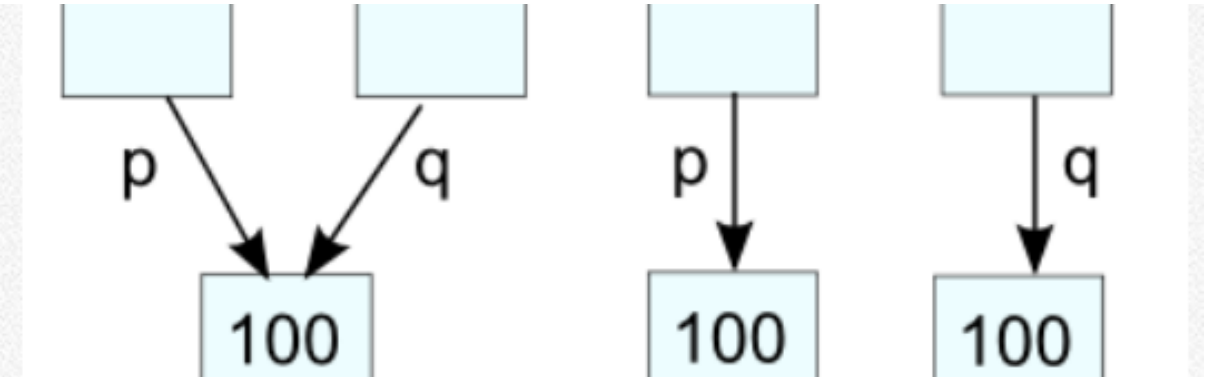


얕은복사 깊은복사

얕은복사 (Shallow copy)

- 객체의 속성 값들을 복사하지만, 객체 내부의 객체(중첩 객체)를 복사하지는 않는다. 복사된 객체와 원본 객체는 같은 객체를 참조하고 있기 때문에, 복사된 객체의 속성 값을 변경하면 원본 객체의 속성값도 함께 변경된다.
- 객체의 `Object.assign()` 메소드나 전개 연산자(`...`)를 사용하여 수행할 수 있다.

깊은복사 (deep copy)

- 객체나 배열 등 **복합 자료형**의 내부에 중첩된 객체들을 포함하여, 객체 전체를 복사하는 방법이다.
  깊은 복사는 내부 구조를 모두 복사하여, 복사된 객체와 원본 객체가 완전히 별개의 객체가 된다.

```
💡복합 자료형은 객체와 배열이다.
  복합 자료형은 객체와 배열을 이용하여 복잡한 데이터를 표현하고 처리할 수 있다.
```

- 단점 : 깊은 복사는 객체의 크기나 중첩된 객체의 수가 많아질수록 복사하는데 걸리는 시간이 오래 걸린다.
- 장점 : 복사한 객체와 원본 객체 간의 참조 관계가 끊어져서 객체를 수정할 때 오류가 발생할 가능성이 낮아진다.

<예시>

얕은복사엔 참조복사와 객체복사가 있다.

```
let me = {id:1,addr:{city:'seoul'}}
```

- **참조복사**

```
let you = me
```

참조복사는 주소값을 복사하기 때문에 메모리를 새로 만들어서 복사하는게 아닌 메모리 참조값자체를 복사한다.

위 메모리처럼 **you**코드는 **me**코드다 라고 이해하면된다.

그래서 **you**의 속성값을 수정하게되면 참조된 주소값이 같기때문에 같이 변경이 된다.

```
let me = {id:1,addr:{city:'seoul'}}
let you = me

you.id = 2

console.log(me.id, you.id) // 2, 2
//you의 id를 변경했지만 me의 id도 같이 2로 출력된다.
```

