함수형 도구

배열에서의 함수형 도구

map, filter, reduce

배열에서의 가장 핵심이 되는 함수형 도구들
셋 다 배열을 순회하며 callback함수를 수행해주는 역할을 함

map

요소를 순회하며, callback을 수행한 결과로 구성된 배열을 반환하는 함수

React에서 받아온 데이터 기반으로 반복되는 컴포넌트를 사용해서
화면을 보여줄 때도 사용됨

```
function Home() {
  const { articles } = useArticlesData();
  return (
    <div>
      {articles.map((article) => (
        <Article data={article} />
      ))}
    </div>
  );
}
```

map

```jsx
function Home() {
  const { articles } = useArticlesData();
  return (
    <div>
      {articles.map((article) => (
        <Article data={article} />
      ))}
    </div>
  );
}
```


```jsx
import { jsx } from 'react/jsx-runtime';

function Home() {
  const { articles } = useArticlesData();
  return jsx('div', {
    children: articles.map((article) =>
      jsx(Article, {
        data: article,
      })
    ),
  });
}
```

map

```jsx
import { jsx } from 'react/jsx-runtime';

function Home() {
  const { articles } = useArticlesData();
  return jsx('div', {
    children: articles.map((article) =>
      jsx(Article, {
        data: article,
      })
    ),
  });
}
```

```jsx
function Article(article) {
  return jsx(Article, {
    data: article,
  });
}

articles.map(Article);
```

```
[data1, data2, data3, data4];
```

→

```
[
  'Article Component의 인자로 data1을 넘긴 반환 객체1',
  'Article Component의 인자로 data2를 넘긴 반환 객체2',
  'Article Component의 인자로 data3를 넘기 반환 객체3',
];
```

이것이 root 객체의 children요소가 되고, 수많은 과정을 거쳐 DOM Element가 되는 것

filter

callback을 통해 배열에서 일부 항목을 뽑아낸 배열을 반환하는 함수

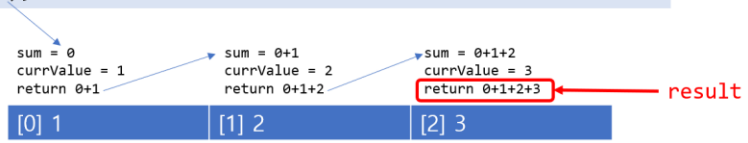
```
[1, null, undefined].filter(Boolean); // [1]
```

알고 계셨나요? Boolean함수를 통해 invalid값을 손쉽게 필터링할 수 있습니다.

reduce

배열을 순회하되, 배열을 반환하지 않고, callback함수를 요소에 맞게 실행한 누적 결과를 반환하는 함수

```
const result = arr.reduce(function add(sum, currValue) {
  return sum + currValue;
}, 0);
```



reduce는 합계 원툴이 아님.
reduce로는 생각보다 많은 일을 할 수 있음

reduce 로 할 수 있는 일

1. 실행취소/실행 복귀

```
const userActions = ['something', 'something2'];
const userActionResult = userAction.reduce(
  (acc, cur) => calculateUserAction(acc, cur),
  {}
);
```

2. join

```
const join = (array: string[]) => {
  return array.reduce((acc, curString) => acc + curString, '');
};
```

3. Min/Max값 구하기

추가로 소개하고 싶은 배열 함수형 도구

Every, Some

Every와 Some은 프로덕트 개발 시 복잡한 권한 관리 등에서 &&와 ||처럼 사용될 수 있음

ex)
1. **선생님이거나**
2. **기관장인 경우**
3. **선생님의 권한을 받은 학생인 경우만**

어떤 기능을 사용할 수 있음.
그러나 전체적으로 기능이 꺼져 있는 경우는 사용 불가함

Every, Some

AS-IS

```
function SomeComponent() {
    ...
  return (
    <div>
    {(isTeacher || isOrganizationRepresentor || isLicensedStudent) && isOn ? <SomeFeature> : null}
    </div>
  );
}
```

ex)
1. 선생님이거나
2. 기관장인 경우
3. 선생님의 권한을 받은 학생인 경우만

어떤 기능을 사용할 수 있음.
그러나 전체적으로 기능이 꺼져 있는 경우는 사용 불가함

Every, Some

TO-BE

```
function SomeComponent() {
    ...
    const permittedUserCondition=[isTeacher,isOrganizationRepresentor,isLicensedStudent];
  return (
    <div>
    {permittedUserCondition.some(Boolean) && isOn ? <SomeFeature> : null}
    </div>
  );
}
```

