

원쌤의 Vue.js 퀵스타트

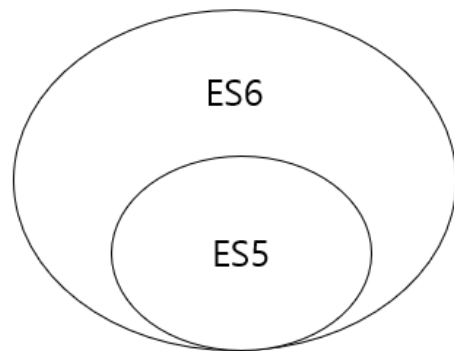
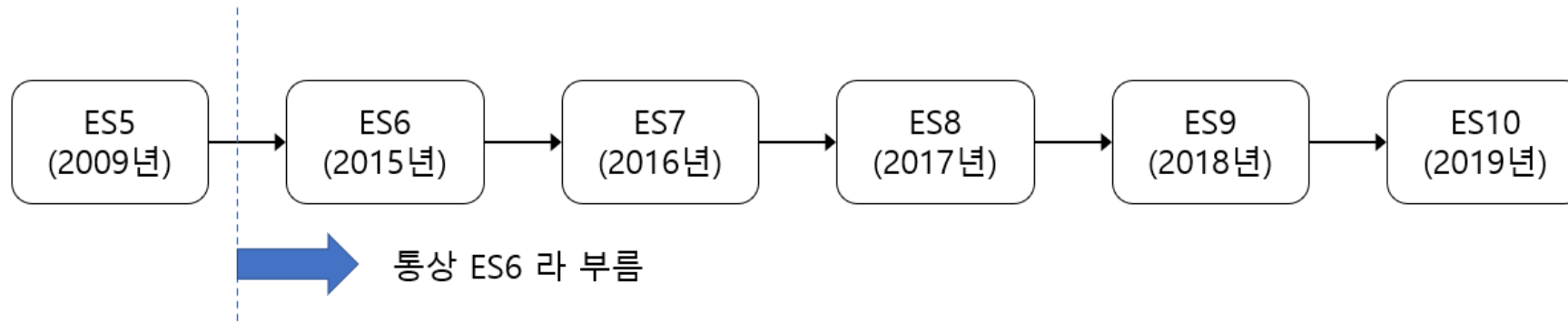
2. Vue.js를 위한 ES6



1. ES6 소개

❖ ES6

- ECMAScript 6
- ECMA-262 기술 규격에 정의된 표준화된 스크립트 프로그래밍 언어

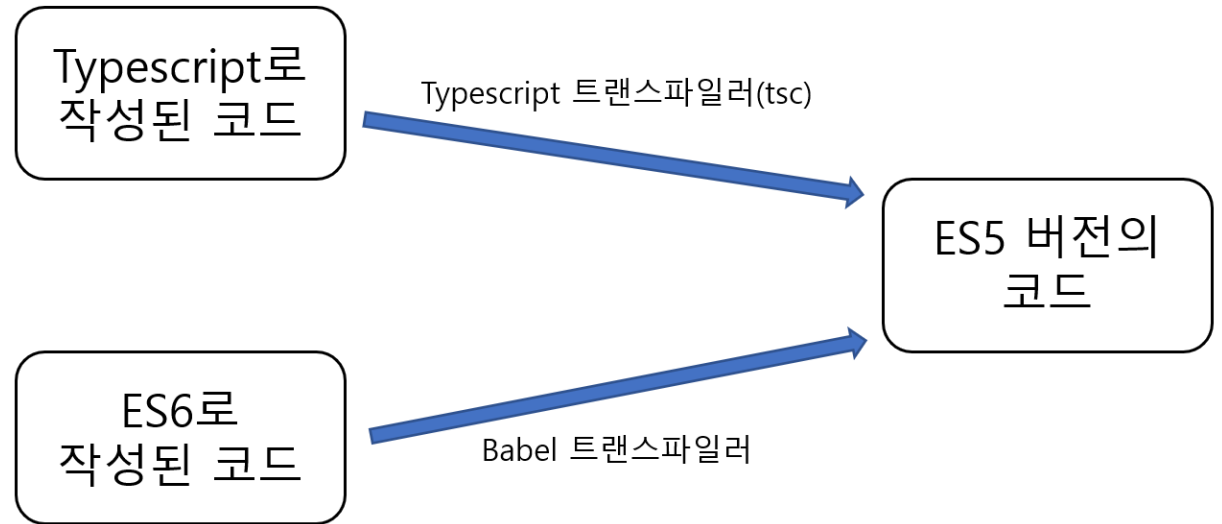


ES6는 이전 버전의 문법을 포함하여 지원하면서
Class, Arrow Function 등 새로운 문법을 추가로 지원함

1. ES6 소개

❖ 트랜스파일러

- Transpile = Translate + Compile
- ES6나 Typescript 언어를 ES5와 같은 이전버전의 자바스크립트 코드로 변환함
- 대표적인 트랜스파일러(Tanspiler)
 - Babel
 - tsc



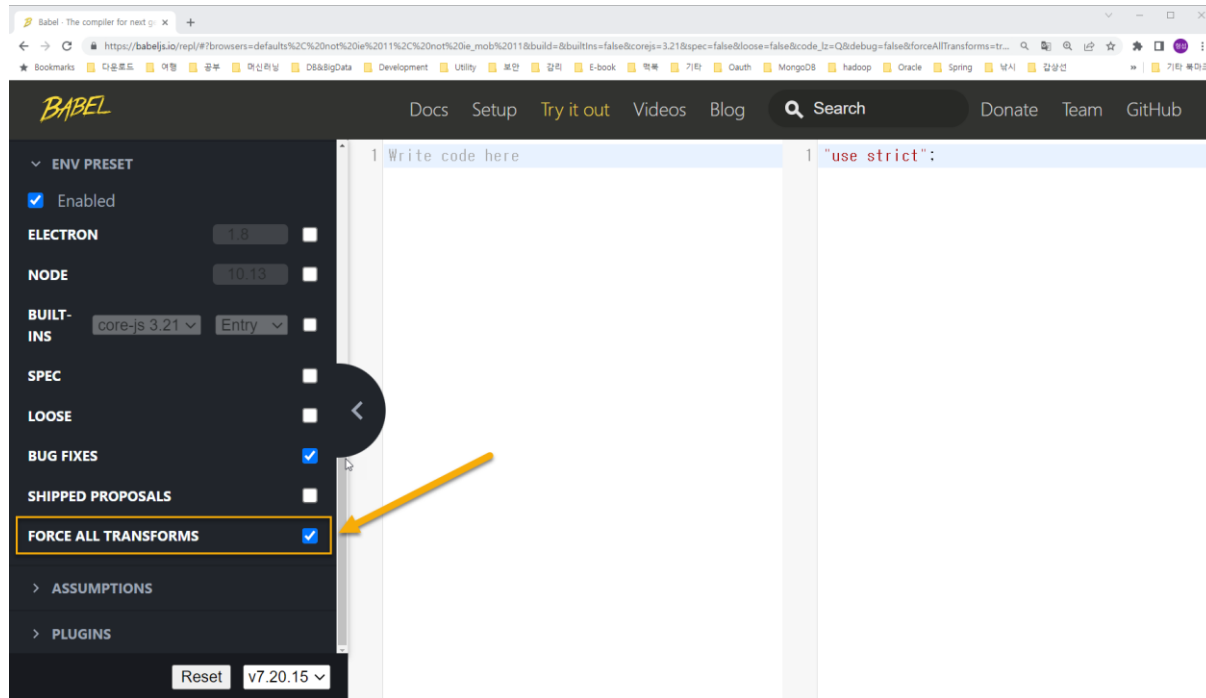
❖ ES6를 학습해야만 하는 이유

- Vue3.x 부터 ES6가 필수
- Vue 3의 반응성이 ES6의 Proxy로 구현되어 있음

1. ES6 소개

❖ES6 학습을 위해 사용하는 추가 도구

- Babel REPL : 브라우저 기반 도구
 - 즉시 트랜스파일 시도
 - <https://babeljs.io/repl>



2. ES6를 위한 프로젝트 설정(1)

❖ ES6 테스트를 위한 프로젝트 생성

- 프로젝트를 위한 디렉토리 생성하고 패키지 생성
 - 터미널 실행 후 ...
 - mkdir es6test
 - cd es6test
 - npm init : 오른쪽 그림 참조

