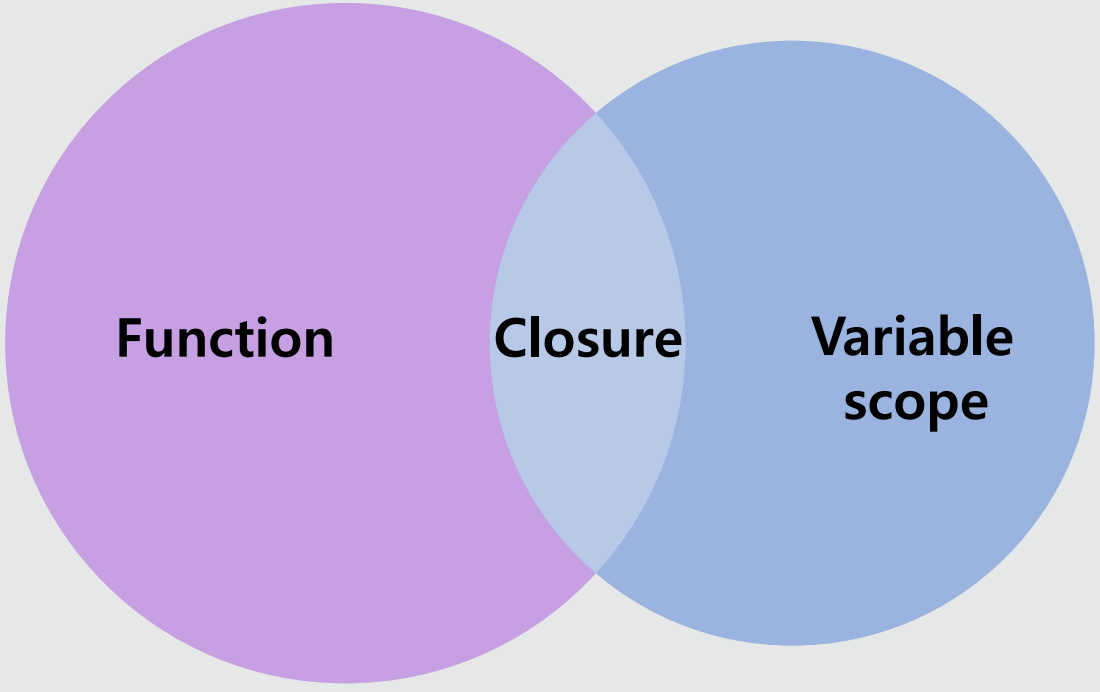
클로저

함수

클로저 함수?



클로저 함수

클로저는 외부 함수의 컨텍스트에 접근할 수 있는 내부 함수를 뜻한다.
외부 함수의 실행이 종료된 후에도,
클로저 함수는 외부 함수가 선언된 어휘적 환경에 접근할 수 있다.

특징

클로저 함수 특징

내부함수는 외부함수에 선언된 변수에 접근이 가능하다는 특징이 있다.

일반 함수는 함수 실행이 끝나고 나면 내부 변수를 사용할 수 없지만,
클로저 함수는 외부 함수의 실행이 끝나더라도 외부 함수 내 변수가 메모리상에 저장된다.
(Lexical 환경을 메모리에 저장하기 때문에 가능한 일)