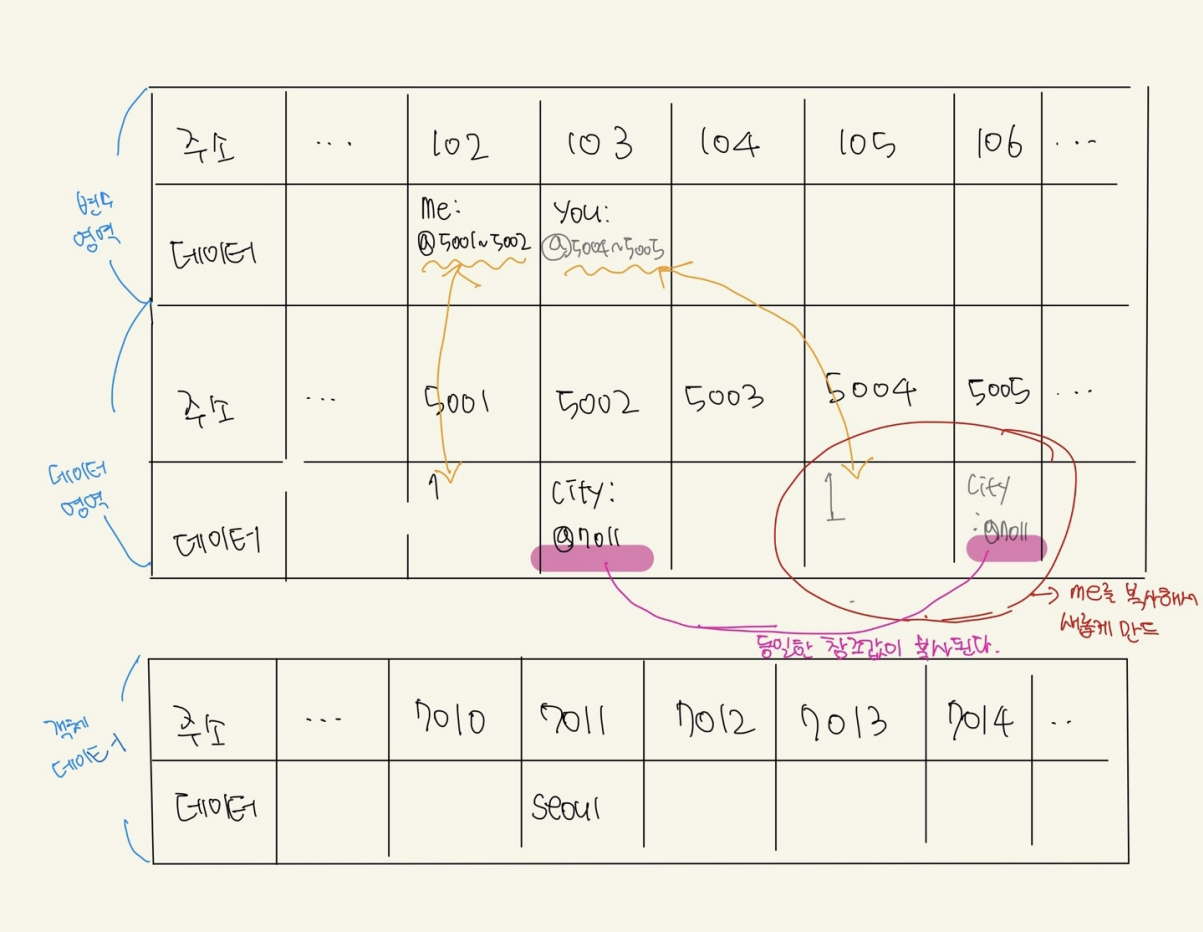
결론적으로 **you.id === me.id**는 같다.

동일하게 city의 속성값을 변경을 해주더라도 **me**와 **you**는 동일하게 변경된다.

- **객체복사**

```
let you = {...me}
```

전개연산자로 객체를 복사할 경우 원래 **me**에 있던 객체를 새로 펼쳐서 **you**에 복사한다.



객체복사는 **me**의 데이터의 참조값이 5001~5002지만 **you**는 **me**의 값을 복사해서 새롭게 만들기 때문에 **you**의 속성값이 변경이 되더라도 원본이 변경되지 않는다.

```
let me = {id:1,addr:{city:'seoul'}}
let you = {...me}

you.id = 2

console.log(me.id, you.id) //1, 2
```

여기에서 **you**의 **id**를 2로 변경했지만

you.id와 **me.id**는 참조하는 주소가 다르기 때문에 원본에 영향이 없는거다.