❖ Visual Studio Code 실행

- 실행 후 통합 생성한 폴더 열기
 - 파일 메뉴 - 폴더 열기
- 메인메뉴에서 '터미널' - '새 터미널' 실행후 패키지 설치

```
npm install --save-dev @babel/core @babel/cli @babel/preset-env core-js
```



```
Windows PowerShell
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (es6test)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\_Vue3QuickStart\Vue3예제\ch02\es6test\package.json:

{
  "name": "es6test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> |
```

2. ES6를 위한 환경 설정(2)

❖ 생성된 package.json 예

생성된 package.json 예

```
{
  "name": "es6test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@babel/cli": "^7.20.7",
    "@babel/core": "^7.20.12",
    "@babel/preset-env": "^7.20.2",
    "core-js": "^3.27.2"
  }
}
```

npm 명령어	설명
npm init	프로젝트 초기화
npm install	package.json의 패키지 설치
npm install --save [패키지명]	패키지를 프로젝트 의존성으로 추가
npm install --save-dev [패키지명]	패키지를 프로젝트 개발 의존성 수준으로 추가
npm install --global [패키지명]	패키지를 전역 수준으로 추가
npm update --save	프로젝트 패키지 업데이트
npm run [스크립트명]	package.json의 스크립트 명령 실행
npm uninstall --save [패키지명]	패키지 삭제
npm cache clean	캐시 삭제

babel.config.json

```
{
  "presets": [
    [ "@babel/env" ]
  ]
}
```

3. ES6를 위한 환경 설정(4)

❖babel.config.json 파일 작성

- 이 설정 파일은 babel 실행을 위한 기본 설정 파일
- Visual Studio Code에서 babel.config.json 파일 추가후 다음과 같이 작성

```
{  
  "presets": [["@babel/env"]]  
}
```

❖테스트 코드 작성

- src 폴더 생성 후 02-01.js 파일 추가

```
let name = "john";  
console.log(`Hello ${name}!!`);
```

```
"scripts": {  
  "build": "babel src -d build"  
},
```

- 작성후 통합 터미널에서 npx babel src -d build 명령어 실행
 - 또는 npm run build
- build 디렉토리의 02-01.js 파일 확인

3. let, const(1)

❖ var

- hoisting : 개발자들에게 이해하기 어려운 부분
 - 변수의 선언을 스코프의 최상단으로 옮기는 행위
- 단계
 - 호이스팅 단계 : 내부에 var 키워드가 지정된 코드를 찾아서 메모리를 미리 할당
 - 실행 단계 : 호이스팅 후에 코드를 실행함.
- 함수단위로 호이스팅
 - 함수단위의 scope만 제공함
 - 블록 수준의 Scope를 지원하지 않음
- var는 중복 선언을 허용함으로써 혼란 야기

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

```
var A1 = 100;  
console.log(A1);  
var A1 = "hello";  
console.log(A1);
```

3. let, const(2)

❖let (이어서)

- block scope를 지원함.

ES6

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

ES5

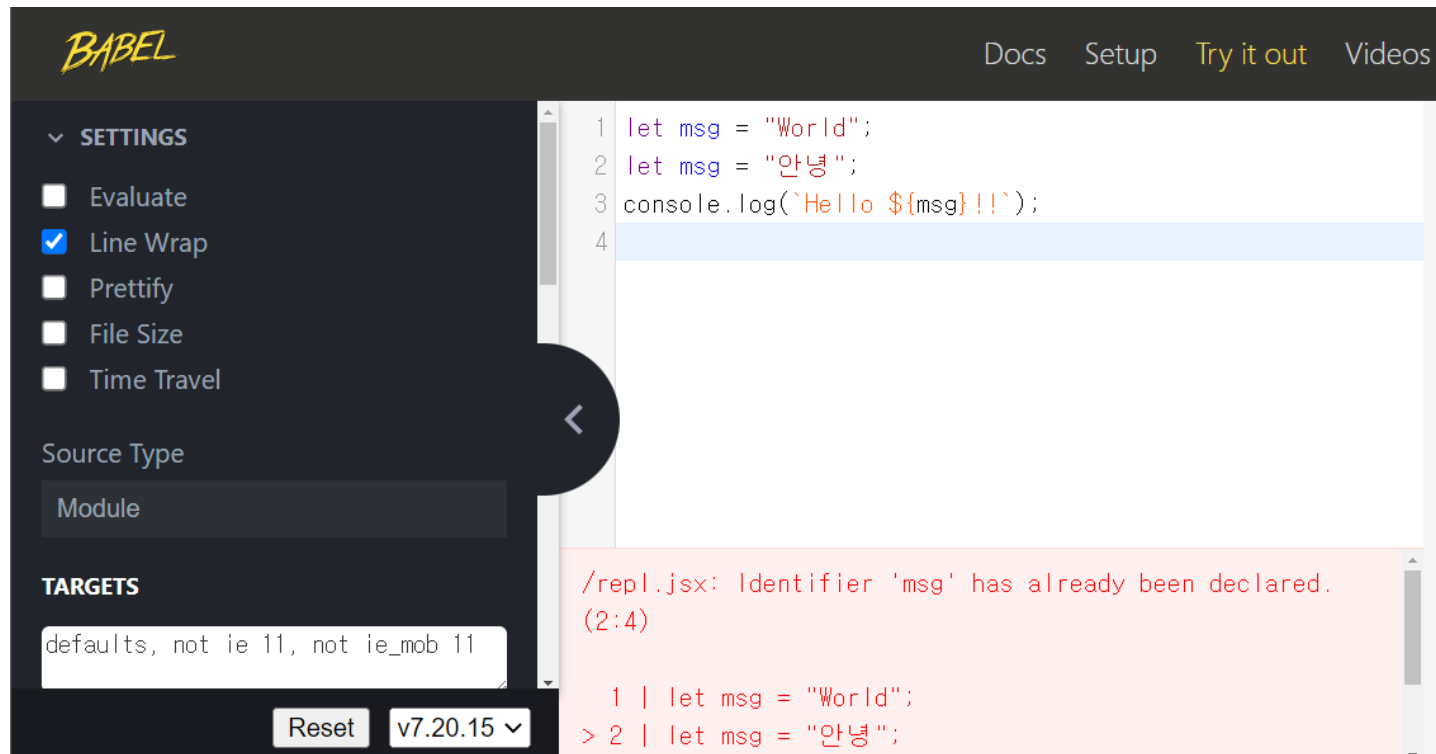
```
"use strict";  
var msg = "GLOBAL";  
function outer(a) {  
  var msg = "OUTER";  
  console.log(msg);  
  if (true) {  
    var _msg = "BLOCK";  
    console.log(_msg);  
  }  
}
```

- 대부분의 var는 let으로 대체가 가능함.

