

## \* 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

$f(\underline{10});$

function     $f(a, b)$  {

}

$\equiv$

arguments = [10]

- 모든 함수에 두 가지는 Built-in 모듈
- 아규먼트 모두를 디폴트로 설정

## \* 아규먼트와 파라미터

$f(10, 20);$

function       $f(a, b) \{$

$\equiv$

}

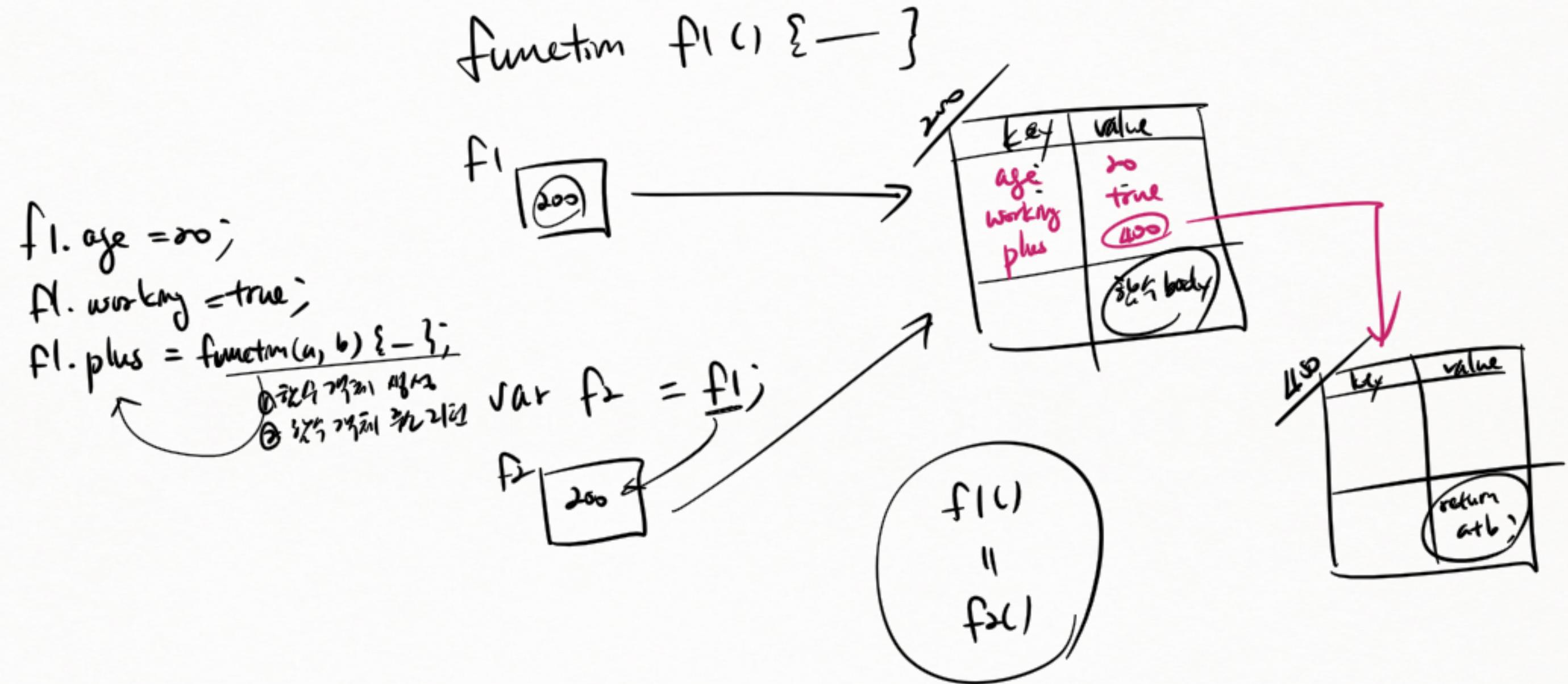
arguments = [10, 20]