다만 **city**의 값을 변경하려고 할때는 어떻게 변경이 될까?

```
let me = { id: 1, addr: { city: 'seoul' } }
let you = { ...me }

you.addr.city = 'LA'

console.log(me, you)
//{ id: 1, addr: { city: 'LA' } } { id: 1, addr: { city: 'LA' } }
```

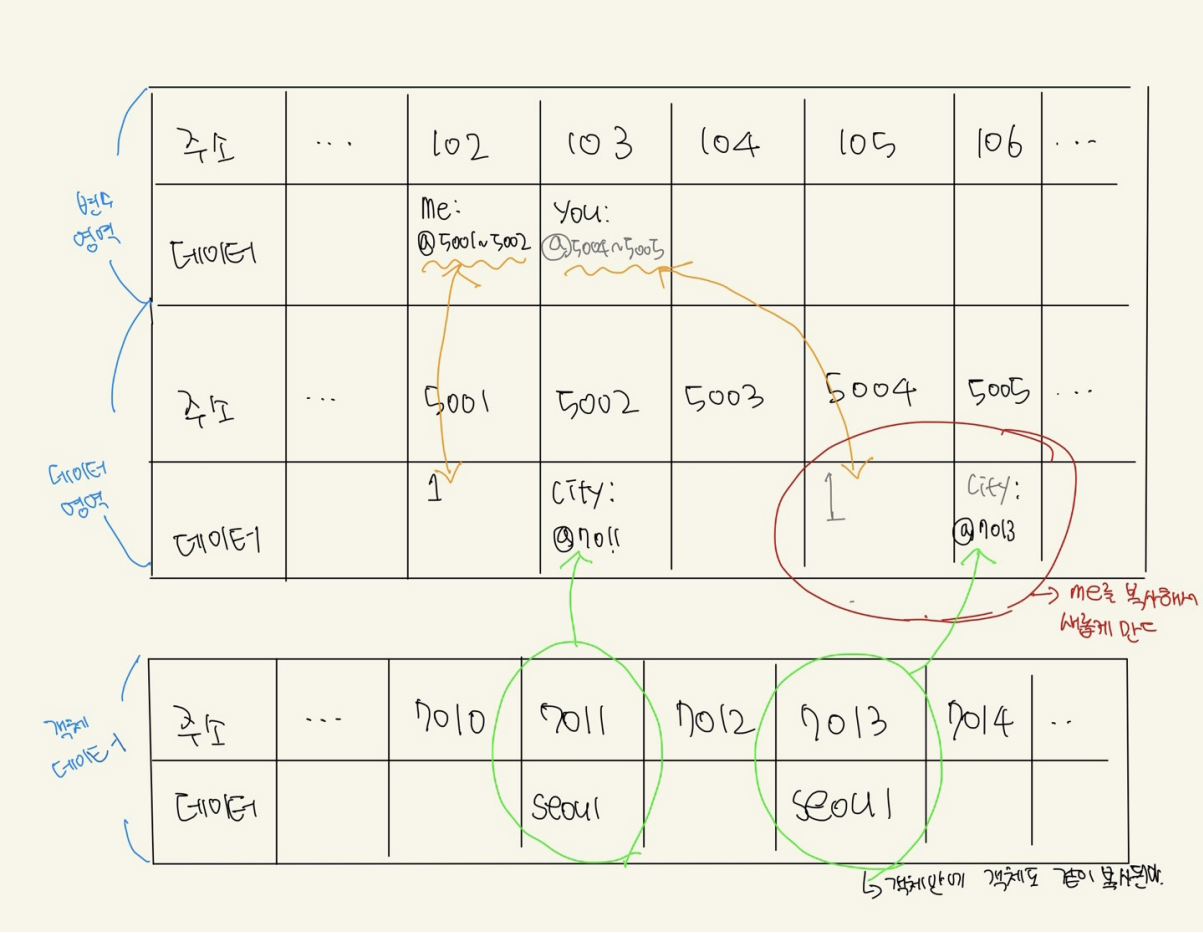
id값을 변경하려고 했을때와는 다르게 **city**의 속성값이 둘다 **'LA'**로 변경된걸 확인할 수 있다.

이유는 메모리의 참조하는 주소값을 확인해보면 알 수 있는데 얕은복사를 이용하면 **addr**안에 **city**의 속성값 주소는 7011그대로 복사가 되기 때문이다.

결론적으로 **you**에 **city**를 변경해도 **me**의 **city**도 변경되서 원본에 영향이 간다.

원본과 완전히 분리를 하기위해서는 얕은복사가 아닌 깊은복사가 나왔다.

깊은복사는 결론부터 얘기하자면 **city**의 참조값도 복사해서 새로 만드는거다.



위 이미지처럼 얕은복사에서는 **city**의 속성 주소값은 7011이였는데 객체의 데이터도 비어있는 7013주소에 넣어줘서 사용을 하게된다. 이렇게 되면 **city**도 새로운 주소값이 생기기 때문에 복사한 **you**의 데이터 값을 변경해도 원본에 영향이 없게 된다.

