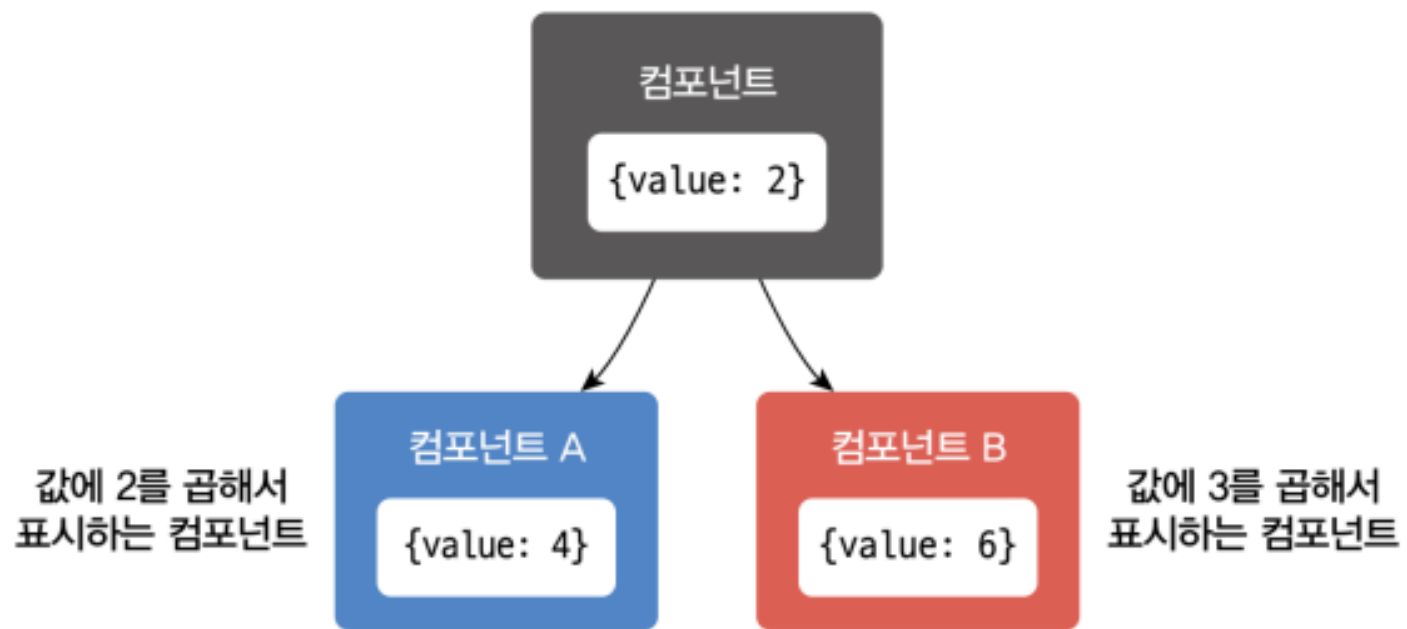


Shared State

State 끌어올려서 컴포넌트 간 State 공유하기

Shared state

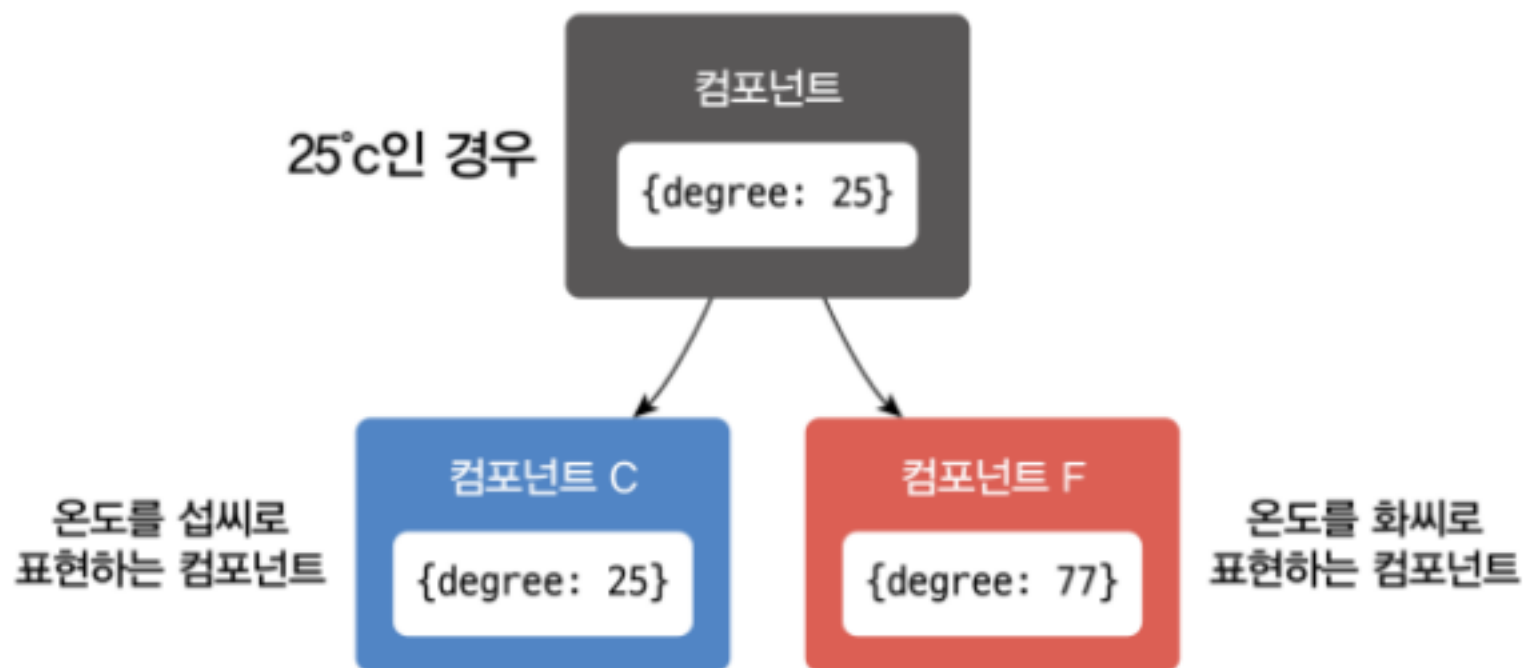
- State 끌어올려서 공통된 상태로 만들어 공유한다.
- 컴포넌트의 상태 공유 예 1)



자식 컴포넌트는 부모컴포넌트의 state에 있는 값에 각각 2와 3을 곱해서 표시하면 된다.