## \* 아규먼트와 파라미터

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

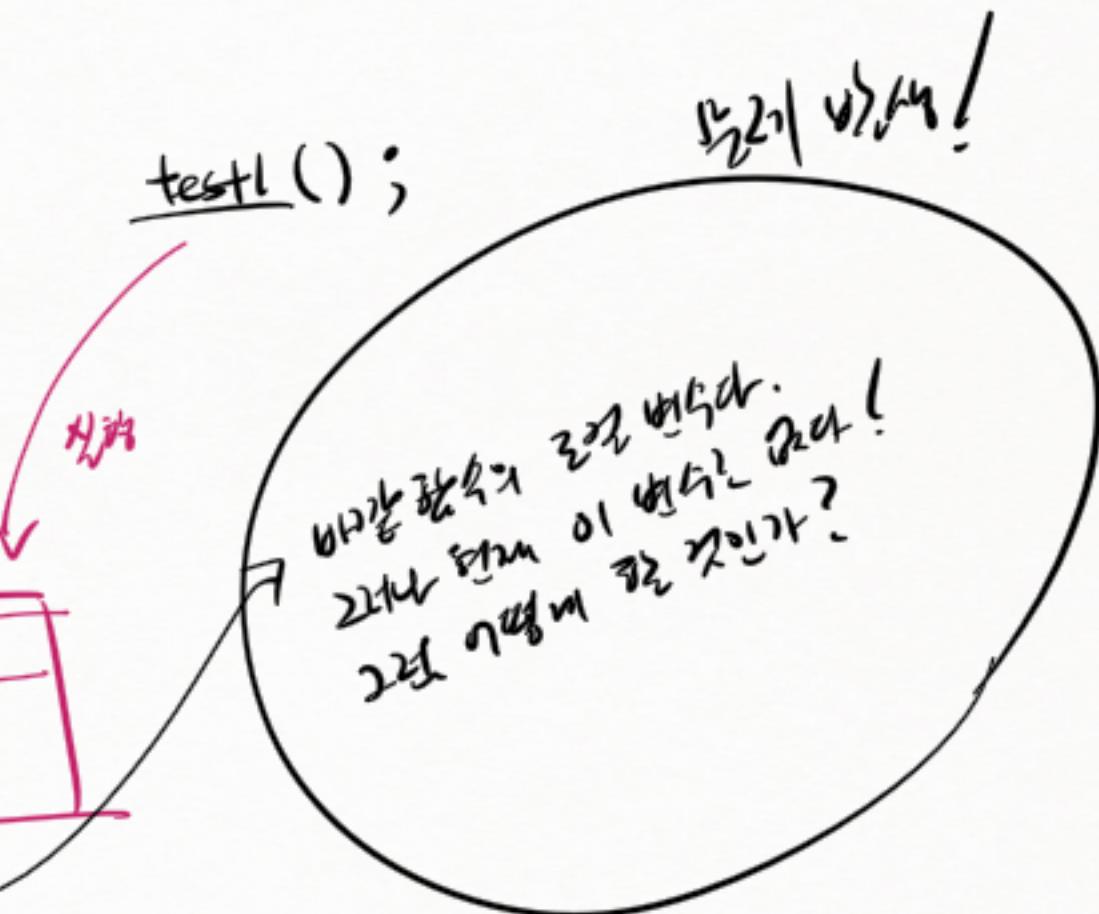
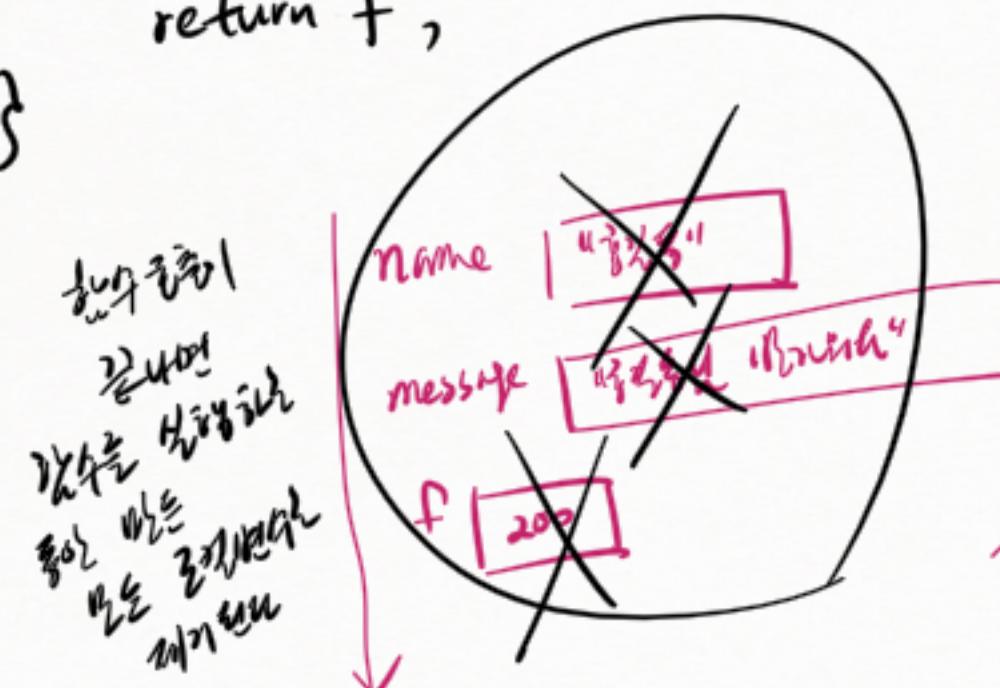
function       $f(a, b)$  {  
     $\equiv$                   arguments = [10, 20, 30, 40]  
}