3. let, const(3)

❖let

- var와 선언하는 방법은 유사하지만...
- 중복 선언을 허용하지 않음
- 블록 수준의 Scope 지원

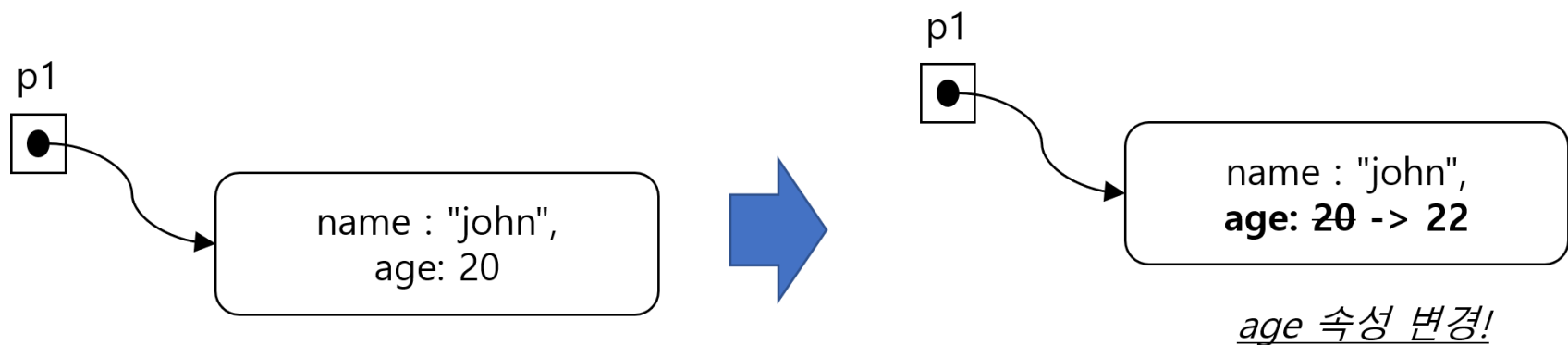


3. let, const(4)

❖ const : 상수 선언

▪ 예제 02-03

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



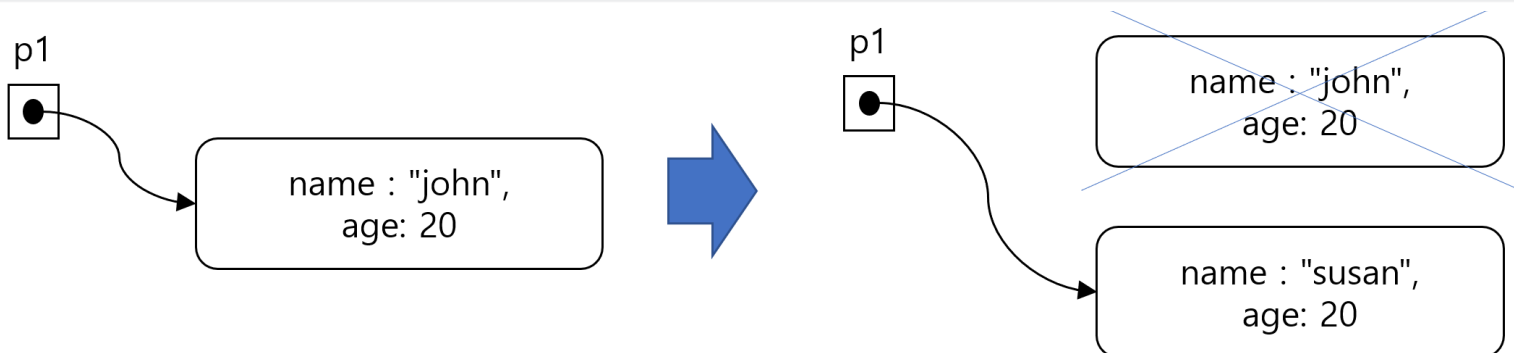
p1이 참조하는 메모리 주소는 변화 없음

3. let, const(5)

■ 예제 02-04

```
const p1 = { name : "john", age : 20 }  
//오류 발생  
p1 = { name:"susan", age: 20 };  
  
console.log(p1);
```

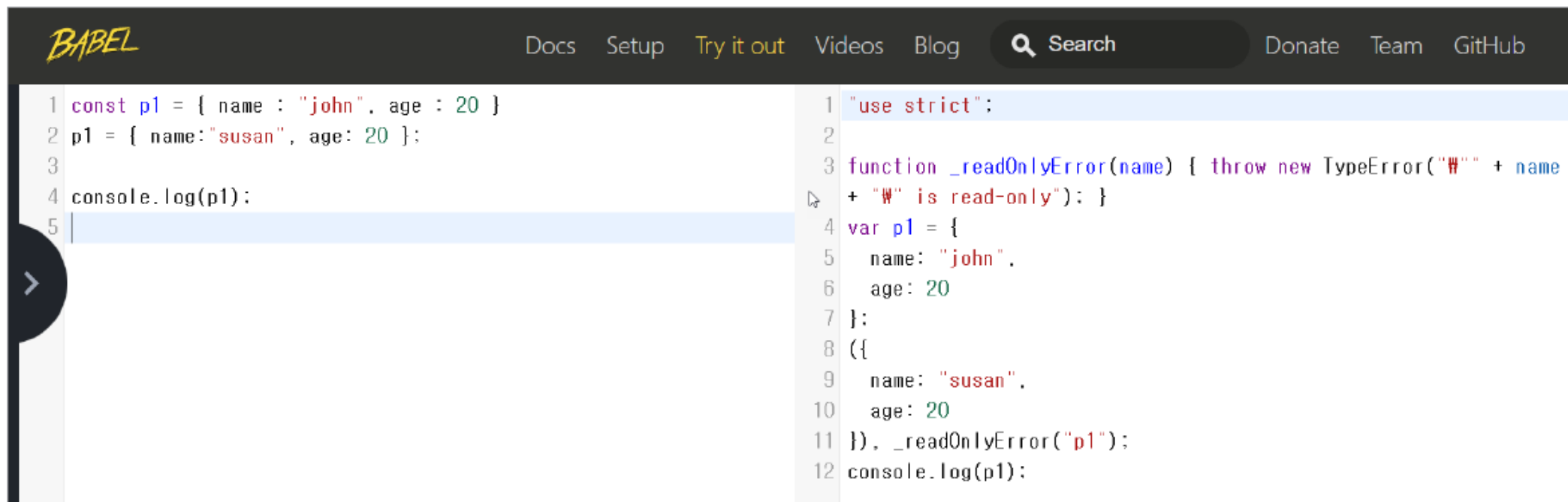
VM38:2 Uncaught TypeError: Assignment to constant variable.
at <anonymous>:2:4



p1이 참조하는 메모리 주소가 바뀌는 것이므로 허용하지 않음.