Shared state

- State 끌어올려서 공통된 상태로 만들어 공유한다.
- 컴포넌트의 상태 공유 예 2)



온도 입력(단위: 화씨)

온도 입력(단위: 섭씨)

자식 컴포넌트는 부모컴포넌트의 degree에 저장된 값을 가져와 섭씨 또는 화씨로 변환하여 표시하면 된다.

Shared state

- **상태 공유란**
- **여러 컴포넌트가 공통으로 필요로 하는 데이터를** 독립적으로 각각 관리하는 대신, 부모 컴포넌트나 상위의 특정 컴포넌트에서 정의하고 그 상태를 필요한 하위 컴포넌트들에 **props로 전달**하는 방식이다.
- **상태 공유의 장점: 상태 동기화로 데이터에 일관성 부여된다.**
 - 상태가 변경되면 해당 상태를 사용하는 모든 컴포넌트가 자동으로 최신 상태 업데이트 되므로 여러 곳에서 같은 데이터를 사용해야 할 때 **데이터 불일치 문제를 방지**할 수 있다.

하위 컴포넌트에서 state 공유하는 방법

- 사용자로부터 온도 입력받아 각각 섭씨온도, 화씨온도로 표현하는 모듈 구현
 - 컴포넌트 구성
 1. 해당 온도에서 물이 끓는지, 끓지 않는지 출력하는 컴포넌트
 - BoilingVerdict
 - => 섭씨온도값을 props로 받아서 물이 끓는 여부를 문자열로 출력하는 기능
 2. BoilingVerdict를 사용하는 부모 컴포넌트 Calculator
 - temperature 상태 정보 가짐

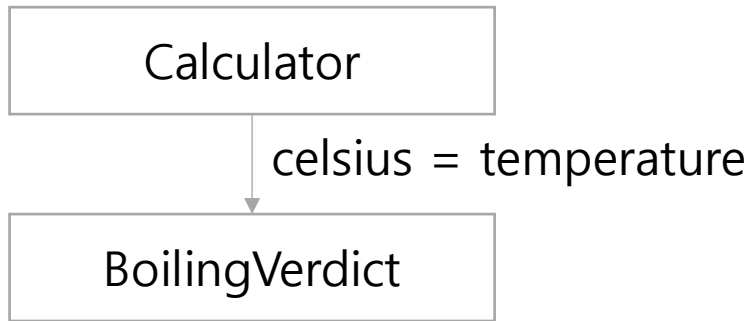
하위 컴포넌트에서 state 공유하는 방법

1. 해당 온도에서 물이 끓는지, 끓지 않는지 출력하는 BoilingVerdict 컴포넌트 구현

```
01 function BoilingVerdict(props) {  
02     if (props.celsius >= 100) {  
03         return <p>물이 끓습니다.</p>;  
04     }  
05     return <p>물이 끓지 않습니다.</p>;  
06 }
```

하위 컴포넌트에서 state 공유하는 방법

2. BoilingVerdict를 사용하는 부모 컴포넌트 Calculator



Calculator

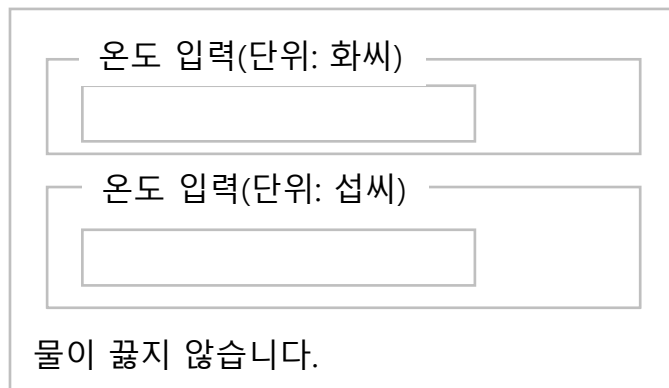
물이 끓지 않습니다. BoilingVerdict

if **temperature** >= 100
"물이 끓습니다."
else
"물이 끓지 않습니다."

```
01 function Calculator(props) {
02   const [temperature, setTemperature] = useState('');
03
04   const handleChange = (event) => {
05     setTemperature(event.target.value);
06   }
07
08   return (
09     <fieldset>
10       <legend>섭씨 온도를 입력하세요:</legend>
11       <input
12         value={temperature}
13         onChange={handleChange} />
14       <BoilingVerdict
15         celsius={parseFloat(temperature)} />
16     </fieldset>
17   )
18 }
```

하위 컴포넌트에서 state 공유하는 방법

- 다음과 같이 사용자가 섭씨온도, 화씨온도를 각각 입력할 있도록 Calculator 컴포넌트의 입력 부분을 컴포넌트로 만들어 보자.
- **사용자의 입력 값은 각 입력창에 동시에 반영되어야 한다.** 입력컴포넌트에서 변경되는 온도값을 어떻게 공유해야 할까?