\* 흡수와 레퍼런스



## \* closure

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



```
var test1 = createGreeting("3218");
```

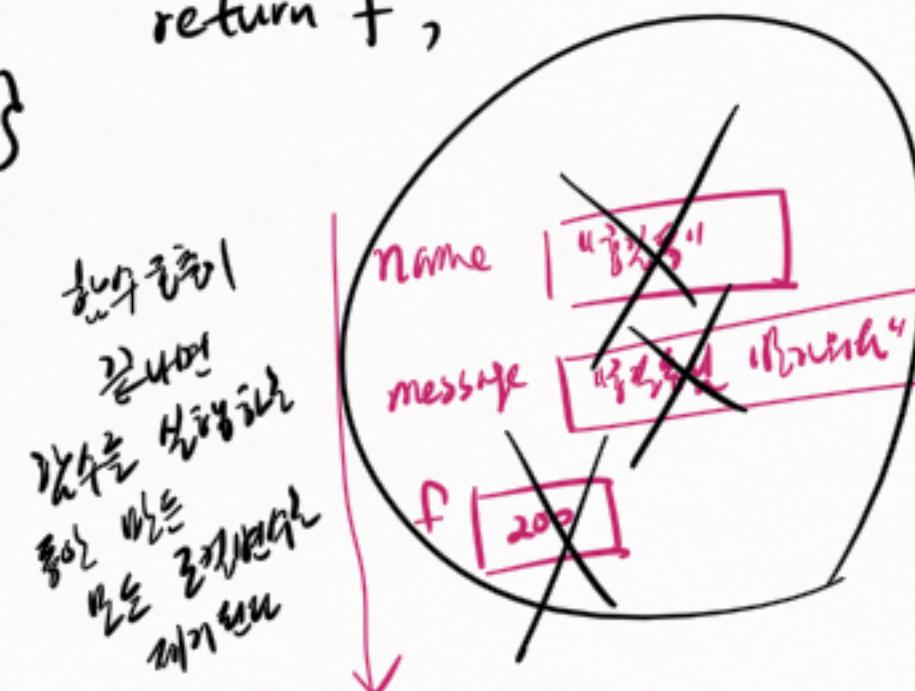
test1 → 200

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  
내부 함수의 로컬변수가  
아니면,  
클로저의 내부를  
복제해온다.

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

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

test1

200

key	value
	console.log (message);

test1()

