

# 변수의 유효범위와 클로저

코드블록

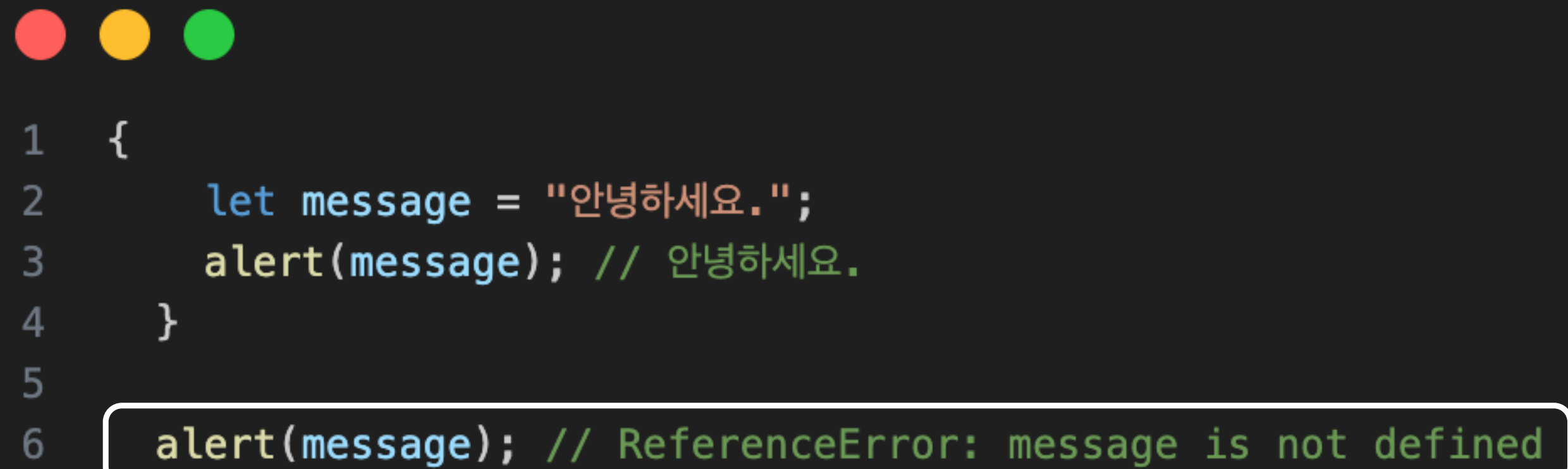
중첩함수

렉시컬 환경

클로저

## 코드 블록

코드 블록 {...} 안에서 선언한 변수는 블록 안에서만 사용



```
1 {
2   let message = "안녕하세요.";
3   alert(message); // 안녕하세요.
4 }
5
6 alert(message); // ReferenceError: message is not defined
```

## 코드 블록

코드 블록 {...} 안에서 선언한 변수는 블록 안에서만 사용

```
1  let phrase = "Hello";  
2  
3  if (true) {  
4    let user = "John";  
5  
6    function sayHi() {  
7      alert(`${phrase}, ${user}`);  
8    }  
9  }  
10  
11  sayHi(); //에러발생  
12
```

if, for, while 등에서도 마찬가지로 {...} 안에서 선언한 변수 오직 블록 안에서만 접근 가능

## 중첩 함수

함수 내부에서 선언한 함수



```
1 function sayHiBye(firstName, lastName) {  
2  
3     // 헬퍼(helper) 중첩 함수  
4     function getFullName() {  
5         return firstName + " " + lastName;  
6     }  
7  
8     alert( "Hello, " + getFullName() );  
9     alert( "Bye, " + getFullName() );  
10  
11 }
```

- 외부 변수에 접근해 이름 전체를 반환해주는 중첩 함수
- 자바스크립트에선 중첩 함수가 흔히 사용된다.

## 중첩 함수

```
1 function makeCounter() {  
2     let count = 0;  
3  
4     return function() {  
5         return count++;  
6     };  
7 }  
8  
9 let counter = makeCounter();  
10  
11 alert( counter() ); // 0  
12 alert( counter() ); // 1  
13 alert( counter() ); // 2
```

호출될 때마다 다음 숫자를 반환해주는 '카운터' 함수

## 중첩 함수



```
1 function makeCounter() {  
2   let count = 0;  
3  
4   return function() {  
5     return count++;  
6   };  
7 }  
8
```

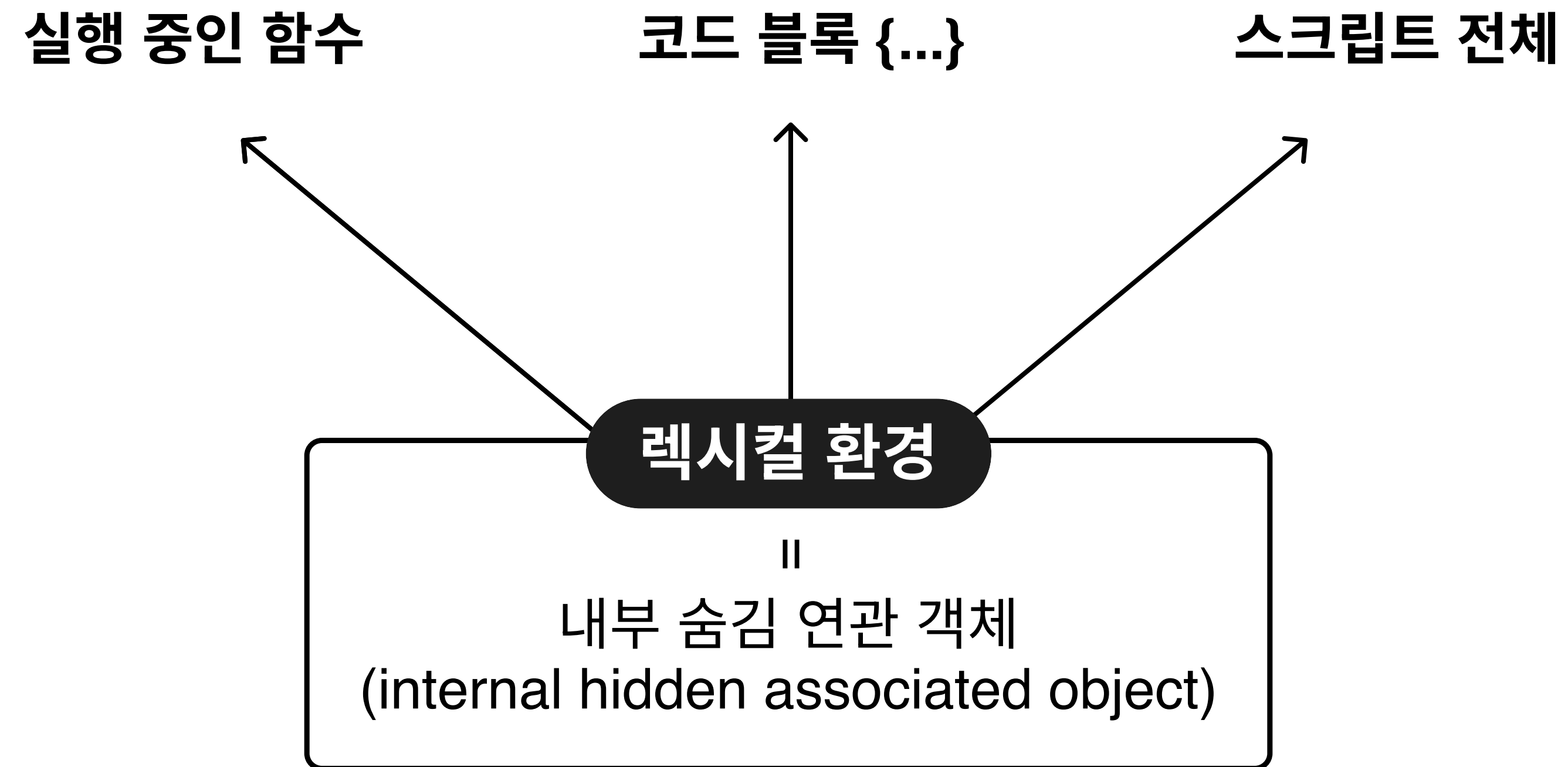
```
9 let counter = makeCounter();  
10  
11 alert( counter() ); // 0  
12 alert( counter() ); // 1  
13 alert( counter() ); // 2
```