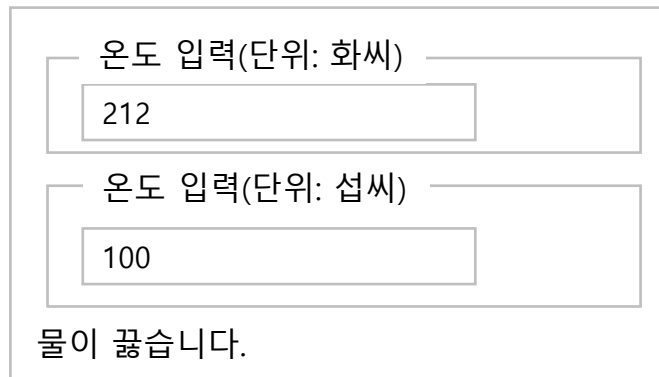


온도 입력(단위: 화씨)

온도 입력(단위: 섭씨)

물이 끓지 않습니다.

초기화면

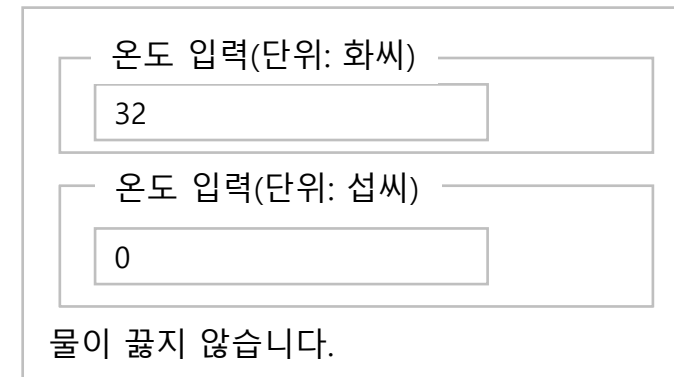


온도 입력(단위: 화씨)

온도 입력(단위: 섭씨)

물이 끓습니다.

화씨온도 입력시 섭씨 입력
창에도 값이 변함



온도 입력(단위: 화씨)

온도 입력(단위: 섭씨)

물이 끓지 않습니다.

섭씨온도 입력시 화씨 입력
창에도 값이 변함

하위 컴포넌트에서 state 공유하는 방법

- Calculator 컴포넌트에 사용자가 입력하는 2가지 종류의 온도값을 동기화시키는 함수 정의

➤ 온도 변환 함수

```
function toCelsius(fahrenheit) {  
    return (fahrenheit - 32) * 5 / 9;  
}  
  
function toFahrenheit(celsius) {  
    return (celsius * 9 / 5) + 32;  
}
```

➤ 온도 변환 함수를 호출하는 함수

```
function tryConvert(temperature, convert) {  
    const input = parseFloat(temperature);  
    if (Number.isNaN(input)) {  
        return '';  
    }  
    const output = convert(input);  
    const rounded = Math.round(output * 1000) / 1000;  
    return rounded.toString();  
}
```

```
tryConvert('abc', toCelsius)      // empty string을 리턴  
tryConvert('10.22', toFahrenheit) // '50.396'을 리턴
```

하위 컴포넌트에서 state 공유하는 방법

사용자 입력부분을 TemperatureInput
컴포넌트로 만들자.

```
<fieldset>
  <legend>섭씨 온도를 입력하세요:</legend>
  <input
    value={temperature}
    onChange={handleChange} />
  <BoilingVerdict
    celsius={parseFloat(temperature)} />
</fieldset>
```

```
01  const scaleNames = {
02    c: '섭씨',
03    f: '화씨'
04  };
05
06  function TemperatureInput(props) {
07    const [temperature, setTemperature] = useState('');
08
09    const handleChange = (event) => {
10      setTemperature(event.target.value);
11    }
12
13    return (
14      <fieldset>
15        <legend>온도를 입력해 주세요(단위: {scaleNames[props.scale]}):</legend>
16        <input value={temperature} onChange={handleChange} />
17      </fieldset>
18    )
19  }
```

하위 컴포넌트에서 state 공유하는 방법

- Calculator 컴포넌트에서 TemperatureInput 컴포넌트 호출 코드

```
01 function Calculator(props) {  
02   return (  
03     <div>  
04       <TemperatureInput scale="c" />  
05       <TemperatureInput scale="f" />  
06     </div>  
07   );  
08 }
```

```
function TemperatureInput(props) {  
  const [temperature, setTemperature] = useState('');  
  
  return (  
    <fieldset>  
      <legend>온도를 입력해 주세요(단위:{scaleNames[props.scale]}):</legend>  
      <input value={temperature} onChange={handleChange} />  
    </fieldset>  
  )  
}
```

-> Temperature를 TemperatureInput의 상태로 정의하면 부모 컴포넌트 Calculator에서 접근할 수 없으므로 BoilingVerdict 컴포넌트로 전달할 수 없다.

```
<BoilingVerdict  
  celsius={parseFloat(temperature)} />
```

하위 컴포넌트에서 state 공유하는 방법

- Shared State 적용-state 끌어올리기

- 1. 하위컴포넌트는 props로 값을 가져온다.

온도값을 TemperatureInput 컴포넌트의 state에서 가져오는 것이 아닌 부모컴포넌트로부터 전달받은 props를 통해서 가져온다.

```
01  return (  
02      ...  
03      // 변경 전: <input value={temperature} onChange={handleChange} />  
04      <input value={props.temperature} onChange={handleChange} />  
05      ...  
06  )
```

하위 컴포넌트에서 state 공유하는 방법

- Shared State 적용-state 끌어올리기

2. 자식컴포넌트에서 변경되는 값을 부모컴포넌트로 전달하기 위해 부모컴포넌트에 정의된 상태 변경 함수 호출한다. 이를 위해서는 자식컴포넌트 호출시 상태변경 함수를 props로 전달한다.
->사용자 입력이 바뀔 때 handleChange() 함수에서는 props를 통해 전달된 부모컴포넌트의 이벤트핸들러를 호출하여 변경된 온도값이 상위 컴포넌트로 전달되도록 한다.

```
01  const handleChange = (event) => {  
02      // 변경 전: setTemperature(event.target.value);  
03      props.onTemperatureChange(event.target.value);  
04  }
```

```
<TemperatureInput  
  temperature={celsius}  
  scale="c"  
  onTemperatureChange={handleCelsuisChange}  
>
```

