KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

함수

[KB] IT's Your Life

KB 국민은행

함수 고급

내부 함수

- 프로그램 개발 시 일어나는 네임 충돌을 막는 방법
- 내부 함수는 함수 내부에 선언

```
function 의부 함수() {
    function 내부 함수1() {
        // 함수 코드
    }

    function 내부 함수2() {
        // 함수 코드
    }

    // 함수 코드
}
```

함수 안에 함수 정의?

```
window.onload=function() {
...
button.onclick=function() {...};
;}
```

이미 봐왔던 것들

js에서는
이러한 형식이 흔하게 사용됨

왜??

namespace 분리.

또 closure라는 매커니즘을 사용 가능하기에

내부 함수 이용으로 함수 충돌을 막는 법

- 내부 함수 사용 시 내부 함수 우선

```
function pythagoras(width, height) {
    function square(x) {
        return x * x;
    }
    return Math.sqrt(square(width) + square(height));
}
```

보편적인 이름의 함수를
특정만의 함수로 wrapping할 때.

내부함수는 다른 외부에서 접근할 수 없음

- 외부에서는 내부 함수를 호출 할 수 없음

09_nestfunction.js

```
function add(a, b) {
    return a + b;
}

function sum(n) {
    var s = 0;
    for (var i = 0; i <= n; i++) {
        s = add(s, i);
    }
    return s;
}

console.log("1~100 = " + sum(100));
```

09_nestfunction2.js

```
function sum(n) {
    function add(a, b) {
        return a + b;
    }

    var s = 0;
    for (var i = 0; i <= n; i++) {
        s = add(s, i);
    }
    return s;
}

console.log("1~100 = " + sum(100));
console.log("2 + 3 = " + add(2 + 3));// 에러
```

09_nestfunction3.js

js는 다른 언어와 다르게 지역변수가 힙에 저장되고,
스코프 객체에 모여진 다음 힙에 보관된다

스택에
호출된 함수 정보
쌓임.

함수가 실행이 끝나면
스택에서 정보가 사라짐

힙에
호출된 함수 관련 scope객체
scope객체 안에는 지역 변수 정보

스택에서 사라진 함수의
scope객체는 가비지가 됨, 사라짐

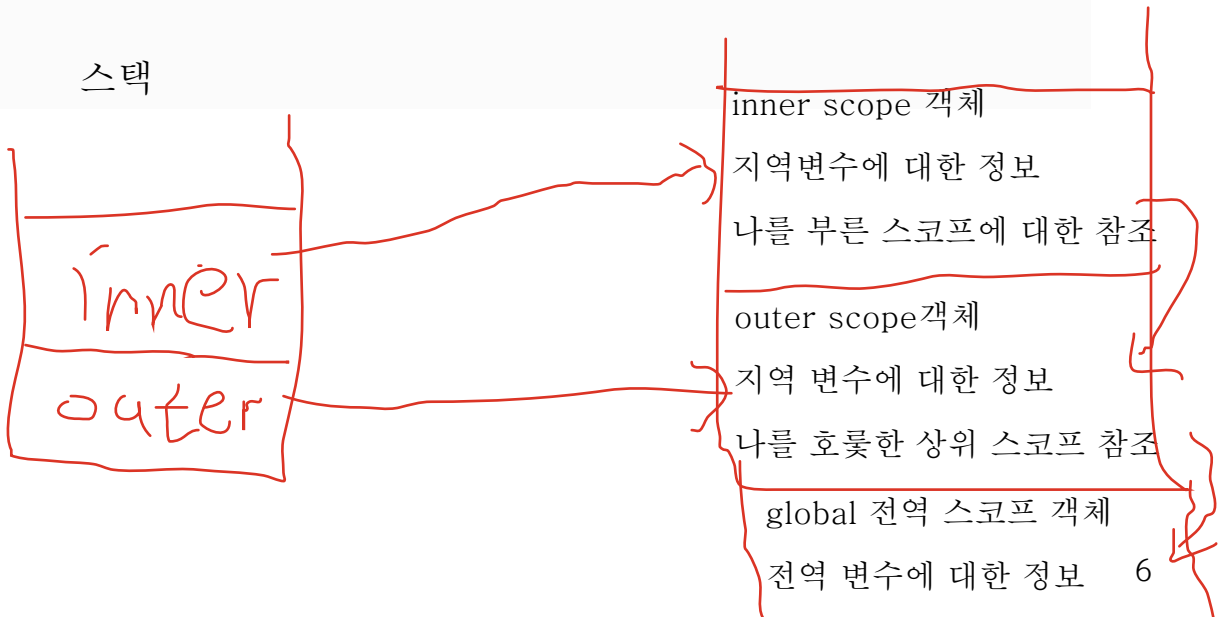
```
function outer() {
    var outvalue = 5678;

    function inner() {
        var invalue = 1234;
        console.log("outvalue = " + outvalue);
    }

    inner();
    console.log("invalue = " + invalue);// 에러
}
outer();
```