test2()

```
var test2 = createGreeting ("Dinner");
```

test2

300

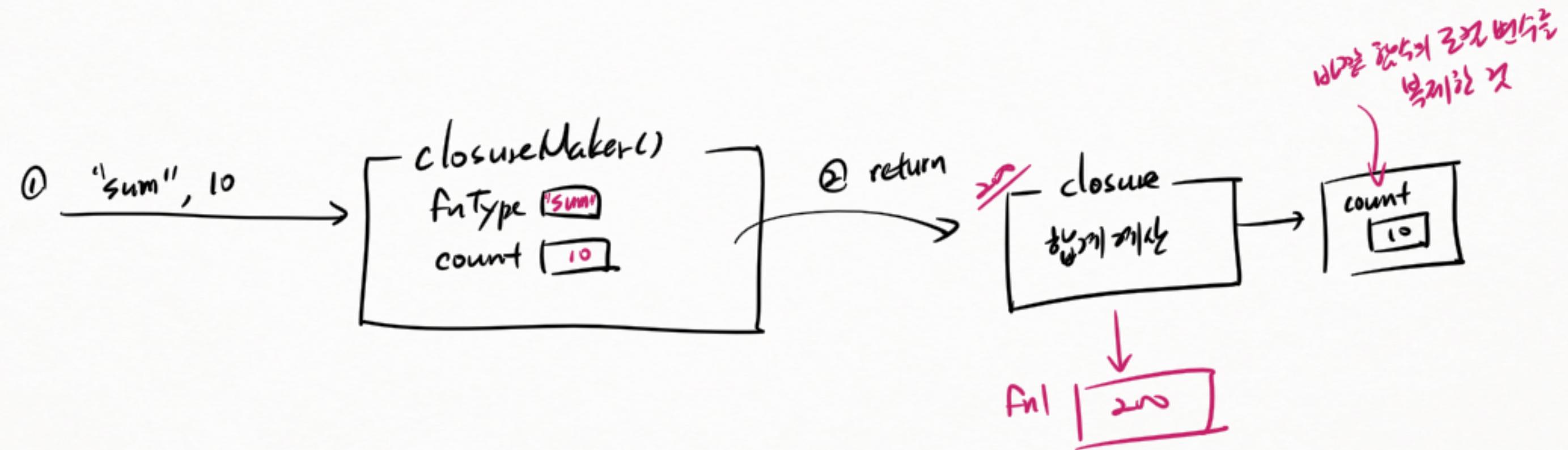
key	value
	console.log (message);

closure 例 2  
variable = 3210392 が 2 つ

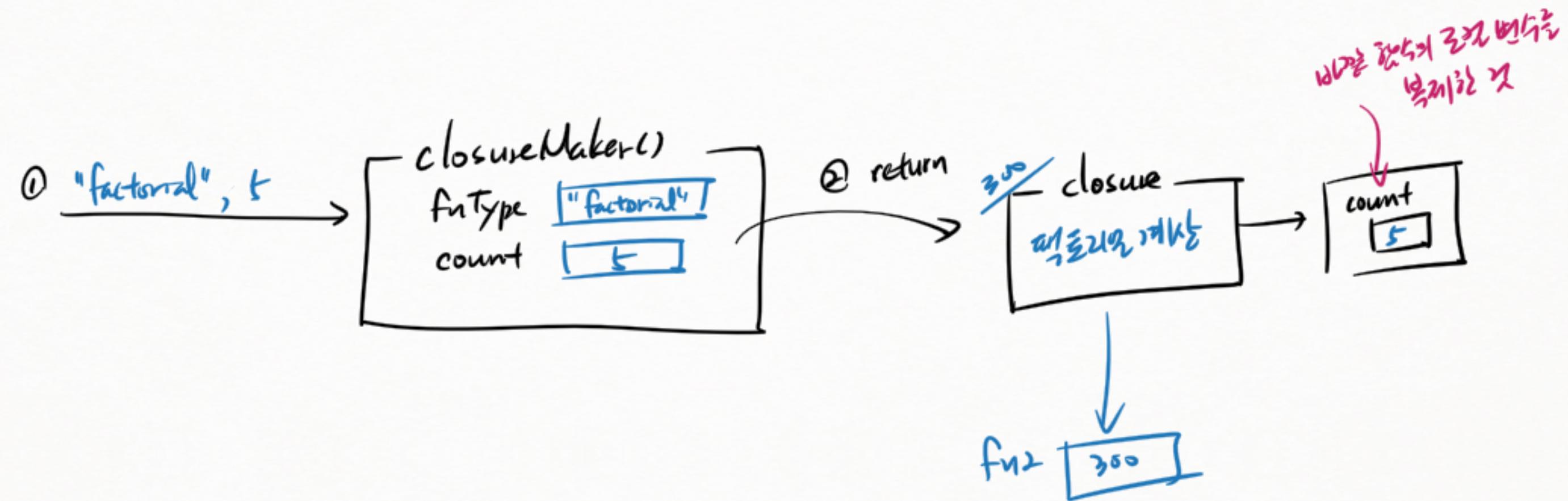
key	value
message	"Hello World"

key	value
message	"Goodbye World"

