
  return(
    <>
      <h1>MyComp</h1>
    </>
  );
}
export default MyComp;
```

3. useRef

- 변하는 값은 유지하되 변할 때마다 렌더링을 발생시키고 싶지 않을 때 사용
- 변수 vs state vs useRef
  - 변수: 리렌더링 되면 초기화됨, 변수 값이 바뀌어도 리렌더링 되지않음
  - state: 리렌더링 되어도 값을 유지, state 값이 바뀔 때마다 리렌더링 됨
  - useRef: 리렌더링 되어도 값을 유지, ref 값이 바뀐다고 리렌더링이 되지는 않음, 스테이트의 변경으로 리렌더링되면 화면에 반영됨

```
const myRef = useRef(0);

const myFunc = () => {
```

```
    myRef.current = myRef.current + 1;
}
```

예제

- 버튼 3개에 onClick 이벤트를 걸어 세개의 차이를 확인하기
- 변하는 값을 볼 수 있게

태그 안에 표시하기

- App.js 수정7

```
import './App.css';
//import MyComp from './components/MyComp';
import React, {useState, useRef} from 'react';

function App() {
  const [count, setCount] = useState(0);
  let score;
  const myRef = useRef(0);
  console.log(myRef);
  console.log(myRef.current);

  //변수
  const clickListener1 = (e)=>{
    score+=1;
    console.log('변수',score);
  };
  //state
  const clickListener2 = (e)=>{
    setCount((preCount)=>preCount+1);
    console.log('state',count);
  };
  //useRef
  const clickListener3 = (e)=>{
    myRef.current = myRef.current+1;
    console.log('ref',myRef.current);
  };
  return(
    <>
      <h1>변수: {score}</h1>
      <h1>count: {count}</h1>
      <h1>ref: {myRef.current}</h1>
      <button onClick={clickListener1}>변수</button>
      <button onClick={clickListener2}>state</button>
      <button onClick={clickListener3}>useRef</button>
    </>
  );
}
export default App;
```

useRef 로 DOM 요소 접근하기

- `const myRef = useRef();`로 설정하고 태그 속성에 `ref={myRef}` 추가
- App.js 수정8

```
import './App.css';
//import MyComp from './components/MyComp';
import React, {useState,useEffect, useRef} from 'react';

function App() {
/*  const [count, setCount] = useState(0);
  let score;
  const myRef = useRef(0);
  console.log(myRef);
  console.log(myRef.current);

  //변수
  const clickListener1 = (e)=>{
    score+=1;
    console.log('변수',score);
  };
  //state
  const clickListener2 = (e)=>{
    setCount((preCount)=>preCount+1);
    console.log('state',count);
  };
  //useRef
  const clickListener3 = (e)=>{
    myRef.current = myRef.current+1;
    console.log('ref',myRef.current);
  }; */
  const inputRef = useRef();
  console.log(inputRef);
  console.log(inputRef.current);
  // useEffect(()=>{
  //   inputRef.current.focus();
  // });
  useEffect(()=>{
    inputRef.current.focus();
  },[]);
  return(
    <>
    <input type="text" ref={inputRef}/>
{/*       <h1>변수: {score}</h1>
      <h1>count: {count}</h1>
      <h1>ref: {myRef.current}</h1>
      <button onClick={clickListener1}>변수</button>
      <button onClick={clickListener2}>state</button>
      <button onClick={clickListener3}>useRef</button> */}
    </>
```

```
    );
}
export default App;
```

참고

- https://ko.legacy.reactjs.org/docs/hooks-reference.html
- https://react.dev/reference/react/hooks