현재 스코프 객체에 없음.
상위 객체인 global객체의 스코프 객체에도 없음
그래서 에러

하지만 함수 실행이 끝난 scope객체를
유지하고 활용하고 싶다면? closure객체를 활용.
닫아놓고 사용한다는 직관적인 네이밍.
closure객체는 실행이 끝난 함수의 scop객체를 참조
조금 뒤에 계속

변수에 대한 검색 순서
변수에 대한 접근을 할때
해당 함수의 스코프 객체로 접근하여 지역변수에 대한 검색 실시
없으면
다음 상위 스코프로 넘어가서 검색 실시

끝까지 없으면 마지막으로
global 전역 객체의 scope객체에서 변수에 대한 검색 실시
=> 해당 과정을 scope체이닝이라고 한다.



익명 함수

- 이름을 가지지 않는 함수
  - 변수에 익명 함수에 대한 참조를 저장하여 사용

```
function(인수목록) { 본체 }
```

- 함수 호출 : 함수 참조(함수명)뒤에 괄호표기후 코드를 실행

```
var fn = function() {
    console.log('Hello javascript')
}

console.log(fn);      // [Function]

fn();      // Hello javascript
```

10_funcliteral.js

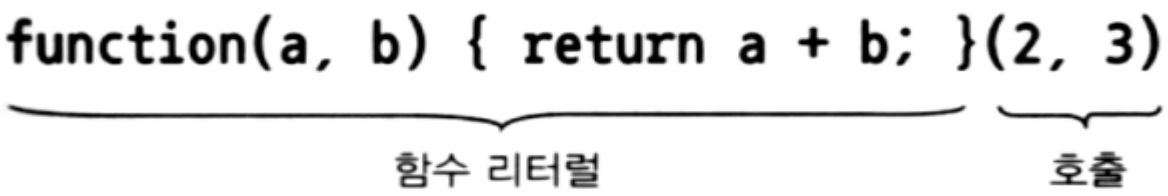
```
var add = function(a, b) {
    return a + b;
};

console.log("2 + 3 = " + add(2, 3));
```

10_funcliteral2.js

```
console.log("2 + 3 = " + add1(2, 3));
console.log("4 + 5 = " + add2(4, 5));  // 에러

function add1(a, b) { return a + b; }
var add2 = function(a, b) { return a + b; };
```

10_funcliteral3.js

```
console.log("2 + 3 = " +
    function(a, b) { return a + b; }(2, 3));
```



11_assignfunc.js

```
var add = function(a, b) {
    return a + b;
}

var plus = add;

console.log("2 + 3 = " + plus(2, 3));
```

함수를 매개변수로 전달하기

- 함수적 프로그래밍

12_funcargument.js

```
var add = function(a, b) {
    return a + b;
}

var multi = function(a, b) {
    return a * b;
}

function calc(a, b, f) {
    return f(a, b);
}

console.log("2 + 3 = " + calc(2, 3, add));
console.log("2 * 3 = " + calc(2, 3, multi));
```

함수를 리턴하는 함수

- 함수를 리턴하는 함수의 사용은 클로저 때문임

```
function outer() {
    return function() {
        console.log('Hello Function...!');
    };
}

// 호출 1
outer()();

// 호출 2
var fn = outer();
fn();
```