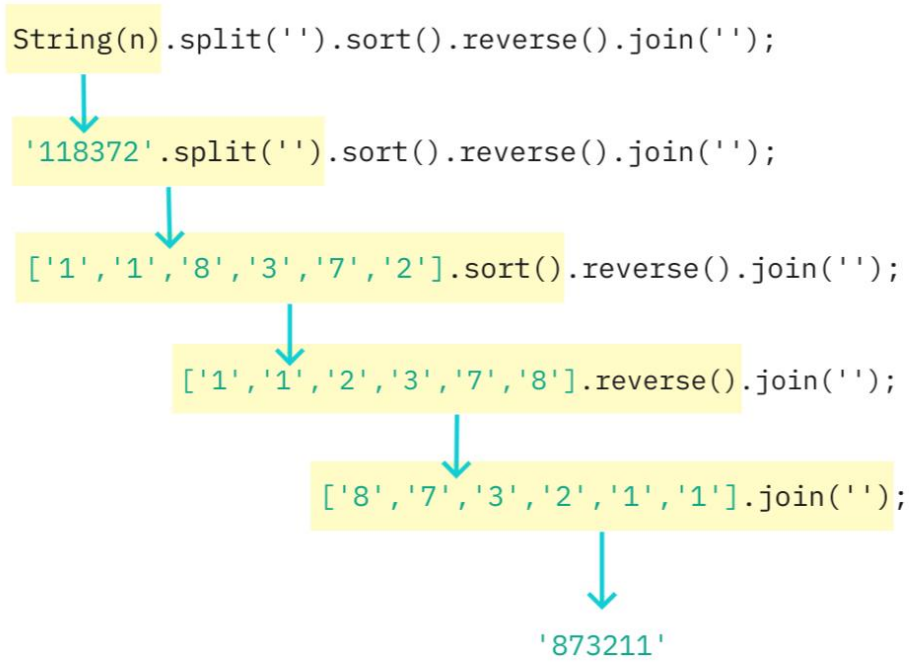
ex)
1. 선생님이거나
2. 기관장인 경우
3. 선생님의 권한을 받은 학생인 경우만

어떤 기능을 사용할 수 있음.
그러나 전체적으로 기능이 꺼져 있는 경우는 사용 불가함

함수형 도구 체이닝

함수형 도구 체이닝의 두 가지 방법

1. 단계에 이름 붙이기
2. Callback 에 이름 붙이기



단계에 이름 붙이기

단계에 이름을 붙이면 훨씬 명확해지고, 구현도 알아보기 쉬움

```
const run = async () => {
  const allDirectories = await getAllDirectoryInCurrentPath(currentPath);
  const packageJsonList = await getExecutableDirectories(allDirectories);
  const choosedPackage = await getChoosedPackage(packageJsonList);
```

https://github.com/d0422/multi-starter/blob/main/src/index.ts

하지만... 콜백 재사용성이 떨어지고, 콜백 자체가 인라인으로 사용되기 쉬움

콜백에 이름 붙이기

콜백에 이름을 붙이면, 콜백을 인라인으로 전달하지 않으며, 단계도 이해하기 쉬워짐

```
export const decodeHtmlEntity = (htmlString: string) => {
  return htmlString
    .replace(/&hellip;/g, '…')
    .replace(/ /g, ' ')
    .replace(/&lt;/g, '<')
    .replace(/&gt;/g, '>')
    .replace(/&amp;/g, '&')
    .replace(/&quot;/g, '"')
    .trim();
};
```

개인적으로 생각하는 1번의 문제점

물론 1번도 좋은 추상화 방법이지만, 1번 방법에 대해서는 아래의 추가적인 문제가 있다고 생각함

1. **단계에 이름을 붙여 놓았으나, 프로젝트 규모가 매우 커져서, 해당 함수를 어디서 사용하는지 명확하게 파악하기 어렵다면?**

2. **추가로, 사용하고 있는 곳에서 해당 단계에 강하게 의존하고 있어 변경이 어려워진다면?**

개인적으로 생각하는 1번의 문제점

일정 스크롤 이상일때 특정 상태를 true로 바꾸는 기능

```tsx
import React, { useEffect, useState } from 'react'

const HelloFEConf: React.FC = () => {
  const [scrolled, setScrolled] = useState(false)

  useEffect(() => {
    const onScroll = () => {
      if (window.scrollY > 500) {
        setScrolled(true)
      } else {
        setScrolled(false)
      }
    }
    window.addEventListener('scroll', onScroll)

    return () => {
      window.removeEventListener('scroll', onScroll)
    }
  }, [setScrolled])

  return <div>...</div>
}
```