하위 컴포넌트에서 state 공유하는 방법

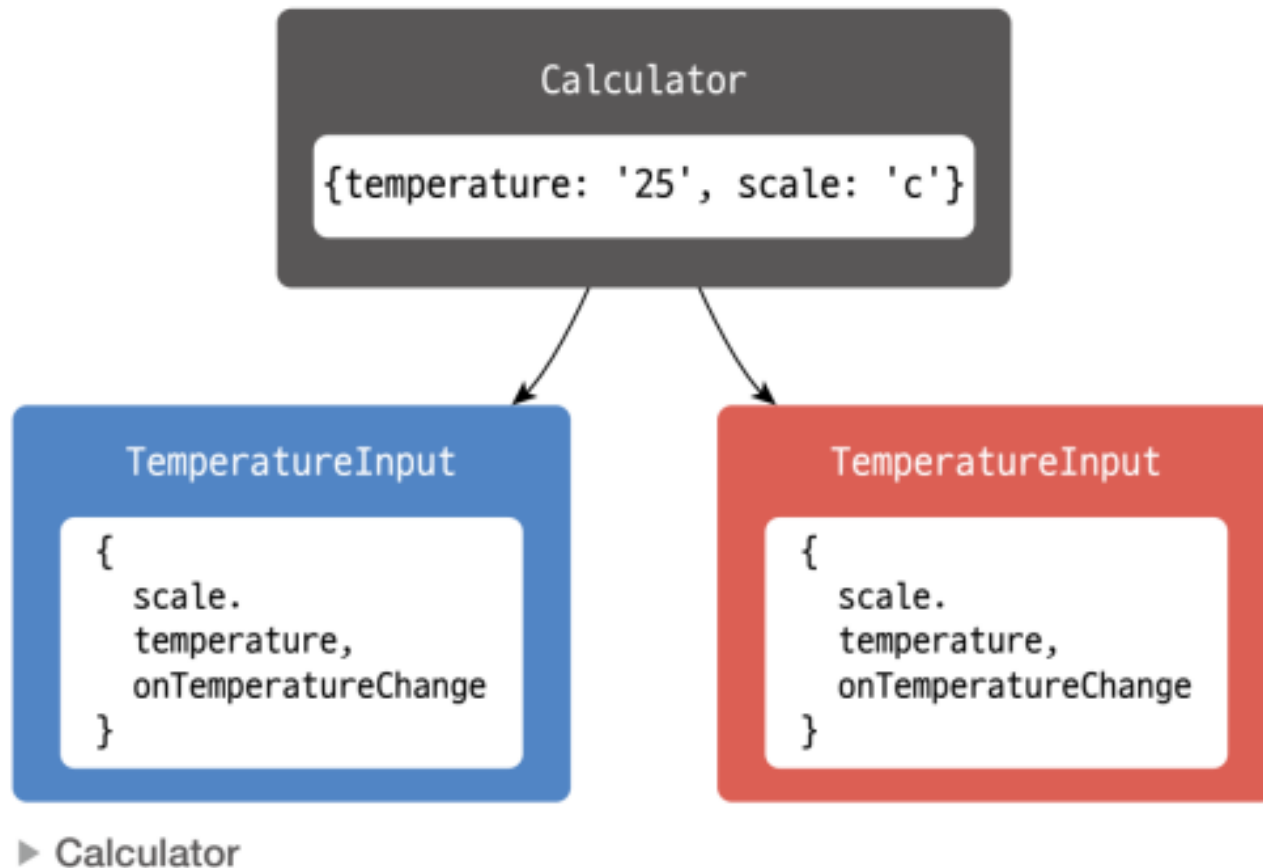
- Shared State 적용-state 끌어올리기
3. 부모컴포넌트에 상태 정보 및 상태 변경 함수 정의, 하위컴포넌트에 전달
TemperatureInput 컴포넌트에 맞춰서 부모 컴포넌트 Calculator 변경

```
01 function Calculator(props) {  
02   const [temperature, setTemperature] = useState('');  
03   const [scale, setScale] = useState('c');  
04  
05   const handleCelsiusChange = (temperature) => {  
06     setTemperature(temperature);  
07     setScale('c');  
08   }  
09  
10   const handleFahrenheitChange = (temperature) => {  
11     setTemperature(temperature);  
12     setScale('f');  
13   }
```