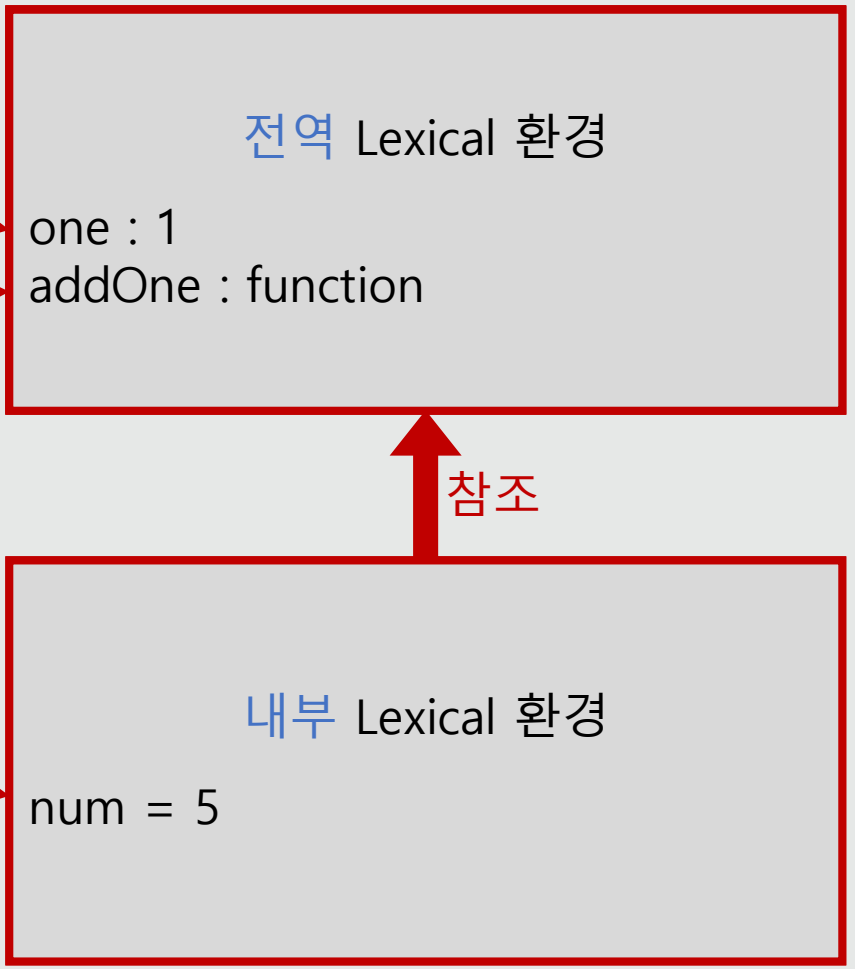
특정 데이터를 scope 안에 가두어 둔 채로 계속 사용할 수 있게 한다.

Lexical – 스크립트 전체, 실행 중인 함수

Scope – 변수에 접근할 수 있는 범위

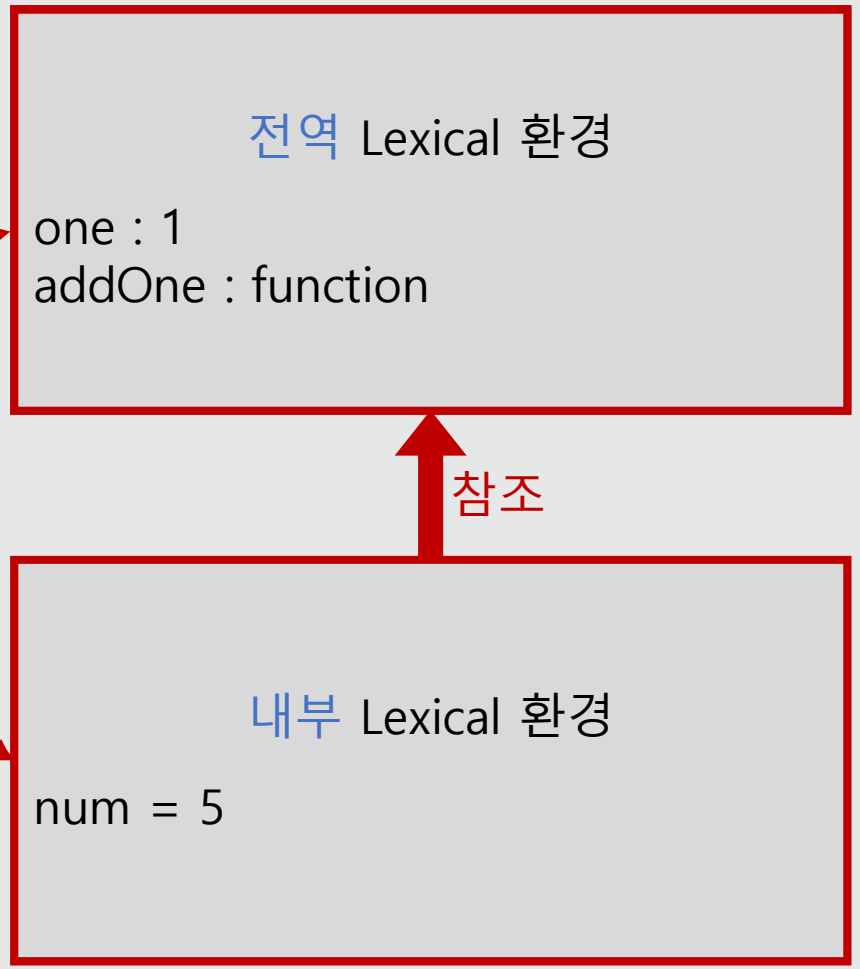
예시 1

```
1    let one;
2    one = 1;
3
4    function addOne(num) {
5        console.log(one + num);
6    }
7
8    addOne(5);
```