3. let, const(5)

❖예제 02-04 트랜스파일 결과



The screenshot shows the Babel REPL interface. On the left, the input code is:
1 `const p1 = { name : "john", age : 20 }`
2 `p1 = { name:"susan", age: 20 };`
3
4 `console.log(p1);`
5
On the right, the transpiled output code is:
1 `"use strict";`
2
3 `function _readOnlyError(name) { throw new TypeError("`" + name`" + " is read-only"); }`
4 `var p1 = {`
5 `name: "john",`
6 `age: 20`
7 `};`
8 `({`
9 `name: "susan",`
10 `age: 20`
11 `}), _readOnlyError("p1");`
12 `console.log(p1);`

함수나 객체를 만든 후에 이것의 메모리 주소가 바뀌지 않도록 하고 싶다면 `const`를 사용하면 됩니다. `const`로 객체를 생성한 경우 객체 내부의 속성이 변경할 수는 있지만 새로운 객체를 생성하여 할당하는 것은 허용하지 않는다는 점을 기억합니다.

4. 기본 파라미터와 가변 파라미터(1)

❖파라미터 값을 전달하지 않았을 때의 기본값을 정의

```
function addContact(name, mobile,
                    home="없음",
                    address="없음",
                    email="없음") {
  var 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");
```



문제 출력 디버그 콘솔 터미널

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> npm run build
```

```
> es6test@1.0.0 build
> babel src -d build
```

```
Successfully compiled 5 files with Babel (792ms).
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node build/02-05.js
```

```
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
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> █
```

4. 기본 파라미터와 가변 파라미터(2)

❖가변 파라미터

- 마지막에 배치해야 함.
- Rest Operator를 지원하지 전에는 arguments를 이용해 가변인자를 처리하였음 → 더이상 arguments를 이용하지 않아도 됨.

[예제 02-06 : src/02-06.js]

```
01 function foodReport(name, age, ...favoriteFoods) {  
02   console.log(name + ", " + age);  
03   console.log(favoriteFoods);  
04 }  
05  
06 foodReport("이몽룡", 20, "짜장면", "냉면", "불고기");  
07 foodReport("홍길동", 16, "초밥");
```

문제 출력 터미널 디버그 콘솔

D:_Vue3QuickStart\Vue3_예제\ch02\es6test>npx babel src -d build
Successfully compiled 6 files with Babel (762ms).

D:_Vue3QuickStart\Vue3_예제\ch02\es6test>node build\02-06.js
이몽룡, 20
['짜장면', '냉면', '불고기']
홍길동, 16
['초밥']

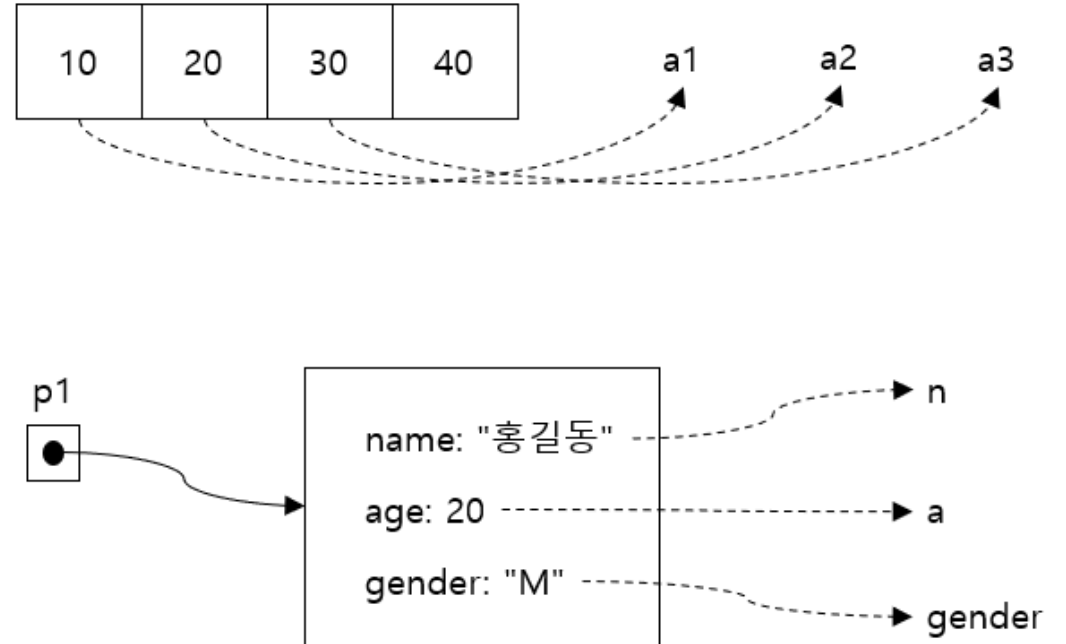
5. 구조분해 할당(1)

❖구조 분해 할당(Destructuring Assignment)

- 배열, 객체의 값들을 여러 변수에 추출하여 할당할 수 있도록 하는 새로운 표현식

[예제 02-07 : src/02-07.js]

```
01 let arr = [10, 20, 30, 40];  
02 let [a1, a2, a3] = arr;  
03 console.log(a1, a2, a3);  
04  
05 let p1 = { name: "홍길동", age: 20, gender: "M" };  
06 let { name: n, age: a, gender } = p1;  
07 console.log(n, a, gender);
```



5. 구조분해 할당(2)

❖구조 분해 할당(이어서)

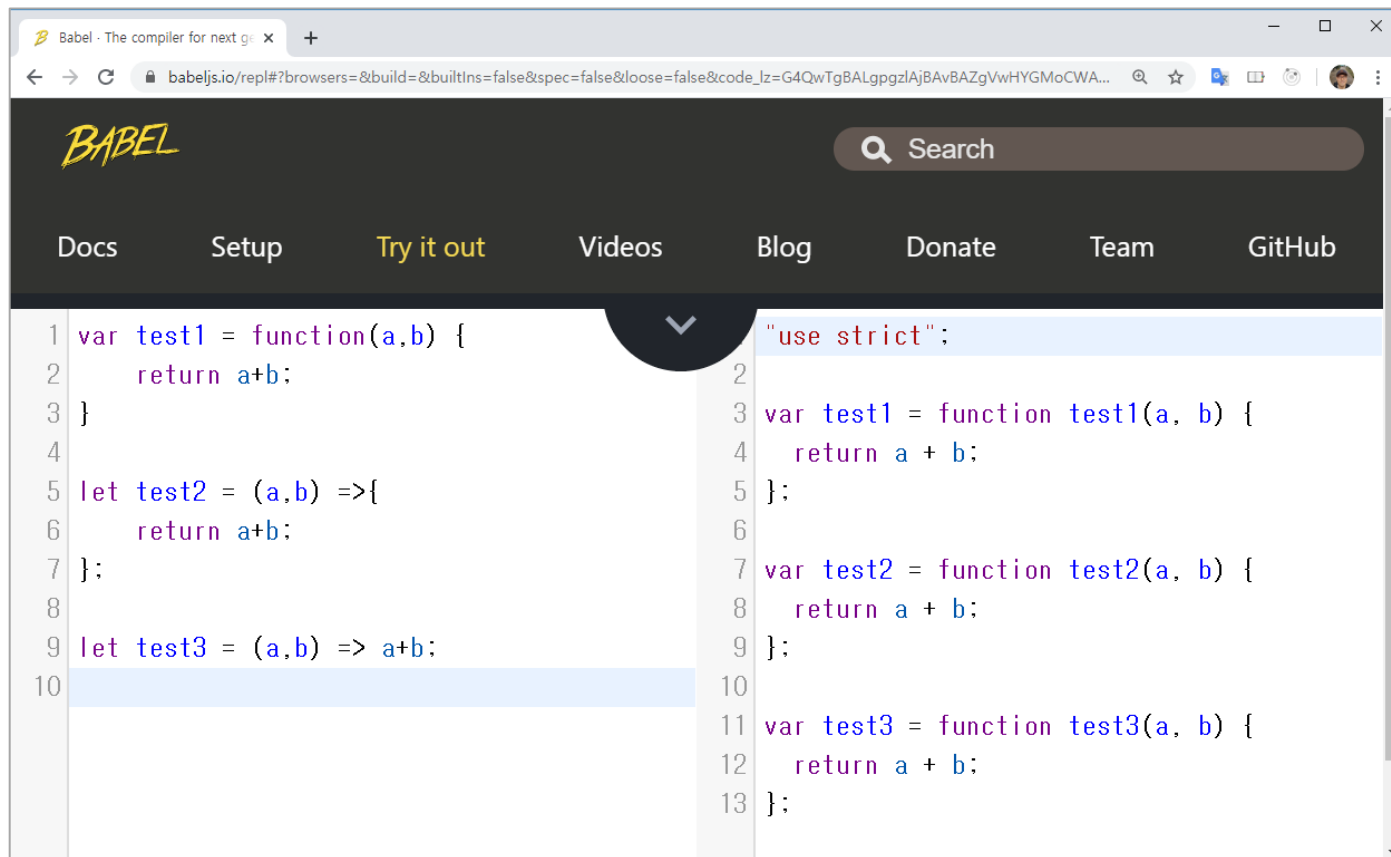
[예제 02-08 : src/02-08.js]

