KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

Vue.js를 위한 ES6 문법

[KB] IT's Your Life

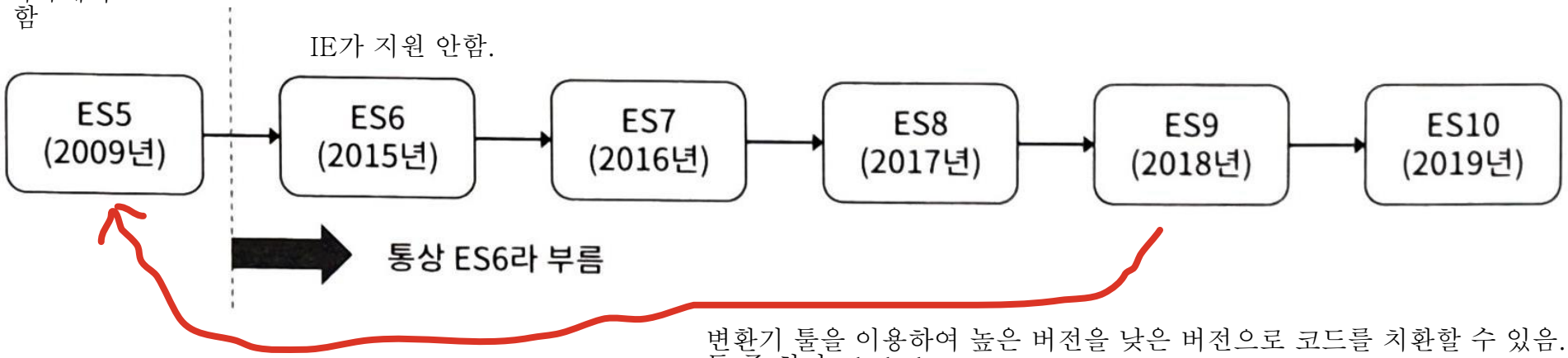
KB 국민은행

ES6(ECMAScript6)

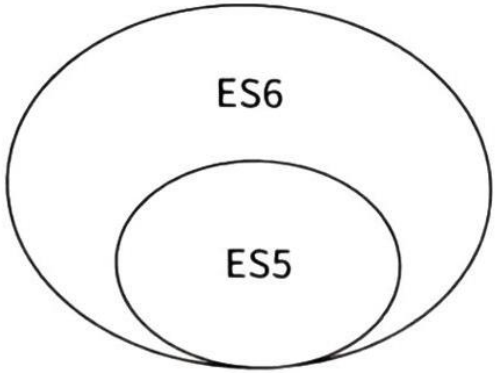
코드를 모듈화하여 배포할 때 고민해야할 것. 어떤 것을 표준으로 잡아야 하냐,,,

보통 scfript언어는 유럽
compile언어는 미국에서
관리, 표준 정립 함

보편적으로 맞추는 표준 es6 (2015년도에 발표된 표준안)



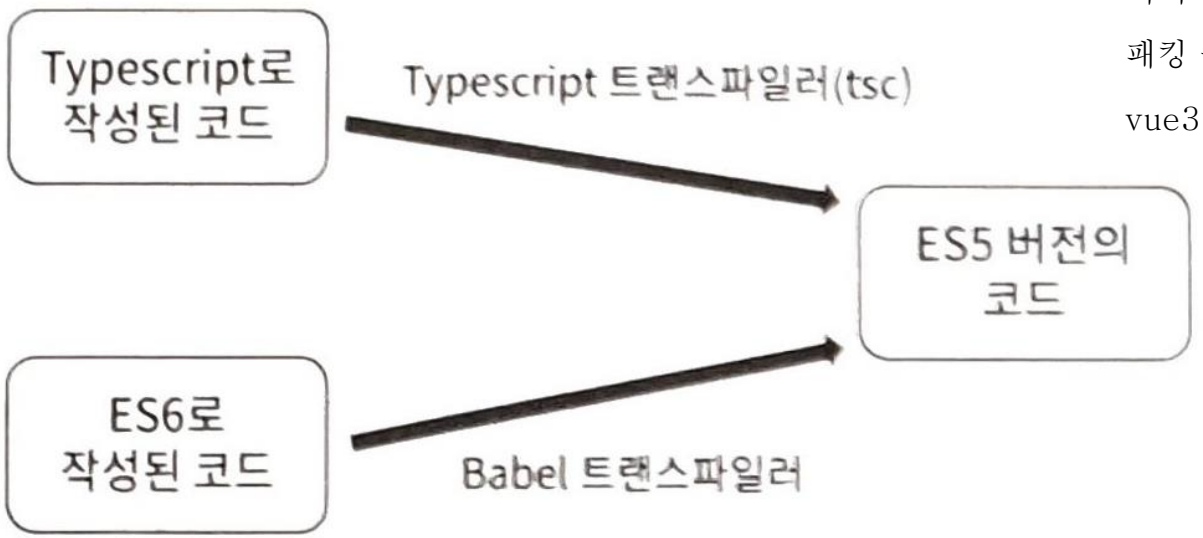
변환기 툴을 이용하여 높은 버전을 낮은 버전으로 코드를 치환할 수 있음.
툴 중 하나 : babel



ES6는 이전 버전의 문법을 포함하여 지원하면서
Class, Arrow Function 등 새로운 문법을 추가로 지원함

ES6 미지원 브라우저에 대한 대처

- 트랜스파일러(Transpiler)를 이용해서 ES6 코드를 ES5 코드로 변환
- babel, tsc(typescript 트래스파일러) 등
- vue.js 개발 환경에서 자동으로 처리 해줌



브라우저 마다
모듈 시스템도 다를 수 있음

=>

모듈 시스템을 써서 작성하고

=>
배포할 때는
모듈 시스템을 안쓰는 비모듈 코드(통합코드)로
패킹하여 배포!

