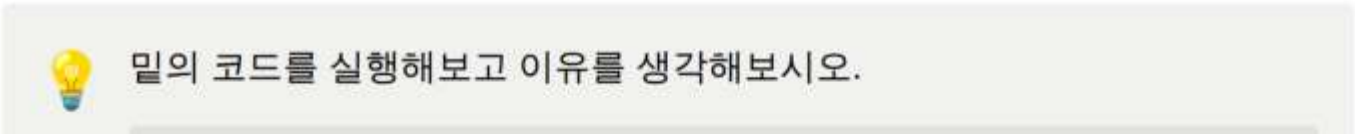



```
> const original = [
  [18, 18, 18, 18],
  [19, 19, 19, 19],
  [20, 20, 20, 20],
  [21, 21, 21, 21],
  ];
  const copy = original.slice();
  console.log(JSON.stringify(original) === JSON.string
  copy[0][0] = 99;
  copy[2].push("02");
  console.log(JSON.stringify(original) === JSON.string
  console.log(original);
  console.log(copy);
true
true
▼ (4) [Array(4), Array(4), Array(5), Array(4)]
  ▶ 0: (4) [99, 18, 18, 18]
  ▶ 1: (4) [19, 19, 19, 19]
  ▶ 2: (5) [20, 20, 20, 20, '02']
  ▶ 3: (4) [21, 21, 21, 21]
    length: 4
  ▶ [[Prototype]]: Array(0)
▼ (4) [Array(4), Array(4), Array(5), Array(4)]
  ▶ 0: (4) [99, 18, 18, 18]
  ▶ 1: (4) [19, 19, 19, 19]
  ▶ 2: (5) [20, 20, 20, 20, '02']
  ▶ 3: (4) [21, 21, 21, 21]
    length: 4
  ▶ [[Prototype]]: Array(0)
```

내가 독학하면서 배운
얕은복사라 하면
얕은복사는 주소값이
같기에 메모리가 적게들고
깊은복사는 주소값이
다르기에 메모리가 더든다
정도로 알고있다
.slice()는 얕은복사
다시 정의를 안해주더라도
같은 주소값을 공유하고
있기에 같이 변하는 모습
그나저나 배열에 숫자랑
문자열 들어가는거 재밌네
ㅋㅋ

```
> const obj = { a: 1 };
  const newObj = Object.assign({}, obj);
  newObj.a = 2;
  console.log(obj);
  console.log(obj === newObj);
▼ {a: 1}
    a: 1
  ▶ [[Prototype]]: Object
false
undefined
>
```

신기하다 1차원에서는
깊은복사가

2차원에서는 얕은복사가
일어나다니 내가 이해한바로는
객체 안에 숫자나 문자열이
있으면 이 숫자나 문자열을
가져오기에 숫자나 문자열을
바꾸면 같이 안바뀐다 하지만
객체안에 객체가 들어가있다면
그안에 숫자를 복사하기위해
객체를 복사 하기에 위의경우는
false 아래의 경우는 같이
바뀌기에 true가 나오는듯 함

```
> const obj = {
  a: 1,
  b: {
  c: 2,
  },
  };
  const newObj = Object.assign({}, obj);
  newObj.b.c = 3;
  console.log(obj);
  console.log(obj.b.c === newObj.b.c);
▼ {a: 1, b: {…}}
    a: 1
  ▶ b: {c: 3}
  ▶ [[Prototype]]: Object
true
undefined
```

```
const obj = { a: 1 };
const newObj = Object.assign({}, obj);
newObj.a = 2;
console.log(obj);
console.log(obj === newObj);
```

```
▶ {a: 1}
false
undefined
```

```
const obj = {
a: 1,
b: {
c: 2,
    },
};
const newObj = { ...obj };
newObj.b.c = 3;
console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

