#간단한 To-Do 앱 만들며 리액트 익히기

- JavaScript Syntax Extenstion
  - 자바스크립트의 확장 문법
  - 리액트에서 JSX를 이용해 화면에서 UI가 보이는 모습을 나타내줌

    >> App.js에서 return 하는 부분
  - JSX를 이용하면 UI를 나타낼 때 자바스크립트(logic)과 HTML 구조(markup)을 같이 사용할 수 있기 때문에 기본 UI에 데이터가 변하는 것들이나 이벤트들이 처리되는 것을 쉽게 구현 가능
  - 리액트에서 의무 사용은 아니지만 편리하므로 사용 권장하며 공식적인 자바스크립트의 문법은 아님
  - 브라우저에서 실행하기 전 바벨을 사용해 일반 자바스크립트 형태의 코드로 변환됨.
  - vs 원래 리액트에서 화면을 그리는 방식
    - React.createElement API를 사용해 엘리먼트를 생성한 후 이 엘리먼트를 in-memory에 저장하고, ReactDOM.redner 함수를 사용해 실제 웹 브라우저에 그려줌

- SPA(Single Page Application)
  - App.js 파일의 소스코드를 변경 시 변경 부분이 화면에 적용됨
  - vs Multi Page Application
    - SPA : 웹 사이트의 전체 페이지를 하나의 페이지에 담아 동적으로 화면을 바꿔가며 표현하는 방식

      index.html에 영역을 정해 요청에 따라 보여줌
    - 전통적인 웹 사이트는 a → b 페이지로 전환 시 a.html 다음에 b.html 보여주는 방식.

      ⇒ index.html만 존재하는 SPA에서는 HTML5 History API를 이용해 페이지 전환 가능(브라우징_

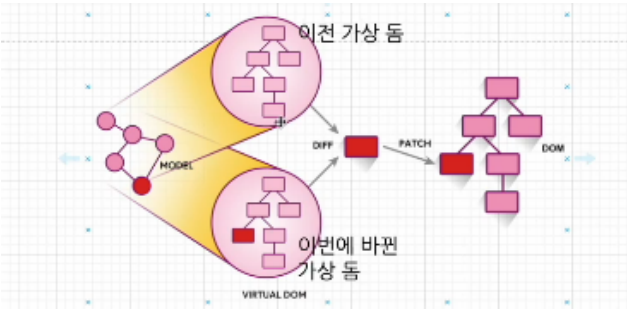
      ⇒ React Router Dom을 이용해 History API를 활용해 페이지를 전환시켜줌

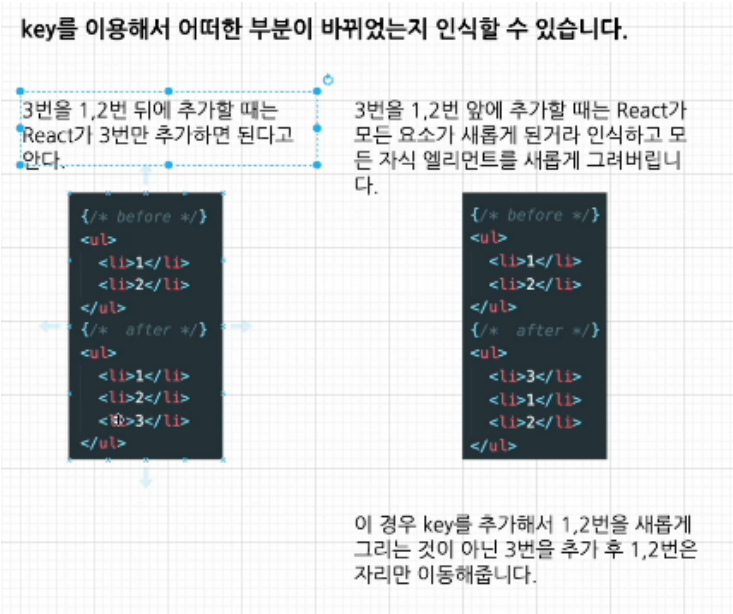
History.back()	세션 기록의 바로 뒤 페이지로 이동하는 비동기 메서드. 브라우저의 뒤로 가기를 누르는 것과 같은 효과를 낸다.
History.forward()	세션 기록의 바로 앞 페이지로 이동하는 비동기 메서드. 브라우저의 앞으로 가기를 누르는 것과 같은 효과를 낸다.
History.go()	특정한 세션 기록으로 이동하게 해 주는 비동기 메서드. 1을 넣어 호출하면 바로 앞 페이지로, -1을 넣어 호출하면 바로 뒤 페이지로 이동한다.
History.pushState()	주어진 데이터를 세션 기록 스택에 넣는다. 직렬화 가능한 모든 Javascript 객체를 저장하는 것이 가능하다.
History.replaceState()	최근 세션 기록 스택의 내용을 주어진 데이터로 교체한다.