counter를 여러 개 만들었을 때,

이 함수들은 서로 독립적일까?

함수와 중첩 함수 내 count 변수엔 어떤 값이 할당될까?

호출될 때마다 다음 숫자를 반환해주는 '카운터' 함수

**렉시컬 환경**

## 렉시컬 환경

### 렉시컬 환경

#### 1. 환경 레코드(Environment Record)

모든 지역 변수를 프로퍼티로 저장하고 있는 객체  
this 값과 같은 기타 정보도 여기에 저장

#### 2. 외부 렉시컬 환경에 대한 참조

외부 코드와 연관됨

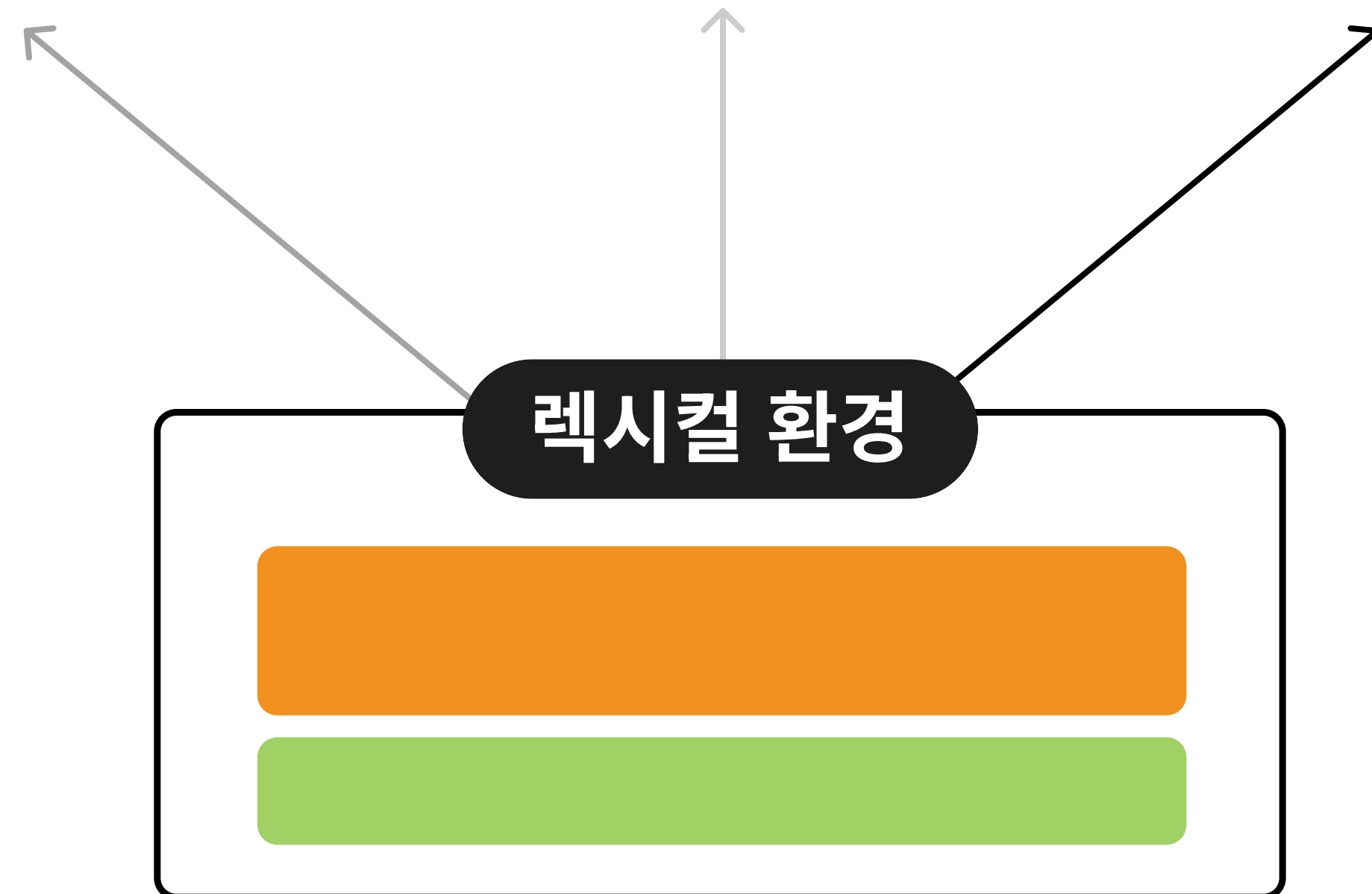


## 렉시컬 환경

실행 중인 함수

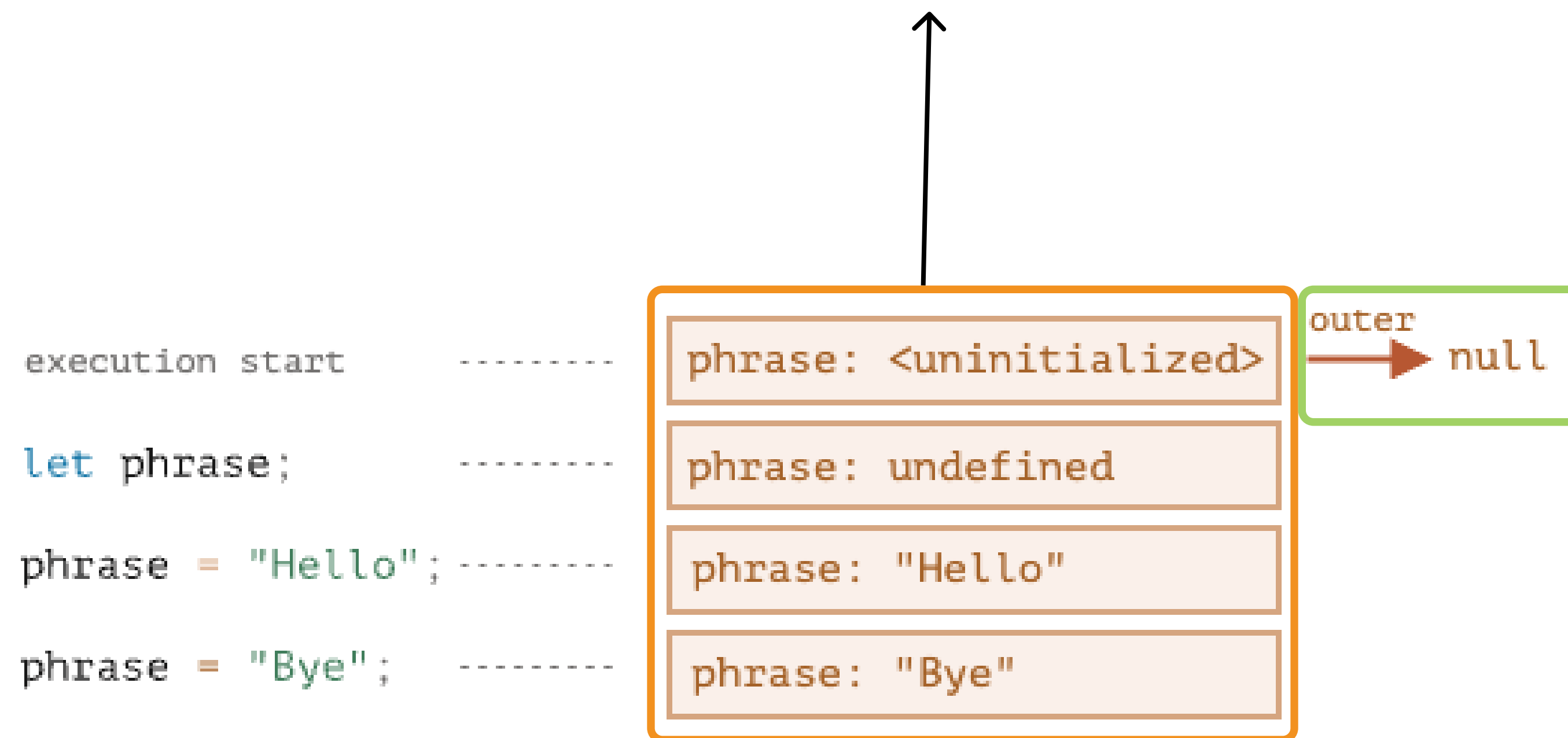