```
01 function addContact1({name, phone, email="이메일 없음", age=0}) {
02   console.log(name, phone, email, age);
03 }
04 addContact1({ name: "이몽룡", phone: "010-3434-8989" })
05
06 function addContact2(contact) {
07   if (!contact.email) contact.email = "이메일없음";
08   if (!contact.age) contact.age = 0;
09   let { name, phone, email, age } = contact;
10   console.log(name, phone, email, age);
11 }
12 addContact2({ name: "이몽룡", phone: "010-3434-8989" })
13
14 function addContact3(name, phone, email="이메일없음", age=0) {
15   console.log(name, phone, email, age);
16 }
17 addContact3("이몽룡", "010-3434-8989")
```

6. 화살표 함수(1)

❖ 화살표 함수

- 트랜스파일된 결과는 같음
- 핵심적인 차이는 this와 관련되어 있음



The screenshot shows the Babel REPL interface. On the left, the input code is:

```
1 var test1 = function(a,b) {  
2   return a+b;  
3 }  
4  
5 let test2 = (a,b) =>{  
6   return a+b;  
7 };  
8  
9 let test3 = (a,b) => a+b;  
10
```

On the right, the output code after transpilation is:

```
2 "use strict";  
3 var test1 = function test1(a, b) {  
4   return a + b;  
5 };  
6  
7 var test2 = function test2(a, b) {  
8   return a + b;  
9 };  
10  
11 var test3 = function test3(a, b) {  
12   return a + b;  
13 };
```

6. 화살표 함수(2)

❖ 자바스크립트의 this

- 현재 호출 중인 메서드를 보유한 객체를 가리킴 (default)

[예제 02-10]

```
01 var obj = { result: 0 };  
02 obj.add = function(x,y) {  
03   this.result = x+y;  
04 }  
05 obj.add(3,4)  
06 console.log(obj)      // { result: 7 }
```

- 위코드를 다음과 같이 실행하면?

```
var add2 = obj.add;  
//호출될 때 add2() 메서드를 보유한 객체가 없으므로 Global(전역)객체가 this가 됨.  
add2(3,4)
```

- this가 바인딩되는 시점?
 - 메서드를 호출할 때마다 this가 바인딩됨.
 - 또한 메서드를 호출할 때 직접 this를 지정할 수 있음 (apply, call 메서드)
 - 또한 this가 미리 바인딩된 새로운 함수를 리턴할 수 있음 (bind 메서드)

6. 화살표 함수(3)

■ 예제 02-12

예제 02-12 : src/02-12.js

```
01: var add = function(x,y) {  
02:     this.result = x+y;  
03: }  
04:  
05: var obj = {};  
06: //1. apply() 사용  
07: //add.apply(obj, [3,4])  
08: //2. call() 사용  
09: //add.call(obj,3,4)  
10: //3. bind() 사용  
11: add = add.bind(obj);  
12: add(3,4)  
13:  
14: console.log(obj);      // { result : 7 }
```

- 11행의 bind() 메서드는 obj를 this로 지정한 새로운 함수를 리턴!!
- 따라서 12행과 같이 특정객체의 메서드 형태로 호출하지 않아도 this는 obj임

6. 화살표 함수(4)

■ 전통적인 함수가 중첩되었을 때의 문제점 이해 : 예제 02-13

```
var obj = { result:0 };
obj.add = function(x,y) {
  function inner() {
    this.result = x+y;
  }
  inner();
}
obj.add(4,5)

console.log(obj);
console.log(result);
```

- add() 메서드 내부에 inner 함수가 정의되어 있음
- 바깥쪽 함수 바로 안쪽 영역의 this? --> obj를 참조함.
- inner() 함수 내부의 this가 obj를 참조할 것인가?
 - 그렇지 않음. inner() 와 같이 호출했기 때문에 inner() 내부의 this는 전역객체를 참조함. 전역변수 result에 결과가 저장
- 이 문제를 해결하려면?
 - apply(), call(), bind()를 이용하거나
 - 화살표 함수를 이용한다.

6. 화살표 함수(5)

■ 문제 해결1 : bind()

```
var obj = { result:0 };
obj.add = function(x,y) {
  function inner() {
    this.result = x+y;
  }
  inner = inner.bind(this);
  inner();
}
obj.add(4,5)

console.log(obj)
```

■ 문제 해결3 : 화살표 함수

```
var obj = { result:0 };
obj.add = function(x,y) {
  var inner = () => {
    this.result = x+y;
  }
  inner()
}
obj.add(4,5)

console.log(obj)
```

- 화살표 함수는 lexical binding이 아님
- 함수가 중첩되었을 때 바깥쪽 함수의 this가 안쪽 함수로 지정됨.
- React 클래스 컴포넌트 작성할 때 알고 있어야 하는 개념

7. 객체 리터럴(1)

❖ 새로운 객체 리터럴

- 객체 속성 표기

```
var name = "홍길동";  
var age = 20;  
var email = "gdhong@test.com";  
  
var obj = { name, age, email };  
  
console.log(obj);
```

- 속성명과 변수명이 같은 경우는 생략 가능

```
var obj = { name: name, age: age, email: email };
```

7. 객체 리터럴(2)

❖ 새로운 객체 리터럴 (이어서)

- 새로운 메서드 표기법

```
let p1 = {
  name : "아이패드",
  price : 200000,
  quantity : 2,
  order : function() {
    if (!this.amount) {
      this.amount = this.quantity * this.price;
    }
    console.log("주문금액 : " + this.amount);
  },
  discount(rate) {
    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();
```

