ES6 Way to Clone an Array 🐑

When we need to copy an array, we often times used slice. But with ES6, you can also use the spread operator to duplicate an array. Pretty nifty, right 🤩

```js
const sheeps = ['🐑', '🐑', '🐑'];

// Old way
const cloneSheeps = sheeps.slice();

// ES6 way
const cloneSheepsES6 = [...sheeps];
```

Contents

Why Can't I Use `=` to Copy an Array?

Because arrays in JS are reference values, so when you try to copy it using the `=` it will only copy the reference to the original array and not the value of the array. To create a real copy of an array, you need to copy over the value of the array under a new value variable. That way this new array does not reference to the old array address in memory.

```js
const sheeps = ['🐑', '🐑', '🐑'];

const fakeSheeps = sheeps;
const cloneSheeps = [...sheeps];

console.log(sheeps === fakeSheeps);
// true --> it's pointing to the same memory space

console.log(sheeps === cloneSheeps);
// false --> it's pointing to a new memory space
```

Problem with Reference Values

If you ever dealt with Redux or any state management framework. You will know immutability is super important. Let me briefly explain. An immutable object is an object where the state can't be modified after it is created. The problem with JavaScript is that `arrays` are mutable. So this can happen:

```js
const sheeps = ['🐑', '🐑'];

const sheeps2 = sheeps;

sheeps2.push('🐺');

console.log(sheeps2);
// [ '🐑', '🐑', '🐺' ]

// Ahhh 😱 , our original sheeps have changed?!
console.log(sheeps);
// [ '🐑', '🐑', '🐺' ]
```

That's why we need to clone an array:

```js
const sheeps = ['🐑', '🐑'];

const sheeps2 = [...sheeps];

// Let's change our sheeps2 array
sheeps2.push('🐺');

console.log(sheeps2);
// [ '🐑', '🐑', '🐺' ]
// ✅ Yay, our original sheeps is not affected!
console.log(sheeps);
// [ '🐑', '🐑' ]
```

Mutable vs Immutable Data Types

Mutable:

- object
- array
- function

Immutable:

All primitives are immutable.

- string
- number
- boolean
- null
- undefined
- symbol

Shallow Copy Only

Please note `spread` only goes one level deep when copying an array. So if you're trying to copy a multi-dimensional arrays, you will have to use other alternatives.

```js
const nums = [[1, 2], [10]];

const cloneNums = [...nums];

// Let's change the first item in the first nested item in our cloned array.
```

```js
cloneNums[0][0] = '👻';

console.log(cloneNums);
// [ [ '👻', 2 ], [ 10 ], [ 300 ] ]

// NOOooo, the original is also affected
console.log(nums);
// [ [ '👻', 2 ], [ 10 ], [ 300 ] ]
```

🤓 Here's an interesting thing I learned. Shallow copy means the first level is copied, deeper levels are **referenced**.

Community Input

`Array.from` is Another Way to Clone Array

```js
const sheeps = ['🐑', '🐑', '🐑'];

const cloneSheeps = Array.from(sheeps);
```

Thanks: @hakankaraduman

- _@hakankaraduman:_ yes, I try not to use spread, it confuses me when reading the code because it does two things, spreading or gathering according to context
- *CJ J*: I think the best way is the one that most closely matches the semantics of the operation. I prefer to use `Array.from`

Resources

- MDN Web Docs - Primitive
- MDN Web Docs - Spread Syntax
- MDN Web Docs - Slice
- Stack Overflow: Why is a spread element unsuitable for copying multidimensional arrays?