여러의 코드덩어리들을 하나로 통합

패킹 툴 webpack.

vue3에서는 vite라는 툴을 권장

정리 브라우저마다 모듈 시스템이 다를 수도 있고 어떤 표준까지 따르는지 몰라. 배포는 해야돼...

모듈 시스템 이용하고 es6 이상 표준으로 개발 => 패킹 툴 vite를 사용해서 통합 => 낮은 표준으로 변환 툴 사용 => 배포

var 변수의 호이스팅

어차피 var 사용 안하는데

그냥 VAR 사용 금지

- 자바스크립트가 실행될 때 먼저 선언문을 먼저 해석 후 실행
- 변수의 선언문이 사용하는 문장 후에 있어도 해당 변수를 인식

스코프 내에서 진입시
변수 선언,함수 선언 먼저
undefined로 초기화 하고

코드 진행

```
console.log(A1);          // undefined
var A1 = "hello";
```

- 함수 단위로 호이스팅
  - 블록 단위로 동작하지 않음
  - 진정한 지역변수가 아님

그냥 var쓰지마

- 변수 선언을 여러 번 할 수 있음 (에러가 아님)

```
var V1 = 100;
console.log(V1);      // 100
var V1 = "hello";
console.log(V1);      // hello
```

let 변수

- ES6에서 도입한 블록 단위의 변수 선언
  - 유효 범위가 블록 단위임
- 호이스팅 지원하지 않음
  - 선언하지 않고 사용하면 에러 발생

다른언어의 지역변수의 유효범위와 같다

02-02.js

```
let msg = "GLOBAL";
function outer() {
    let msg = "OUTER";
    console.log(msg);
    if (true) {
        let msg = "BLOCK";
        console.log(msg);
    }
}
outer()
```

```
OUTER
BLOCK
```

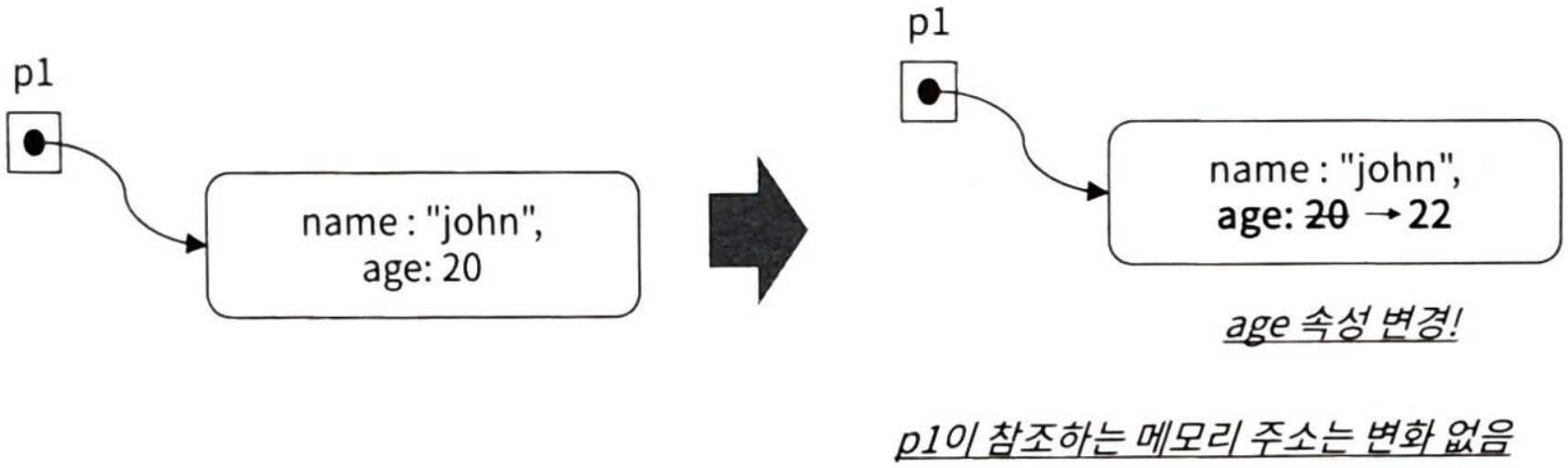
const

- 값 변경 불가 → 상수 선언
- 범위(scope)는 let과 동일
- 참조 변수인 경우 참조는 변경 불가
  - 필드는 수정 가능

02-03.js

```
const p1 = { name: 'john', age: 20 };
p1.age = 22;
console.log(p1);

p1 = { name: 'lee', age: 25 };    안돼 이건
```