```
discount: function discount(rate) {
  if (rate > 0 && rate < 0.8) {
    this.amount = (1 - rate) * this.price * this.quantity;
  }
  console.log(100 * rate + "% 할인된 금액으로 구매합니다.");
}
```

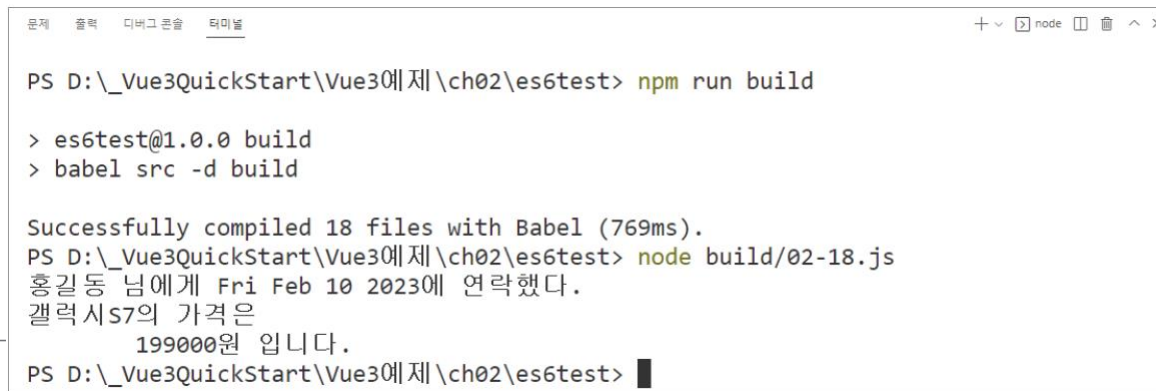
8. 템플릿 리터럴(1)

❖ backtick(`)으로 묶여진 문자열

- 템플릿 대입문(`\${}`) 로 문자열 끼워넣기 기능 제공
 - 템플릿 대입문에 수식 구문, 변수, 함수 호출 구문 등 모든 표현식이 올 수 있음.
 - 템플릿 문자열을 다른 템플릿 문자열 안에 배치하는 것도 가능
 - `\${` 을 나타내려면 \$ 또는 {을 이스케이프시킴

예제 02-18 : src/02-18.js

```
01: const d1 = new Date();
02: let name = "홍길동";
03: let r1 = `${name} 님에게 ${d1.toDateString()}에 연락했다.`;
04: console.log(r1);
05:
06: let product = "갤럭시S7";
07: let price = 199000;
08: let str = `${product}의 가격은
09:           ${price}원 입니다.`;
10: console.log(str);
```



```
문제 출력 디버그 콘솔 터미널
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> npm run build

> es6test@1.0.0 build
> babel src -d build

Successfully compiled 18 files with Babel (769ms).
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node build/02-18.js
홍길동 님에게 Fri Feb 10 2023에 연락했다.
갤럭시S7의 가격은
199000원 입니다.
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> █
```

9. 모듈(1)

❖Module

- 여러 디렉토리와 파일에 나눠서 코드를 작성할 수 있도록 함.
- 자바스크립트 파일은 모듈로써 импорт 될 수 있음

❖Export

- 모듈안에서 선언된 모든 것은 local(private)
- 모듈 내부의 것들을 public으로 선언하고 다른 모듈에서 이용할 수 있도록 하려면 export 해야 함.
- export 대상 항목
 - let, const, var, function, class

```
export let a= 1000;  
export function f1(a) { ... }  
export { n1, n2 as othername, ... }  
//export할 때 기존의 이름이 아닌 다른 이름을 사용하고 싶다면 as를 이용함.
```

❖Import

- 익스포트된 모듈은 다른 모듈에서 import 구문을 이용해 참조할 수 있음

9. 모듈(2)

❖ Basic Example

[예제 02-19 : src/modules/02-19-module.js]

```
01 let base = 100;  
02 const add = (x) => base+x;  
03 const multiply = (x) => base*x;  
04  
05 export { add, multiply };
```

[예제 02-20 : src/02-20-main.js]

```
01 import { add, multiply } from './modules/02-19-module'  
02  
03 console.log(add(4));  
04 console.log(multiply(4));
```

9. 모듈(3)

❖ Default export

- default export를 사용해 단일 값을 익스포트, 임포트 할 수 있음

예제 02-19 변경

```
01: let base = 100;
02: const add = (x) => base+x;
03: const multiply = (x) => base*x;
04: const getBase = ()=>base;
05:
06: export { add, multiply };
07: export default getBase;
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> npm run build
```

```
> es6test@1.0.0 build
> babel src -d build
```

```
Successfully compiled 20 files with Babel (821ms).
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node ./build/02-20-main.js
```

```
104
```

```
100
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> █
```

예제 02-20 변경

```
01: import getBase, { add } from './modules/02-19-module';
02:
03: console.log(add(4));
04: console.log(getBase());
```

10. Promise(1)

❖비동기 처리를 위한 콜백 처리

- Callback Hell : 콜백함수들이 중첩되어 지옥을 경험함
 - 디버깅 어려움.
 - 예외처리 어려움
- ES6 Promise는 Callback Hell 문제 해결

```
// Promise 객체의 생성
const p = new Promise((resolve, reject) => {
  //비동기 작업 수행
  //이 내부에서 resolve(result)함수를 호출하면 then에 등록해둔 함수가 호출됨
  // reject(error)가 호출되거나 Error가 발생되면 catch에 등록해둔 함수가 호출됨.
});

p.then((result)=> {

})
.catch((error)=> {

})
```

```
getData(function(a){
  getMoreData(a, function(b){
    getMoreData(b, function(c){
      getMoreData(c, function(d){
        getMoreData(d, function(e){
          ...
        });
      });
    });
  });
});
```


10. Promise(2)

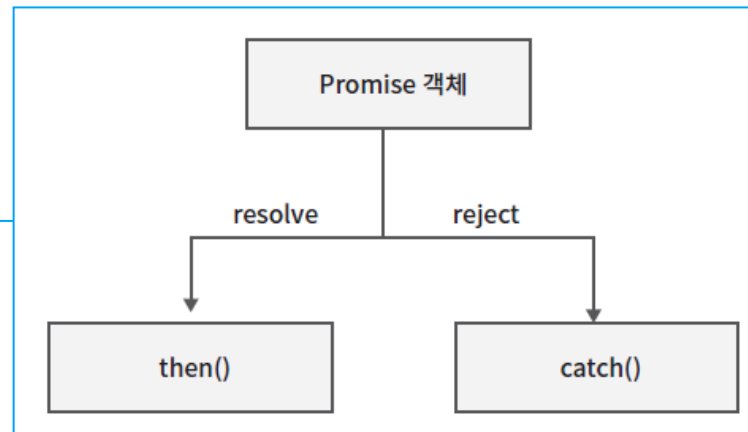
❖ Promise 패턴

- 자바스크립트 비동기 처리를 수행하는 추상적인 패턴

```
const p = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    var 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!");
```



```
문제 출력 디버그 콘솔 터미널  
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node src/02-21.js  
## Promise 객체 생성!  
처리 결과 : 0.35236729567388303  
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test>
```

그림 02-24 : 예제 02-21의 resolve 호출

```
문제 출력 디버그 콘솔 터미널  
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node src/02-21.js  
## Promise 객체 생성!  
오류 : 생성된 숫자가 0.8이상임 - 0.9434341856369346  
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test>
```

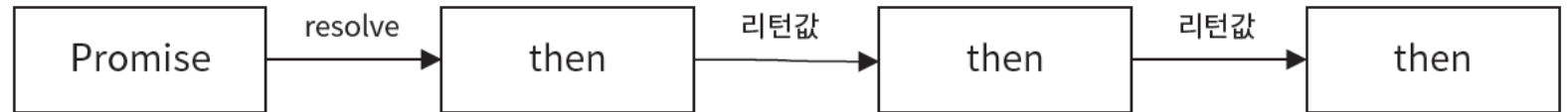
그림 02-25 : 예제 02-21의 reject 호출

10. Promise(3)

❖ Promise Chaining

- then 메서드의 리턴값은 다시 Promise 객체 리턴 가능 → 연속적인 작업 처리시에 유용함
- Promise 객체를 직접 생성하여 리턴할 수도 있음