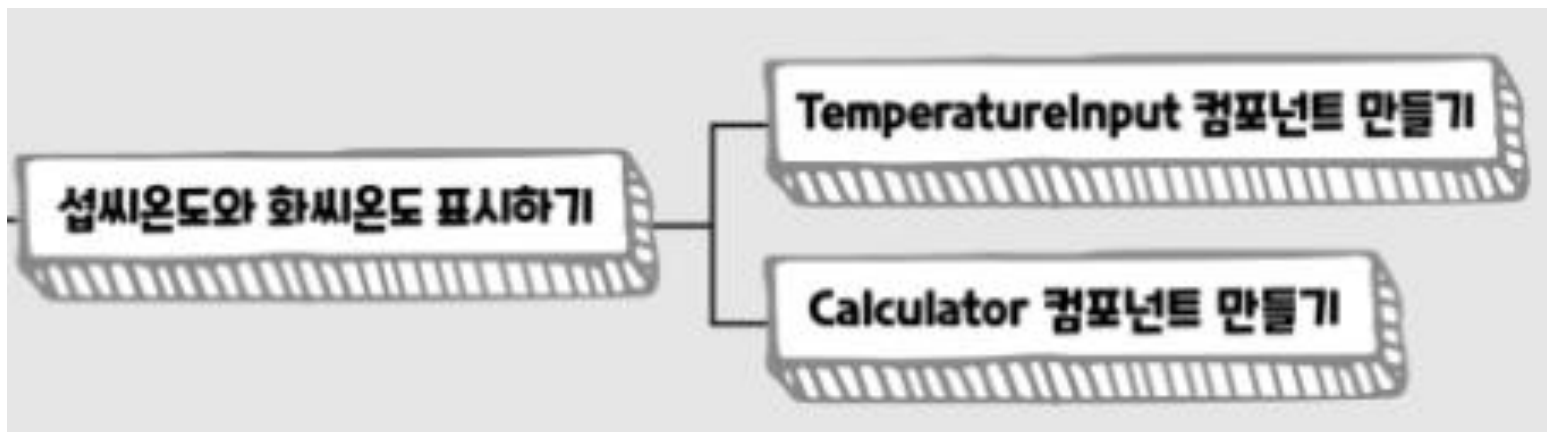
```
15   const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) :  
    temperature;  
16   const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit)  
    : temperature;  
17  
18   return (  
19     <div>  
20       <TemperatureInput  
21         scale="c"  
22         temperature={celsius}  
23         onTemperatureChange={handleCelsiusChange} />  
24       <TemperatureInput  
25         scale="f"  
26         temperature={fahrenheit}  
27         onTemperatureChange={handleFahrenheitChange} />  
28       <BoilingVerdict  
29         celsius={parseFloat(celsius)} />  
30     </div>  
31   );  
32 }
```

하위 컴포넌트에서 state 공유하는 방법

- Calculator 컴포넌트의 state로 temperature와 scale 선언하여 온도값과 단위를 저장.
- 변환 함수를 통해 섭씨와 화씨온도를 구해진 온도값과 단위 및 사용자 입력값 변경시 업데이트하는 이벤트함수를 TemperatureInput 컴포넌트에 props로 전달.



Shared State 구현 실습



Temperature.jsx

Calculators.jsx

온도를 입력해 주세요(단위:화씨)

온도를 입력해 주세요(단위:섭씨)

물이 끓지 않습니다.

온도를 입력해 주세요(단위:화씨)

온도를 입력해 주세요(단위:섭씨)

물이 끓습니다.

온도를 입력해 주세요(단위:화씨)

온도를 입력해 주세요(단위:섭씨)

물이 끓지 않습니다.

Shared State 구현 실습

Temperature.jsx

```
const scaleNames = {
  f: "화씨",
  c: "섭씨",
};

function TemperatureInput(props) {
  const handleChange = (event) => {
    //사용자입력이 발생하면 부모컴포넌트의 temperature 변경 함수 호출
  };

  return (
    <fieldset>
      <legend>온도를 입력해 주세요(단위:{scaleNames[ //전달받은scale ]})</legend>
      //입력 양식 지정 value=전달받은 온도, onChange 이벤트 처리 함수 호출
    </fieldset>
  );
}
```

Shared State 구현 실습

Calculator.jsx

```
function toCelsius(fahrenheit) { //화씨를 섭씨로 변경하는 함수
  return ((fahrenheit - 32) * 5) / 9;
}
function toFahrenheit(celsius) { //섭씨를 화씨로 변경하는 함수
  return (celsius * 9) / 5 + 32;
}
function tryConvert(temperature, convert) { //온도와 변경함수 전달 받음
  const input = parseFloat(temperature);
  if (Number.isNaN(input)) { //숫자가 아닐 경우 빈문자열의 리턴
    return "";
  }
  const output = convert(input); //변환함수 호출
  const rounded = Math.round(output * 1000) / 1000; //소수점 이하 3자리까지만 표시
  return rounded.toString();
}
```

Shared State 구현 실습

Calculator.jsx

```
function Calculator(props) {  
  const [temperature] = useState(""); // 온도 상태 정보  
  const [scale] = useState("c"); // 단위 상태 정보  
  
  const celsius =  
    // scale이 'f' 이면 tryConvert 함수 호출, 아닐 경우 temperature 값 선택  
    // scale이 'c' 이면 tryConvert 함수 호출, 아닐 경우 temperature 값 선택  
  const fahrenheit =  
    // scale이 'c' 이면 tryConvert 함수 호출, 아닐 경우 temperature 값 선택  
  const handleFahrenheitChange = (temperature) => {  
    // temperature, scale 상태값 저장  
  };  
  const handleCelsiusChange = (temperature) => {  
    // temperature, scale 상태값 저장  
  };  
};
```

Shared State 구현 실습

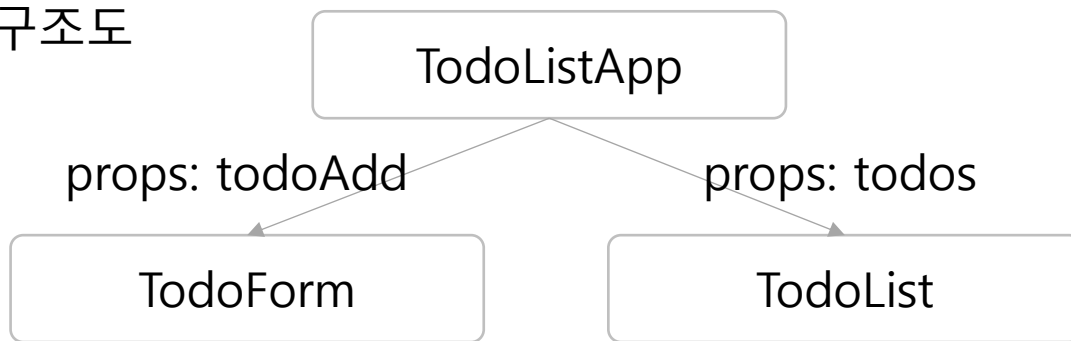
Calculator.jsx

```
return (  
  <div>  
    <TemperatureInput  
      //컴포넌트 호출시 scale, temperature, onTemperatureChange를  
      props로 전달  
    />  
    <TemperatureInput  
      //컴포넌트 호출시 scale, temperature, onTemperatureChange를  
      props로 전달  
    />  
    <BoilingVerdict celsius={parseFloat(celsius)} />  
  </div>  
}
```

응용실습

- 사용자가 할일 입력창에 내용 입력후 추가 버튼을 누르면 목록으로 추가되도록 할일 목록 페이지를 만들어 보자.

