

## \* destructuring

```
var {body} = document;
```

K	V
:	
body	200
:	

~~200~~ <body> — </body>

document.getElementsByTagName("body")[0]

body == document.body ==

## \* function



function prototype (C/C++)  
= method signature (Java)

function 함수명 (파라미터, 파라미터, ...)

Function body {  
    `문장1;`  
    :  
    return 표현식;  
}

리턴값 : "aaa", 20, true, {}-[], [ ]  
변수 : a, score, sum ...  
식 : a + "hello", a \* 2, 함수호출 ...

함수호출  
(함수 선언시켜놓은 것)  
⇒     함수호출 (인자, 인자, ...);  
                ↑      ↓  
                아주먼드 (argument)

## \* 아규먼트와 파라미터

f(10);

function f(a, b) {  
 arguments = [10]  
}

• 모든 함수에 접근 가능한 Built-in 변수  
• 아기언은 모두를 위한 기본적인 변수

## \* 아규먼트와 파라미터

$f(10, 20);$

function       $f(a, b) \{$

$\equiv$

}

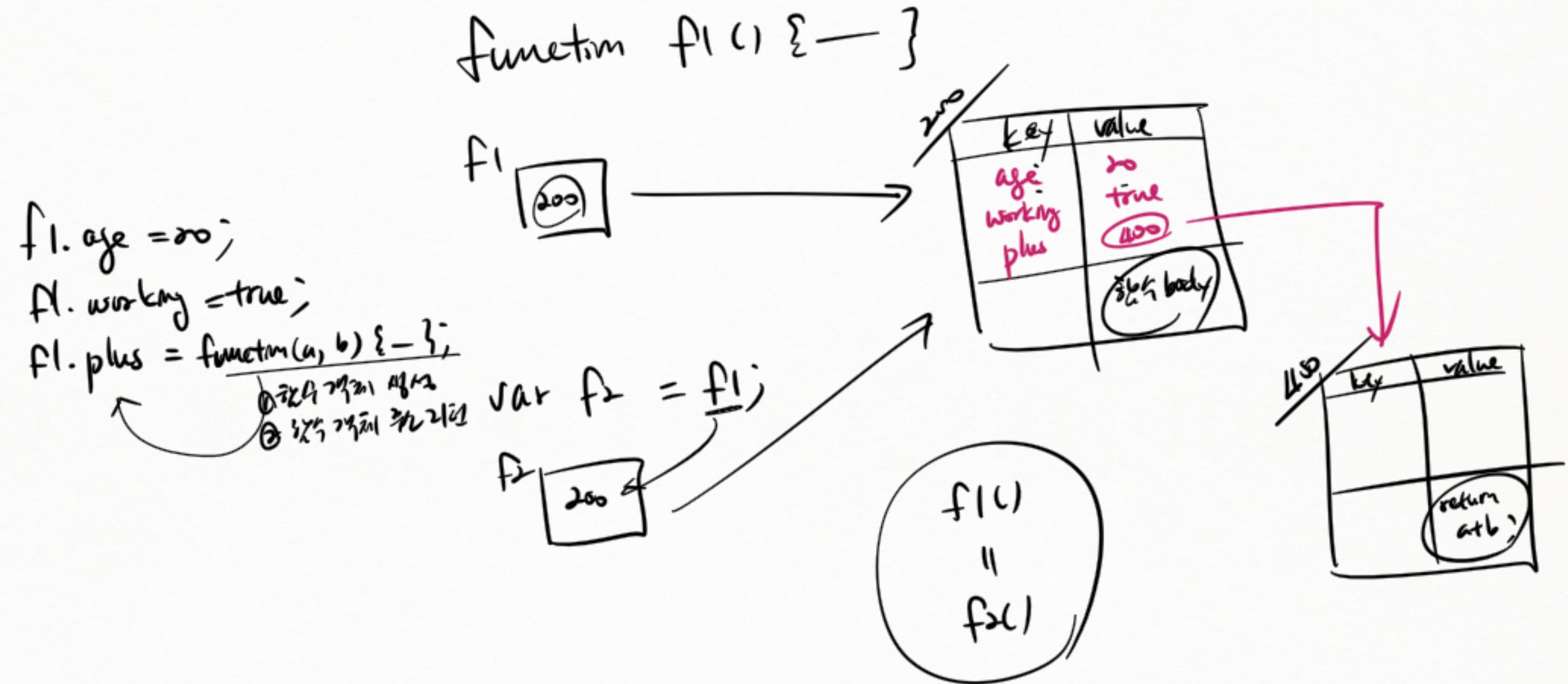
arguments = [10, 20]

## \* 아규먼트와 파라미터

$f(10, 20, 30, 40);$

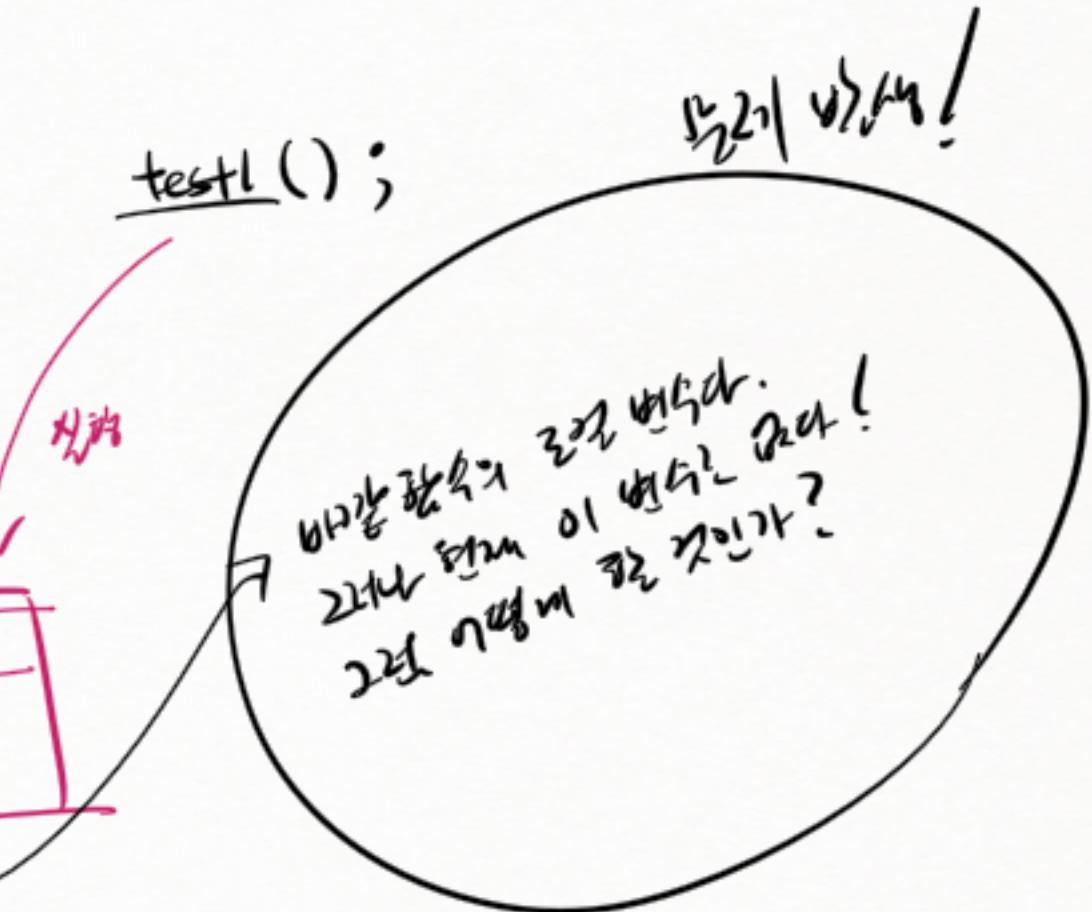
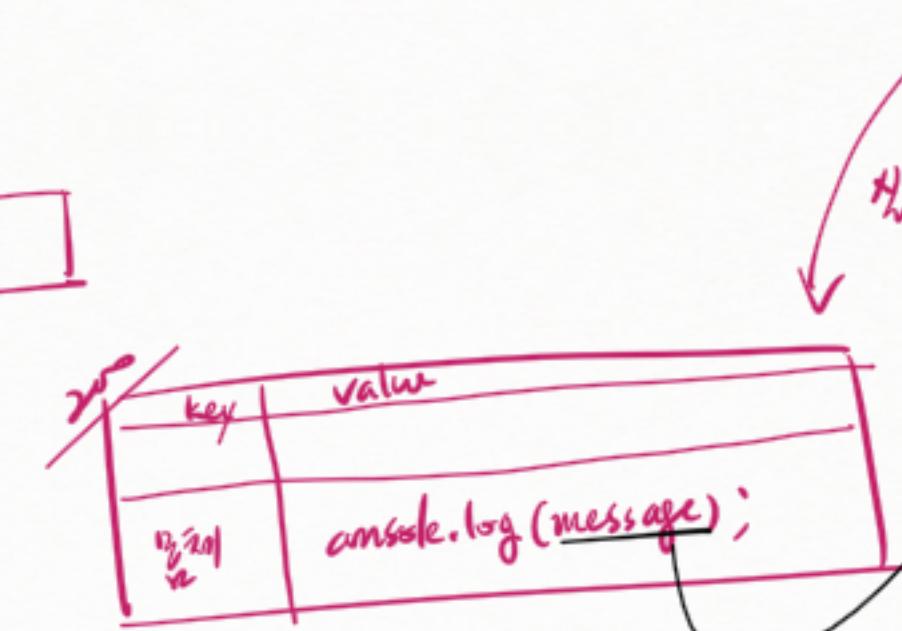
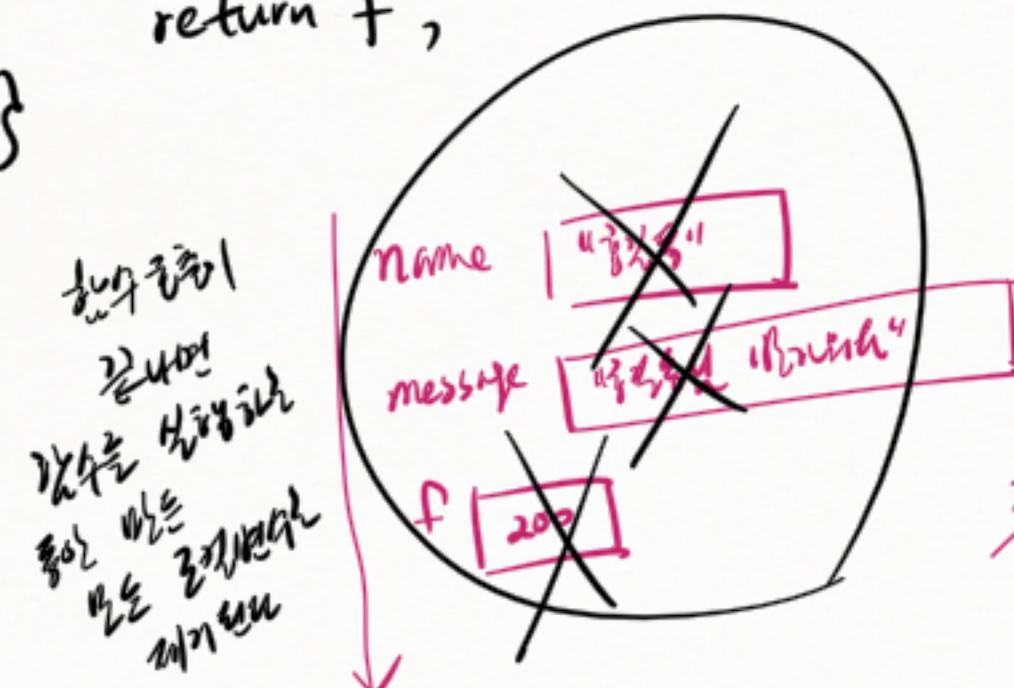
function       $f(a, b)$  {  
     $\equiv$                   arguments = [10, 20, 30, 40]  
}