깊은복사를 하는 방법은 주로 사용하는 2가지가 있다.

1. JSON.stringify(), JSON.parse()

```
let me = { id: 1, addr: { city: 'seoul' } }

let stringYou = JSON.stringify(me)
//객체를 문자열로 바꿔준다. => {"id":1,"addr":{"city":"seoul"}}
let you = JSON.parse(stringYou)
//다시 객체형식으로 바꿔준다. =>{ id: 1, addr: { city: 'seoul' } }

you.addr.city = 'LA'

console.log(me, you)
// { id: 1, addr: { city: 'seoul' } } { id: 1, addr: { city: 'LA' } }
```

JSON.stringify() ⇒ 객체를 문자열로 바꿔준다.

JSON.parse() ⇒ 문자열로 바꼈던걸 객체형식으로 바꿔준다.

두가지를 메서드를 이용해서 깊은복사를 해주게되면 you의 city의 속성값을 변경해도 원본에는 영향이 없게된다.

2. lodash 라이브러리 사용

   lodash란?
   자바스크립트에서 사용할 수 있는 **유틸리티 라이브러리**이며, 기본 자바스크립트 함수의 기능을 확장하고, 개발자가 일상적으로 사용하는 유용한 함수들을 모아서 제공합니다.

   Lodash는 배열, 숫자, 문자열, 함수, 객체 등 다양한 자료형에 대한 작업을 수행할 수 있는 다양한 메서드를 제공합니다.

   필요한 메서드만 선택적으로 사용할 수 있고 Node.js와 브라우저 모두에서 사용할 수 있으며, NPM을 통해 쉽게 설치할 수 있습니다.

   ```
   💡유틸 라이브러리란?
     프로그래밍 언어나 프레임워크에서 제공하지 않는 기능을 보완하거나
     개발자가 자주 사용하는 유용한 함수들을 모아놓은 라이브러리
   ```

   ▼ rodash 라이브러리 설치방법

   Node.js환경에서 터미널을 통해 라이브러리를 설치한다.

   ```
   npm install lodash
   ```

   ▼ 라이브러리 설치도중 에러

   라이브러리를 설치도중에 에러가 발생할 수 있다.

   ```
   npm ERR! code EACCES
   npm ERR! syscall link
   npm ERR! path /Users/choeyuli/.npm/_cacache/tmp/e4ef943e
   npm ERR! dest /Users/choeyuli/.npm/_cacache/content-v2/sha512/55/14/0d5b47eaa57ad9294380191a44a9fe513521f70fdbe850ed47eb051b8
   npm ERR! errno EACCES
   npm ERR!
   npm ERR! Your cache folder contains root-owned files, due to a bug in
   npm ERR! previous versions of npm which has since been addressed.
   npm ERR!
   npm ERR! To permanently fix this problem, please run:
   npm ERR!   sudo chown -R 501:20 "/Users/choeyuli/.npm"

   npm ERR! A complete log of this run can be found in:
   npm ERR!     /Users/choeyuli/.npm/_logs/2023-04-13T15_32_47_366Z-debug-0.log
   ```

   이 에러는 npm이 특정 경로에 권한이 없어서 발생하는 문제이며

   ```
   sudo chown -R 501:20 "/Users/choeyuli/.npm"
   ```

   경로에 있는 모든 파일과 폴더의 소유자를 현재 사용자(501)로 변경하고, 그룹을 staff(20)으로 변경한다.

   이렇게 하면 이후에 npm에서 발생할 수 있는 권한 관련 문제를 방지할 수 있다.

   이후에는 lodash를 설치할 때 다시 에러가 발생하지 않는다.

lodash를 이용한 깊은복사 방법

cloneDeep 메소드를 사용하여 객체나 배열을 재귀적으로 순회하며, 해당 객체나 배열의 모든 요소들을 새로운 객체나 배열로 복제할 수 있고, 이를통해 깊은 복사를 수행할 수 있다.

```
const _ = require('lodash');
//Lodash함수를 호출하기전에 해당 라이브러리를 import하는 의미

let me = { id: 1, addr: { city: 'seoul' } }
let you = _.cloneDeep(me);

you.addr.city = 'LA'

console.log(me, you)
//{ id: 1, addr: { city: 'seoul' } } { id: 1, addr: { city: 'LA' } }
```