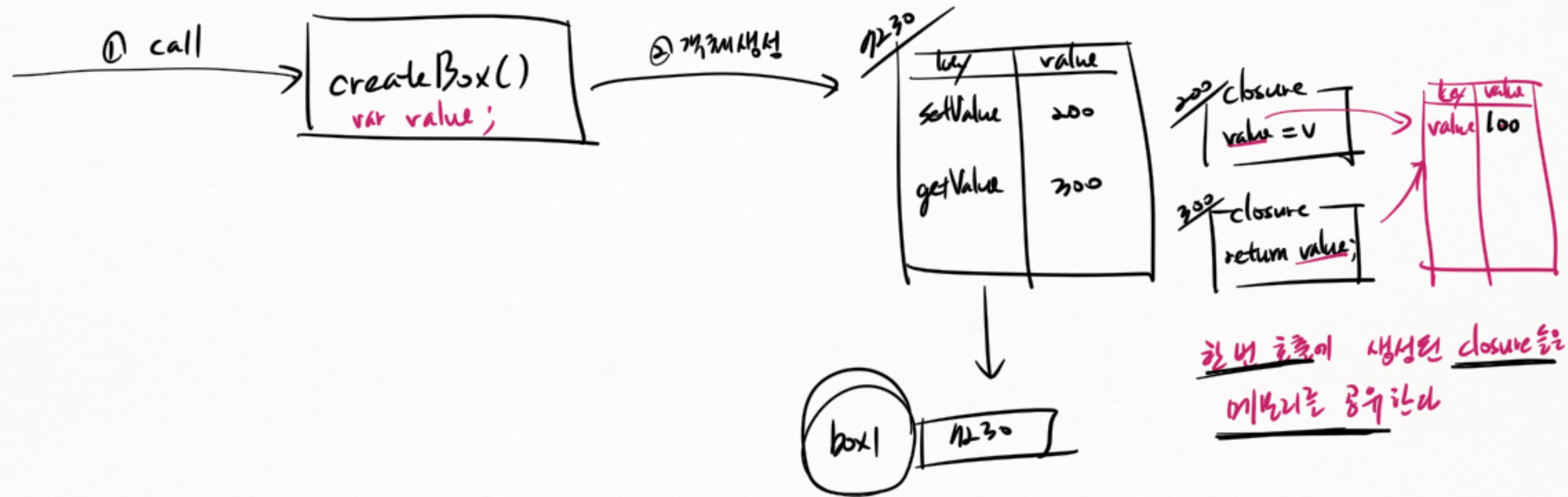
## \* closure № II



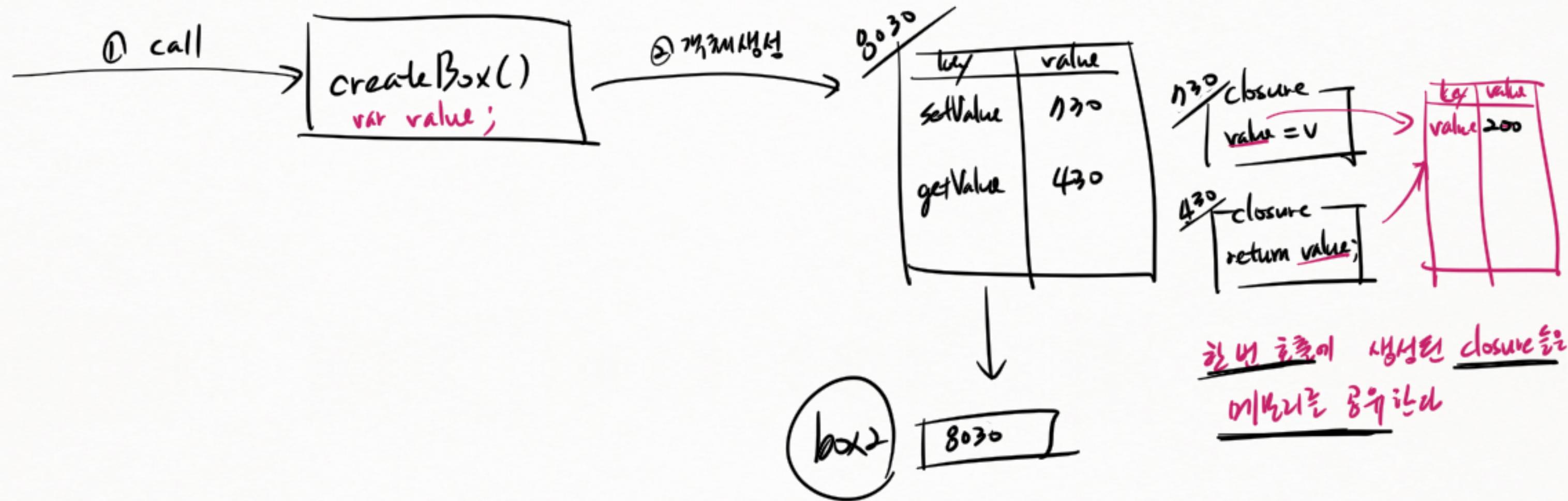
## \* closure № II



\* 4th Ⅲ : box1

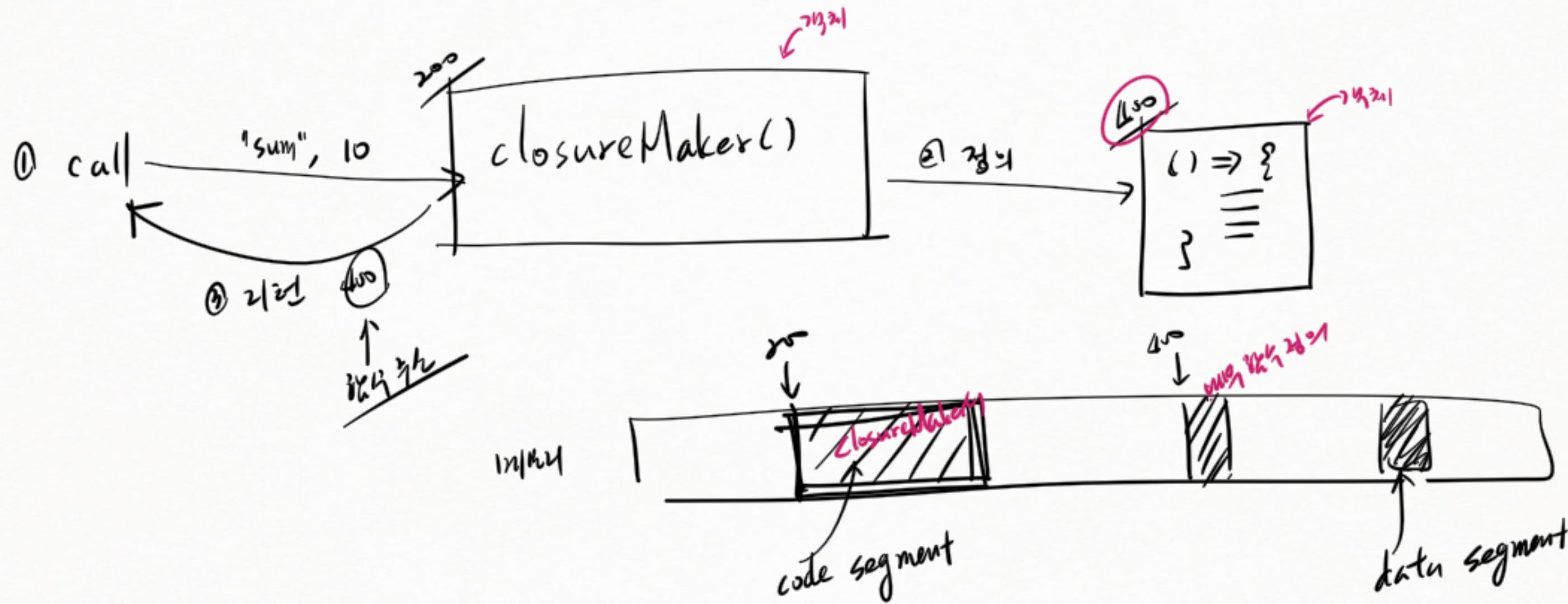


\* 4th Ⅲ : box 2

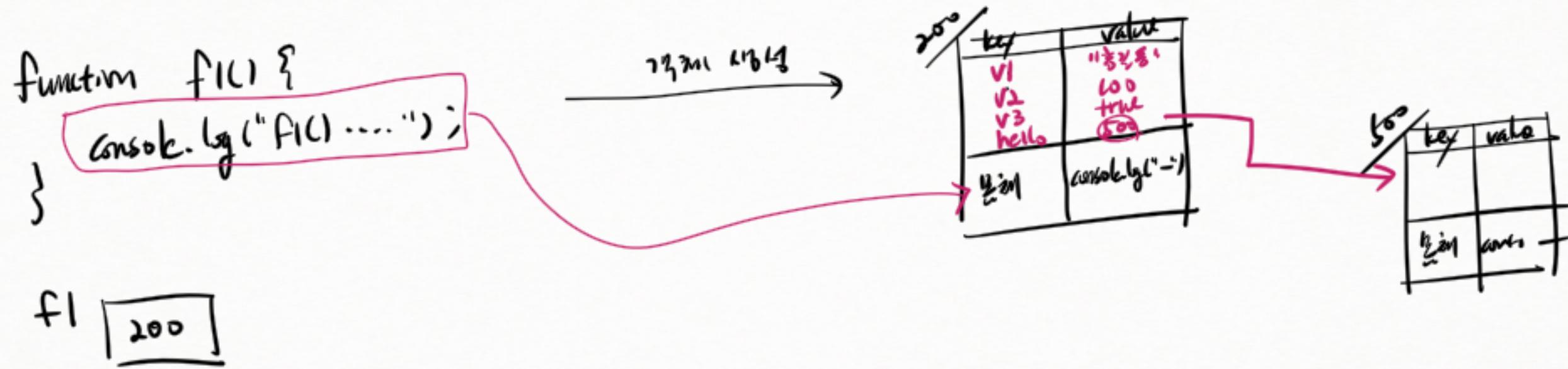


```
function() { return "안녕" }  
          ↓  