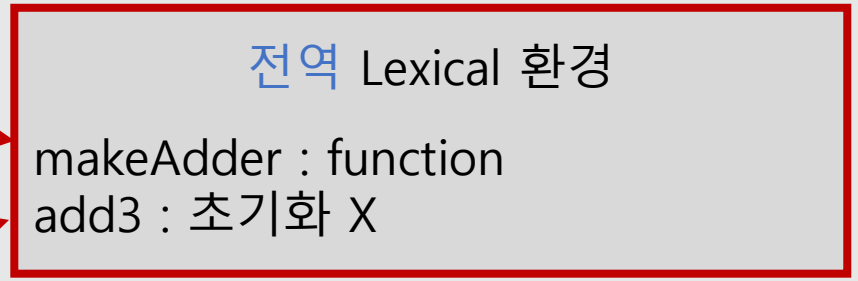
예시 1

```
1    let one;
2    one = 1;
3
4    function addOne(num) {
5        console.log(one + num);
6    }
7
8    addOne(5);
```



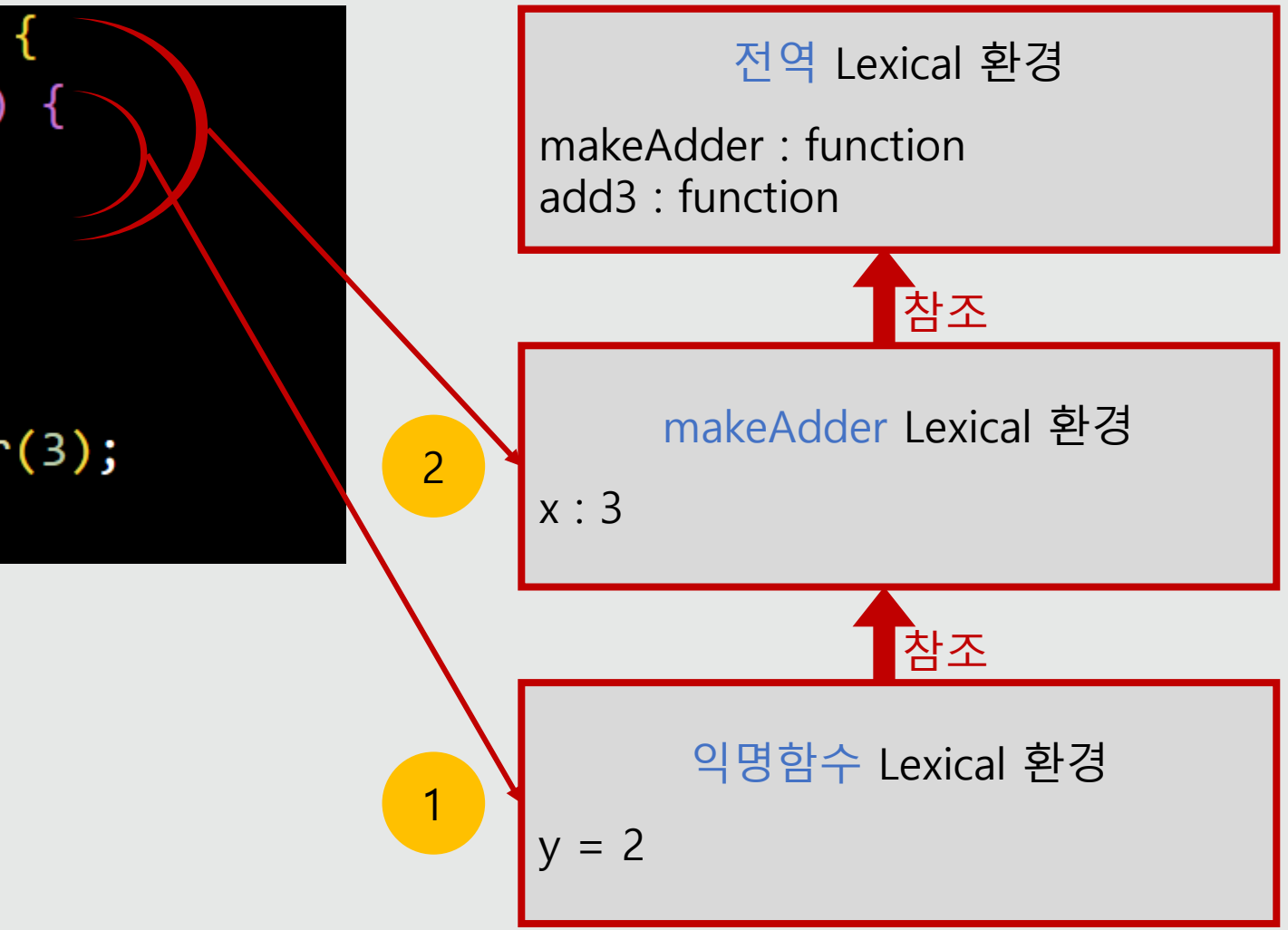
예시 2

```
1   function makeAdder(x) {
2       return function(y) {
3           return x + y;
4       }
5   }
6
7   const add3 = makeAdder(3);
8   console.log(add3(2));
```