코드 블록 {...}

스크립트 전체



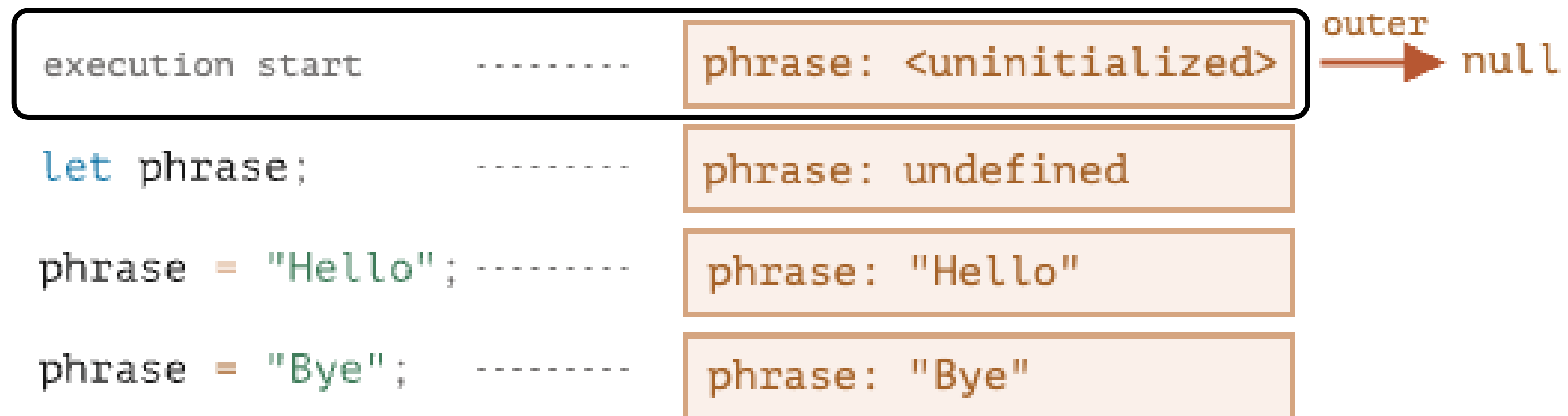
## 렉시컬 환경

### 전역 렉시컬 환경(global Lexical Environment)



## 렉시컬 환경

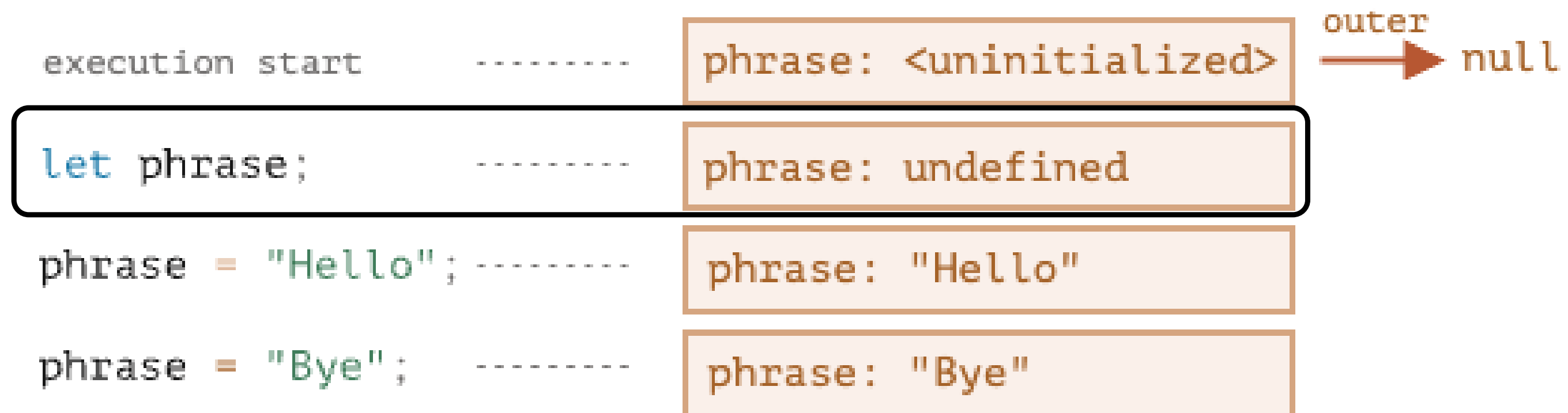
## 1. 변수



1. 스크립트가 시작되면 스크립트 내에서 선언한 변수 전체가 렉시컬 환경에 올라간다.
  - 이때 변수의 상태는 특수 내부 상태인 '**uninitialized**'
  - 자바스크립트 엔진은 변수를 **인지 O** / let을 만나기 전까진 이 변수를 **참조 X**