```
var p = new Promise((resolve, reject)=> {  
  resolve("first!")  
})  
  
p.then((msg)=> {  
  console.log(msg);  
  return "second";  
})  
.then((msg)=>{  
  console.log(msg);  
  return "third";  
})  
.then((msg)=>{  
  console.log(msg);  
})
```

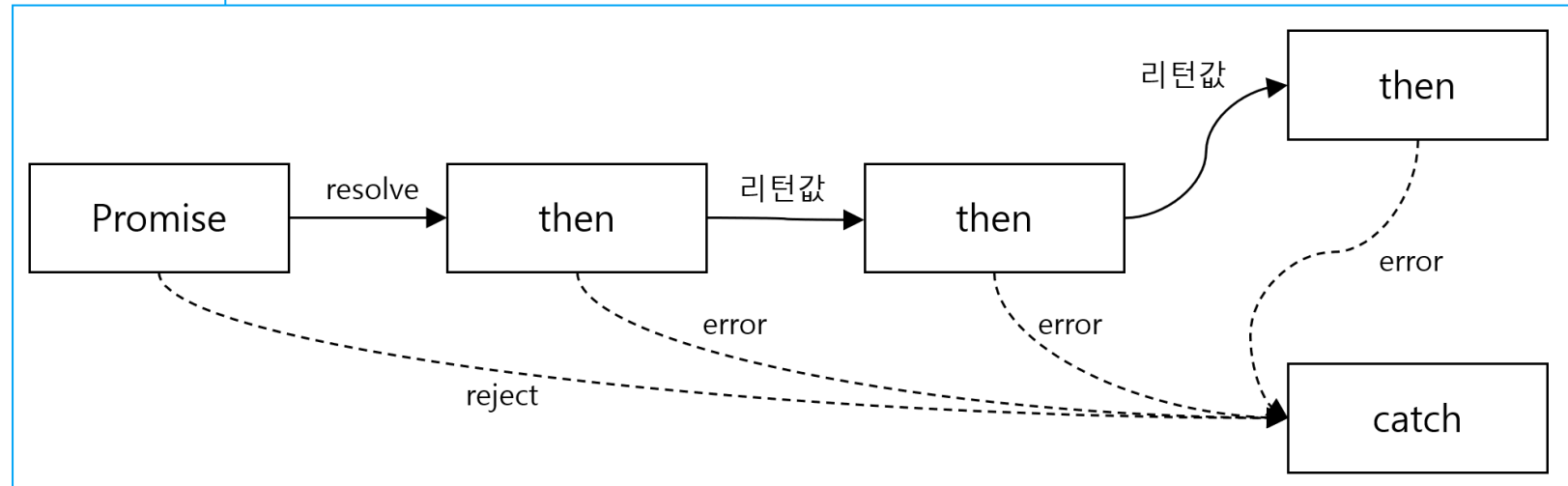


10. Promise(4)

❖ Promise Chain에 catch 추가

- then() 내부에서 오류가 발생하면 가장 가까운 catch()에 등록된 함수가 호출됨

```
var 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);  
});
```



11. 전개 연산자(1)

❖ Spread Operator

- 객체나 배열을 복제할 때 자주 사용함
- 기존 객체,배열을 그대로 둔 채 새로운 객체, 배열을 생성함.

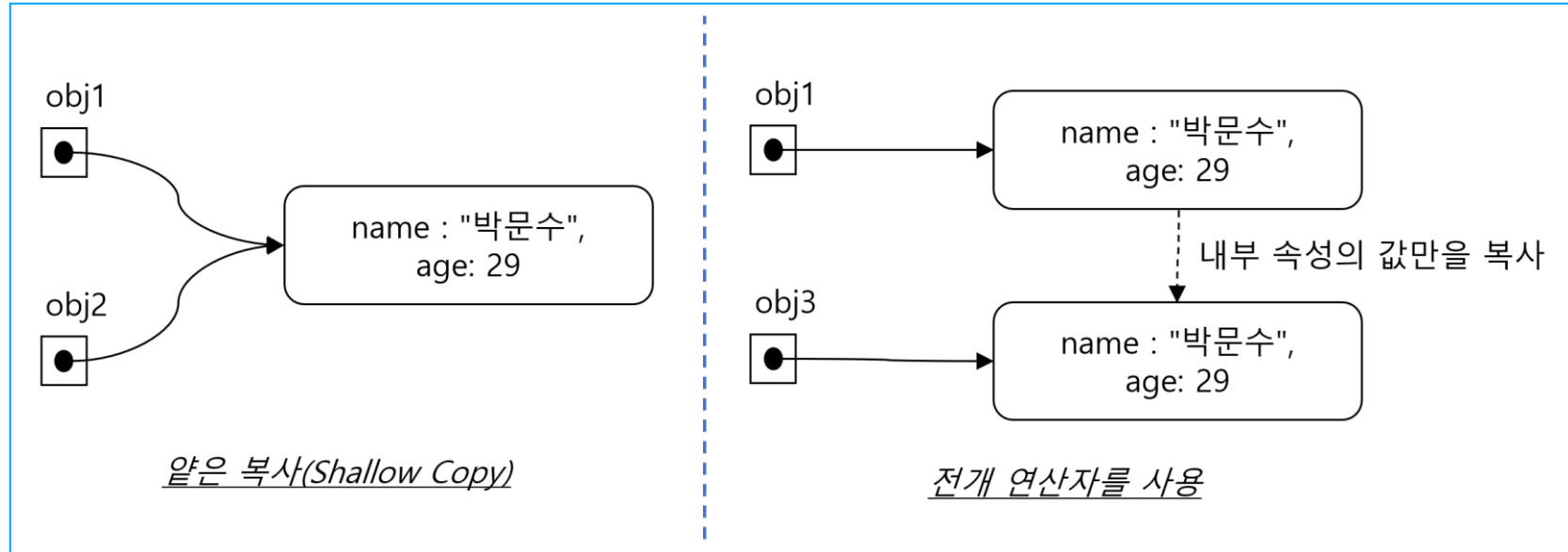
```
let obj1 = { name: "박문수", age: 29 };  
let obj2 = obj1; //shallow copy! obj1, obj2는 동일한 객체를 참조  
let obj3 = { ...obj1 }; //객체 내부의 값은 복사하지만 obj3, obj1은 다른 객체  
let obj4 = { ...obj1, email: "mspark@gmail.com" }; //새로운 속성 추가
```

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

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

11. 전개 연산자(2)

❖ 얕은 복사와 전개 연산자 사용 비교

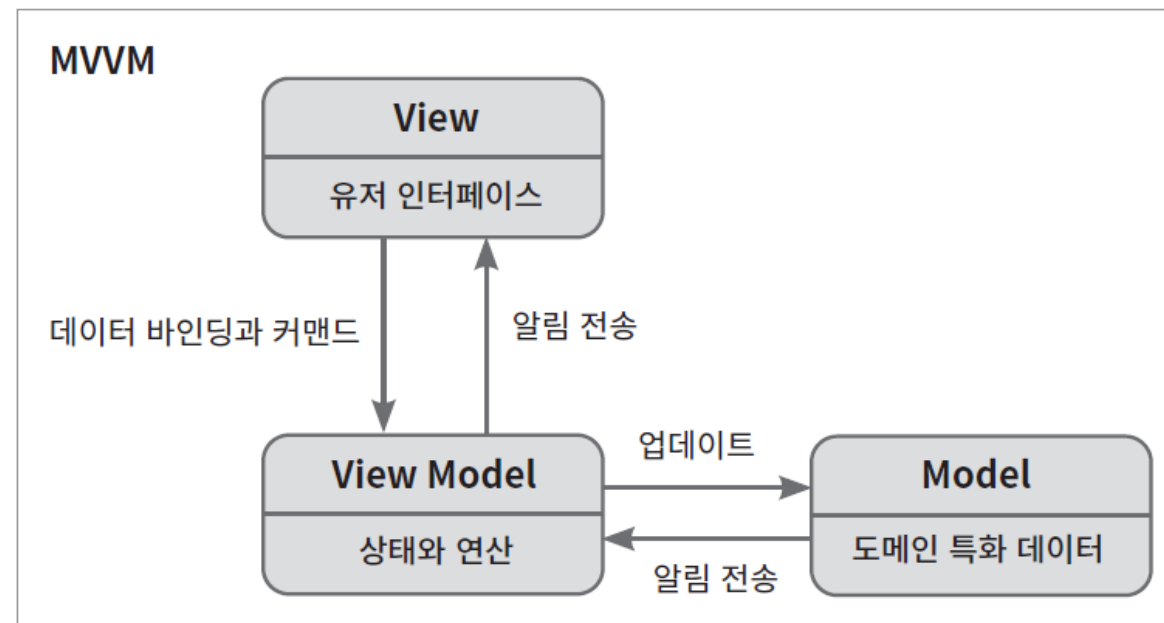