\* 흡수와 레퍼런스



\* closure

```
function createGreeting(name) {  
    var message = name + "님 반갑습니다";  
  
    var f = function() { console.log(message); };  
  
    return f;  
}
```



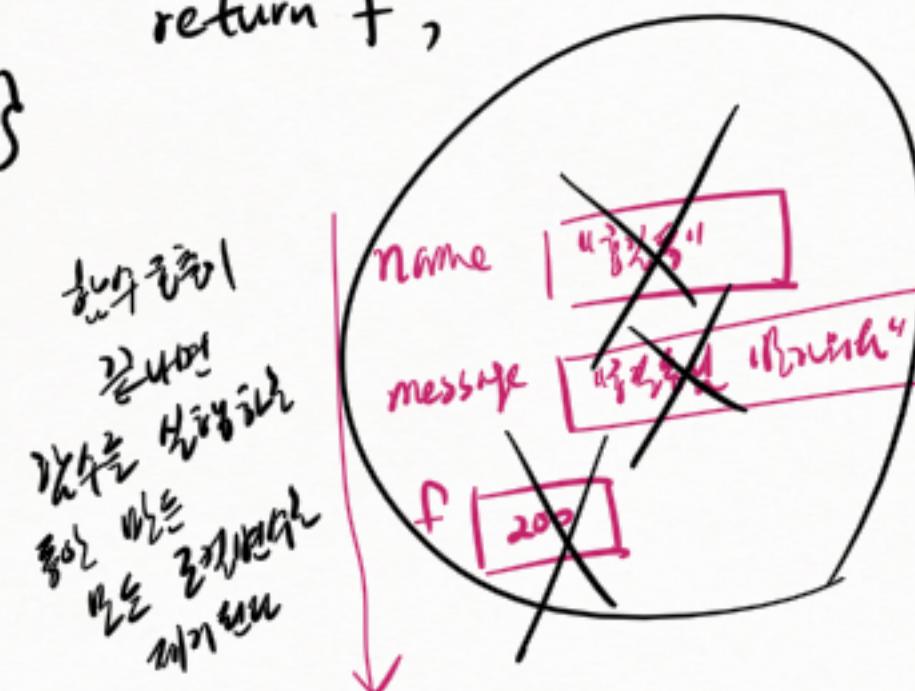
```
var test1 = createGreeting("3 ½ %");
```

test1 → 200

- name
- message
- \*

\* closure : 100% 깨끗이 청결 상태를  
갖고 있어야 한다!

```
function createGreeting (name) {  
    var message = name + "님 반갑습니다";  
    var f = function () { console.log(message); } ;
```



closure 를 만들 때  
클로저가 사용하는  
값을 전역의 로컬변수로  
저장,  
클로저의 값을 만들 때  
변경된다.

key	value
12345 var	anssle.log ( <u>message</u> );

key	value
message	"

복지제도 예산을 가리킨다.

```
var test1 = createGreeting("Hello");
```

test1 → 200

200

- name
- message
- \*

test()

your first  
message right  
now  
right!

## \* closure 例

test(1)  
test2()

var test1 = createGreeting ("Hello");

test1

200

key	value
	console.log(message);

closure 例 2  
var test1 = createGreeting ("Hello");  
var test2 = createGreeting ("World");

key	value
message	"Hello World"

var test2 = createGreeting ("World");

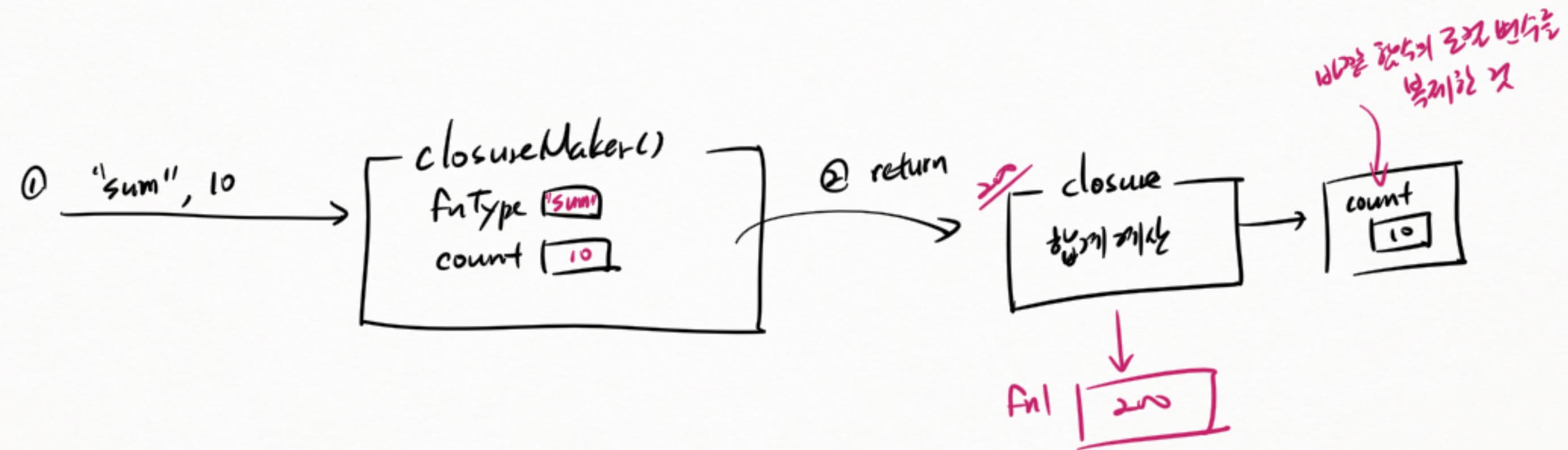
test2

300

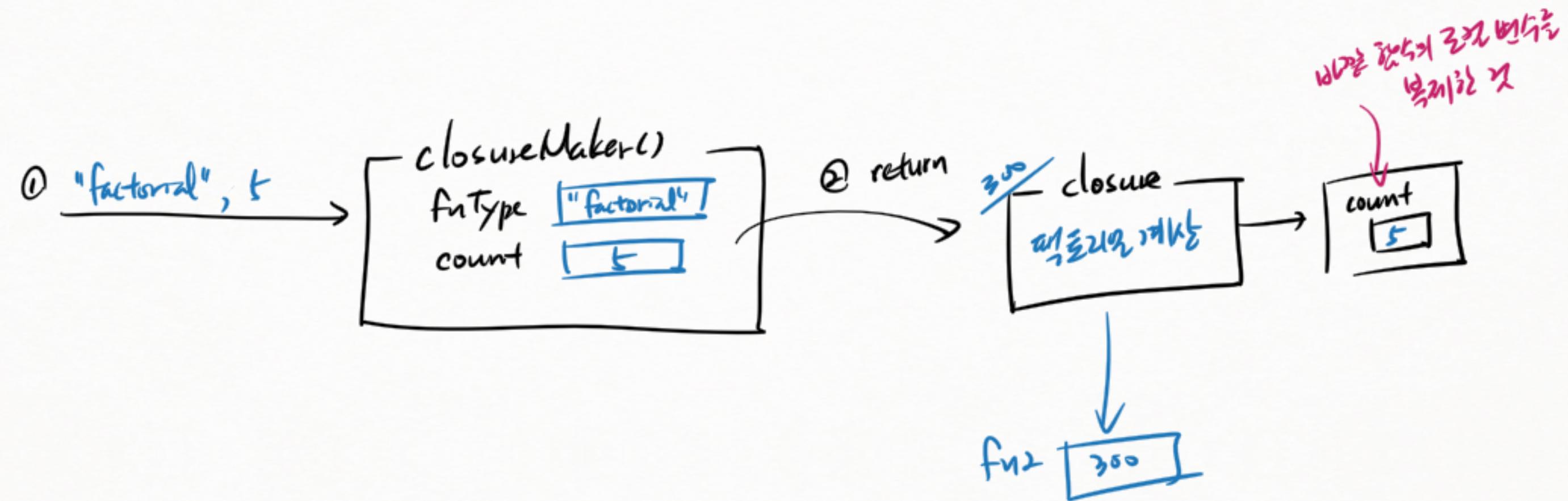
key	value
	console.log(message);

key	value
message	"Hello World"

