2️⃣

State

useState

지금까지는 페이지를 렌더링할 때 정적인(Static) 요소들만을 사용했습니다. 그런데 웹에서는 사용자와의 상호작용을 통해 화면이 바뀌는 경우가 많습니다. 이럴 때 `useState` 를 사용하면 "상태 관리"가 가능합니다.

> 💡 나중에 배울 Hooks 중 하나가 바로 `useState` 입니다.

아래 예시는 +1, -1 버튼을 클릭하면 화면의 숫자가 변하는 예제입니다. 첫번째 챕터에서 구현했던 예제와 동일합니다. 차이점은 이번에는 리액트의 `useState` 를 이용했다는 점입니다.

https://codesandbox.io/s/infallible-sinoussi-zc83r?file=/src/App.js

코드를 볼까요?

```
import React, { useState } from "react"; // import useState

function Counter() {
  const [number, setNumber] = useState(0); // default == 0

  const onIncrease = () => {
    setNumber(number + 1);
  };

  const onDecrease = () => {
```

```
    setNumber(number - 1);
  };

  return (
    <div>
      <h1>{number}</h1>
      <button onClick={onIncrease}>+1</button>
      <button onClick={onDecrease}>-1</button>
    </div>
  );
}

export default Counter;
```

`return` 부분을 자세히 보면 JSX `<button>` 에 `onClick={onIncrease}` 와 같은 애트리뷰트를 볼 수 있습니다. `onIncrease` 는 `Counter` 컴포넌트에 정의된 함수입니다. 이 함수가 하는 일은 `useState` 의 결과값인 `number` 의 값에 1을 더하는 것입니다.

그럼 `useState` 가 하는 일이 뭘까요?

```
const [number, setNumber] = useState(0);
// const numberState = useState(0);
// const number = numberState[0];
// const setNumber = numberState[1];
```

`useState`는 기본값을 입력으로 받아 배열을 리턴하는데, 첫 번째 원소는 현재 상태를 나타내는 변수, 두번째는 이 상태를 변경할 수 있는 Setter 함수입니다.

```
const onIncrease = () => {
  setNumber(number + 1);
};
```

`onIncrease` 함수에서 이 `setNumber` 가 바로 `number` 의 상태를 변경할 수 있는 Setter 함수입니다. State는 리액트에서 정말 중요하면서 자주 쓰이는 개념이기 때문에 확실히 이해하고 넘어가시는 걸 추천합니다.

Example: Input status

Input.js라는 파일을 만들어 아래 내용을 입력합니다.

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

function Input() {
  const [inputs, setInputs] = useState({
```

```
    subject: "",
    score: "",
  });

  const onChange = (event) => {
    const { value, name } = event.target; // e.target : DOM
    setInputs({
      ...inputs, // copy inputs object => subject: '', score: ''
      [name]: value, // dynamic assign: create new entry with string value
    });
  };

  const onReset = () => {
    setInputs({
      subject: "",
      score: "",
    });
  };

  return (
    <div>
      <div>
        <h2>
          성적:
          {inputs.subject ? inputs.subject : "수학"} (
          {inputs.score ? inputs.score : 99})
        </h2>
      </div>
      <input
        name="subject"
        placeholder="수학"
        onChange={onChange}
        value={inputs.subject}
      />
      <input
        name="score"
        placeholder="99"
        onChange={onChange}
        value={inputs.score}
      />
      <button onClick={onReset}>Init</button>
    </div>
  );
}

export default Input;
```

App.js에서 위 컴포넌트를 렌더링합니다.

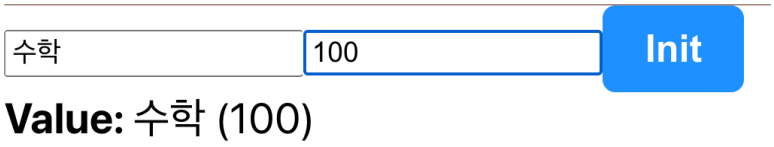
```
import React from 'react';
import Input from './Input';

function App() {
  return (
    <Input />
```

```
  );
}

export default App;
```

Dev 서버를 실행시키면 다음과 같은 화면이 나옵니다. 입력창에 텍스트를 입력하면 화면의 `Value:` 에 해당하는 텍스트가 바뀌게 되고, `Initialize` 버튼을 누르면 입력창과 텍스트가 빈 텍스트로 대체됩니다.



이게 가능한 이유는 `onChange` 가 `<input>` 엘리먼트의 이벤트이기 때문입니다. 이를 함수 `onTextChange` 에서 `e` 라는 파라미터로 받아올 수 있고, `e.target` 은 이벤트가 발생한 DOM 을 가리킵니다.

이제 다음 코드를 보겠습니다.

```
const onChange = (event) => {
  const { value, name } = event.target; // e.target : DOM
  setInputs({
    ...inputs, // copy inputs object
    [name]: value // create new entry with name key
  });
};
```

`setInputs` 부분이 약간 난해한데, 여기서 스프레드를 통해 `inputs` 객체를 복사한 다음, `name` 키의 값만 업데이트하고 있습니다. 여러 스테이트를 다루는 경우, 리액트는 기존 state 변수를 대체하기 때문에 다른 state 변수의 손실을 막기 위해 이런 방법을 사용합니다.

참고로 여기서 객체를 복사하는 이유가 한 가지 더 있습니다. `<input>` 엘리먼트가 자체적으로 state를 가지고 있기 때문입니다. HTML state와 리액트 state를 따로 관리하지 않고 리액트 state만을 사용하기 위해서입니다. 이런 컴포넌트를 "Controlled component" 라고 부릅니다.
참고 : https://reactjs.org/docs/forms.html#controlled-components