- map()
  - map 메소드는 배열 내 모든 요소 **각각에 대해** 주어진 함수를 호출한 결과를 모아 새로운 배열을 반환
  - https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map

- JSX Key 속성
  - 리액트에서 요소의 리스트를 나열할 때는 Key가 필요함

    리액트가 변경, 추가, 제거된 항목을 식별하는데 키가 필요. 요소에 안정적인 ID를 부여하려면 배열 내부 요소에 Key를 할당해야 함!
  - 리액트는 가상 돔을 이용해 바뀐 부분만 실제 돔에 적용함





왼쪽 케이스의 경우 문제가 없지만 오른쪽 케이스의 경우 Key가 없으면 모든 자식 엘리먼트를 새로 그려야하므로 Key를 이용해 바뀐 부분만 식별.

- Filter 메소드
  - 주어진 함수의 테스트를 통과하는 모든 요소를 모아 새로운 배열로 반환
  - https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

- React State

  https://ko.reactjs.org/docs/react-component.html#setstate
  - 리액트에서 데이터가 변할 때 화면을 다시 렌더링하기 위해서는 React State를 사용함
  - state는 어떤 컴포넌트에 한정하여 사용되는 데이터를 포함하며, 데이터는 변경이 가능. 사용자가 자유롭게 정의할 수 있고, 일반적인 자바스크립트 객체여야함.
  - React State : 컴포넌트의 렌더링 결과물에 영향을 주는 데이터를 갖고있는 객체, State가 변경되면 컴포넌트는 re-rendering. 또한, State는 컴포넌트 안에서 관리
  - setState(updater, [callback])
    - setState()는 컴포넌트 state의 변경사항을 대기열에 집어넣고, React에게 해당 컴포넌트와 그 자식들이 갱신된 state를 사용해 다시 렌더링해야 함을 알림→ 이벤트 핸들러나 서버 응답으로 UI 갱신시 많이 사용하는 메서드
    - updater : (state, props) ⇒ stateChange
      - state는 변경 사항이 적용되는 시점에 컴포넌트가 가지는 state에 대한 참조로 직접 변경하면 안됨.
      - state와 props를 기반으로 새로운 객체를 만들어 변경 사항을 표현해야 함

```
this.setState((state, props) => {
  return {counter : state.counter + props.step};
});

this.setState({quantity : 2}) //파라미터로 객체 전달도 가능
```

  - callback : setState() 실행 완료 후 컴포넌트가 렌더링된 후 실행될 함수에 대한 콜백으로 생략 가능.