출처: https://www.youtube.com/watch?v=YP7d9ae_VzI

개인적으로 생각하는 1번의 문제점

리팩터링? -> 이름 붙이기

```
const HelloFEConf: React.FC = () => {
  const scrolled = useScrolled()

  return <div>...</div>
}
```

```
function useScrolled() {
  const [scrolled, setScrolled] = useState(false)

  useEffect(() => {
    const onScroll = () => {
      if (window.scrollY > 500) {
        setScrolled(true)
      } else {
        setScrolled(false)
      }
    }
    window.addEventListener('scroll', onScroll)

    return () => {
      window.removeEventListener('scroll', onScroll)
    }
  }, [setScrolled])

  return scrolled
}
```

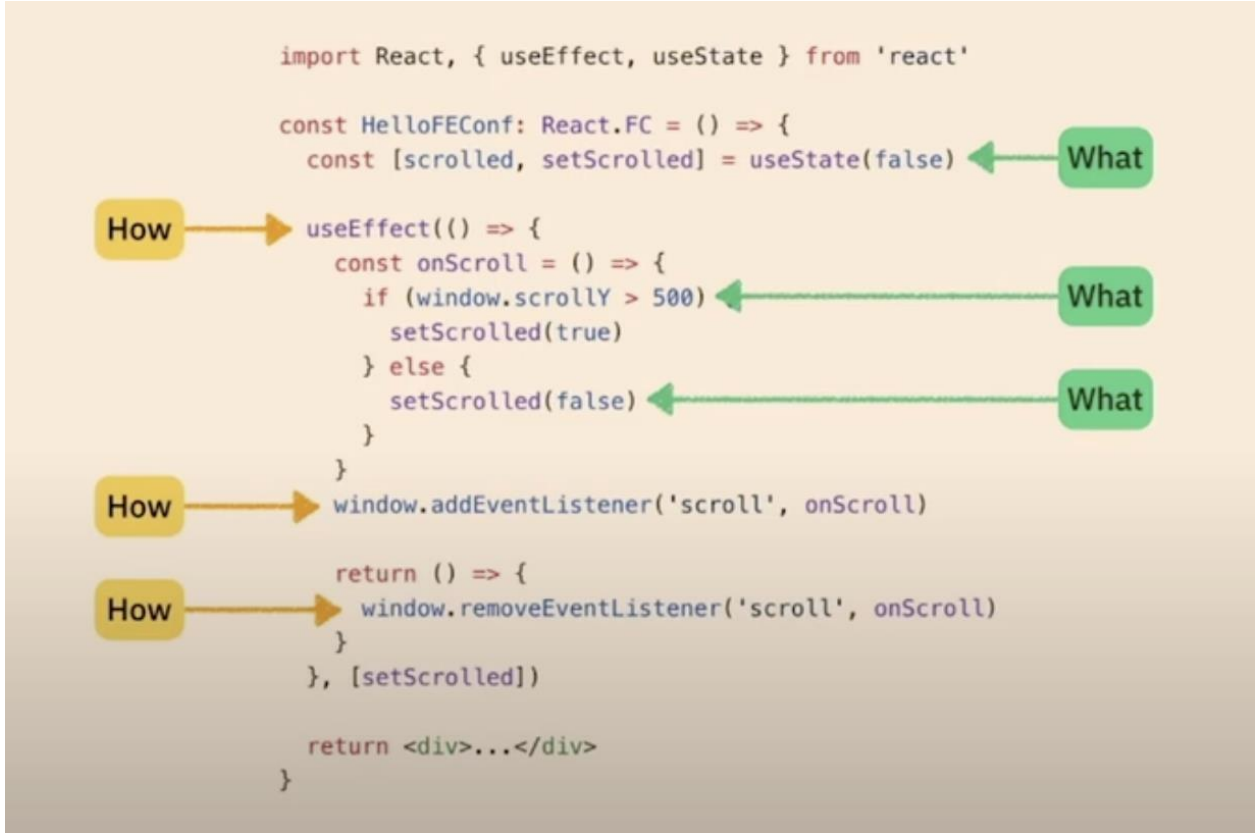
응집성이 올라감!
그런데...