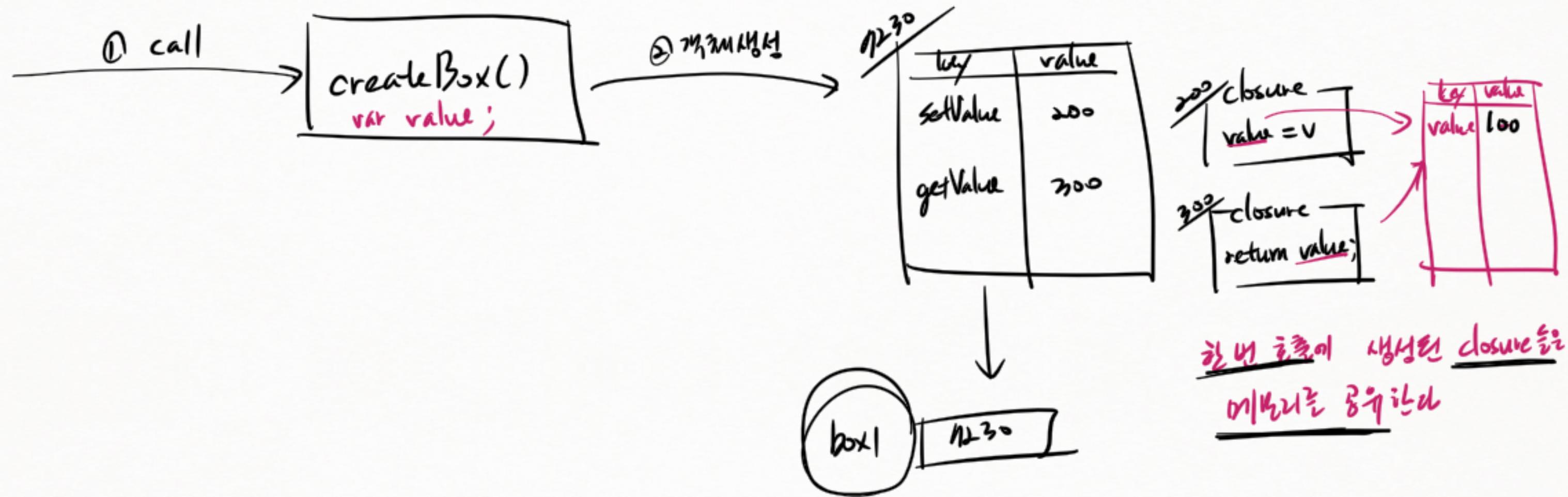
## \* closure № II



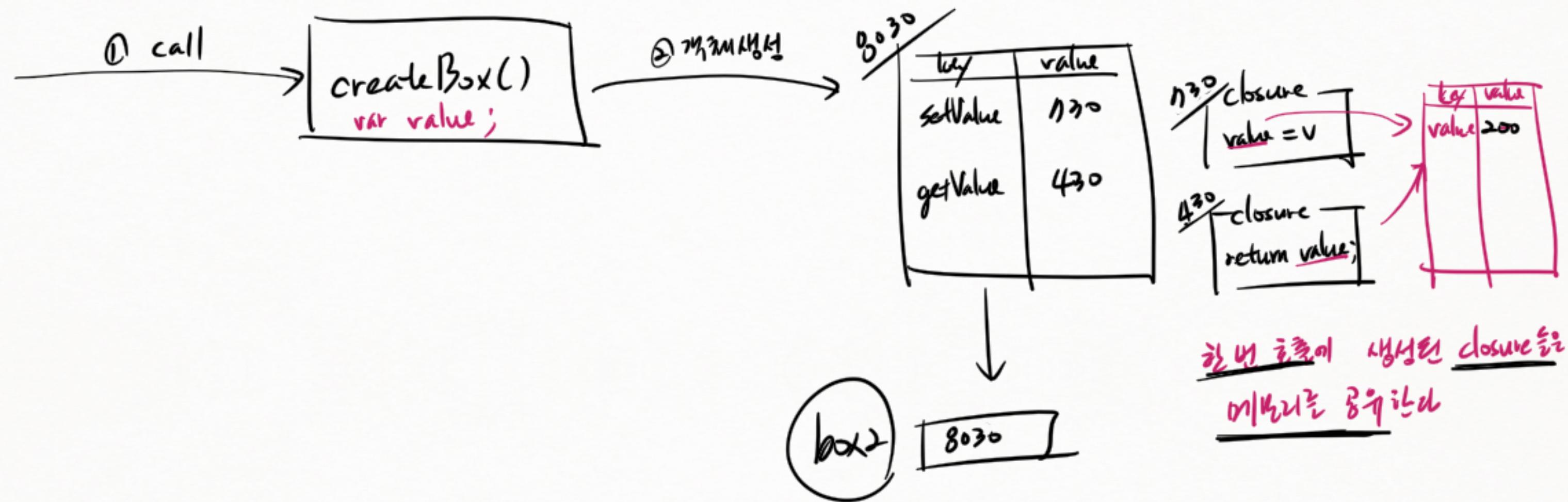
## \* closure № II



\* 4th Ⅲ : box1

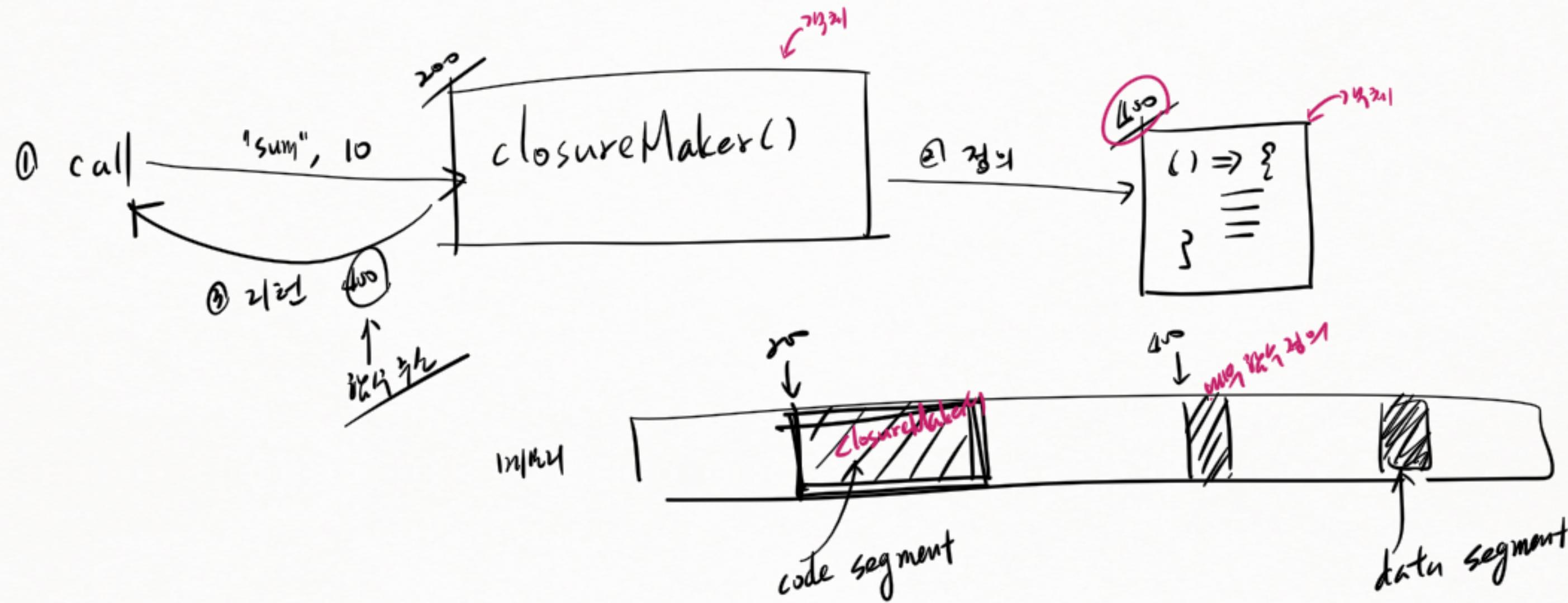


\* 4th Ⅲ : box 2

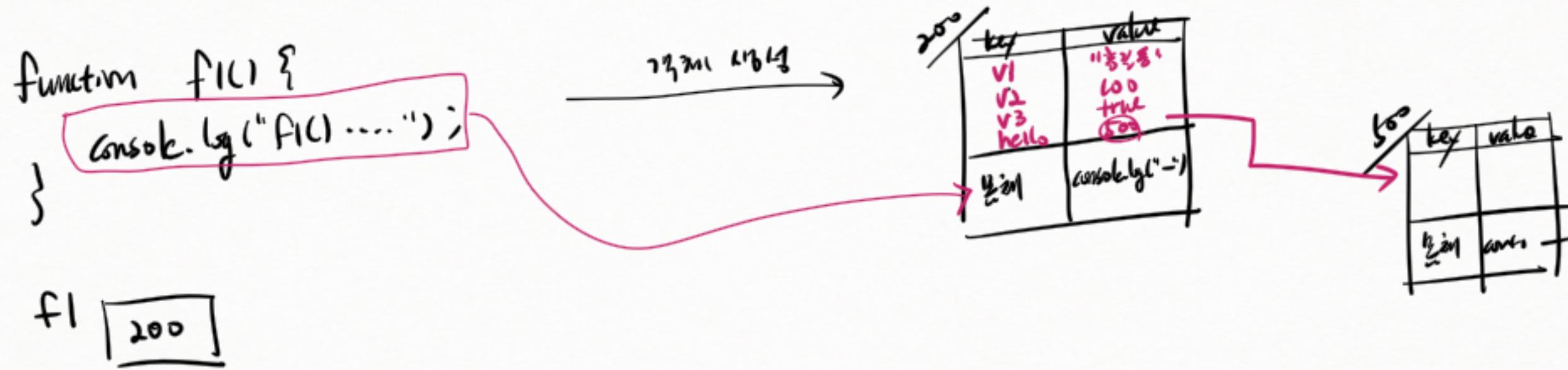


```
function() { return "안녕" }  
          ↓  
console.log( (값주기)() )  
          ↑  
값주기를 가지는 함수 호출  
  
          ( ) => "안녕"  
          ↓  
console.log( (값주기)() )
```

## \* closure 2단



\* 值 (值)



`f1();`  $\xrightarrow{\text{语句块}} \text{值} \Rightarrow \text{值 (function call)}$

`f1.v1 = "Hello";`

`f1.v2 = 100;`

`f1.v3 = true;`

`f1.hello = function() {  
 console.log("Hello!");  
};`

`f1.hello();`  $\xrightarrow{\text{语句块}} \text{值 (值)}$

\* 동기화 와 비동기화  
Synchronize      Asynchronize

Synchronize(동기화)

기본적인 문법  
변수 선언  
함수 정의  
변수 초기화  
결과 출력  
var a = 100;  
var b = 200;  
var result = plus(a, b);  
console.log(result);

Asynchronize(비동기화)

var a = 100;  
var b = 200;  
var result = 0;  
function calculate() {  
 result = a + b;  
}  
window.setTimeout(plus, 5000);  
console.log(result);

호이스팅 예제  
변수  
값 초기화

\* eval()

"JavaScript is"  
↓  
eval( ) → this

\* onclick 속성

HTML의 onclick = 함수명;

↑  
호출할 때마다  
이벤트 핸들러  
 onclick 이벤트  
 메시지 처리  
 응답

event property  
↑  
(callback)  
" " event handler  
" " event listener

Web Browser

## \* JSON.parse()

JSON 풀기

① JavaScript 객체를 문자열로 풀기

이후 브루트 채우기  
여러 브루트 채우기

② 문자열은 " "으로 풀기

③ 프로퍼티명은 문자열로 풀기

④ 헷갈리지 말기

- 같은 브루트 채우기 가능!
- 다른 객체 풀기 가능
- 배열 풀기 가능

가져온 JSON 형식으로 정의한 문자열

JSON.parse()