## 렉시컬 환경

## 1. 변수

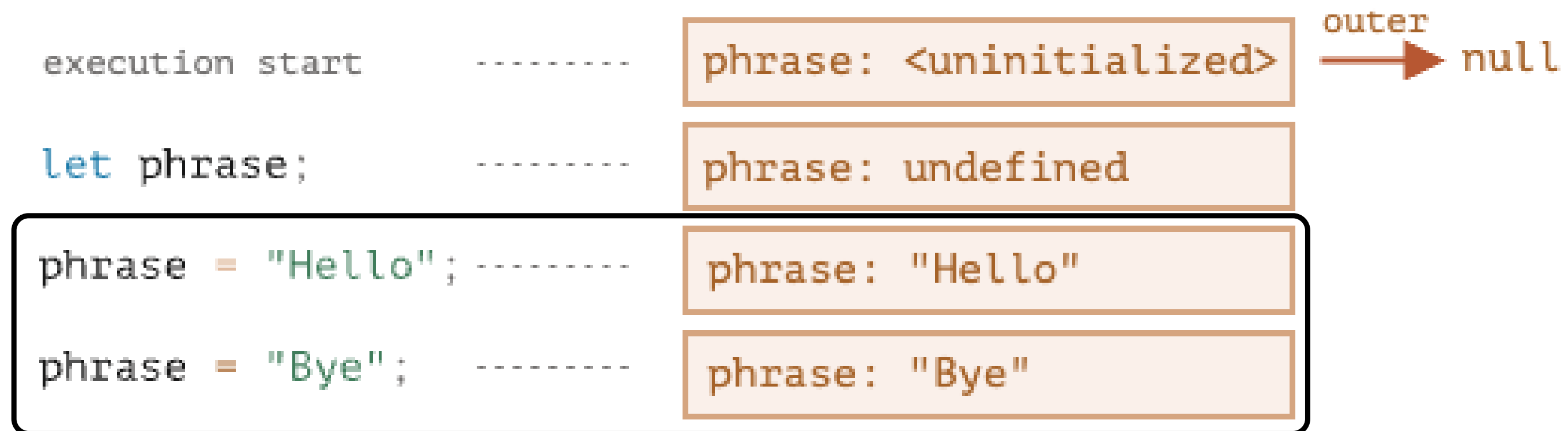


## 2. let phrase로 선언

- 아직 값을 할당하기 전이기 때문에 프로퍼티 값은 **undefined**
- phrase는 이 시점 이후부터 사용 가능

## 렉시컬 환경

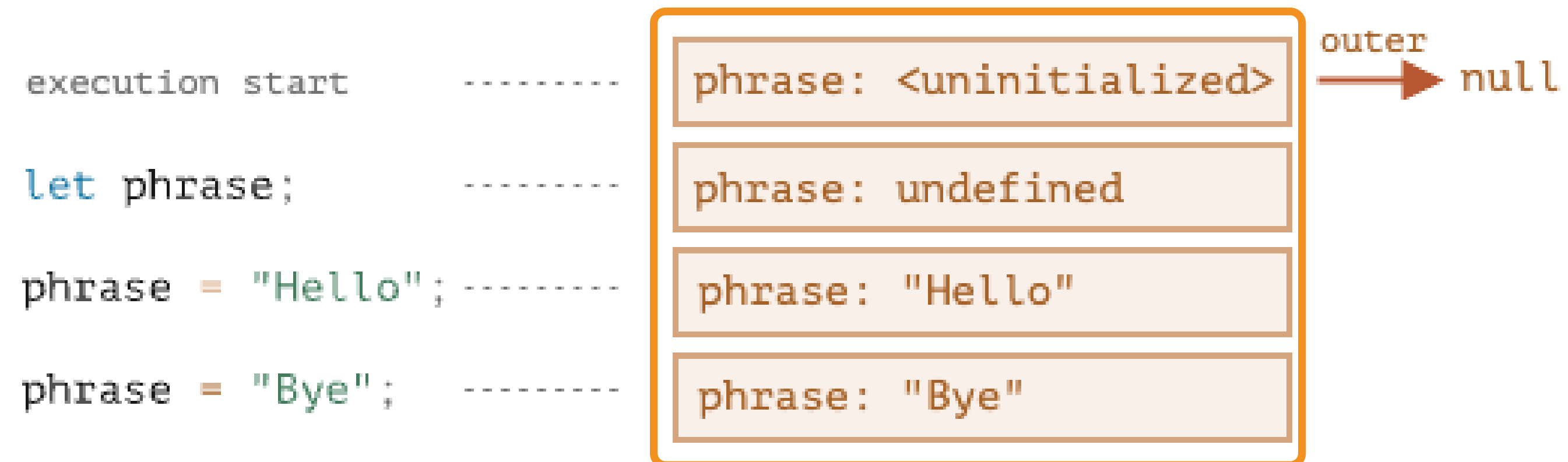
## 1. 변수



## 3. phrase에 값을 할당

## 렉시컬 환경

## 1. 변수



현재 실행 중인 함수와 코드 블록, 스크립트와 연관됨



phrase와 같은 변수 = **환경 레코드**의 프로퍼티  
변수를 변경하면 환경 레코드의 프로퍼티가 변경된다.