예시 2

```
1   function makeAdder(x) {
2       return function(y) {
3           return x + y;
4       }
5   }
6
7   const add3 = makeAdder(3);
8   console.log(add3(2));
```



사용하는 이유?

사용하는 이유

1. 상태 유지
    - 현재 상태를 기억하고 변경된 최신 상태를 유지할 수 있다.
2. 정보의 접근 제한 (캡슐화)
    - 클래스 기반 언어의 private 키워드를 흉내낼 수 있다.
3. 전역변수 사용 억제 (모듈화에 유리)
    - 클로저를 통해 데이터와 메소드를 묶어다닐 수 있기에 클로저는 모듈화에 유리하다.

단점

메모리 측면에서 손해를 볼 수 있다.

코드가 복잡해질 수 있다.

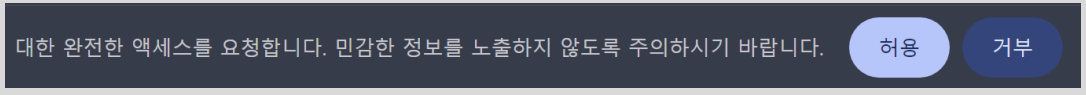
실습

개발자 도구에서 소스 편집 준비

바탕화면에 폴더 만들기

아무 브라우저에 접속 후, F12(개발자 도구)

Sources -> File system(Workspace)

만든 폴더 가져온 후, 허용버튼 누르기 

개발자 도구에서 New file을 만들고 index.html로 하면
만든 폴더 안에도 index.html이 만들어진 것을 볼 수 있다.

파일을 드래그해서 브라우저에 열기

실습

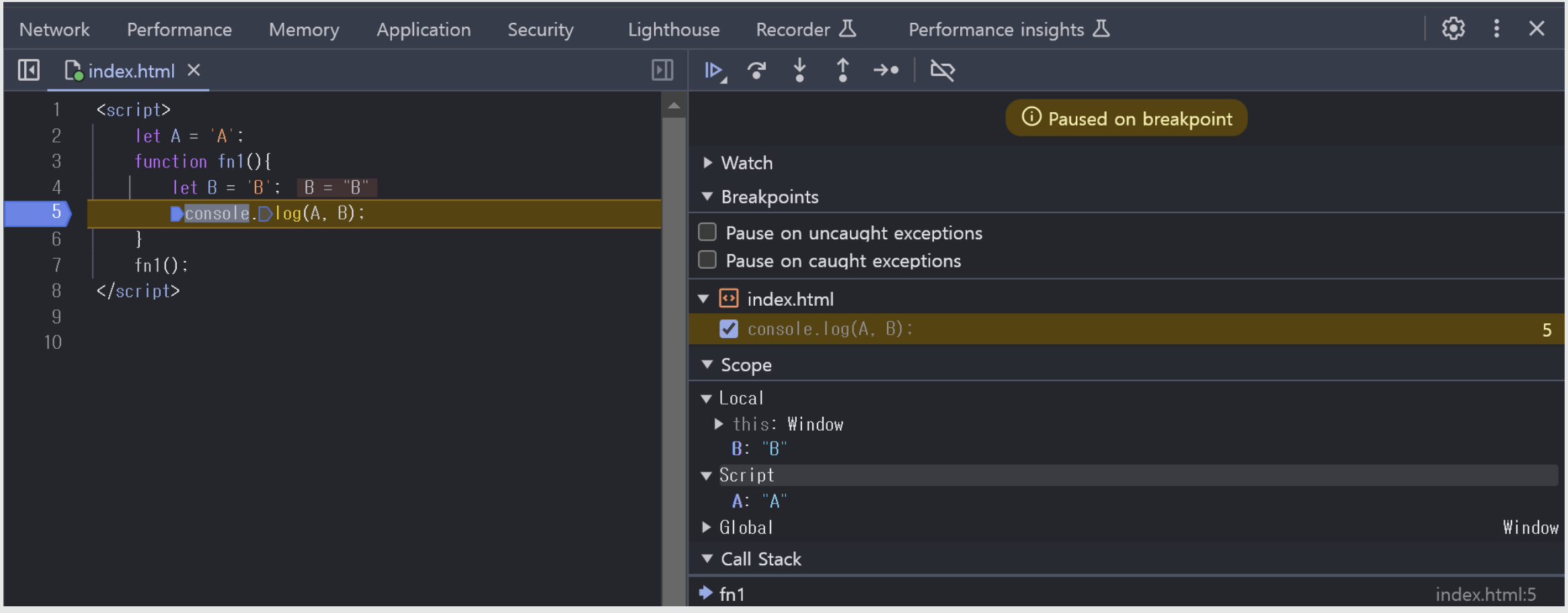
index.html

```
1  <script>
2      let A = 'A';
3      function fn1(){
4          let B = 'B';
5          console.log(A, B);
6      }
7      fn1();
8  </script>
9
```

```
<script>
    let A = 'A';
    function fn1(){
        let B = 'B';
        console.log(A, B);
    }
    fn1();
</script>
```