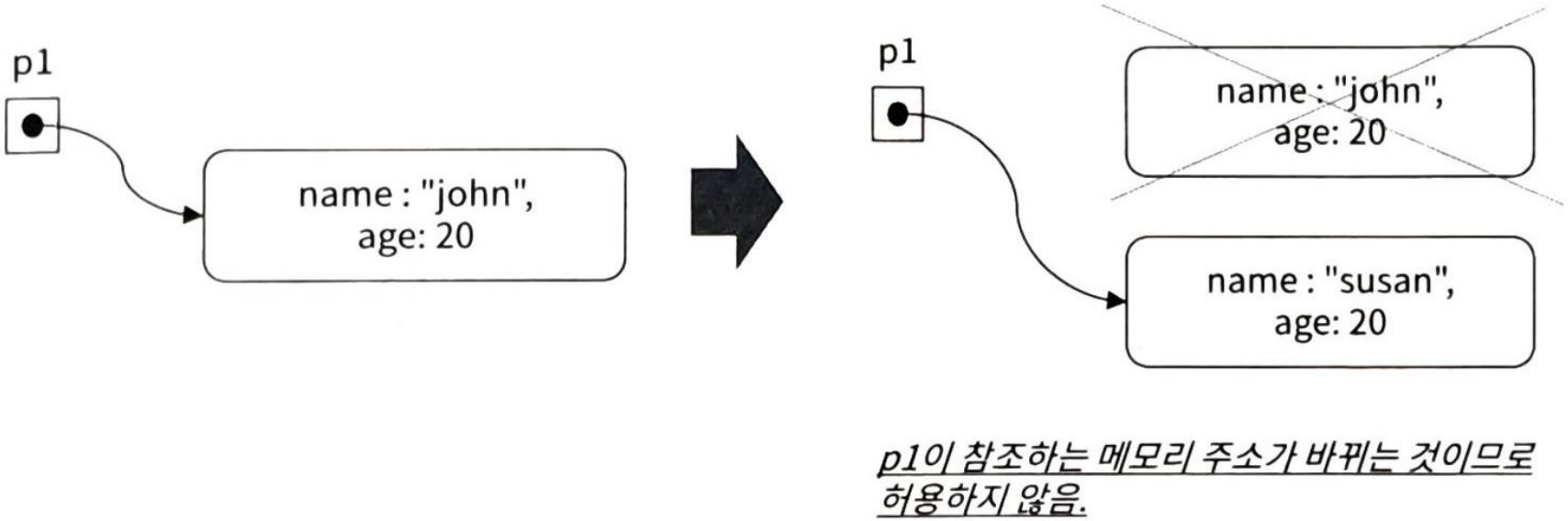
```
{ name: 'john', age: 22 }
```



02-04.js

```
const p1 = { name : "john", age : 20 }
p1 = { name:"susan", age: 20 };

console.log(p1);
```

type error가 난다



p1이 참조하는 메모리 주소가 바뀌는 것이므로 허용하지 않음.

기본 파라미터 Default Parameter

- 함수 호출 시 인수를 생략했을 때 가지는 기본값을 지정

  function func(arg1, arg2 = 기본값, arg3 = 기본값, ...) { ... }

- 반드시 뒤부분에 지정
  - 중간에만 기본값 지정하는 것은 불가능

다른 언어처럼 뒷부분부터 디폴트값 지정

02-05.js

```
function addContact(name, mobile, home="없음", address="없음", email="없음") {
    let str = `name=${name}, mobile=${mobile}, home=${home},` +
              ` address=${address}, email=${email}`;
    console.log(str);
}

addContact("홍길동", "010-222-3331")
addContact("이몽룡", "010-222-3331", "02-3422-9900", "서울시");
```

```
name=홍길동, mobile=010-222-3331, home=없음, address=없음, email=없음
name=이몽룡, mobile=010-222-3331, home=02-3422-9900, address=서울시, email=없음
```

가변 파라미터 Rest Paramter

- 전달하는 파라미터의 개수를 가변적으로 적용
- 매개변수 앞에 ... 연산자 지정
  - 전달된 인수를 매개변수로 매칭한 후,
  - 매칭되지 않은 인수를 모아 배열로 지정 → 가변 파라미터로 전달

```
function func(arg1, arg2, ...argv) { ... }
```

- 1개만 지정가능, 마지막 매개변수에만 지정가능

02-06.js

rest연산자로고도 불림.
매개변수 매칭하고 남은것들 모아놓는 곳

```
function foodReport(name, age, ...favoriteFoods) {
    console.log(name + ", " + age);
    console.log(favoriteFoods);
}

foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");
foodReport("홍길동", 16, "초밥");
```

나머지 인자값들을 배열 형식으로 저장함

```
이몽룡, 20
[ '짜장면', '냉면', '불고기' ]
홍길동, 16
[ '초밥' ]
```

구조분해 할당 Destructing Assignment

- 배열, 객체의 값들을 추출하여 한 번에 여러 변수에 할당할 수 있는 기능

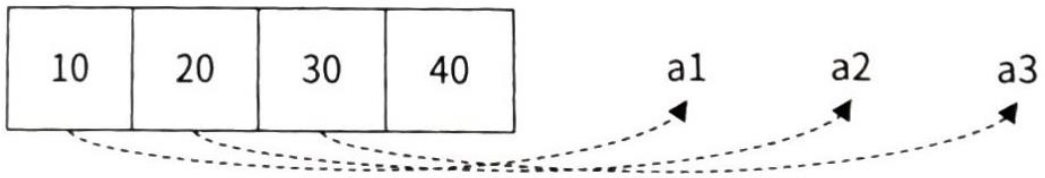
번외 js에서 중요한 prototype 개념
js는 원래 class라는 개념이 없었음.

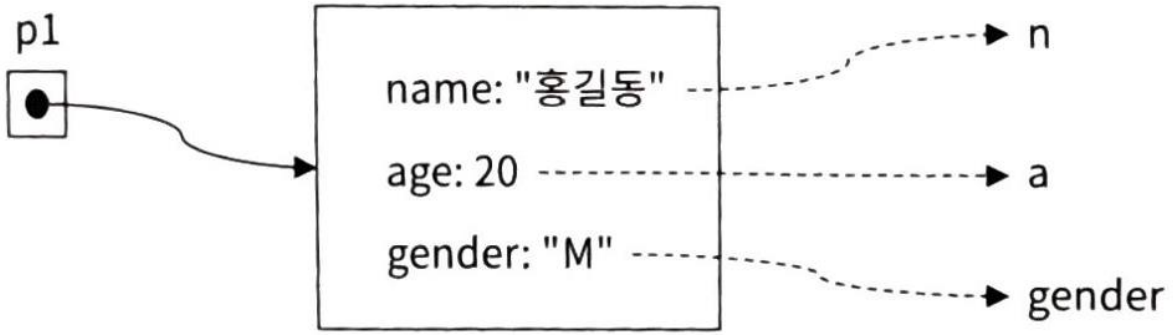
02-07.js

```
let arr = [10,20,30,40];
let [a1,a2,a3] = arr;
console.log(a1, a2, a3);

let p1 = { name:"홍길동", age:20, gender:"M" };
let { name: n, age:a, gender } = p1;
console.log(n, a, gender);
```

```
10 20 30
홍길동 20 M
```