## 렉시컬 환경

## 2. 함수선언문

execution start

let phrase = "Hello";

```
function say(name) {  
  alert( `${phrase}, ${name}` );  
}
```

phrase: <uninitialized>  
say: function

outer  
→ null

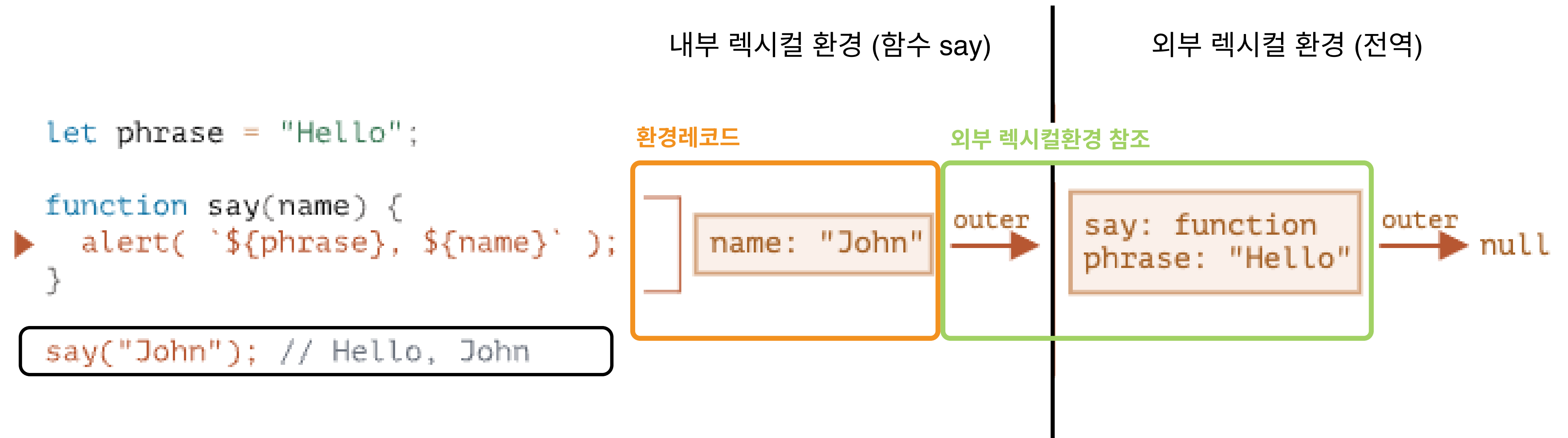
...

함수 선언문(function declaration)으로 선언한 함수 -> 일반 변수와는 달리 바로 초기화!!

let say = function(name)...같은 함수 표현식은 해당 X

## 렉시컬 환경

## 3. 내부와 외부 렉시컬 환경



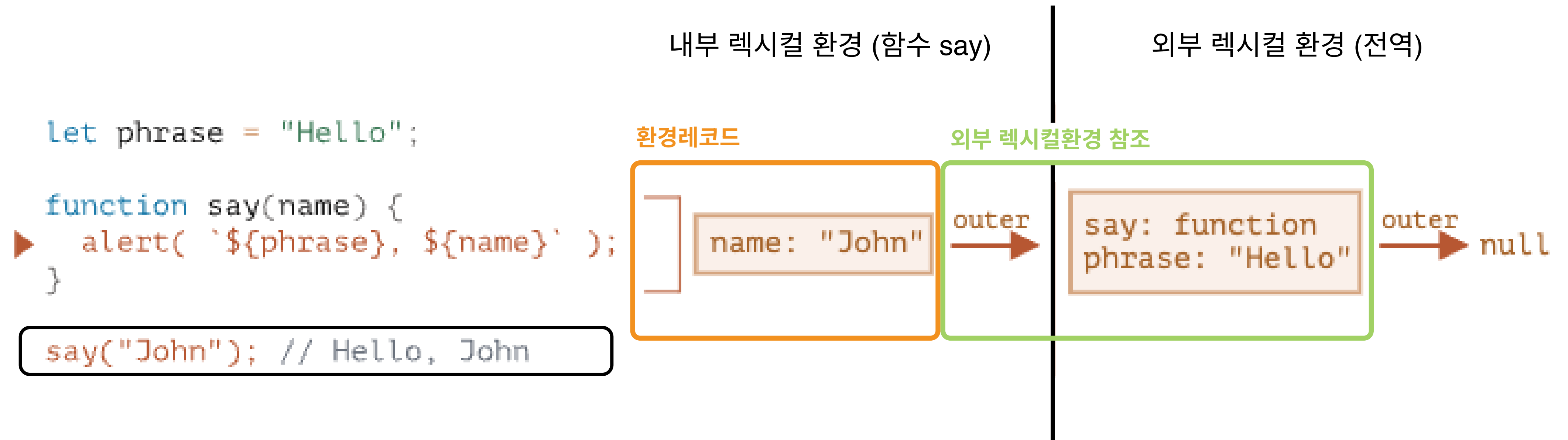
함수 say의 렉시컬 환경 -> 변수 name / say("John")을 호출했기 때문에, name의 값은 "John"