코드

실습



> **5번 라인에서 break하고 실행**

실습

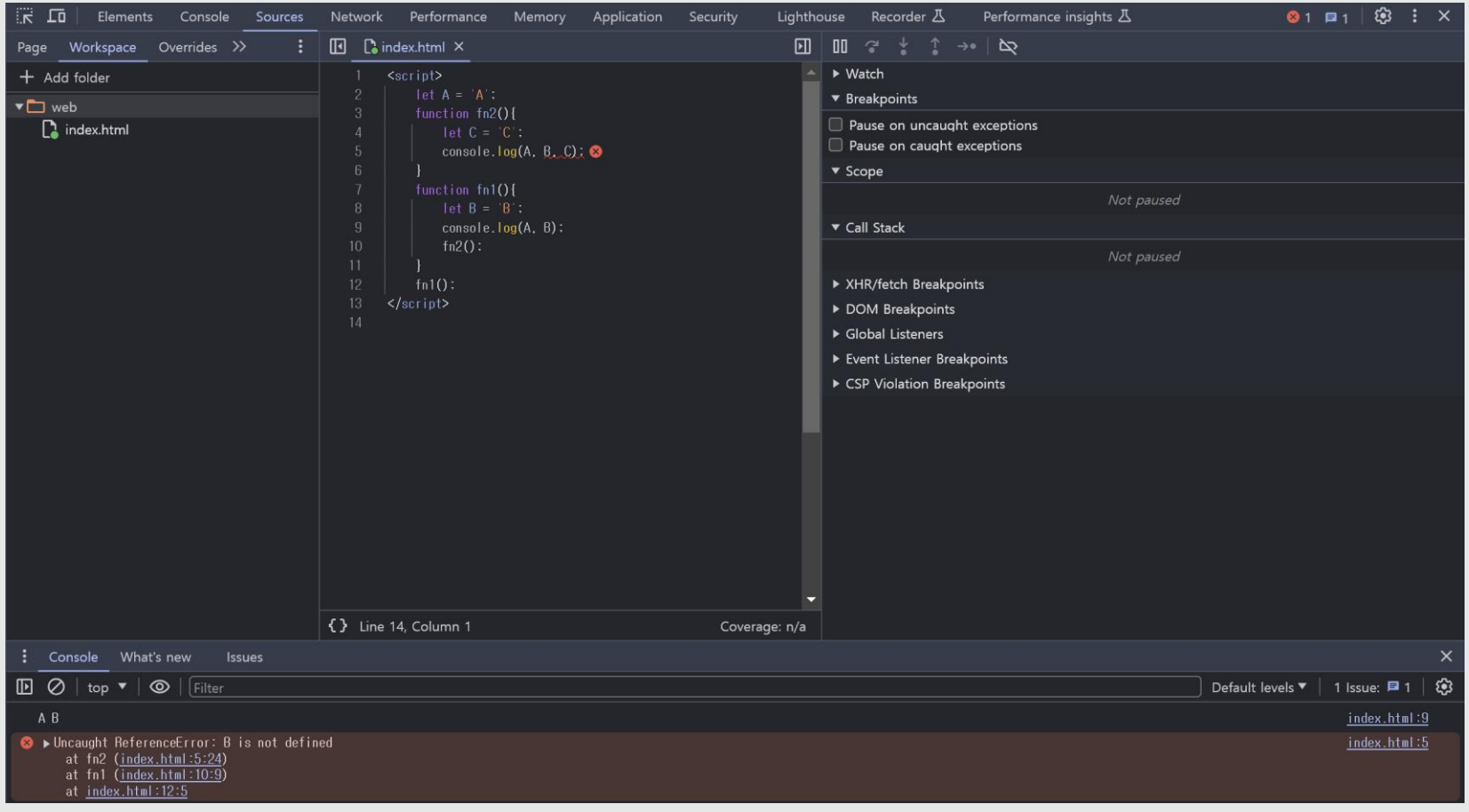
index.html

```
1   <script>
2       let A = 'A';
3       function fn2(){
4           let C = 'C';
5           console.log(A, B, C);
6       }
7       function fn1(){
8           let B = 'B';
9           console.log(A, B);
10          fn2();
11      }
12      fn1();
13  </script>
```

```
<script>
    let A = 'A';
    function fn2(){
        let C = 'C';
        console.log(A, B, C);
    }
    function fn1(){
        let B = 'B';
        console.log(A, B);
        fn2();
    }
    fn1();
</script>
```