```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node src/02-23.js
{ name: '박문수', age: 19 }
{ name: '박문수', age: 19 }
{ name: '박문수', age: 29 }
true
false
[ 100, 200, 300 ]
[ 'hello', 100, 200, 300, 'world' ]
```

12. Proxy(1)

❖Proxy란?

- 객체의 속성을 읽어오거나 설정하는 작업을 가로채기 위해 래핑할 수 있도록 하는 객체
 - 객체의 속성에 접근할 때 개발자가 지정한 작업을 수행하도록 할 수 있음
- Vue3의 반응성은 Proxy를 통해서 제공됨
 - 내부적으로 사용되므로 개발자가 직접 Proxy 객체를 생성할 일은 없음
- Vue3의 model 객체의 속성이 변경될 때 ...
 - proxy가 '알림 전송'을 수행



12. Proxy(2)

❖ Proxy 예제 : 예제 02-24

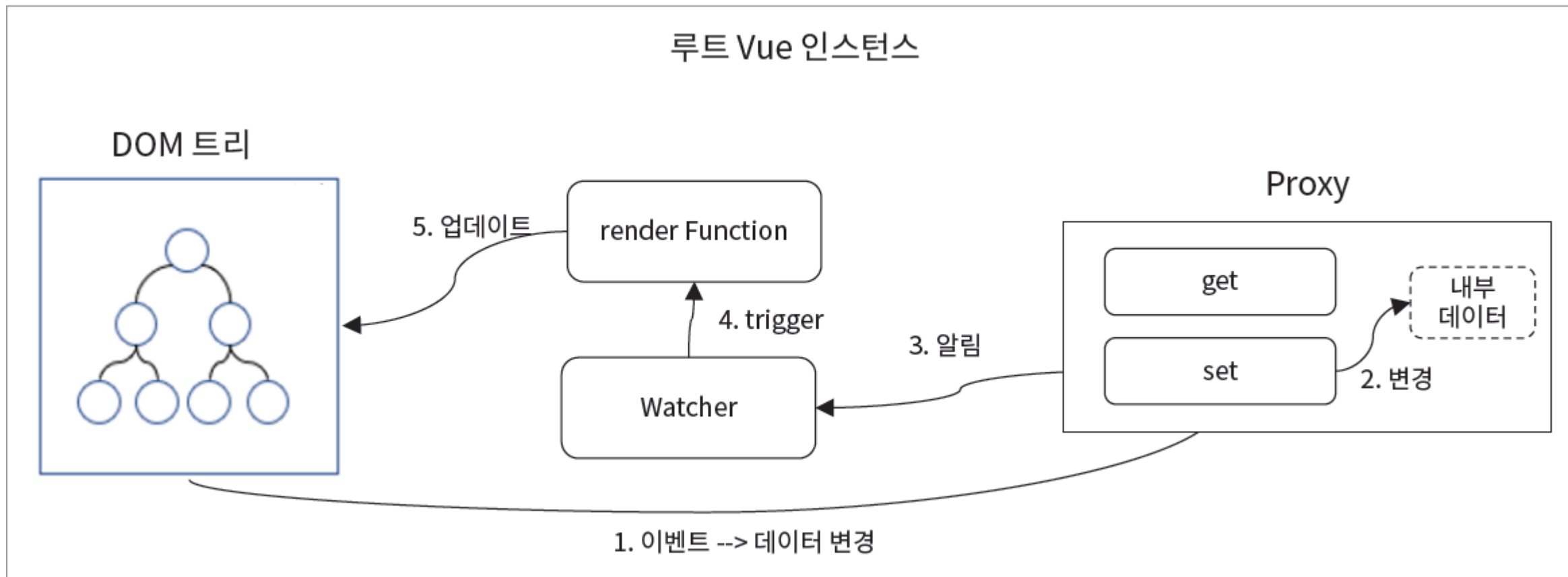
예제 02-24 : src/02-24.js

```
01: let obj = { name : "홍길동", age : 20 };
02: const proxy = new Proxy(obj, {
03:   get: function(target, key) {
04:     console.log("## get " + key)
05:     if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
06:     return target[key];
07:   },
08:   set : function(target, key, value) {
09:     console.log("## set " + key)
10:     if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
11:     target[key] = value;
12:   }
13: })
14:
15: console.log(proxy.name);      //읽기 작업 get 호출
16: proxy.name = "이몽룡";      //쓰기 작업 set 호출
17: proxy.age = 30;              //쓰기 작업 set 호출
```

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node src/02-24.js
## get name
홍길동
## set name
## set age
```


12. Proxy(3)

❖Vue3의 반응성 개요도



12. Proxy(4)

❖배열에 대한 Proxy 생성

예제 02-25 : src/02-25.js

```
01: var arr = [10,20,30];
02:
03: const proxy = new Proxy(arr, {
04:   get: function(target, key, receiver) {
05:     console.log("## get " + key)
06:     if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
07:     return target[key];
08:   },
09:   set : function(target, key, value) {
10:     console.log("## set " + key)
11:     if (!target[key]) throw new Error(`존재하지 않는 속성(${key})입니다`);
12:     target[key] = value;
13:   }
14: })
15:
16: proxy[1] = 99;
```

문제 출력 디버그 콘솔 터미널

+ v node [icon] [icon] ^ x

```
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> node src/02-25.js
## set 1
PS D:\_Vue3QuickStart\Vue3예제\ch02\es6test> █
```

13. 마무리

이제까지 ES6 문법 요소 중 Vue 3 개발에 필요한 것들 위주로 살펴보았습니다. Vue 2까지는 ES6 문법이 필수가 아니었지만 Vue 3부터는 반드시 익혀야 합니다. ES6는 Vue 개발뿐만 아니라 다른 프론트엔드 프레임워크 개발에 꼭 필요한 것이니 반드시 꼼꼼하게 학습하세요.