- 전개연산자(Spread Operator)

  https://www.w3schools.com/react/react_es6_spread.asp

  https://paperblock.tistory.com/62

  - ECMAScript6에서 추가. 특정 객체 또는 배열의 값을 다른 객체, 배열로 복제하거나 옮길 때 사용

1. 배열에서의 Spread Operator

✅ 배열 병합 (Array Concatenation)

기존에 두 개의 배열을 결합하는 데 있어서 concat 메서드를 이용하곤 했습니다. ES6에서는 spread 연산자를 이용하여 좀 더 깔끔한 배열 병합이 가능합니다.

```
// 기존 방식
var arr1 = [1,2,3];
var arr2 = [4,5,6];

var arr = arr1.concat(arr2);
console.log(arr); // [ 1, 2, 3, 4, 5, 6 ]

// ES6 spread operator
var arr1 = [1,2,3];
var arr2 = [4,5,6];

var arr = [...arr1, ...arr2];
console.log(arr); // [ 1, 2, 3, 4, 5, 6 ]
```

또한 Spread 연산자를 이용하면 다양한 형태의 배열 병합을 비교적 간단하게 수행할 수 있습니다.

```
var arr1 = [1,2,3];
var arr2 = [4,5,6];
arr1.push(...arr2)

console.log(arr1); // [1,2,3,4,5,6]


var arr1 = [1,2];
var arr2 = [0, ...arr1, 3, 4];

console.log(arr2); // [0, 1, 2, 3, 4]
```

✅ 배열 복사 (Copying Arrays)

JavaScript에서 배열을 새로운 변수에 할당하는 경우 새로운 배열은 기존 배열을 **참조**합니다. 따라서 새로운 배열을 변경하는 경우 원본 배열 역시 변경됩니다.

```
// 단순 변수 할당
var arr1 = ['철수','영희'];
var arr2 = arr1;

arr2.push('바둑이');
console.log(arr2); // [ "철수", "영희", "바둑이" ]
// 원본 배열도 변경됩니다.
console.log(arr1); // [ "철수", "영희", "바둑이" ]
```

배열 참조가 아닌 배열 복사를 위해서 기존에는 slice 또는 ES5의 map을 이용하여 배열을 복사했습니다.

```
// Javascript array 복사
var arr1 = ['철수','영희'];
var arr2 = arr1.slice();

arr2.push('바둑이');
console.log(arr2); // [ "철수", "영희", "바둑이" ]
// 원본 배열은 변경되지 않습니다.
console.log(arr1); // [ "철수", "영희" ]


// ES5 map
var arr1 = ['철수','영희'];
var arr2 = arr1.map((item) => item);

arr2.push('바둑이');
console.log(arr2); // [ "철수", "영희", "바둑이" ]
// 원본 배열은 변경되지 않습니다.
console.log(arr1); // [ "철수", "영희" ]
```

ES6의 Spread 연산자를 사용하면, 다음과 같이 새로운 **복사된 배열**을 생성할 수 있습니다.

```
// ES6 spread operator
var arr1 = ['철수','영희'];
var arr2 = [...arr1];

arr2.push('바둑이');

console.log(arr2); // [ "철수", "영희", "바둑이" ]
// 원본 배열은 변경되지 않습니다.
console.log(arr1); // [ "철수", "영희" ]
```

참고로 Spread 연산자를 이용한 복사는 얕은(shallow) 복사를 수행하며, 배열 안에 객체가 있는 경우에는 객체 자체는 복사되지 않고 원본 값을 참조합니다. 따라서 원본 배열 내의 객체를 변경하는 경우 새로운 배열 내의 객체 값도 변경됩니다.

```
var arr1 = [{name: '철수', age: 10}];
var arr2 = [...arr1];
```

```
arr2[0].name = '영희';

console.log(arr1); // [ {name:'영희', age: 10}]
console.log(arr2); // [ {name:'영희', age: 10}]
```

2. 함수에서의 Spread Operator

✅ Rest Parameter

함수를 호출할 때 함수의 매개변수(parameter)를 spread operator로 작성한 형태를 Rest parameter라고 부릅니다. Rest 파라미터를 사용하면 함수의 파라미터로 오는 값들을 모아서 "배열"에 집어넣습니다. 이를 통해서 깔끔한 함수 표현을 적용할 수 있습니다.