↑  
JavaScript의  
Built-in 객체

가져온  
가져온 객체

JavaScript  
Object  
Notation

+

JavaScript  
Object Literal  
풀기

## \* Data 포맷 : binary vs text

Data

{ 이름: ABC  
나이: 20  
재직: true

→  
바이트 단위로 구조화 따라  
저장

① binary

이름길이	이름	나이	재직여부
00	03	41	42

...  
163 00 14 01

↳ 이를 때 바이트 규칙에 따라 읽는다

\* 파일 크기가 가장 작은 편이다

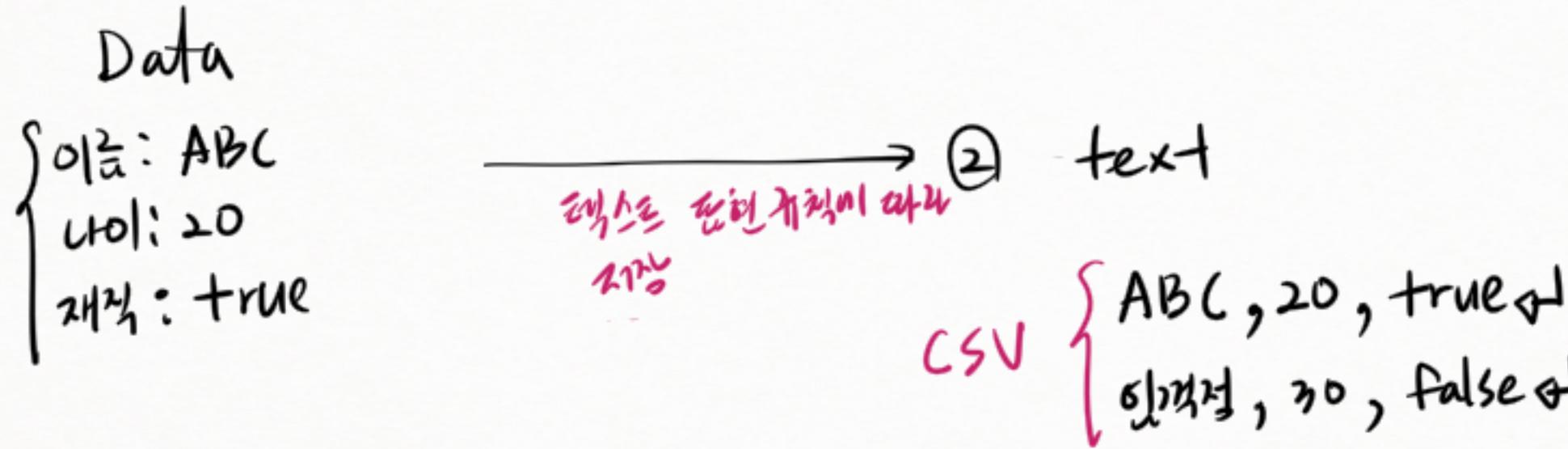
↳ 예? 데이터에 대한 목록이나 설명이 있다  
(meta data)



바이트 저장 규칙을 몰라도  
데이터를 제대로 읽을 수 있다

\* 맥스로 편집기로 데이터를 제대로 볼 수 있다.

## \* Data 포맷 : binary vs text



- \* 텍스트로 데이터를 쉽게 접근할 수 있다
- \* binary 형식에 비해 파일 크기는 커진다.

## \* Text 파일 표기

① CSV

Comma-Separated Value

홍길동, 20, true ↪

임꺽정, 30, false ↪

• 간접적.

• 한 칸에 한 데이터

• 각 항목의 정보가 없다

↓  
직접적으로 데이터가 흘러온다

• 매우 간단한 데이터를 다룰 때 좋다

② XML

extensible Markup Language

```
<members>
  <member>
    <name>홍길동</name>
    <age>20</age>
    <working>true</working>
    <schools>
      <school>
        <name>부드천중</name>
        <state>경기</state>
      </school>
      <school>
        <name>부드천고</name>
        <state>경기</state>
      </school>
    </schools>
  </member>
  :
</members>
```

meta data: 태그를 설명하는 태그.

- 개별 구조의 데이터 표현 가능
- 각 항목의 의미를 표현 가능

↓  
특정 항목의 항목을 찾기 쉬워

- data 뿐 metadata가 아니지 않다.  
↓  
파일 크기가 크다.

## \* Text 파일 형식

### ① JSON

JavaScript Object Notation

```
[  
  { "name": "홍길동",  
    "age": 20,  
    "working": true,  
    "schools": [  
      { "name": "마르코폴로", "state": "경기" },  
      { "name": "비즈쿱", "state": "경기" }  
    ]  
  },  
  :  
]
```

- XML 보다 더 간결한 meta data
- JavaScript 와 의사 표로ini 형식이다

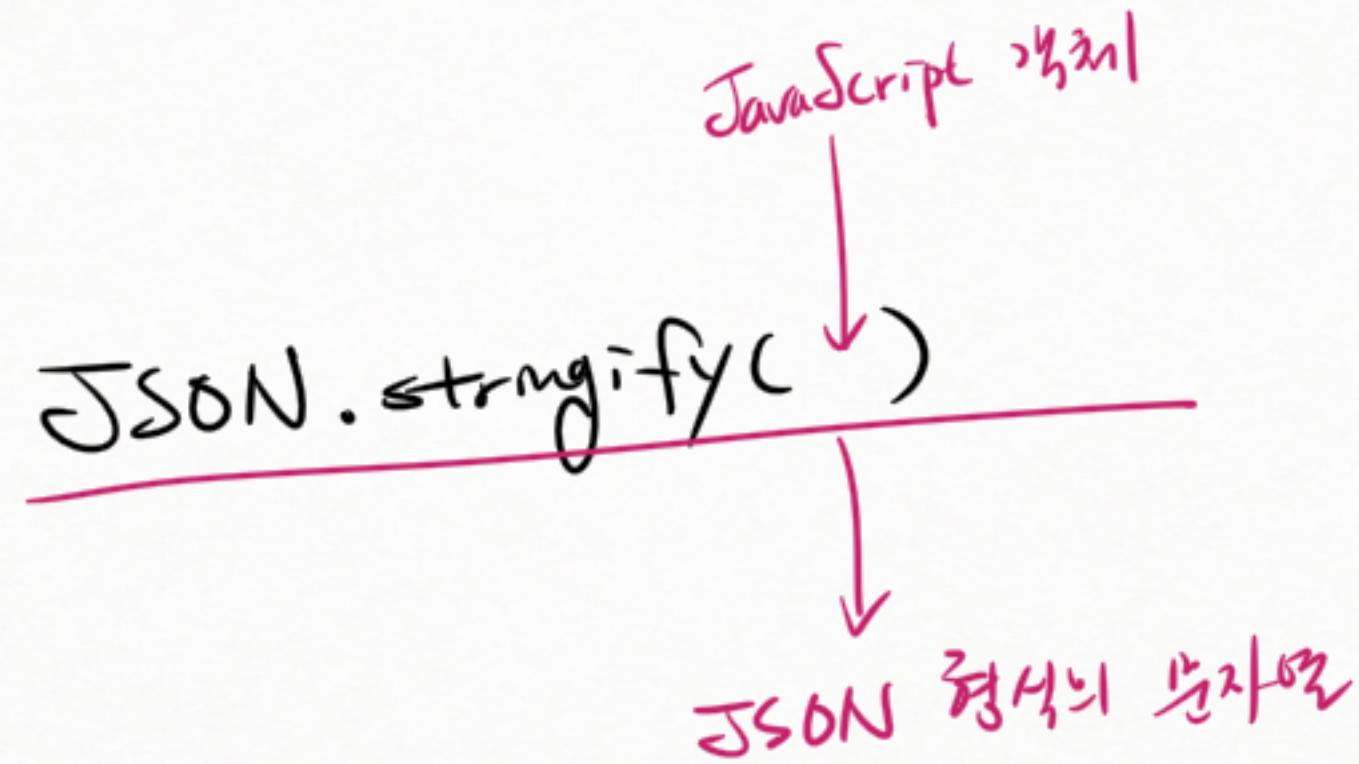
### ④ YAML

Yet Another Markup Language  
↳ YAML ain't markup Language

```
name: 홍길동  
age: 20  
working: true  
schools:  
  - school:  
    name: 마르코폴로  
    state: 경기  
  - school:  
    name: 비즈쿱  
    state: 경기
```

- JSON 보다 더 간결
- 들여쓰기 (indent)로 세이팅 구조를 표현

## \* JSON.stringify()



211 211

\* 12월 105회 ~ 드로잉 106회

① Képalkotásokhoz gyűjtemények (pl: Java, C++, ...)

```
class Student {  
    String name;  
    int age;  
    boolean working;  
}
```

Diagram illustrating static type binding:

```

graph TD
    A["(all: Java, C++, ...)"] -- "Static type binding" --> B["Student obj = new Student();"]
    B -- "Object creation" --> C["obj"]
    C -- "Value" --> D["200"]
    B -- "Object creation" --> E["name: '홍길동'"]
    B -- "Object creation" --> F["age: 20"]
    B -- "Object creation" --> G["working: true"]

```

The diagram shows the creation of a `Student` object named `obj`. The variable `obj` is assigned the value `200`, which is highlighted in red. Below the variable, there is a box containing the value `200`, also highlighted in red. To the right of the assignment, there is a pink bracket grouping `name`, `age`, and `working`, with the label `Object creation` above it. Below this bracket, there are three boxes representing the object's fields: `name` with value `"홍길동"`, `age` with value `20`, and `working` with value `true`.

```
obj.name = "3218"
```

ob). age = 20;

6b). Working = true

obj. tel = "02-111-2222"

Comprile  $\ell_{\eta}^{\pm} = m \ell_{\eta}^{\pm}$

\* JavaScript 와 프로토타이핑 비교

## ② 프로토타이핑 (prototyping) 방법으로 객체 만들기

함수로 접근!



var obj = new Object();

① obj[3]	tx	value
String valueof :	-	-
name	"홍길동"	
age	20	
working	true	

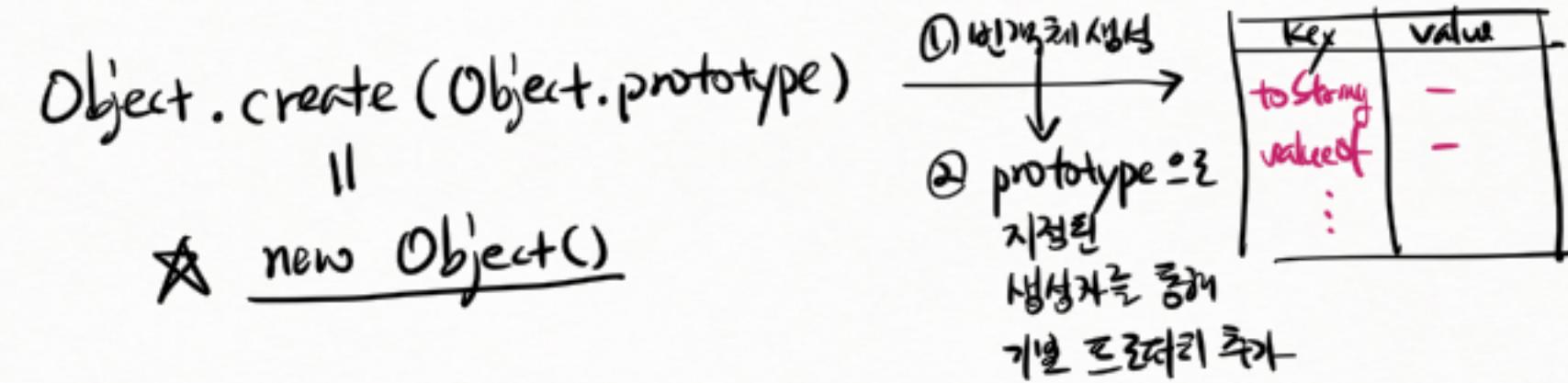
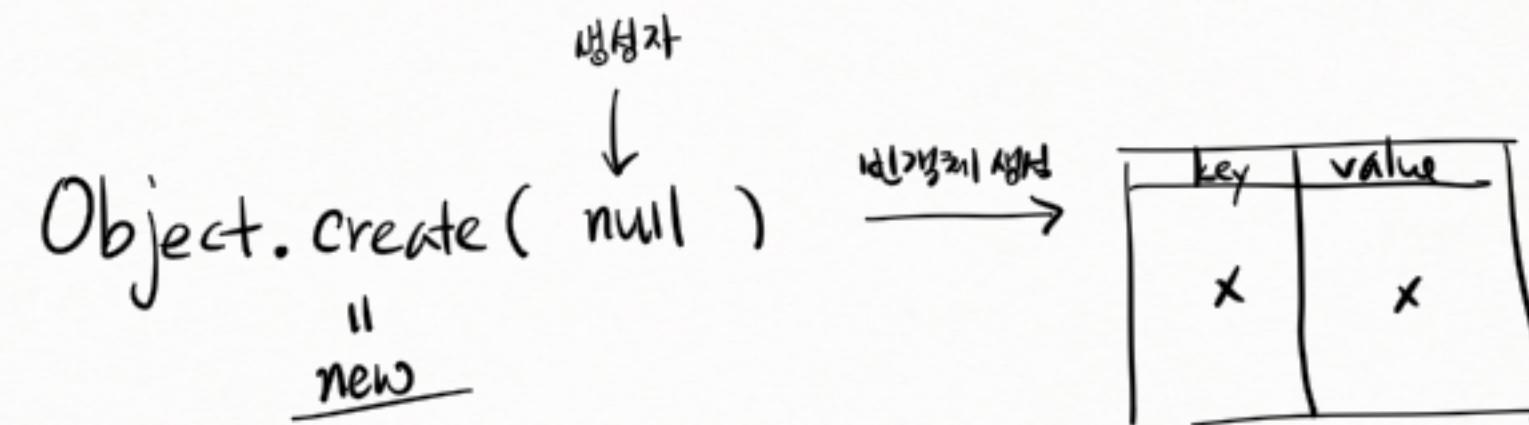
obj.name = "홍길동";