1. Scroll 값인 500값을 바꾸려면?

2. 스크롤 됐을 때, 상태 변경이 아닌, 어떤 액션을 바로 일으키고 싶다면?

개인적으로 생각하는 1번의 문제점

how와 what을 분리해서 생각하기



How는 Hook에서
What은 사용하는 곳에서 지정하도록 변경

개인적으로 생각하는 1번의 문제점

how와 what을 분리해서 생각하기

How

```
function useScrollEffect(
  listener: (scrollY: number) => void,
  deps?: any[]
) {
  useEffect(() => {
    const onScroll = () => {
      listener(window.scrollY)
    }
    window.addEventListener('scroll', onScroll)

    return () => {
      window.removeEventListener('scroll', onScroll)
    }
  }, deps)
}
```

What

```
const HelloFEConf: React.FC = () => {
  const [scrolled, setScrolled] = useState(false)

  useWindowScrollEffect(
    (scrollY) => {
      if (scrollY > 500) {
        setScrolled(true)
      } else {
        setScrolled(false)
      }
    },
    [setScrolled]
  )

  return <div>...</div>
}
```

이 방법은 hook 자체의 재사용성도 높이고, 의존성 문제도 줄일 수 있게 됨
구현부에는 how만 들어있어서 사용하는 사람은 what만 정의하면 됨

개인적으로 생각하는 1번의 문제점

callback에 이름 붙이기는 고차함수(how)와, 어떤 일을 하는지(how)를 각각 선언하고, 구현부에서 이를 합쳐 what만을 표기하는 방식

```
const transformedUsers = users
  .filter(filterByScore)
  .map(capitalizeName)
  .map(categorizeByAge)
  .map(extractRelevantData);
```

```
const filterByScore = (user) => user.score >= 50;

const capitalizeName = (user) => ({
  ...user,
  name: user.name.toUpperCase(),
});

const categorizeByAge = (user) => ({
  ...
});

const extractRelevantData = (user) => ({
  ...
});
```

체인 최적화

map, filter, reduce는 copy-on-write를 기반으로 작동하기 때문에, 체이닝이 길어지는 경우, 매번 배열이 새롭게 만들어짐
GC가 높은 성능으로 해결해주긴 하지만, 그래도 최적화할 수 있음

1. map, filter를 두 번 사용하는 경우 callback을 합치는 형태

2. map다음 reduce를 사용하는 경우, reduce한번만 수행하는 형태

물론, 최적화를 할 수 있다는 것이지, 하는 것을 추천한다는 것은 아님.
명확한 단계로 선언적으로 관리 되었을 때, 더 이해하기 쉬움

다양한 함수형 도구

es-toolkit

https://es-toolkit.slash.page/ko/

groupBy

주어진 키 생성 함수에 따라서 배열의 요소를 분류해요.

이 함수는 파라미터로 배열과 각 요소에서 키를 생성하는 함수를 받아요. 키는 함수에서 생성된 키이고, 값은 그 키를 공유하는 요소끼리 묶은 배열인 객체를 반환해요.

인터페이스

```typescript
function groupBy<T, K extends PropertyKey>(arr: T[], getKeyFromItem: (item: T)
```

파라미터

- `arr` (`T[]`): 요소를 분류할 배열.
- `getKeyFromItem` (`(item: T) => K`): 요소에서 키를 생성하는 함수.

반환 값

(`Record<K, T[]>`): 키에 따라 요소가 분류된 객체를 반환해요.

countBy

배열에 속해 있는 요소를 `mapper` 함수가 반환하는 값 기준으로 분류하고, 개수를 세요.

인터페이스

```typescript
function countBy<T, K extends PropertyKey>(arr: T[], mapper: (item: T) => K): R
```

파라미터

- `arr` (`T[]`): 요소의 갯수를 세고자 하는 배열.
- `mapper` (`(item: T) => K`): 요소를 분류할 기준이 되는 값을 반환하는 함수.

반환 값

(`Record<K, number>`) 각 요소가 분류별로 몇 개 있는지를 계산한 객체.

꼿