1. 컴포넌트 구조도



2. 초기 화면

TodoForm.jsx

할일 목록

추가

3. 사용자 입력->추가버튼 클릭 후

TodoList.jsx

할일 목록

추가

- 운동
- 리액트복습

응용 실습: TodoListApp.jsx


- TodoListApp 컴포넌트에서 TodoForm 컴포넌트와 TodoList 컴포넌트의 todos 상태를 공유한다.

```
function TodoListApp() {  
  const [todos, setTodos] = useState([]);  
  
  // 새로운 할 일 추가 함수 addTodo 정의  
  
  return (  
    <div>  
      <h1>할 일 목록</h1>  
  
      <div>todos 배열 상태 변경</div>  
  
      <div>  
        TodoForm 컴포넌트 호출: props addTodo 전달  
        TodoList 컴포넌트 호출: props todos 전달  
      </div>  
    </div>  
  );  
}
```

응용 실습: TodoForm.jsx

- 사용자 입력을 받고 추가 버튼을 누르면 입력값은 부모컴포넌트 TodoListApp의 addTodo 함수로 전달된다.

```
function TodoForm({ addTodo }) {  
  const [task, setTask] = useState("");  
  
  const handleSubmit = (e) => {  
    e.preventDefault();  
    if (task.trim()) {  
      부모컴포넌트의 addTodo 함수에 할일(task) 전달  
      setTask(""); // 입력 필드 초기화  
    }  
  };  
};
```



A UI mockup of the TodoForm component. It consists of a light gray rounded rectangle containing a white text input field on the left and a gray button with the text '추가' (Add) on the right.

```
return (  
  <form onSubmit={handleSubmit}>  
    <input  
      type="text"  
      placeholder="할 일 입력"  
      value={task}  
      입력창 상태 변경될 경우 task 상태 변경  
    />  
    <button type="submit">추가</button>  
  </form>  
);  
}
```

응용실습: TodoList.jsx

- TodoList 컴포넌트는 부모컴포넌트 TodoListApp로부터 전달받은 todos 배열의 항목 출력 수행

```
function TodoList({ todos }) {  
  return (  
    <ul>  
        
    </ul>  
  );  
}
```

부모컴포넌트로부터 전달받은 todos 배열 항목 출력
map() 사용

할일 목록

추가

- 운동
- 리액트복습

응용 실습

- 할일 항목 오른쪽에 삭제 버튼 추가
- 사용자가 삭제 버튼을 누르면 항목이 목록에서 삭제되도록 코드를 수정하세요.

할일 목록

추가

- 운동 삭제
- 리액트 복습 삭제

- 사용자가 "삭제" 버튼 클릭 → deleteTodo 호출
- todos 상태가 업데이트 (항목 제거 filter() 함수 적용)
`todos.filter((_, i) => i !== index)`