전역 렉시컬 환경 -> phrase와 함수 say



## 렉시컬 환경

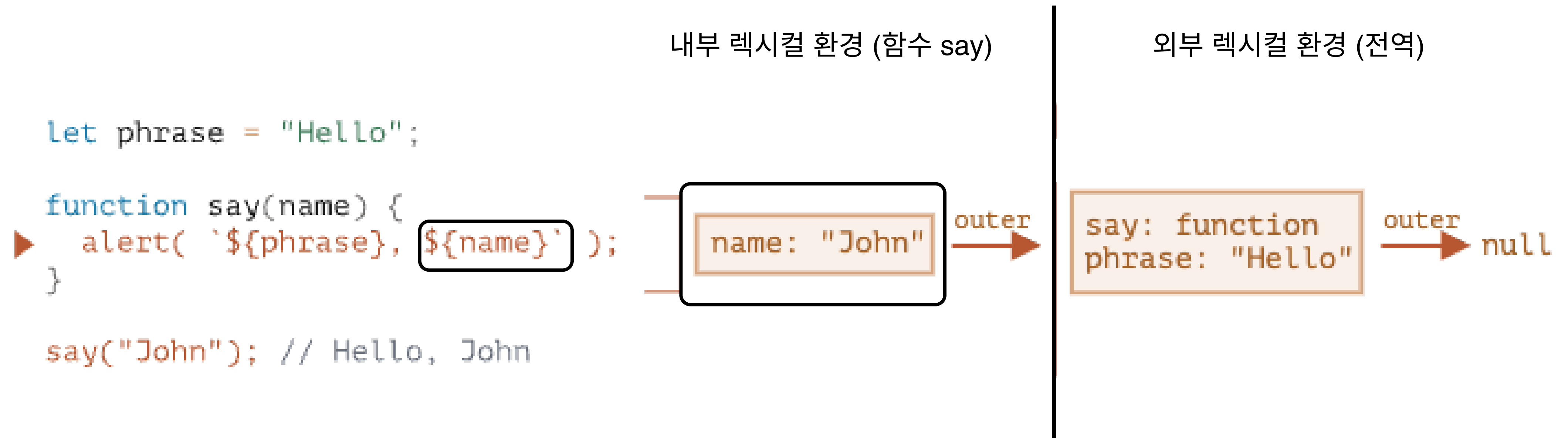
## 3. 내부와 외부 렉시컬 환경



코드에서 변수에 접근할 땐, 먼저 내부 렉시컬 환경을 검색 범위로 잡는다.  
없다면 외부 렉시컬 환경으로 확장시킨다.

## 렉시컬 환경

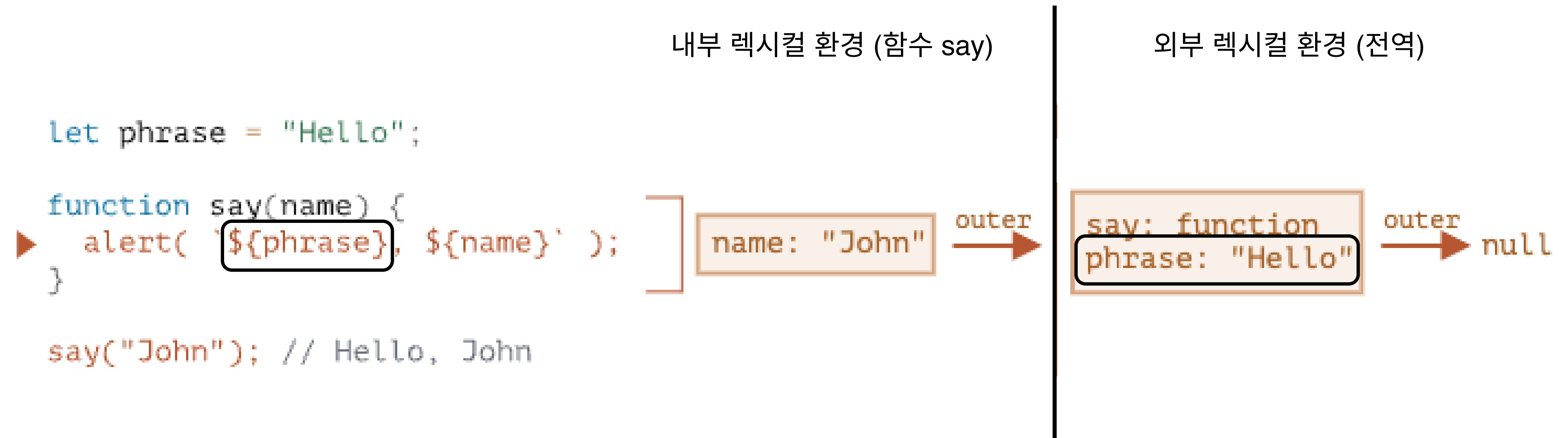
## 3. 내부와 외부 렉시컬 환경



1. 함수 say 내부의 alert에서 변수 name에 접근 -> 내부 렉시컬 환경에서 변수 name 찾기

## 렉시컬 환경

## 3. 내부와 외부 렉시컬 환경



2. alert에서 변수 phrase에 접근 -> 내부 렉시컬 환경에서 못찾음

-> 검색 범위는 외부 렉시컬 환경으로 확장됨 -> 외부 렉시컬 환경에서 phrase를 찾음

## 렉시컬 환경

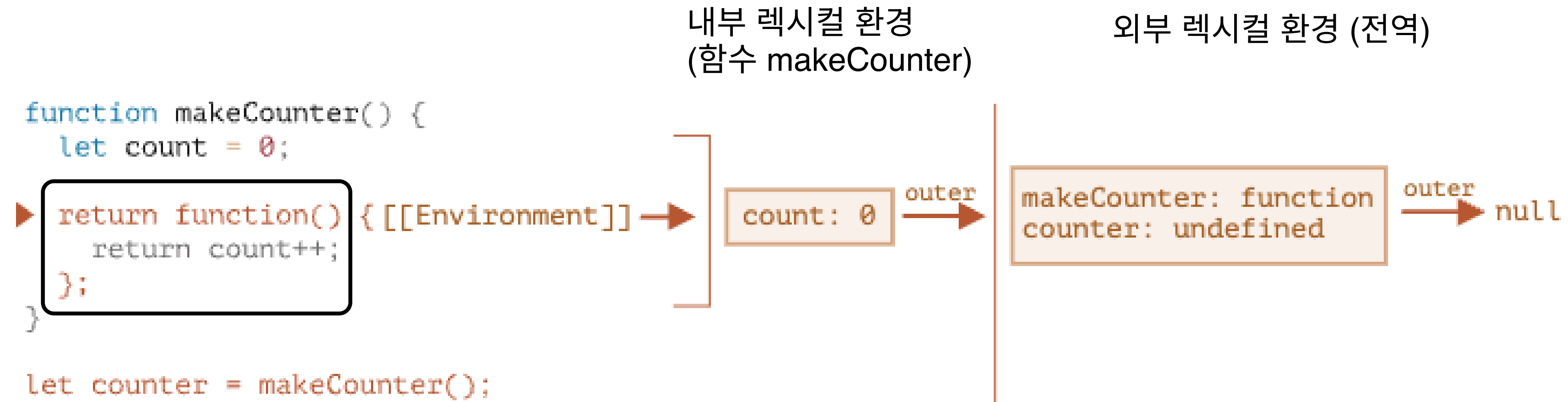
## 4. 함수를 반환하는 함수

```
1 function makeCounter() {  
2     let count = 0;  
3  
4     return function() {  
5         return count++;  
6     };  
7 }  
8  
9 let counter = makeCounter();  
10  
11 alert( counter() ); // 0  
12 alert( counter() ); // 1  
13 alert( counter() ); // 2
```