name 값을 받지만
n변수명을 사용하겠다는 뜻





02-08.js

매개변수도 같은 형식

```
function addContact1({name, phone, email="이메일없음", age=0}) {
    console.log(name,phone,email,age);
}
addContact1({ name : "이몽룡", phone : "010-3434-8989" })

function addContact2(contact) {
    if (!contact.email) contact.email = "이메일없음";
    if (!contact.age) contact.age = 0;
    let { name, phone, email, age} = contact;
    console.log(name,phone,email,age);
}
addContact2({ name : "이몽룡", phone : "010-3434-8989" })

function addContact3(name,phone,email="이메일없음",age=0) {
    console.log(name,phone,email,age);
}
addContact3("이몽룡","010-3434-8989")
```

매개변수를 받을 때 분해 할 당

키워드 인수 전달 방식

```
이몽룡 010-3434-8989 이메일없음 0
이몽룡 010-3434-8989 이메일없음 0
이몽룡 010-3434-8989 이메일없음 0
```

화살표 함수의 형식

- 기존 함수 표현식 간결화
- 함수의 본체가 한 줄 문장인 경우
  - { } 생략
  - 한 줄을 표현식으로 해석하여 자동 리턴(return 생략)

02-09.js

```
const test1 = function(a,b) {
    return a+b;
}

const test2 = (a,b) =>{
    return a+b;
};

const test3 = (a,b) => a+b;

console.log(test1(3,4));
console.log(test2(3,4));
console.log(test3(3,4));
```

```
7
7
7
```

자바스크립트의 this

- 메서드, 함수가 호출될 때마다 현재 호출중인 메서드를 보유한 객체가 this로 연결
- 현재 호출 중인 메서드를 보유한 객체 없다면 전역 객체가 연결
  - 전역 객체
    - 브라우저에서는 window 객체

02-10.js

```
let obj = { result: 0 };
obj.add = function(x,y) {
    this.result = x+y;
}
obj.add(3,4)
console.log(obj)
```

{ result: 7, add: [Function (anonymous)] }

02-11.js

```
let obj = { result: 0 };

obj.add = function(x,y) {
   this.result = x+y;
}

let add2 = obj.add;

console.log(add2 === obj.add)      //true, 동일한 함수

add2(3,4)              //전역변수 result에 7이 할당됨.
console.log(obj);      //{ result: 0 }
console.log(result);   //7
```

참조가 같으니까!

전역객체.add2(3,4)가 되어버려. 일반 함수 형식은 this가 전역객체

```
true
{ result: 0, add: [Function (anonymous)] }
7
```

자바스크립트의 this 지정

- 함수나 메서드를 호출할 때 this를 변경할 수 있음
- bind()
  - 지정한 객체를 this로 미리 연결한(binding) 새로운 함수를 리턴

한번 결정후 계쏙됨

- apply(), call()
  - 지정한 객체를 this로 연결한 후 함수를 직접 호출

한번만 바꾸는 일회용.

둘의 차이점은
매개변수의 형식, 가변이냐 배열이냐 차이

둘의 기능은 똑같음

02-12.js

```
let add = function(x,y) {
    this.result = x+y;
}

let obj = {};
//1. apply() 사용
//add.apply(obj, [3,4])
//2. call() 사용
//add.call(obj,3,4)
//3. bind() 사용

add = add.bind(obj);
add(3,4)

console.log(obj);       // { result : 7 }
```

전달된 객체 고정

```
{ result: 7 }
```

- **화살표 함수에서의 this**
  - 함수를 정의하는 영역의 this를 그대로 전달

02-13.js

```
let obj = { result:0 };

obj.add = function(x,y) {
  function inner() {
     this.result = x+y;  inner내부함수는 일반 함수다. 메서드가 아니다!
  }
  inner();
}

obj.add(3,4)

console.log(obj)         // { result: 0 }
console.log(result)      // 7
```

window

02-14.js

```
let obj = { result:0 };

obj.add = function(x,y) {
  function inner() {
     this.result = x+y;
  }
  inner = inner.bind(this);
  inner();
}
obj.add(3,4)

console.log(obj)        // { result: 7 }
```

02-15.js

```
let obj = { result:0 };
obj.add = function(x,y) {
  const inner = () => {
     this.result = x+y;
  }
  inner()
}
obj.add(3,4)
console.log(obj)        // { result: 7 }
```

화살표 함수안에서의 this는 바깥 함수의 this를 상속받는다!!

호출될 때 obj객체가 호출하므로 add함수 안에 있는 inner함수는 add함수의 this를 상속받아 실행되므로 obj가 타겟이다

객체 리터럴

기존 객체 리터럴

```
let name = "홍길동";
let age = 20;
let obj = {
  name: name,
  age: age
};
```

ES6 객체 리터럴

- 변수명이 객체의 속성명과 일치하는 경우 변수명 제시 가능

```
let name = "홍길동";
let age = 20;
let obj = { name,  age };
```

메서드인 경우 function 키워드 없이 바로 선언

02-16.js

```
let name = "홍길동";
let age = 20;
let email = "gdhong@test.com";

//let obj = { name: name, age: age, email: email };
let obj = { name, age, email };
console.log(obj);
```

```
{ name: '홍길동', age: 20, email: 'gdhong@test.com' }
```