console.log( (값주기)() )  
          ↑  
값주기를 가지는 함수 호출  
  
          ( ) => "안녕"  
          ↓  
console.log( (값주기)() )
```

## \* 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?  
이후 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

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

↳ 이를 때 바이트 규칙에 따라 읽는다

\* 파일 크기가 가장 작은 편이다

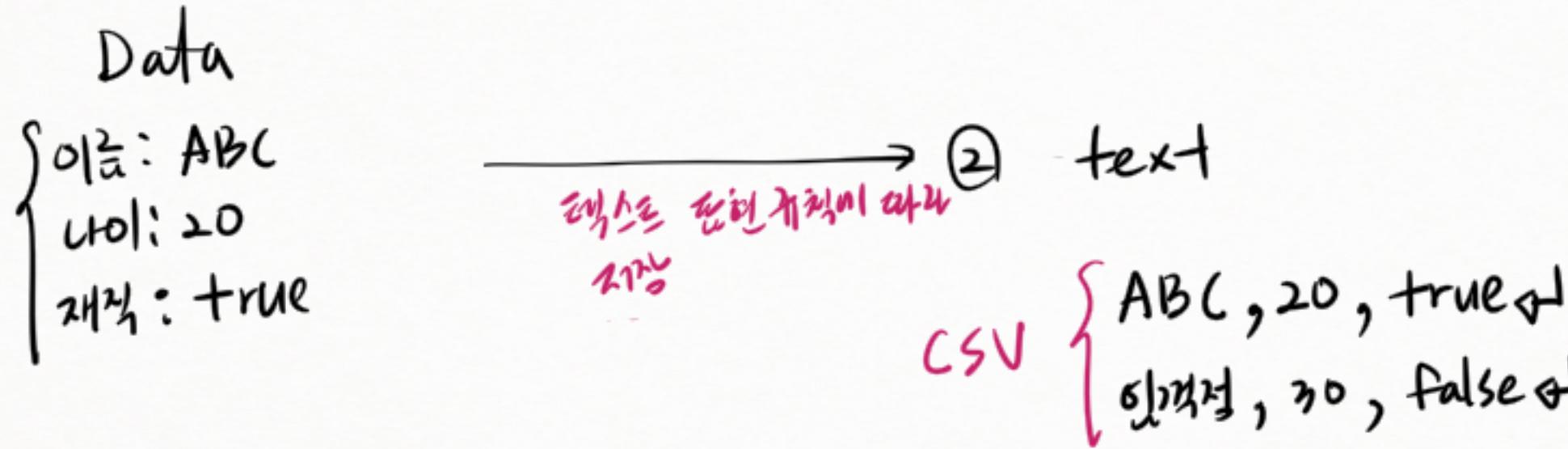
↳ 예? 데이터에 대한 목록이나 설명이 있다  
(meta data)



바이트 저장 규칙을 몰라도  
데이터를 차대로 읽을 수 있다

\* 맥스로 편집기로 데이터를 차대로 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 다룬다.

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 커다.