할 일 목록

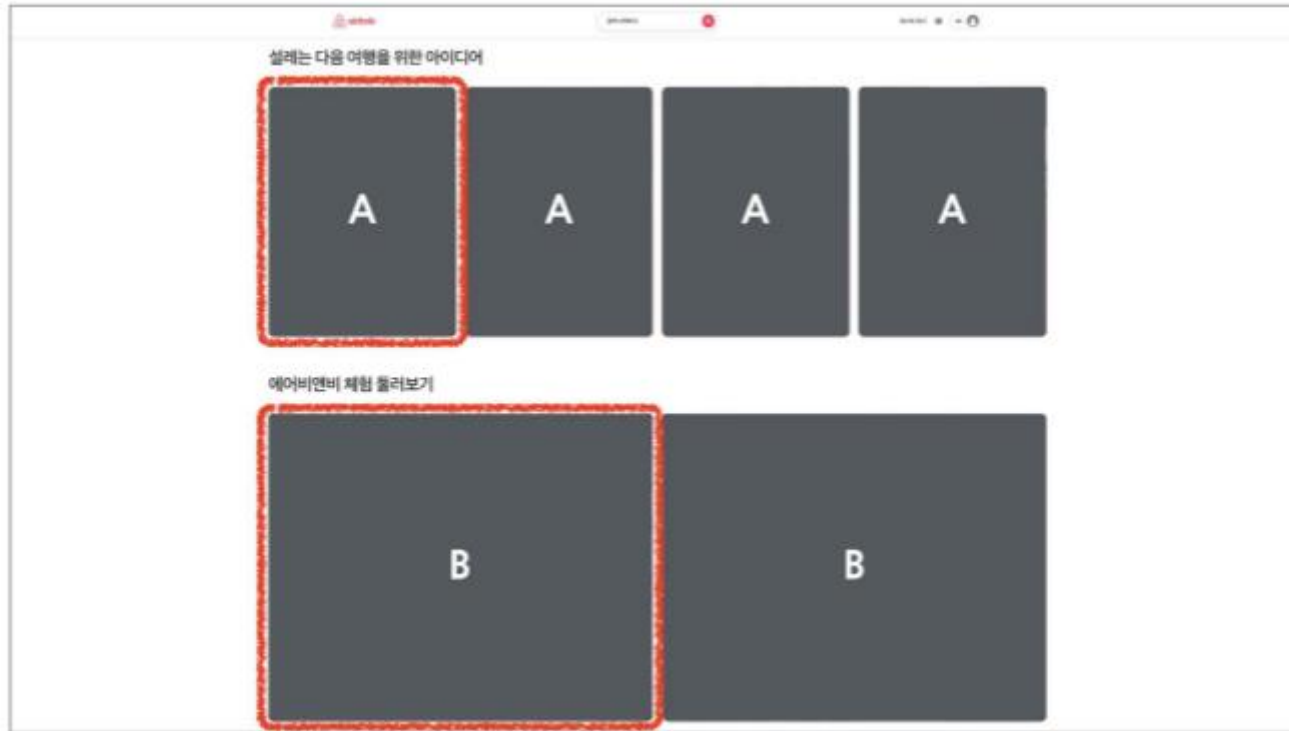
추가

- 리액트 복습 삭제

컴포넌트 합성

합성

- 여러 개의 컴포넌트를 합쳐서 새로운 컴포넌트를 만드는 것
 - A 컴포넌트와 B 컴포넌트를 합쳐서 새로운 페이지 컴포넌트를 만듦



합성방법1-Containment

- 컴포넌트에서 다른 컴포넌트 담기
 - **children** 속성 사용
 - 하위 컴포넌트를 포함하는 형태의 합성 방법
 - 사이드바나 다이얼로그 같이 어떤 내용이 담길지 모르는 박스 형태의 컴포넌트에 주로 사용

```
function FancyBorder(props) {  
  return (  
    <div  
      style={{ padding: "8px",  
        backgroundColor: props.color || "white" }}  
    >  
      {props.children}  
      <p>여기는 리액트 공부방입니다.</p>  
    </div>  
  );  
}
```

어서오세요

우리 사이트에 방문하신 것을 환영합니다.

여기는 리액트 공부방입니다.

props.children 존재할 경우

여기는 리액트 공부방입니다.

props.children 존재하지 않을 경우

합성방법1-Containment

- children 속성은 배열 형식
 - 따라서 컴포넌트는 여러 개의 하위 컴포넌트를 가질 수 있다

```
React.createElement (  
  type,  
  [props],  
  [... children]  
)
```

```
function WelcomeDialog(props) {  
  return (  
    <div>  
      <FancyBorder color="#eaea00">  
        <h1>어서오세요</h1>  
        <p>우리 사이트에 방문하신 것을 환영합니다.</p>  
      </FancyBorder>  
    </div>  
  );  
}
```

FancyBorder
컴포넌트의
children

합성방법1-Containment

- **별도의 props를 정의해서** 각각 원하는 컴포넌트를 삽입.
 - App 컴포넌트에서 SplitPane 컴포넌트 호출시 left, right라는 두개의 props 정의하여 그 안에 각각 다른 컴포넌트를 넣어준다.

```
function SplitPane(props) {  
  return (  
    <div className="SplitPane">  
      <div className="SplitPane-left">  
        {props.left}  
      </div>  
      <div className="SplitPane-right">  
        {props.right}  
      </div>  
    </div>  
  );  
}
```

```
function App(props) {  
  return (  
    <SplitPane  
      left={  
        <Contacts />  
      }  
      right={  
        <Chat />  
      }  
    />  
  );  
}
```

합성방법2-Specialization

- 범용적인 개념을 구별이 되도록 구체화하는 것
 - props를 이용
 - 예-웰컴다이얼로그는 다이얼로그의 특별한 케이스
 - Dialog 컴포넌트는 전달되는 title, message 값에 따라 경고 또는 인사말 다이얼로그가 됨

```
function Dialog(props) {  
  return (  
    <FancyBorder color="#33ffee">  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
    </FancyBorder>  
  );  
}
```

```
function WelcomeDialog(props) {  
  return (  
    <div>  
      <Dialog  
        title="어서오세요"  
        message="우리 사이트에 방문하신 것을 환영합니다."  
      />  
    </div>  
  );  
}
```

어서오세요

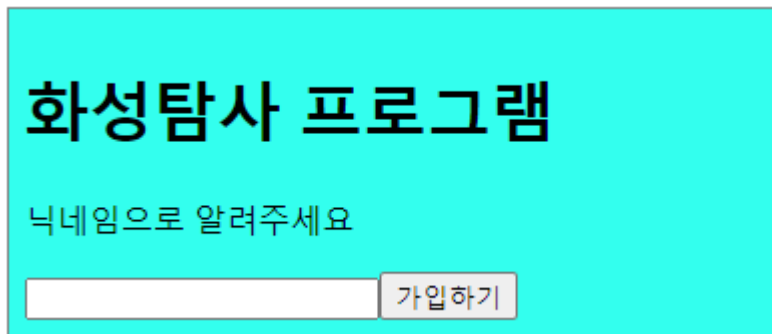
우리 사이트에 방문하신 것을 환영합니다.

여기는 리액트 공부방입니다.