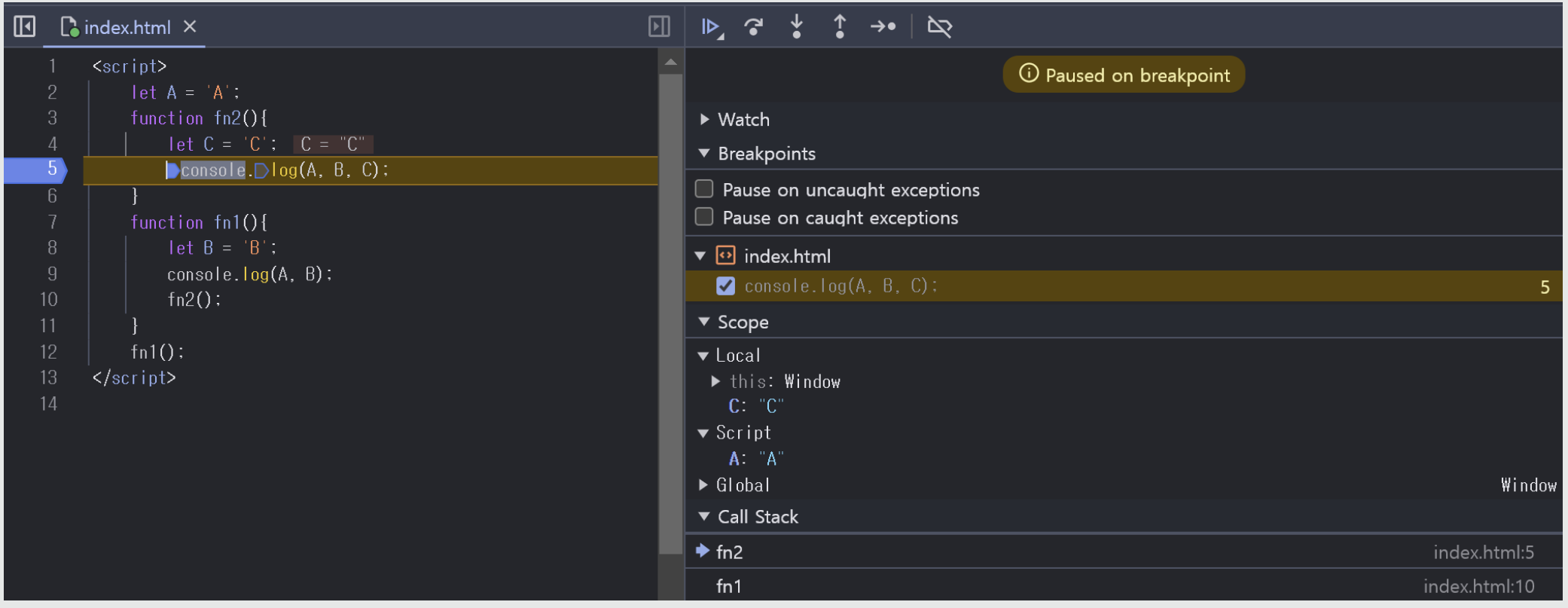
코드

실습


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
Page
Workspace
Overrides
Add folder
web
index.html
<script>
let A = 'A';
function fn2(){
let C = 'C';
console.log(A, B, C);
}
function fn1(){
let B = 'B';
console.log(A, B);
fn2();
}
fn1();
</script>
Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
Scope
Not paused
Call Stack
Not paused
XHR/fetch Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints
CSP Violation Breakpoints
Line 14, Column 1
Coverage: n/a
Console
What's new
Issues
top
Filter
Default levels
1 Issue: 1
A B
index.html:9
Uncaught ReferenceError: B is not defined
index.html:5
at fn2 (index.html:5:24)
at fn1 (index.html:10:9)
at index.html:12:5