```
▼ {a: 1, b: {…}}
    a: 1
  ▶ b: {c: 3}
  ▶ [[Prototype]]: Object
true
```

깊은복사랑 뭐가 다른거지?
깊은복사를 직접
만들어낸건가? 객체를
계속해서 복사해내는데
객체가 아닐때까지 복사하네
깊은복사랑 완전히
같은성능일까?

방금것과 다른점이라면
Const newObj = { . . .oobj};인데
… 연산자가 편해보이긴 한다
앞에서해온 방법과 차이점이
있나? 지금까진 없는듯

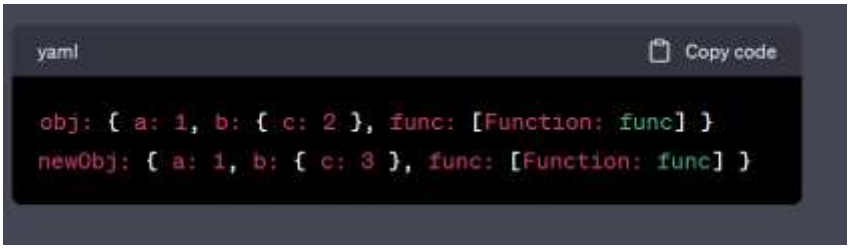
```
function deepCopy(obj) {
if (obj === null || typeof obj !== "object") {
return obj;
}
let copy = {};
for (let key in obj) {
copy[key] = deepCopy(obj[key]);
}
return copy;
}
const obj = {
a: 1,
b: {
c: 2,
},
func: function () {
return this.a;
},
};
const newObj = deepCopy(obj);
newObj.b.c = 3;
console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

```
▼ {a: 1, b: {…}, func: f}
    a: 1
  ▶ b: {c: 2}
  ▶ func: f ()
  ▶ [[Prototype]]: Object
false
undefined
```

```
// & npm i lodash 으로 설치
const lodash = require("lodash");
const obj = {
a: 1,
b: {
c: 2,
},
func: function () {
return this.a;
},
};
const newObj = lodash.cloneDeep(obj);
newObj.b.c = 3;
console.log(obj); // { a: 1, b: { c: 2 }, func: [Function: func] }
console.log(obj.b.c === newObj.b.c); // false
```

```
▶Uncaught ReferenceError: require is not defined          VM1215:2
    at <anonymous>:2:16
```

크롬에서 f12눌러서 치면
안나오니 ai의 힘을 빌려서
알아보았다



여기선 newObj도 출력해준다 c부분이
차이가 나는걸로보아 올바르게 나온듯
하다 cloneDeep메서드는
Lodash라이브러리에 포함된것으로
깊은복사를 해주는 함수라고 하는데
중첩된 객체나 배열내부의 것을
재귀적으로 복사하는 방법이라고
하는듯함

옛날에 C++독학하면서 깊은복사를 한것이다
쓰는 방법은 가물가물해도 어떤역할을
하는지는 기억한다

```
#include<iostream>

class deepcopy {
private:
int* data;
int size;
public:
// 생성자 : 배열 크기를 받아서 배열을 할당하고 초기화
deepcopy(int s) : size(s) {
data = new int[size];
for (int i = 0; i < size; i++)
{
data[i] = i;//배열원소에 0부터 size-1까지의 ㅏ값을 저장
}
}
//복사 생성자: 다른 객체를 복사해 새 객체를 생성
deepcopy(const deepcopy& other) :size(other.size) {
data = new int[size];
memcpy(data, other.data, size * sizeof(int));

}
~deepcopy()
{
delete[]data;
}
void printdata() {
for (int i = 0; i < size; i++)
{
std::cout << data[i] << " ";

}
std::cout << std::endl;
}
};
int main()
{
deepcopy original(5);
original.printdata();

deepcopy copy = original;
copy.printdata();
}
```

옛날에 C++독학하면서
깊은복사를 한것이다
쓰는 방법은 가물가물해도
어떤역할을 하는지는
기억한다

여기서 내가한것은
깊은복사로 그냥
깊은복사만 하였다 형이
만든 다른예시를 만들려면
원본을 따로 설정을
해줘야하는데
<-이쯤에 새로운 함수로
SetData(int index int value){
data[index] = value;
}
이렇게 하면 값을 바꿀수
있을듯
Main에다가
original.SetData(2,10)하면
원본만 값이 바뀔듯