02-17.js

```
let p1 = {
    name : "아이패드",
    price : 200000,
    quantity : 2,
    order : function() {   // 기존 방법
        if (!this.amount) {
            this.amount = this.quantity * this.price;
        }
        console.log("주문금액 : " + this.amount);
    },
    discount(rate) { // ES6의 메서드 선언
        if (rate > 0 && rate < 0.8) {
            this.amount = (1-rate) * this.price * this.quantity;
        }
        console.log((100*rate) + "% 할인된 금액으로 구매합니다.");
    }
}
p1.discount(0.2);
p1.order();
```

function 키워드 안씀

```
20% 할인된 금액으로 구매합니다.
주문금액 : 320000
```

템플릿 리터럴(Template Literal)

- ``로 묶여진 문자열에서 템플릿 대입문(${표현식})을 이용해 동적으로 문자열을 끼워 넣어 구성하는 법
- 개행 문자를 포함하여 여러 줄로 작성 가능

02-18.js

```
const d1 = new Date();
let name = "홍길동";
let r1 = `${name} 님에게 ${d1.toDateString() }에 연락했다.`;
console.log(r1);

let product = "갤럭시S7";
let price = 199000;
let str = `${product}의 가격은
        ${price}원 입니다.`;
console.log(str);
```

```
홍길동 님에게 Mon Jan 15 2024에 연락했다.
갤럭시S7의 가격은
        199000원 입니다.
```

ES6 모듈

- 모듈
  - JS 코드를 포함하고 있는 파일
- import, export 구문을 이용해서 모듈을 가져오거나, 내보낼 수 있음
- 모듈 내부에서 선언된 모든 변수, 함수, 객체, 클래스는 지역적인(local) 것으로 간주
- 재사용 가능한 모듈을 만들려면 반드시 외부에 공개하는 것을 export 해야 함

```
export let a = 1000;
export function f1(a) { ... }
export { n1, n2 as othername, ...}        // as를 사용해서 기존 이름(n2) 대신 다른 이름(othername)으로 사용
```

- export된 모듈은 다른 모듈에서 import 구문으로 참조하여 사용

npm init -y

package.json

```
{
  …
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  …
}
```

modules/02-19-module.js

```
let base = 100;
const add = (x) => base+x;
const multiply = (x) => base*x;

export { add, multiply };
```

```
let base = 100;
export const add = (x) => base+x;
export const multiply = (x) => base*x;
```

02-20-main.js

```
import { add } from './modules/02-19-module.js';

console.log(add(4));
```

02-19-module.js

```
let base = 100;
const add = (x) => base+x;
const multiply = (x) => base*x;
const getBase = ()=>base;

export { add, multiply };
export default getBase;
```

```
let base = 100;
export const add = (x) => base+x;
export const multiply = (x) => base*x;

export default () => base;
```

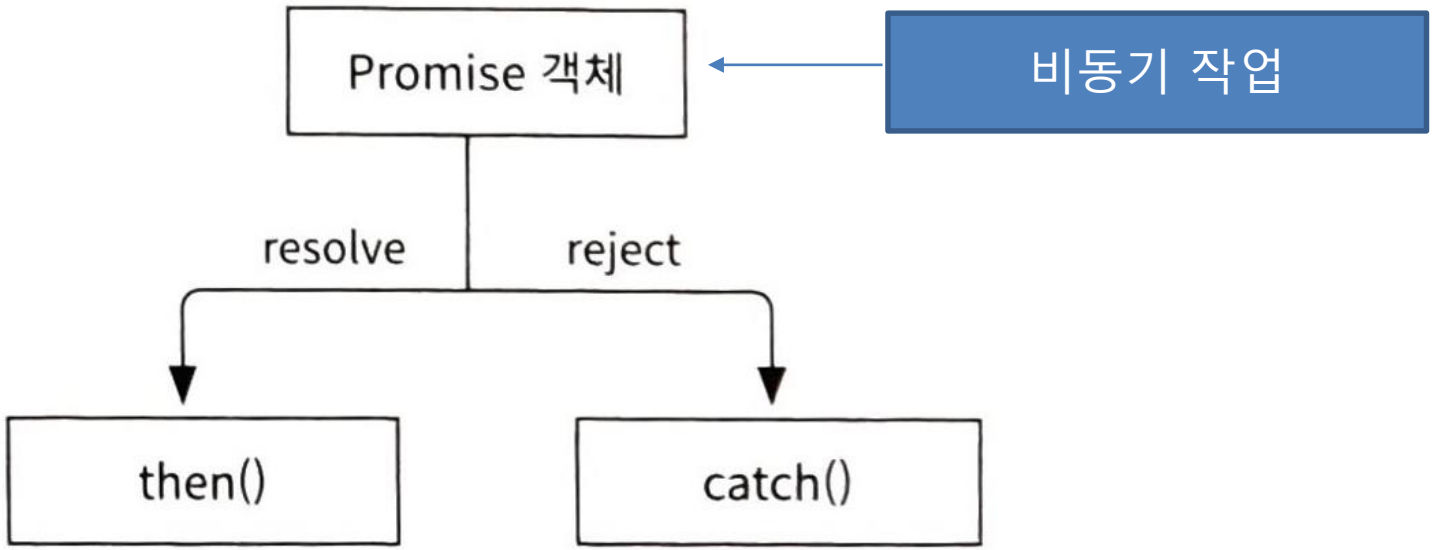
02-20-main.js

```
import getBase, { add } from './modules/02-19-module.js';

console.log(add(4));
console.log(getBase());
```

Promise 객체

- 비동기 처리 지원
- 콜백 함수
  - 비동기 작업의 순차 처리 시 콜백 함수의 중첩 문제 발생

- Promise 패턴



02-21.js Promise 객체

```
const p = new Promise((resolve, reject) => {
    setTimeout(()=> {
        let num = Math.random();        //0~1사이의 난수 발생
        if (num >= 0.8) {
            reject("생성된 숫자가 0.8이상임 - " + num);
        }
        resolve(num);
    }, 2000)
})

p.then((result)=> {
    console.log("처리 결과 : ", result)
})
.catch((error)=>{
    console.log("오류 : ", error)
})

console.log("## Promise 객체 생성!");
```

```
## Promise 객체 생성!
처리 결과 :  0.32435778048671526

## Promise 객체 생성!
오류 :  생성된 숫자가 0.8이상임 - 0.811963599046789
```