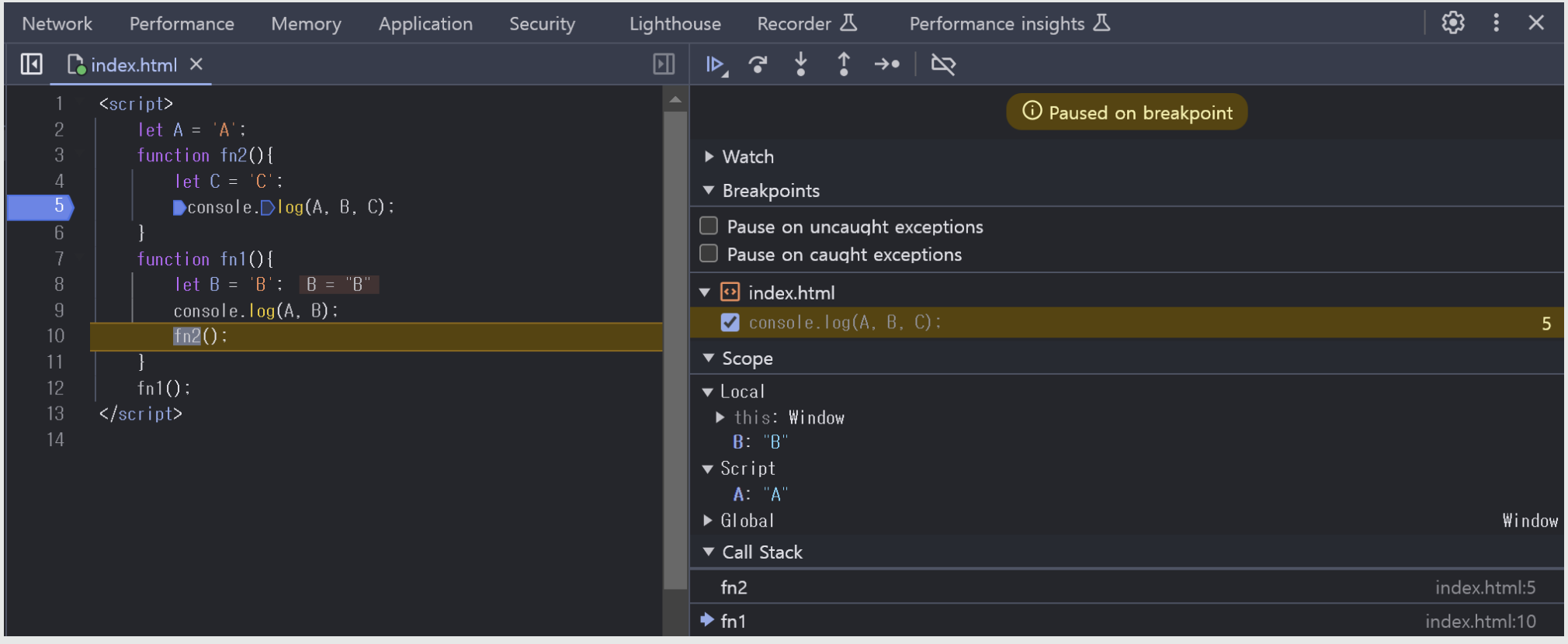
> **오류 발생**

실습



5번 라인에서 break하고 실행

실습


Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
index.html
1 <script>
2 let A = 'A';
3 function fn2(){
4 let C = 'C';
5 console.log(A, B, C);
6 }
7 function fn1(){
8 let B = 'B'; B = "B"
9 console.log(A, B);
10 fn2();
11 }
12 fn1();
13 </script>
14
Paused on breakpoint
Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
index.html
console.log(A, B, C); 5
Scope
Local
this: Window
B: "B"
Script
A: "A"
Global Window
Call Stack
fn2 index.html:5
fn1 index.html:10