아래 더하기 예제를 살펴볼까요?

```
function add(...rest) {
  let sum = 0;
  for (let item of rest) {
    sum += item;
  }
  return sum;
}

console.log( add(1) ); // 1
console.log( add(1, 2) ); // 3
console.log( add(1, 2, 3) ); // 6
```

위의 결과에서 보이듯이, Rest parameter를 이용하면 인자의 개수에 상관없이 모든 인자를 더하는 함수를 쉽게 구현할 수 있습니다. (기존 JavaScript에서는 "**arguments**"를 이용하여 처리할 수 있습니다.)

다음과 같이 기본 인자를 섞어서 사용하는 방법도 가능합니다.

```
function addMul(method, ...rest){
  if (method === 'add') {
    let sum = 0;
    for (let item of rest) {
      sum += item;
    }
     return sum;
  } else if (method === 'multiply'){
    let mul = 1;
    for (let item of rest) {
      mul *= item;
    }
     return mul;
  }

}

console.log( addMul('add', 1,2,3,4) ); // 10
console.log( addMul('multiply', 1,2,3,4) ); // 24
```

단, Rest파라미터는 항상 제일 마지막 파라미터로 있어야 합니다. function addMul (...rest, method){ }와 같은 방식으로는 사용할 수 없습니다.

✅ 함수 호출 인자로 사용

위의 경우와 반대로, 함수 정의 쪽이 아니라 함수를 Call 할 때 spread operator를 사용할 수 있습니다. 기존에는 배열로 되어있는 내용을 함수의 인자로 넣어주려면 손으로(?) 풀어서 넣어주던지, **Apply**를 이용했어야 하지만 spread operator를 이용하면 배열 형태에서 바로 함수 인자로 넣어줄 수 있습니다.

가장 간단한 예로 Math 함수를 예로 들어볼까요?

```
var numbers = [9, 4, 7, 1];
Math.min(...numbers); // 1
```

Map 하고 섞으면 다음과 같이도 응용이 가능하겠네요!

```
var input = [{name:'철수',age:12}, {name:'영희',age:12}, {name:'바둑이',age:2}];

//가장 어린 나이 구하기.
var minAge = Math.min( ...input.map((item) => item.age) );
console.log(minAge); // 2
```

3. 객체에서의 Spread Operator

ES2018 (ES9)에서는 객체와 관련된 사항이 추가되었습니다. 배열에 대한 Spread Opeator를 지원하는 최근의 브라우저는 객체에 대한 Spread Operator 역시 지원합니다.

✅ 객체 복사 또는 업데이트

객체에서 spread operator를 이용하여 객체의 복사 또는 프로퍼티를 업데이트 할 수 있습니다.

간단한 State Management 구현을 위해서 다음과 같은 방식으로 응용하여 사용하기도 합니다.

```
var currentState = { name: '철수', species: 'human'};
currentState = { ...currentState, age: 10};

console.log(currentState)// {name: "철수", species: "human", age: 10}

currentState = { ...currentState, name: '영희', age: 11};
console.log(currentState); // {name: "영희", species: "human", age: 11}
```

위에서 두번째 예제는 **객체의 프로퍼티를 오버라이드** 함으로써 객체가 업데이트되는 것을 이용한 내용입니다.

4. Destructuring

Spread Operator는 배열이나 객체에서의 destructuring에 사용될 수 있습니다. 이 경우에 의미적으로는 spread operator라기보다는 rest parameter에 가까운 형태가 되겠네요.

```
var a, b, rest;
[a, b] = [10, 20];

console.log(a); // 10
console.log(b); // 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest); // [30,40,50]

({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});
console.log(a); // 10
console.log(b); // 20
console.log(rest); // {c: 30, d: 40}
```

⇒ to-do-app 내 사용

```
this.setState({todoData : [...this.state.todoData, newTodo]});
```

#1 기본 구조

React.Component