obj.age = 20;

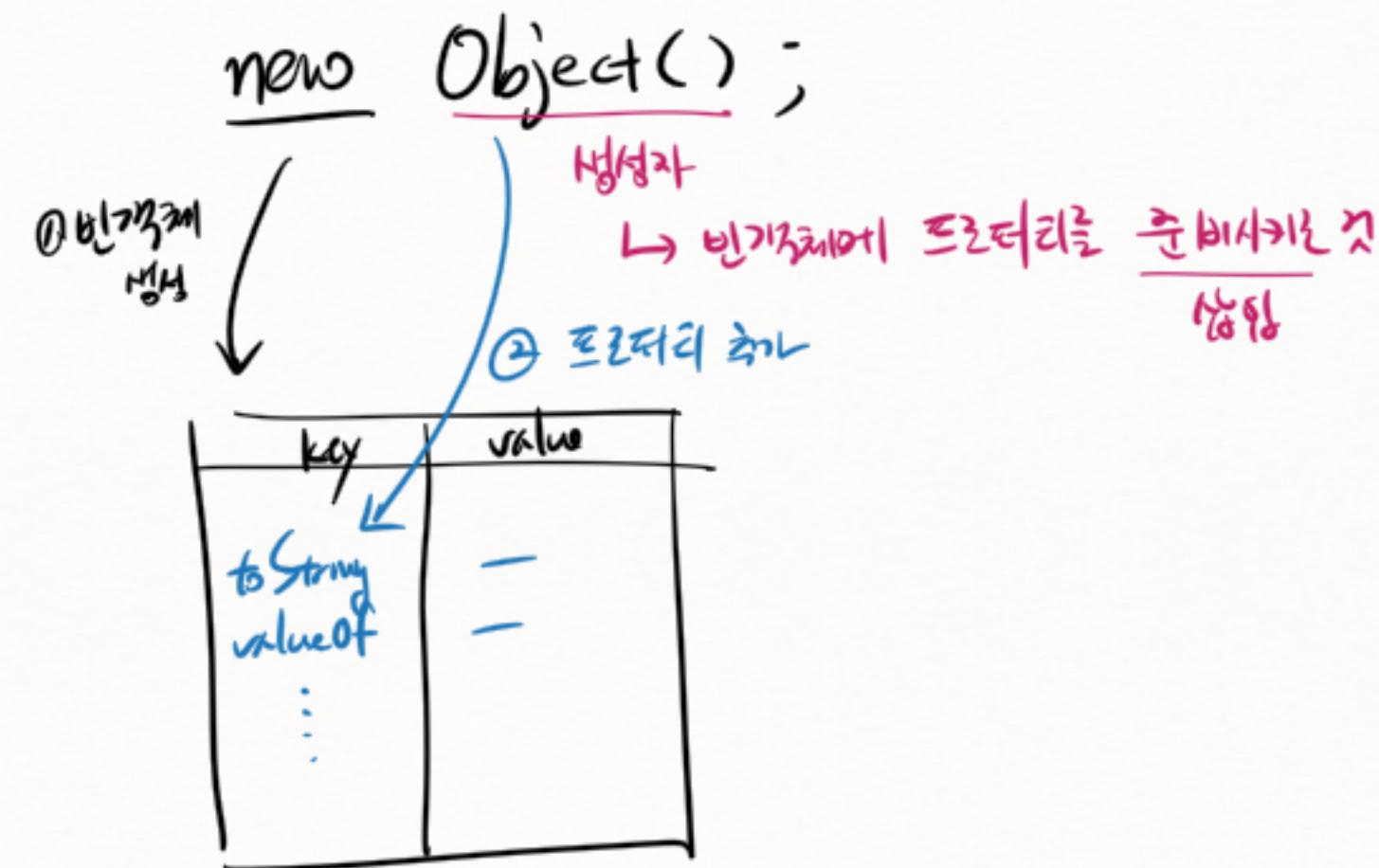
obj.working = true;

기존에 있는 속성  
기존에 있는 속성을 찾는다  
기본 프로퍼티 추가

## \* 객체 생성



## \* 생성자 (constructor)



\* hasOwnProperty ("프로퍼티인가")

↓  
기존에 추가시킨 프로퍼티인가 검사

var obj = new Object()

obj.hasOwnProperty("toString") → false  
" " ("valueOf") → false

obj.plus3 = () => {};  
↑  
값이 없음

obj.title = " ";  
obj["content"] = " ";  
obj["viewCount"] = " ";  
obj.plus1 = f1;  
obj.plus2 = function() { };  
↑  
값이 없음

key	value
toString	-
valueOf	-
:	-
title	-
content	-
viewCount	-
plus1	-
plus2	-
plus3	-

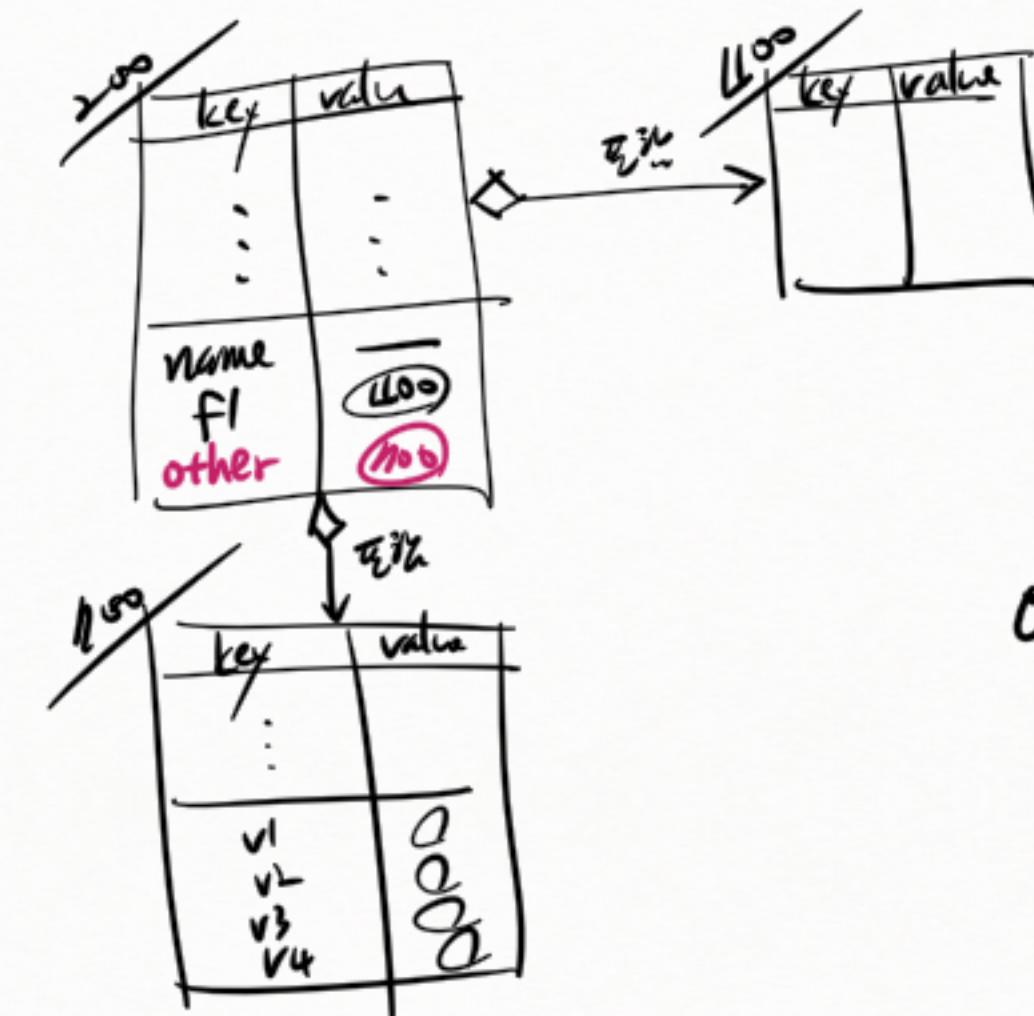
\* گیگانی اتے چھپاں کیوں نہیں

let obj = new Object();  
100

obj.name = "—";  
obj.fl = () => {};

let obj2 = new Object();

obj2.v1 = 0;  
obj2.v2 = 0;  
obj2.v3 = 0;  
obj2.v4 = 0;