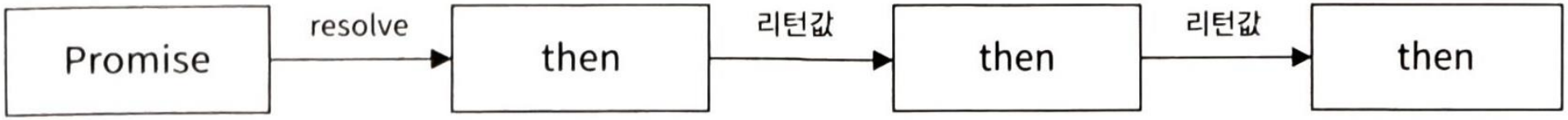
02-22.js Promise 체인 – 비동기 작업의 순차 실행

```
let p = new Promise((resolve, reject)=> {
    resolve("first!")
})

p.then((msg)=> {
    console.log(msg);
    // throw new Error("## 에러!!")
    return "second";
})
.then((msg)=>{
    console.log(msg);
    return "third";
})
.then((msg)=>{
    console.log(msg);
})
.catch((error)=> {
    console.log("오류 발생 ==>  " + error)
})
```

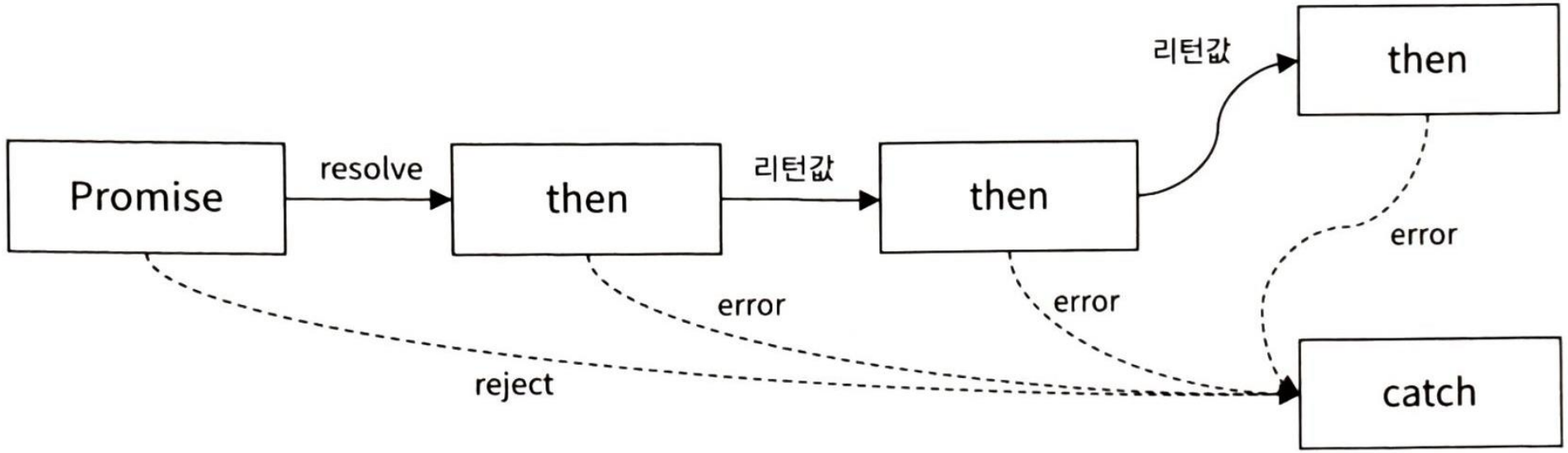
```
first!
second
third


first!
오류 발생 ==>  Error: ## 에러!!
```

Promise 체인

정상 처리 시



오류 발생 시



가변 매개변수 선언때 봤던 친구죠?

전개 연산자 spread operator

- ...연산자
- 배열이나 객체를 ...연산자와 함께 객체 또는 배열에서 사용
  → 객체, 배열 내의 값을 분해된 값으로 전달
- 기존 값을 복사하거나, 기존 값을 복사하면서 새로운 항목 추가시 사용
  - let obj1 = { ... }
  - let obj2 = obj1
  - let obj3 = { ...obj1 }
  - let obj4 = { ...obj1, color }

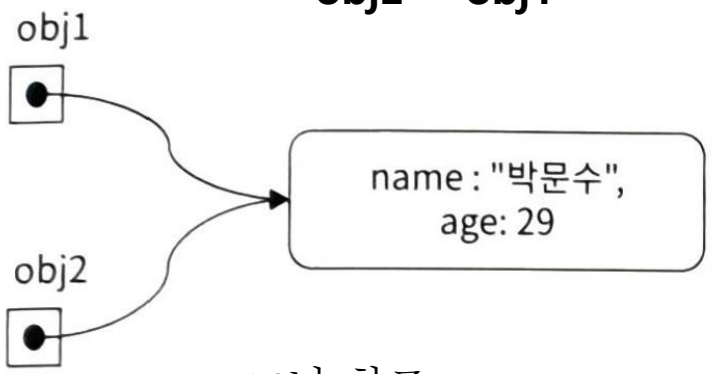
그렇다면 깊은 복사는??
npm i lodash라는 모듈을 설치하고
깊은 복사를 하는 함수를
사용해라.
아님 구현하0던가
deepClone이라는 모듈 함수

