Ask Company

리액트와 함께 장고 시작하기 / 리액트

immer를 활용한 손쉬운 불변객체 다루기

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

불변성 (Immutable)

리액트에서는 불변성을 유지하면서, 상탯값을 업데이트해야만 합니다.

하지만 Vanilla JS에서는 생각보다 직관적이지 않은 코드이며, 코드 작성하기도 어렵습니다. ☹

array의 filter/map/reduce/slice/push 등의 항연에서 ~ ☹

```
// 이건 그럭저럭 할만합니다.

const todo = {
  name: 'immer 이해하기',
  is_completed: false,
};

const newState = {
  ...todo,
  is_completed: true,
};
```

```
// 사과/레몬을 제거하고, 딸기를 끼워넣고 싶습니다.

const fruits = ["오렌지", "사과", "레몬", "바나나"];

// #1) [BAD] 제거된 객체들을 반환하며, fruits 객체를 변경합니다.
fruits.splice(1, 2, "딸기");

// #2) [GOOD] fruits를 변경하지 않고, 새로운 객체 newFruits를 생성
const newFruits = [
  ...fruits.slice(0, 1),
  "딸기",
  ...fruits.slice(3),
];

// #3) [GOOD] immer를 사용할 경우, 익숙한 코드로 불변성을 지킬 수 있습니다.
const newFruits = produce(fruits, draft => {
  draft.splice(1, 2, "딸기");
});
```

immer

설치: yarn add immer

```
const {produce} = require("immer"); // nodejs에서의 모듈 시스템
// import { produce } from "immer"; // 리액트에서는 이렇게.

const fruits = ["오렌지", "사과", "레몬", "바나나"];

                          base 객체   receipt 함수
const newFruits = produce(fruits, draft => {
  draft.splice(1, 2, "딸기");
});

console.log("fruits :", fruits);
console.log("newFruits :", newFruits);
```

새로운 라이브러리를 배울 필요없이, 일반 자바스크립트 객체 쓰듯이 사용하면, 불변객체로서 처리해줍니다.

https://immerjs.github.io/immer/docs/introduction

Quiz: immer를 쓰지 않고, immer를 쓰고 ~

baseState는 변경하지 않고, 새로운 newState object는 ...

두번째 object의 done=true로 세팅되고

끝에 { todo: "Tweet about it" } 가 추가되어야만 합니다.

```
const baseState = [
  {
    todo: "Learn typescript",
    done: true
  },
  {
    todo: "Try immer",
    done: false
  }
];
```

```
[
  { todo: 'Learn typescript', done: true },
  { todo: 'Try immer', done: true },
  { todo: 'Tweet about it' }
];
```

My Codes ~

```
const newState1 = [
  ...baseState.map(
    (tweet, index) => {
      return index !== 1
        ? tweet
        : { ...tweet, done: true };
    }
  ),
  {
    todo: "Tweet about it",
  }
];
```

```
const newState2 = produce(baseState, draftState => {
  draftState.push({todo: "Tweet about it"})
  draftState[1].done = true
})
```

immer를 활용하여, onKeyDown 루틴 개선

```
onKeyDown = (event) => {
  if ( event.keyCode === 13 ) {
    const { todoList, current } = this.state;

    this.setState({
      todoList: [...todoList, current],
      current: ''
    });
  }
}
```

```
import { produce } from 'immer';


onKeyDown = (event) => {
  if ( event.keyCode === 13 ) {
    const newState = produce(this.state, draft => {
      draft.todoList.push(draft.current);
      draft.current = '';
    });

    this.setState(newState);
  }
}
```

immer를 안 쓰는 것이
가독성이 더 좋을 때도 있습니다. ☺

상탯값 setter에서의 immer

produce의 첫번째 인자가 함수라면

그 인자는 receipt 함수로 사용되며,

업데이트 함수를 반환합니다.

→ 상탯값 setter와 엮어쓰기 좋아요.

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

function App() {
  const [state, setState] = useState({ score: 10 });

  const onClick1 = () => {
    setState({
      score: state.score + 1,
    });
  };

  const onClick2 = () => {
    setState(prevState =>
      produce(prevState, draft => {
        draft.score += 1;
      })
    );
  };

  const onClick3 = () => {
    setState(produce(draft => {
      draft.score += 1;
    }));
  };

  // 생략 ...
}
```

Life is short.
You need Python and Django.

I will be your pacemaker.