2️⃣

2회차 Basic Mission - 노진현

1. 할 일 목록 제목 옆에 새로운 버튼을 만들어서 그 버튼을 누를 시에 모든 할 일이 지워지는 기능을 만드세요. (함수 컴포넌트를 사용합니다.)

1.1 '전체 삭제' 버튼 생성

```
//App.js
//생략

return (
    <div className="flex items-center justify-center w-screen h-screen bg-blue-100">
      <div className="w-full p-6 m-4 bg-white rounded shadow-sm lg:w-3/4 lg:max-w-lg">
        <div className="flex justify-between mb-3 items-center">
          <h1>할 일 목록</h1>
          <button
          className="p-2 text-white bg-red-300 hover:bg-red-500 shadow-md rounded"
          >
            전체삭제
          </button>
        </div>

        <Lists
          todoData={todoData}
          setTodoData={setTodoData}
          handleClick={handleClick}
        />
        <Form setValue={setValue} value={value} handleSubmit={handleSubmit} />
      </div>
    </div>
  );
```

App.js에서 `<h1>할 일 목록</h1>` 상위 `<div>` 에서 이미 flex justify-between 속성을 갖고 있어서 `<button>` 과 style을 위한 `className`만 추가하였습니다.

1.2 '전체 삭제' 버튼 클릭시 모든 할 일이 지워지는 기능 구현

```
//App.js
//생략

const handleDeleteAll = () => {
    setTodoData([]);
  };

return (
    <div className="flex items-center justify-center w-screen h-screen bg-blue-100">
      <div className="w-full p-6 m-4 bg-white rounded shadow-sm lg:w-3/4 lg:max-w-lg">
        <div className="flex justify-between mb-3 items-center">
          <h1>할 일 목록</h1>
          <button
          className="p-2 text-white bg-red-300 hover:bg-red-500 shadow-md rounded"
          onClick={handleDeleteAll}
          >
            전체삭제
          </button>
        </div>
```

```
      <Lists
        todoData={todoData}
        setTodoData={setTodoData}
        handleClick={handleClick}
      />
      <Form setValue={setValue} value={value} handleSubmit={handleSubmit} />
    </div>
  </div>
);
```

- 버튼에 onClick을 부여하고 `handleDeleteAll` 함수를 생성했습니다.
- 함수는 빈 배열을 `setTodoData` 인자로 전달하여 state에 반영하는 것으로 List가 모두 삭제되도록 작성하였습니다.

2. 할 일 목록을 수정하는 기능을 만들어 주세요.

2.1 '수정' 버튼 만들기

```
<button
  className="text-sm mx-3 my-2 float-left text-gray-500"
>
  수정
</button>
```

2.2 수정 버튼 클릭 시 List에 Input Form이 생길 수 있도록 하기

2.2.1 수정/완료 구분을 위한 state 생성

```
const [isEditing, setEditState] = useState(false);
```

수정 버튼을 누르면 isEditing 을 true로 만들고, 수정이 완료되면 isEditing을 false로 두는 것으로 구분합니다.

함수 컴포넌트에서 작업중이기 때문에 useState Hook을 사용하며, 초기값으로 `false`를 넘겨줍니다.

```
<button
  className="text-sm mx-3 my-2 float-left text-gray-500"
  onClick={() => {
    setEditState(true);
  }}
>
  수정
</button>
```

2.2.2 isEditing의 값에 따라서 return 다르게 주기 (UI 구분해주기)

```
if (isEditing) {
      //수정 버튼 클릭 시 return
      return (
        <div
          key={id}
          {...provided.draggableProps}
          ref={provided.innerRef}
          {...provided.dragHandleProps}
```

```
            className={`${
              snapshot.isDragging ? "bg-gray-300" : "bg-gray-100"
            } flex items-center justify-between w-full px-4 py-1 my-2 text-gray-600 bg-gray-100 border rounded`}
          >
            <div className="items-center w-4/5 ">
              <form onSubmit={handleEditSubmit}>
                <input
                  className="w-full px-3 py-2 border border-blue-200 rounded"
                  type="text"
                  name="value"
                  value={editedValue}
                  placeholder="수정할 내용을 입력하세요."
                  onChange={handleEditChange}
                />
              </form>
            </div>
            <div className="items-center">
              <button
                className="text-sm mx-3 my-2 float-left text-gray-500"
                type="submit"
                onClick={handleEditSubmit}
              >
                완료
              </button>
              <button
                className="text-sm px-1 my-2 float-right bg-red-400 text-white rounded"
                onClick={() => {
                  handleClick(id);
                }}
              >
                X
              </button>
            </div>
          </div>
        );
      } else {
        // 수정 버튼 누르기 전 / 수정 버튼을 누른 후 완료 버튼을 누른 후 return
        return (
        //생략
        )
      }
```

IsEditing이 True 일 때, 즉 수정 중일 때에는 return에 수정 Form을 전달하고, '완료' 버튼 클릭 시 input value가 todoData에 반영될 수 있도록 해야한다. 그 전에 Form 컴포넌트와 마찬가지로 Form 값을 다루기 위해서는 수정 Form에도 키보드 타이핑에 대응할 handleChange 함수와 Form 제출 시 값을 반영할handleSubmit 함수가 필요하다.

2.3 input form value state 생성

```
const [editedValue, setEditedValue] = useState("");
```

Form의 input Value를 반영할 state를 생성해준다.

2.4 수정 '완료' 버튼 클릭 시 수정한 값이 반영되도록 함수 작성

```
//Edit Form Change => setState
const handleEditChange = (e) => {
  setEditedValue(e.target.value);
};

//Edit Form Submit => setTodoData
const handleEditSubmit = (e) => {
  e.preventDefault();
```

```
  let newTodoData = todoData.map((data) => {
    if (data.id === id) {
      data.title = editedValue;
    }
    return data;
  });

  setTodoData(newTodoData);
  setEditState(false);
};
```

handleEditChange에서 키보드 입력에 맞춰 `editedValue` state를 업데이트 해주고, handleEditSubmit에서는 `editedValue`를 기존 `todoData`에 set해 준다. 이 때, `isEditing` state의 값을 false로 set 해주면서 수정 종료 상태로 반영한다.