obj.other = obj2;  
100  
ئىچىكى

\* think of this

100

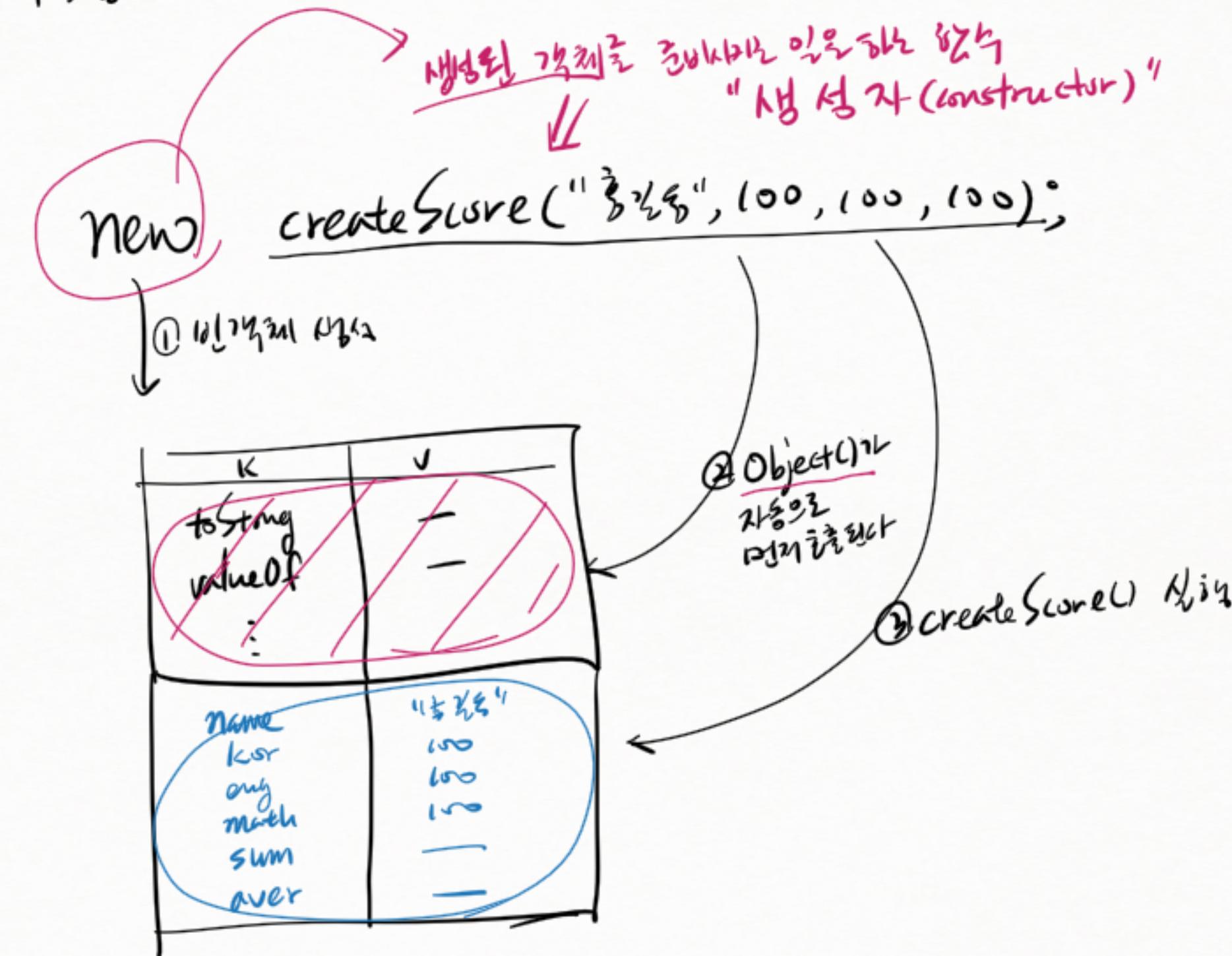
K	V
name	-
kor	-
eng	-
math	-
toString	200

100

K	V
100	100

obj.toString = function() {  
 return this.name + ...;  
}  
  
100 → return this.name 100.

\* new 484



## \* Higher prototype

`<생성자>`  
**Score()**

prototype

K	V
sum	<code>f(){} -&gt;</code>
aver	<code>f(){} -&gt;</code>

*Score()가 더 높은 prototype!*  
*즉, 더 높은 prototype*  
*보다 더 높은 prototype!*

`new`

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

`var scores = [200, 300, 400]`

`scores[0].sum();`

↓ call

`Score.prototype.sum()`

`scores[0].aver();`

↓ call

`Score.prototype.aver()`

`scores[2].sum();`

↓ call

`Score.prototype.sum()`

\* 생성자와 일상기능

function f() {}

var obj = new f();  
      ↑  
      new가 생성자임을 알기

① new ~~250~~  $\xrightarrow{\text{Object}}$  K V  
      ↓  
② Object  $\xrightarrow{\text{toString}}$  toString value  
      ↓  
③ f()  $\xrightarrow{\cdot}$  . .

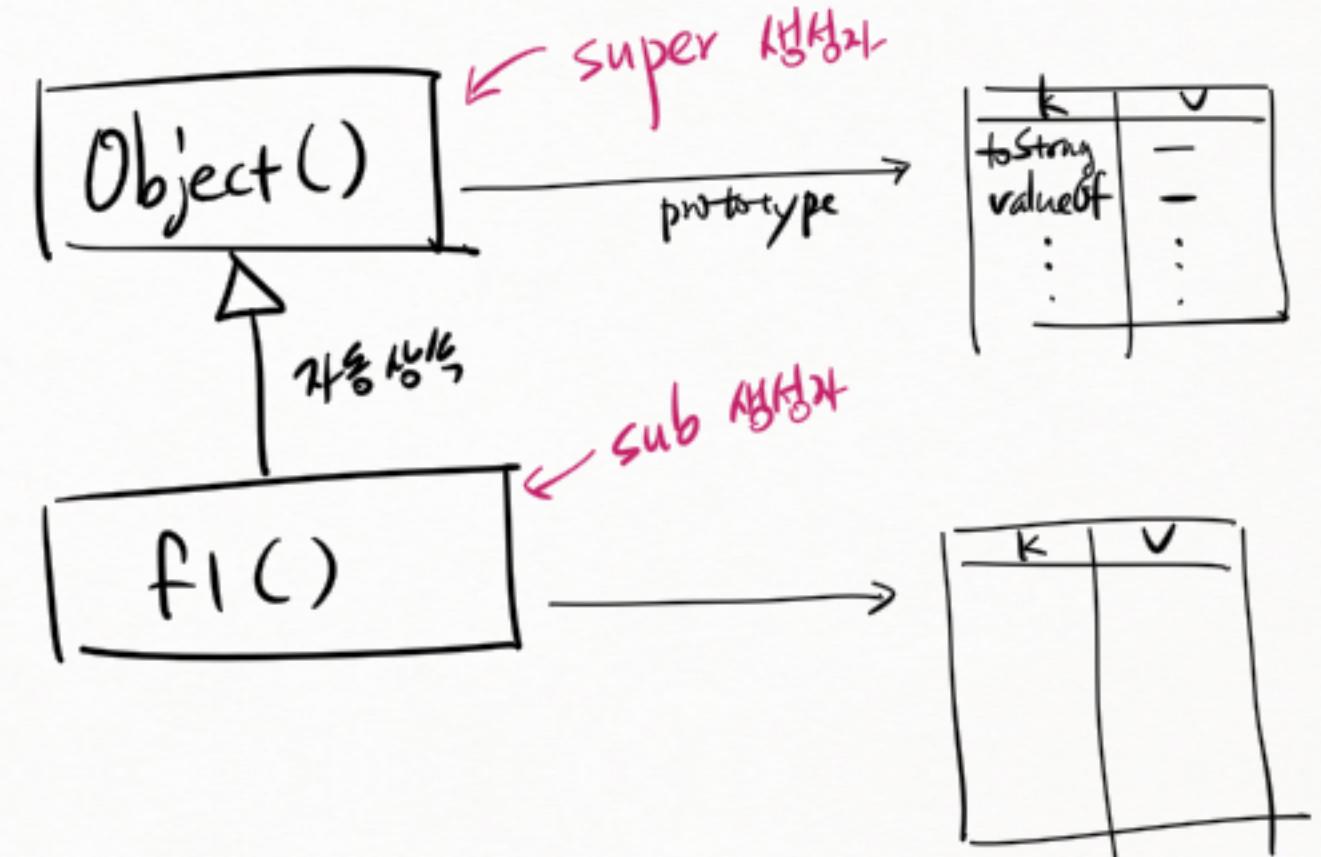
K	V
toString	—
value	—
:	:
.	.

var obj = f();  
      ↑  
      undefined

## \* 생성자와 Object()

`var obj = new f1();`

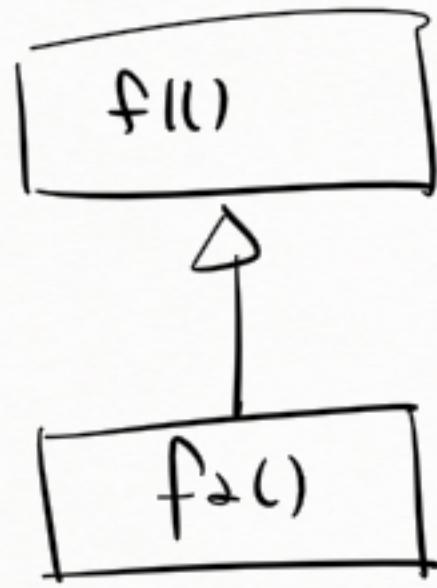
- ① new: 빈 객체 생성
- ② f1()의 super 생성자 호출  
↳ Object()
- ③ f1() 호출



obj. toString()

- ① 기본적인 출력.
- ② 생성자 f1.prototype에서 출력.
- ③ super 생성자 Object.prototype에서 출력.

## \* 생성자를 상속하기



```

f1(n) {
    this.name = n;
}
  
```

f2(n, k, e, m) ?

f1(n); ← 일반 함수 호출방법으로는 f2()가 this 라는

f1.call(this, n);  
this.kor = k;  
this.eng = e;  
this.math = m;

}

변수의 받았을 때까지 주소를  
f1()에 전달할 때마다

var obj = 제작자 주소 전달

new f2("홍길동", 100, 90, 80);