복사하고 새로운 속성 추가

혹은 기존 속성값 업데이트

참조!

obj2 = obj1

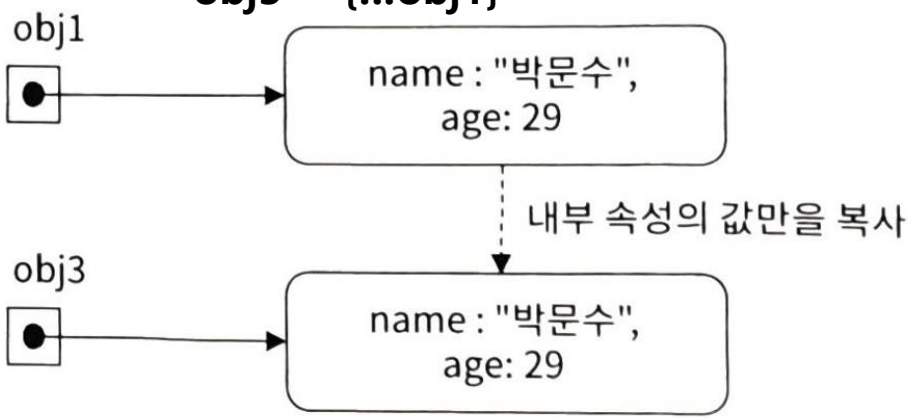


그냥 참조

얕은 복사(Shallow Copy)

obj1의 모든 요소 복사
근데 이거 얕은 복사다!

obj3 = {...obj1}



전개 연산자를 사용

02-23.js

```
let obj1 = { name:"박문수", age:29 };
let obj2 = obj1;          //shallow copy! obj1, obj2는 동일한 객체를 참조
let obj3 = { ...obj1 };    //객체 내부의 값은 복사하지만 obj3, obj1은 다른 객체
let obj4 = { ...obj1, email:"mspark@gmail.com" };  //새로운 속성 추가

obj2.age = 19;
console.log(obj1);  //{ name:"박문수", age:19 }
console.log(obj2);  //{ name:"박문수", age:19 }   obj1과 동일!!
console.log(obj3);  //{ name:"박문수", age:29 }   age가 바뀌지 않음
console.log(obj1 == obj2); //true
console.log(obj1 == obj3); //false
```

객체 말고도
배열에서도 똑같이 적용한다.

```
let arr1 = [ 100, 200, 300 ];
let arr2 = [ "hello", ...arr1, "world"];
console.log(arr1);      // [ 100, 200, 300 ]
console.log(arr2);      // [ "hello", 100, 200, 300, "world" ]
```

```
{ name: '박문수', age: 19 }
{ name: '박문수', age: 19 }
{ name: '박문수', age: 29 }
true
false
[ 100, 200, 300 ]
[ 'hello', 100, 200, 300, 'world' ]
```

11 Proxy

사전적 의미 : 대리인, 대신해준다. 외부에서 봤을 때 proxy가 감싸고 잇는 객체는 그냥 객체인줄 알고 사용하는 것.
proxy는 해당 객체를 감싸며 해당객체의 메소드가 호출 될때마다 사전작업들을 할 수 잇따. 물론 외부에선 모름.

Proxy

== 투명성
vue에서는 앞서 말한 사전작업이 모니터링후 변경값을 화면에 적용하는 일이다.

- 객체의 속성을 읽어오거나 설정하는 작업을 가로채기 위해 래핑할 수 있도록 하는 객체
- 객체의 속성에 접근할 때 개발자가 원하는 지정된 작업을 수행하도록 할 수 있음
- Vue 내부에서Proxy를 사용
  - 개발자가 직접 개발하지 않음
- 반응성(reactivity) 지원을 위한 기법

모델데이터 객체가 수정되면 화면이 수정된다.
how?
proxy 객체가 모니터링함.

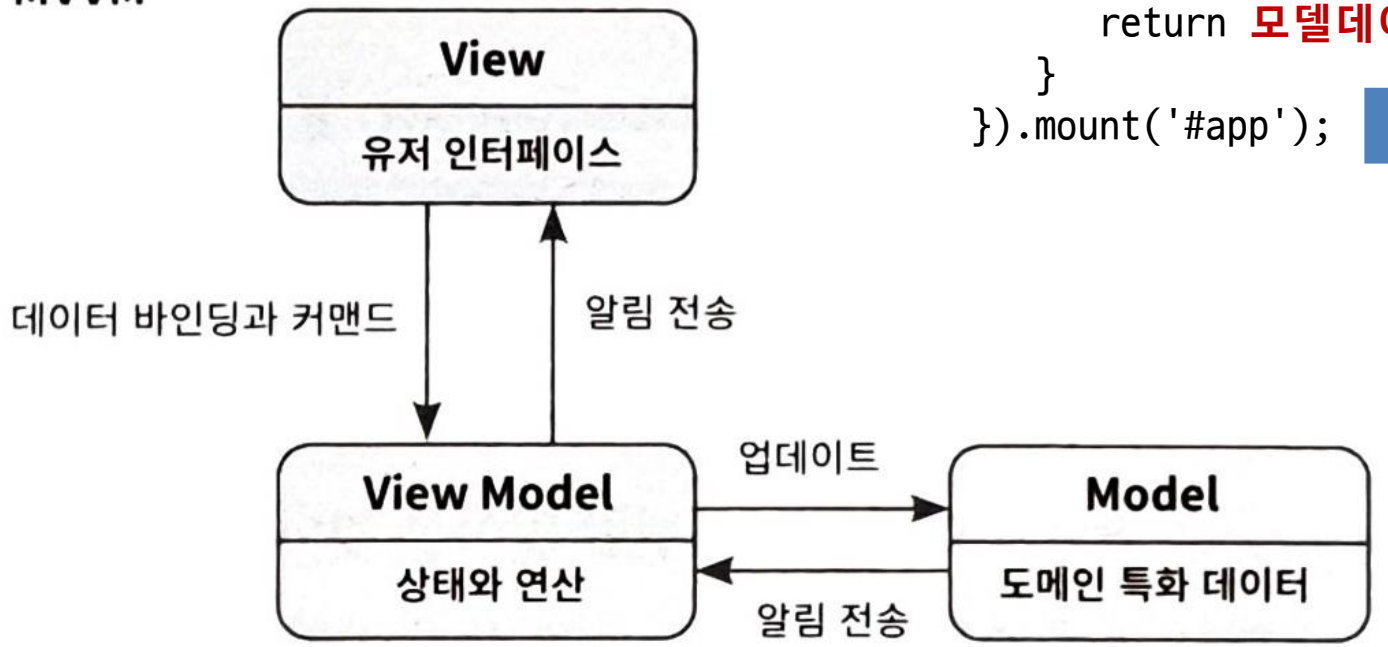
프레임웍마다 방식이 약간 다를 수 있다.