호출될 때마다 다음 숫자를 반환해주는 '카운터' 함수

## 렉시컬 환경

## 4. 함수를 반환하는 함수

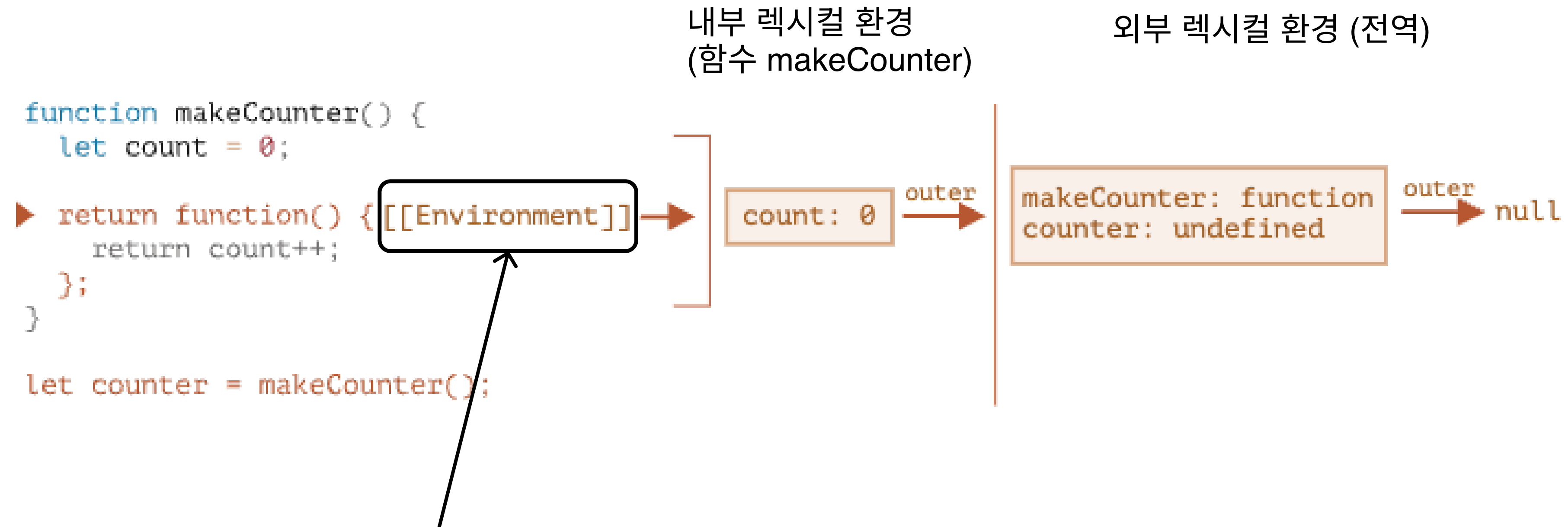


함수 say()와의 차이점

- return count++이라는 본문이 한줄 짜리인 **중첩 함수**가 만들어진다
- makeCounter()는 함수를 반환하는 함수

## 렉시컬 환경

### 4. 함수를 반환하는 함수

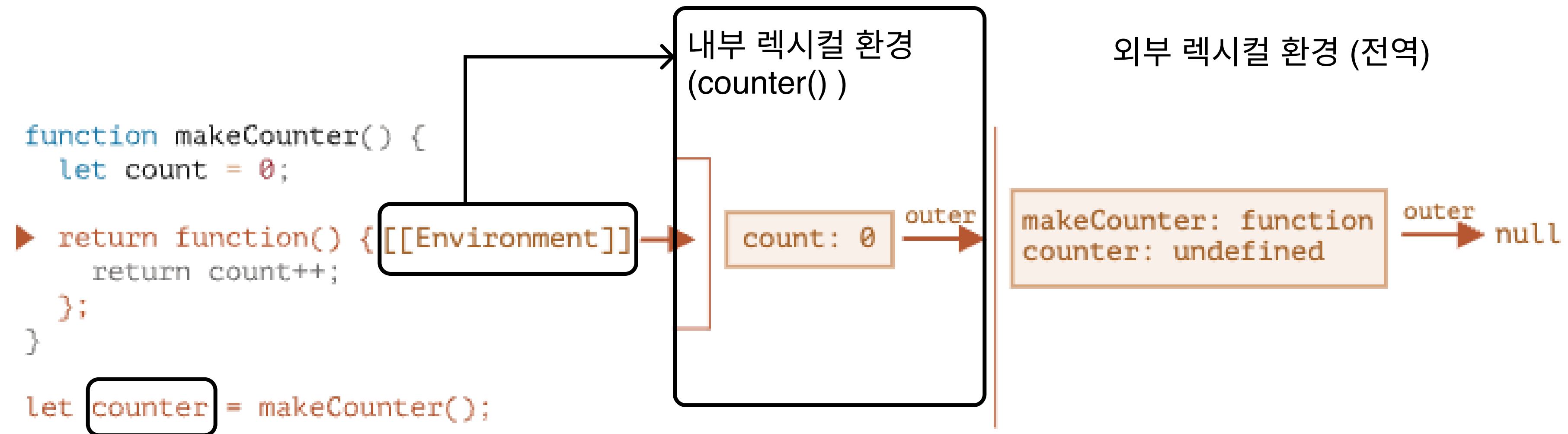


- 함수는 `[[Environment]]`라 불리는 숨김 프로퍼티를 가진다.
- 여기에 함수가 만들어진 곳의 렉시컬 환경에 대한 참조가 저장된다.

**즉 모든 함수는 함수가 생성된 곳의 렉시컬 환경을 기억한다!**

## 렉시컬 환경

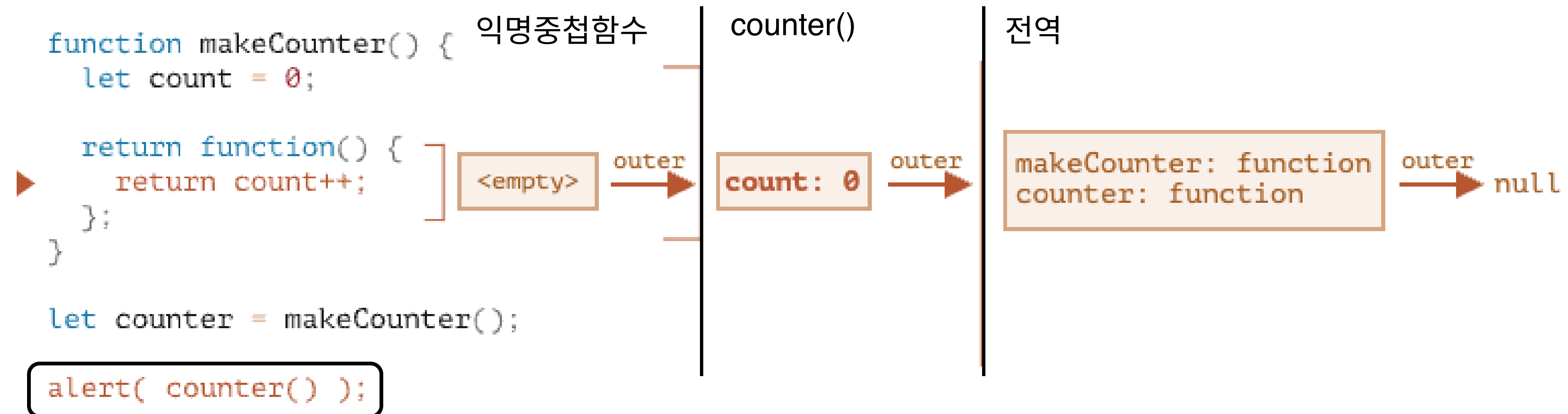
## 4. 함수를 반환하는 함수



counter.{{Environment}} -> {count: 0}이 있는 렉시컬 환경에 대한 참조가 저장  
호출 장소와 상관없이 함수가 자신이 태어난 곳을 기억할 수 있다.