fn1에 접근

실습

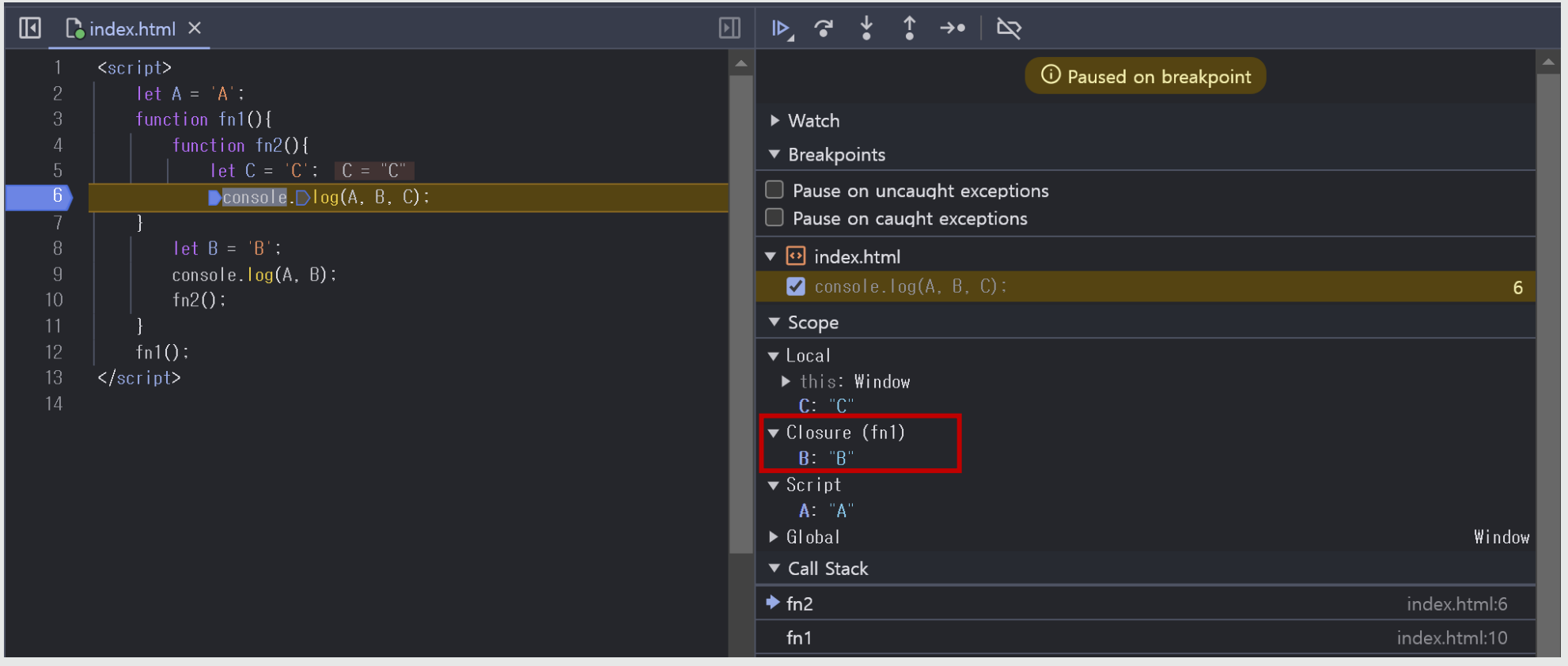
index.html

```
1  <script>
2      let A = 'A';
3      function fn1(){
4          function fn2(){
5              let C = 'C';
6              console.log(A, B, C);
7      }
8          let B = 'B';
9          console.log(A, B);
10         fn2();
11     }
12     fn1();
13 </script>
```

```
<script>
    let A = 'A';
    function fn1(){
        function fn2(){
            let C = 'C';
            console.log(A, B, C);
    }
        let B = 'B';
        console.log(A, B);
        fn2();
    }
    fn1();
</script>
```

오류 해결

실습



오류 해결

6번줄 break

Reference

https://velog.io/@jellyjw/JS-클로저
https://enjoydev.life/blog/javascript/6-closure
https://developer.mozilla.org/ko/docs/Web/JavaScript/Closures
https://poiemaweb.com/js-closure
https://adjh54.tistory.com/64

유튜브
https://www.youtube.com/watch?v=tpI2oXQkGZs
https://www.youtube.com/watch?v=bwwaSwf7vkE
https://www.youtube.com/watch?v=Bgv01qVTo8Q – 개발자 도구에서 소스 편집하기

감사

합니다