Containment와 Specialization 함께 사용한 예

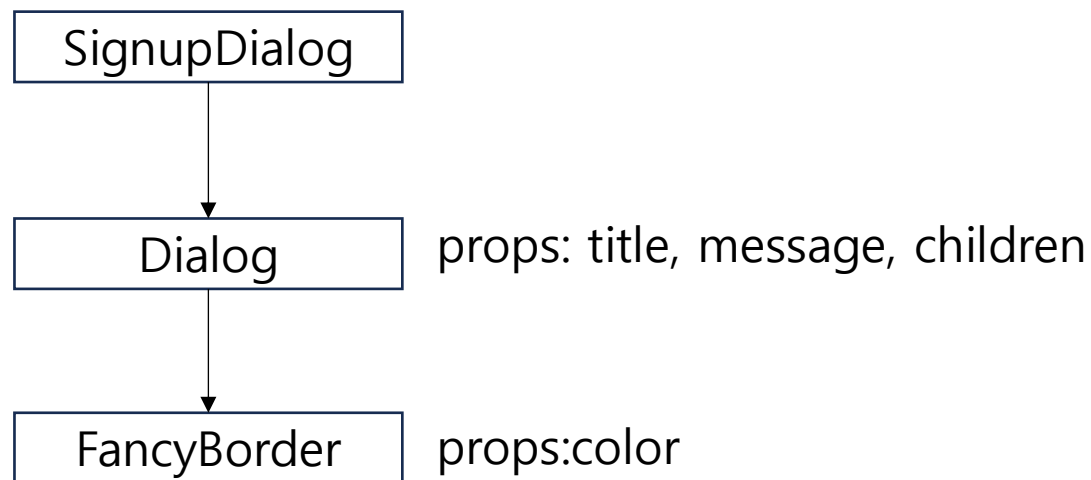
```
function Dialog(props) {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title">  
        {props.title}      Specialization 합성을 위해 title, message 속성 지정  
      </h1>  
      <p className="Dialog-message">  
        {props.message}  
      </p>  
      {props.children}    Containment 합성을 위해 props.children 속성 지정  
    </FancyBorder>  
  );  
}
```


Containment와 Specialization 함께 사용한 예



화성탐사 프로그램

닉네임으로 알려주세요



Containment와 Specialization 함께 사용한 예

```
function SignUpDialog(props) {  
  const [nickname, setNickname] = useState('');  
  
  const handleChange = (event) => {  
    setNickname(event.target.value);  
  }  
  
  const handleSignUp = () => {  
    alert(`어서 오세요, ${nickname}님!`);  
  }  
}
```

```
return (  
  <Dialog  
    props {  
      title="화성 탐사 프로그램"  
      message="닉네임을 입력해 주세요.">  
      children {  
        <input  
          value={nickname}  
          onChange={handleChange} />  
        <button onClick={handleSignUp}>  
          가입하기  
        </button>  
      }  
    </Dialog>  
  );  
}
```

실습-Card 컴포넌트 만들기

- Card 컴포넌트로 ProfileCard 만들기

Inje Lee

안녕하세요 소플입니다.

저는 리엑트를 사용해서 개발하고 있습니다.

```
function ProfileCard(props) {  
  return (  
    <Card title="Inje Lee" backgroundColor="#4ea04e">  
      <h1>안녕하세요 소플입니다.</h1>  
      <p>저는 리엑트를 사용해서 개발하고 있습니다.</p>  
    </Card>  
  );  
}
```

실습-Card 컴포넌트 만들기

- Card 컴포넌트로 ProfileCard 만들기

```
function Card(props) {  
  const { title, backgroundColor, children } = props;  
  return (  
    <div  
      style={{  
        margin: 8,  
        padding: 8,  
        borderRadius: 8,  
        boxShadow: "0px 0px 4px grey",  
        backgroundColor: backgroundColor || "white",  
      }}  
    >  
      {title && <h1>{title}</h1>}  
      {children}  
    </div>  
  );  
}
```

Card 컴포넌트 활용 실습

Card 컴포넌트를 활용하여 title, content 입력 할일 추가 버튼을 누르면 내용 추가되도록 구현해 보자.

1. 초기 화면

할일 추가하기

할일추가

3. 내용 입력 없이 버튼 클릭하면 경고창 띄우기

'제목과 내용을 입력하세요'

2. 사용자 입력->버튼 클릭 후 화면

할일 추가하기

할일추가

오늘의 할일

1시간 운동하기

학습 목표

React 공부하기

Card 컴포넌트 활용 실습

Card 컴포넌트를 활용하여 title, content 입력 할일 추가 버튼을 누르면 내용 추가되도록 구현해 보자.

할일

localhost:3000 내용:
제목과 내용을 입력해주세요.

확인

제목을

할일 추가하기

기계학습

신경망 동작 원리 이해하기

리액트

컴포넌트 합성 기법

Card 컴포넌트 활용 실습

```
function App() {
  const [cards, setCards] = useState([]);
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");

  // 할일 추가 버튼 클릭 시 새로운 카드 추가
  const handleAddCard = () => {
    if (!title || !content) {
      alert("제목과 내용을 입력해주세요.");
      return;
    }

    const newCard = {
      title: title,
      backgroundColor: `#${Math.floor(Math.random() * 16777215).toString(16)}`,
      content: content,
    };

    setCards([...cards, newCard]);
    setTitle(""); // 입력 필드 초기화
    setContent(""); // 입력 필드 초기화
  };
}
```

Card 컴포넌트 활용 실습

```
return (  
  <div style={{ padding: 16 }}>  
    <h1>할일 추가하기</h1>  
    <div style={{ marginBottom: 16 }}>  
      <input  
        type="text"  
        placeholder="제목을 입력하세요"  
        value={title}  
        onChange={(e) => setTitle(e.target.value)}  
        style={{  
          marginRight: 8,  
          padding: 8,  
          border: "1px solid #ccc",  
          borderRadius: 4,  
        }}  
      />  
    </div>  
  </div>  
)
```


Card 컴포넌트 활용 실습

```
<input
  type="text"
  placeholder="내용을 입력하세요"
  value={content}
  onChange={(e) => setContent(e.target.value)}
  style={{
    marginRight: 8,
    padding: 8,
    border: "1px solid #ccc",
    borderRadius: 4,
  }}
/>
```

Card 컴포넌트 활용 실습

```
<button
  onClick={handleAddCard}
  style={{
    padding: 8,
    backgroundColor: "#4CAF50",
    color: "white",
    border: "none",
    borderRadius: 4,
    cursor: "pointer",
  }}
>
  할일 추가
</button>
</div>
```

Card 컴포넌트 활용 실습

```
<div>
  {cards.map((card, index) => (
    <Card
      key={index}
      title={card.title}
      backgroundColor={card.backgroundColor}
    >
      <p>{card.content}</p>
    </Card>
  ))}
</div>
</div>
);
}
```