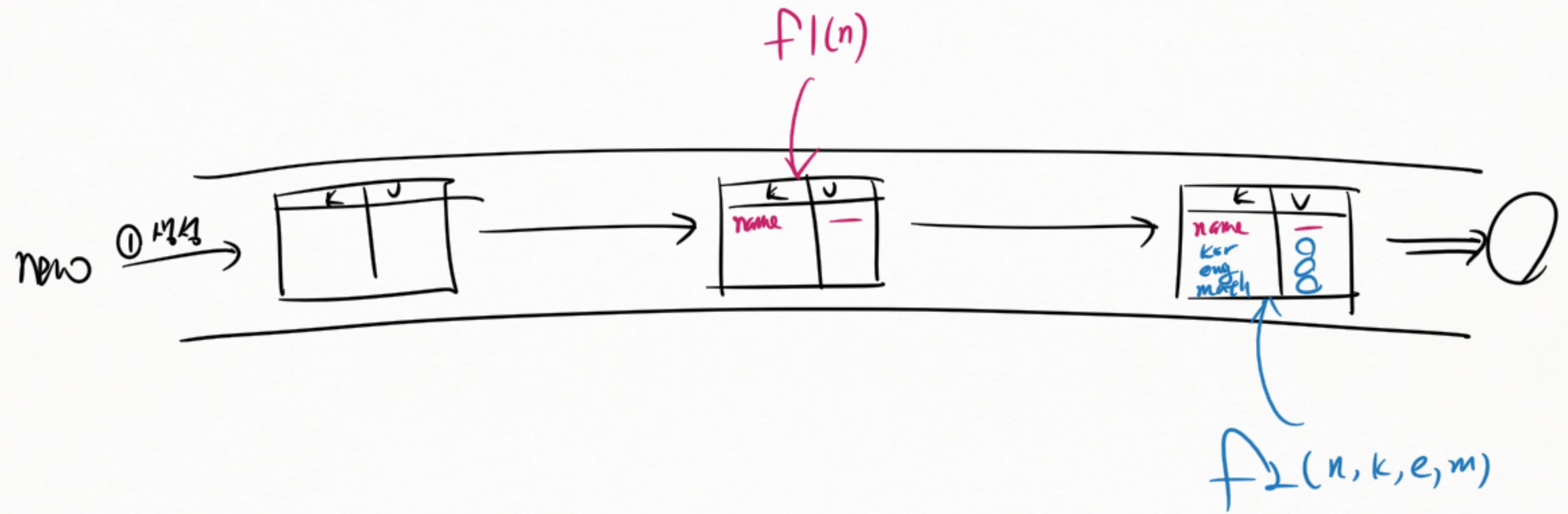
<u>200</u>	K	V

## \* 함수를 호출하는 방법

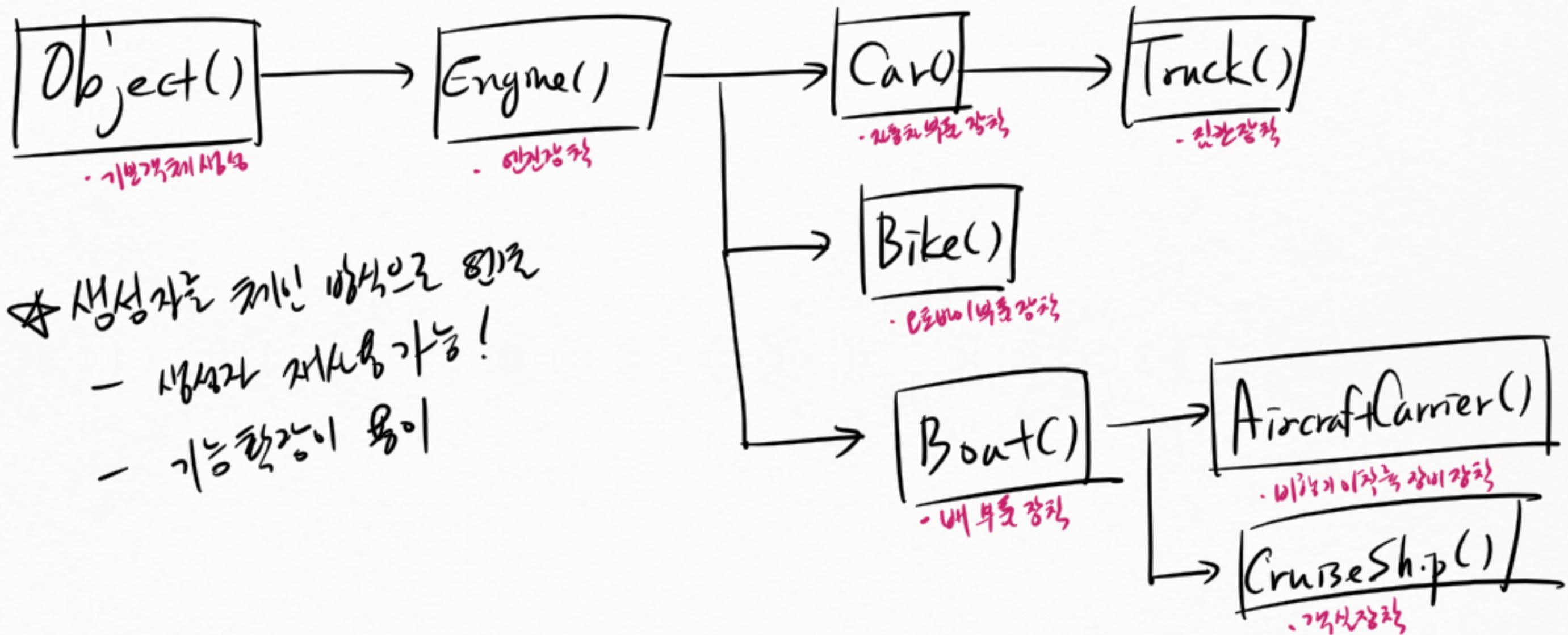
① 함수명(아구먼트, ...);  
예) f1();

② 함수명.call(개체주소, 아구먼트, ...);  
예) f1.call();

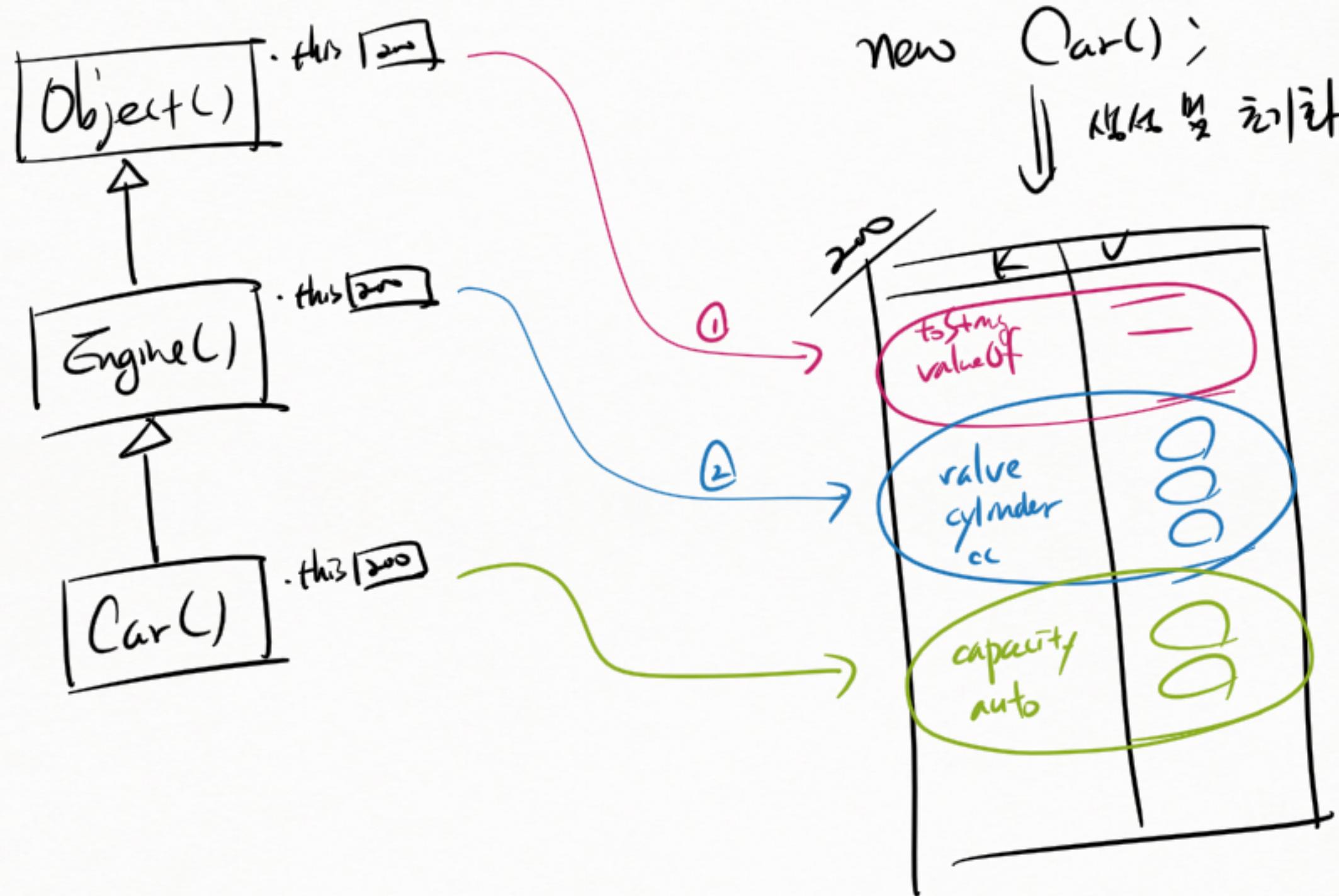
\* 생성자와 전파하여 배운



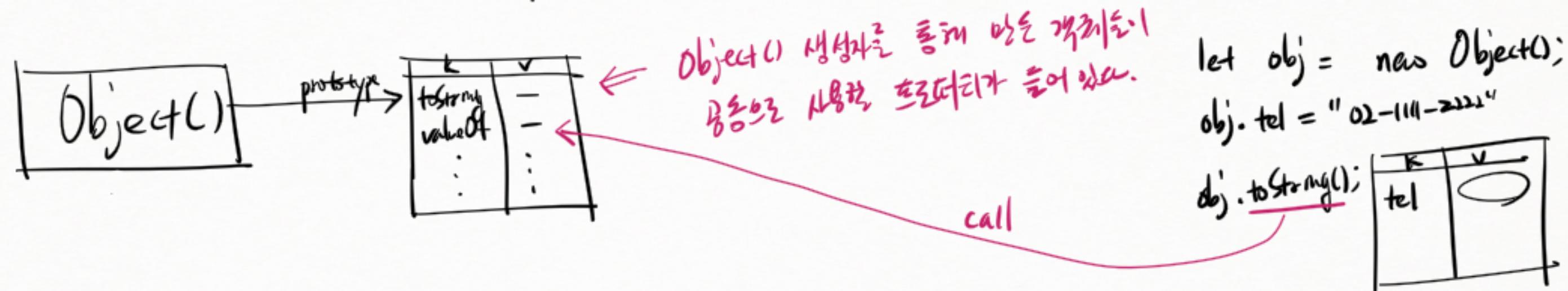
\* 생성자를 차인으로 초기화되어 super-sub로 만드는 이유?