## 렉시컬 환경

### 4. 함수를 반환하는 함수

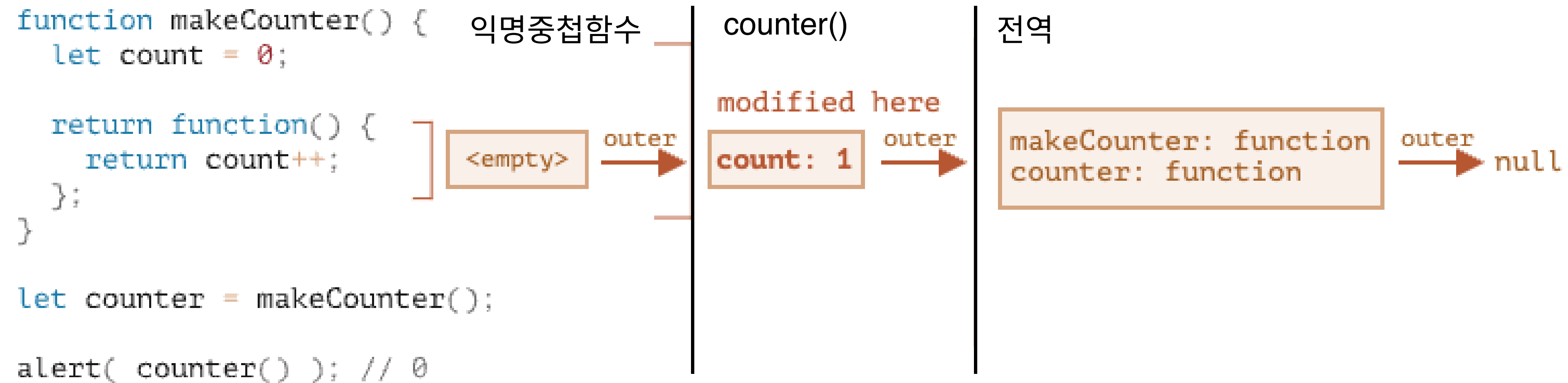


1. 익명중첩함수 자체 렉시컬 환경 -> 없음 ( `<empty>` )
2. 외부 `counter()` 렉시컬환경 -> `count` 찾음 ( `0` )
3. 0으로 alert



## 렉시컬 환경

### 4. 함수를 반환하는 함수



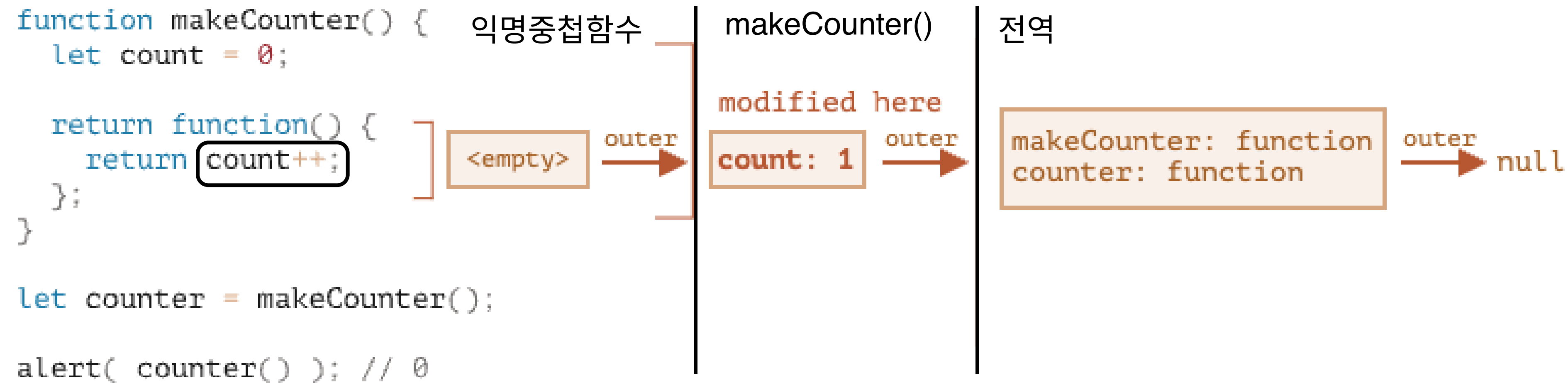
변숫값 갱신은 변수가 저장된 렉시컬 환경에서 이뤄진다!

4. `count++`가 실행되면서 `count` 값이 1 증가

5. 실행종료

## 렉시컬 환경

### 4. 함수를 반환하는 함수



즉 `count` 변수가 저장된 **counter()**의 렉시컬환경에서 변경이 된다!

-> 다음에 `counter()`를 호출하면 1로 출력되고 `count` 변수가 2로 증가

## 클로저 closure

외부 변수를 기억하고 이 외부 변수에 접근할 수 있는 함수

자바스크립트에서 함수는 `[[Environment]]`를 통해 자신이 어디서 만들어졌는지(렉시컬 환경)를 기억  
함수 본문에선 `[[Environment]]`를 사용해 외부 변수에 접근합니다.

-> 자바스크립트에선 모든 함수가 자연스럽게 클로저

**클로저****closure**

외부 변수를 기억하고 이 외부 변수에 접근할 수 있는 함수

## 모든 함수가 자연스럽게 클로저

자바스크립트에서 함수는 `[[Environment]]`를 통해 자신이 어디서 만들어졌는지(렉시컬 환경)를 기억  
함수 본문에선 `[[Environment]]`를 사용해 외부 변수에 접근합니다.

하지만

**new Function**은 예외

-> 자바스크립트에선 모든 함수가 자연스럽게 클로저

**THANK YOU**