vue에선 proxy객체가.

```
let vm = Vue.createApp({
    name: "App",
    data() {
        return 모델데이터객체;
    }
}).mount('#app');
```

Proxy 객체로 래핑됨

MVVM



js proxy지원한느 방식

02-24.js Proxy로 객체 래핑하기

```
let obj = { name : "홍길동", age :20 };
const proxy = new Proxy(obj, {//첫 매개변수가 original 객체(래핑할 객체)
    get: function(target, key) {
        console.log("## get " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
        return target[key];
    },
    set : function(target, key, value) {
        console.log("## set " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
        target[key] = value;
        return true
    }
})

console.log(proxy.name);        //읽기 작업 get 호출
proxy.name = "이몽룡";          //쓰기 작업 set 호출
proxy.age = 30;                 //쓰기 작업 set 호출

console.log(proxy);
```

original 객체

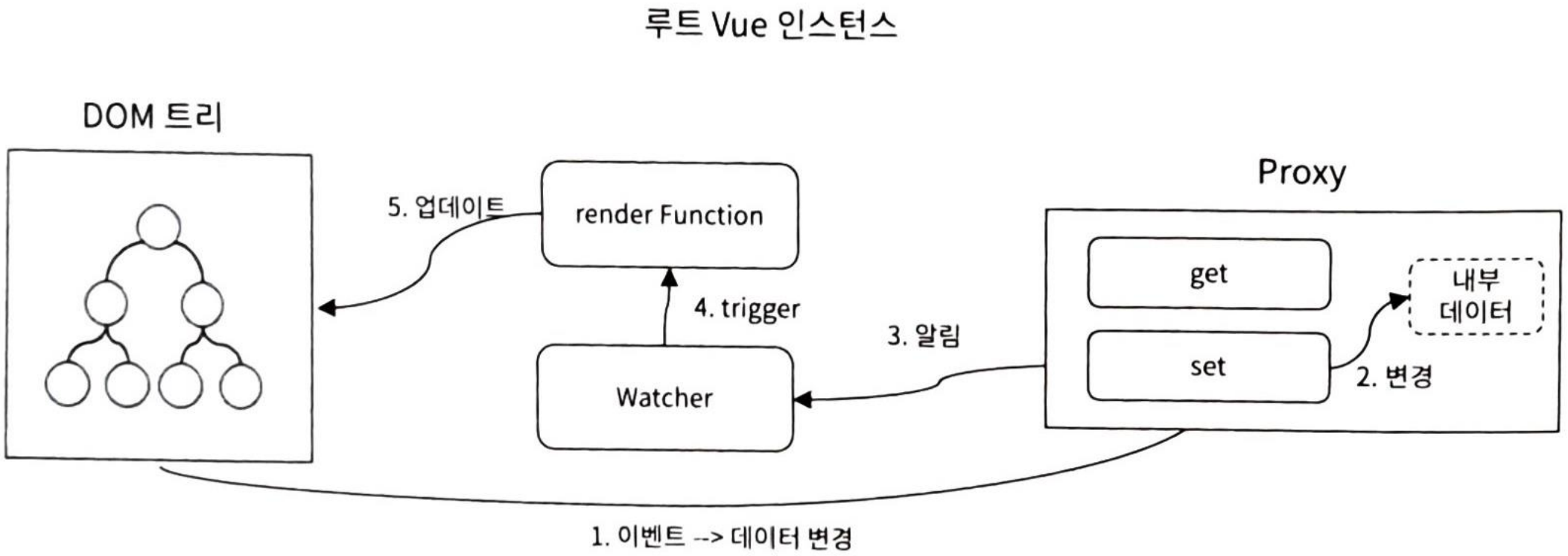
get메서드 (getter) : 대입문 진행시 호출됨. 대입문 오른쪽일때 겠쬬?

set메서드 (setter) : 대입문 진행시 호출됨. 대입문 왼쪽에 있을 때겠죠

vuejs의 proxy객체는 데이터 수정후 화면 갱신하는 코드가 있다.

```
## get name
홍길동
## set name
## set age
{ name: '이몽룡', age: 30 }
```

Vue3의 반응성


루트 Vue 인스턴스
DOM 트리
5. 업데이트
render Function
4. trigger
Watcher
3. 알림
Proxy
get
set
내부 데이터
2. 변경
1. 이벤트 --> 데이터 변경

02-25.js Proxy로 배열 래핑하기

```
let arr = [10,20,30];

const proxy = new Proxy(arr, {
    get: function(target, key, receiver) {
        console.log("## get " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
        return target[key];
    },
    set : function(target, key, value) {
        console.log("## set " + key)
        if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
        target[key] = value;
    }
})

proxy[1] = 99;
//proxy[4] = 100;     //오류발생
```