## \* 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  
언어로의 암호화

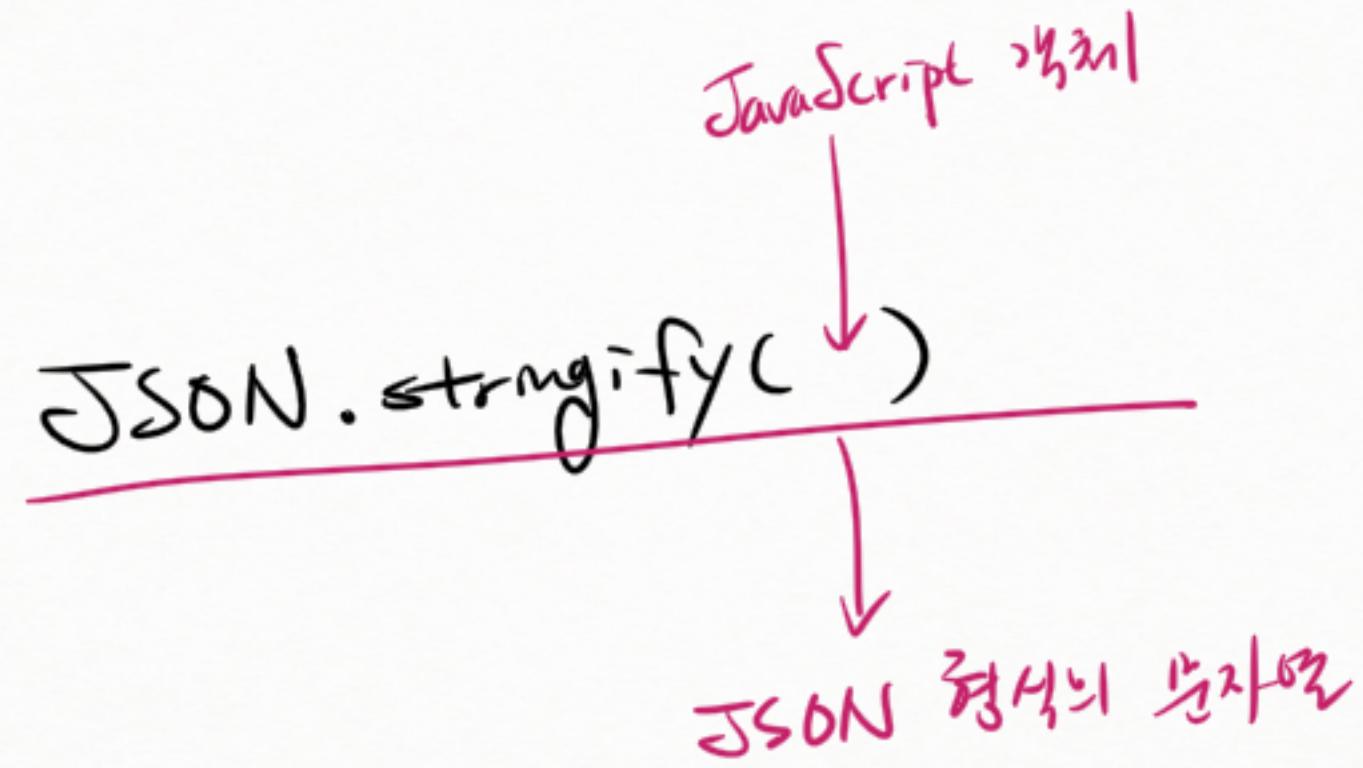
### ④ YAML

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  
세이프 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 separate boxes: one for `name` containing `"홍길동"`, one for `age` containing `20`, and one for `working` containing `true`. Each of these three boxes is also highlighted in red.

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

obj). age = 20;

6b). Working = true

obj. tel ~~= "02-111-2222"~~

Complex  $e^{2\pi i} = -1$

\* JavaScript 와 프로토타이핑 비교

## ② 프로토타이핑 (prototyping) 방법으로 객체 만들기

함수로 접근!



```
var obj = new Object();
```

① 비주얼 형태	② 기본 프로퍼리 추가										
<p>Object</p> <table border="1"><thead><tr><th>key</th><th>value</th></tr></thead><tbody><tr><td>String valueOf etc</td><td>- -</td></tr><tr><td>name</td><td>"홍길동"</td></tr><tr><td>age</td><td>20</td></tr><tr><td>working</td><td>true</td></tr></tbody></table>	key	value	String valueOf etc	- -	name	"홍길동"	age	20	working	true	<p>기본 프로퍼리 추가 할 때마다 같은 기본 프로퍼리가 생성되는 것</p>
key	value										
String valueOf etc	- -										
name	"홍길동"										
age	20										
working	true										

```
obj.name = "홍길동";
```

```
obj.age = 20;
```

```
obj.working = true;
```