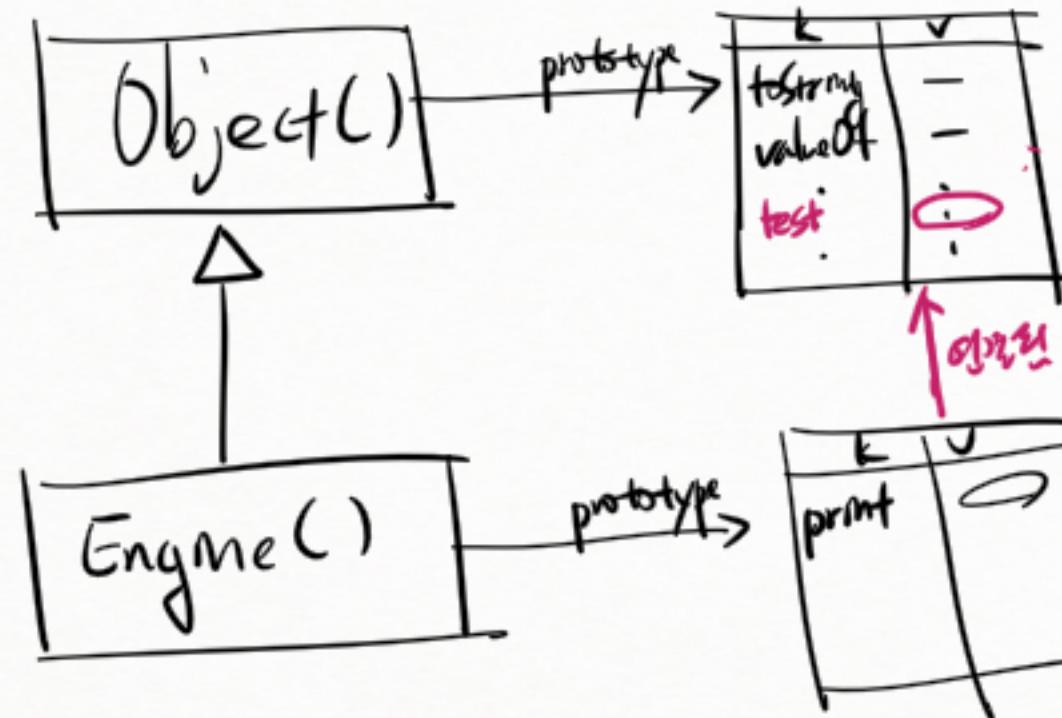
\* 생성자 체인 예:



## \* 자식 생성자의 prototype 사용하기



## \* 자식 생성자의 prototype 활용



`Engine()` 생성자는 `__proto__`를  
기반으로 `Object`의 프로토 타입을 상속

`let el = new Engine(...)`

K	V
value	0
cylinder	0
cc	0

`el. test();`

↳ `Object.prototype.test()`

`el. print();`

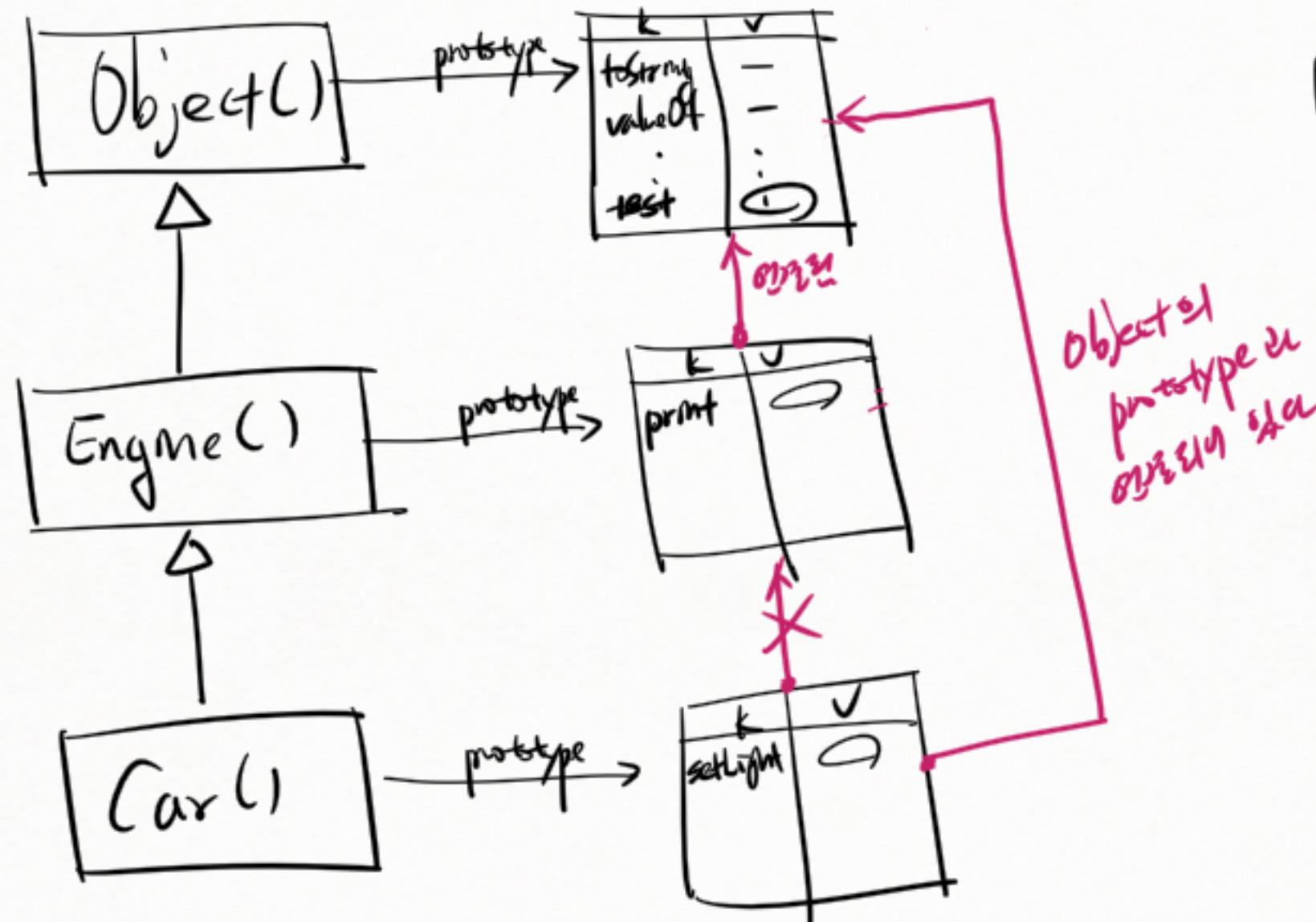
↳ `Engine.prototype.print()`

{ ① `Object.prototype.test()`는 정의되어 있지 않음

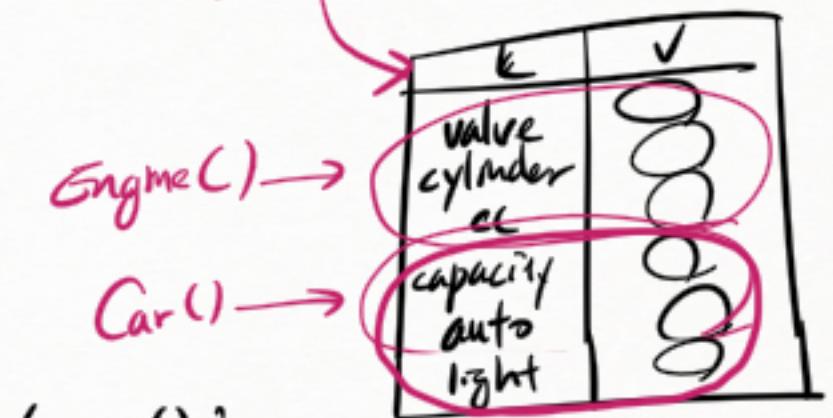
② `Engine.prototype`에 정의되어 있음

③ `Engine` 생성자의 `prototype`에 정의되어 있음.

\* 자식 생성자의 prototype 활용



`let c1 = new Car(...);`



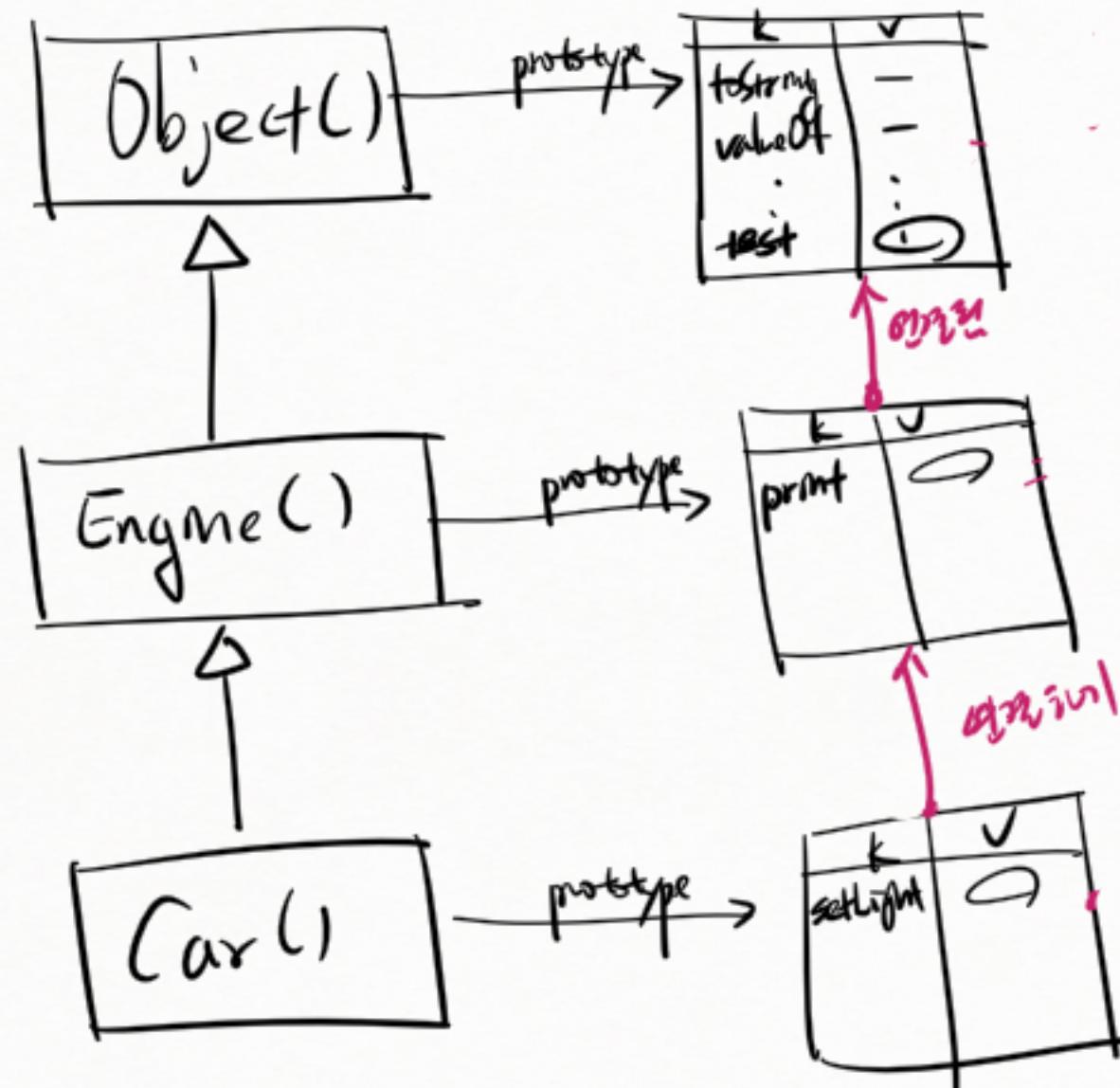
`c1.toString();`  
↳ Object.prototype.toString()

`c1.setLight();`  
↳ Car.prototype.setLight()

`c1.print();`  
↳ ~~Engine.prototype.print()~~

Car() 생성자 허무|무기  
Engine.prototype이 연결되어 있지 않음

## \* 자바스크립트의 prototype 기관



Object.setPrototypeOf(  
Car.prototype,  
Engine.prototype);  
Object.setPrototypeOf(  
Engine.prototype,  
Object.prototype);