함수를 반환하는 함수

함수를 반환 받음

클로저

지역 변수의 유효 범위

클로저 :
함수가 선언될때의 스코프를 기억하는 기능!!

```
function test(name) {
    var output = 'Hello ' + name + '...!';
}

console.log(output)
```

래퍼 함수가 종료된 이후에도
클로저는 그 함수가 선언된 환경을 참조할 수 있다

- 함수 안의 지역 변수는 함수 외부에서 사용 불가능
- 지역 변수는 함수 실행 시 생성되고 종료 시 사라짐

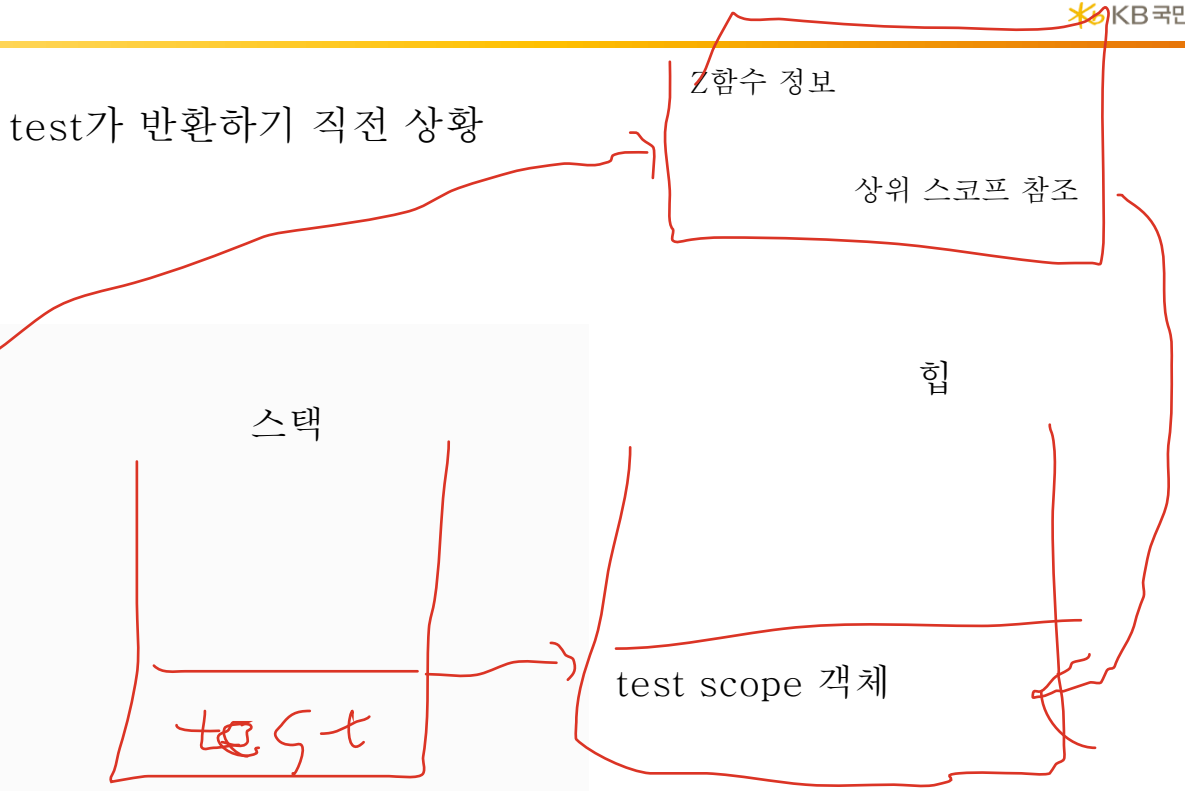
하지만 js는 closure깁법으로 사용 가능

KB 국민은행

클로저

- 클로저 특징 : 규칙 위반 가능

```
function test(name) {
    var output = 'Hello ' + name + '...!';

    return function() {
        console.log(output)
    }
}

test('Javascript')();
```



- 지역 변수를 남겨두는 현상
- test( ) 함수로 생성된 공간
- 리턴된 함수 자체
- 살아남은 지역 변수 output(반드시 리턴된 클로저 함수 사용)

test가 반환하고 종료되어 스택에서 사라지고
test scope객체가 가비지가 되어도

z함수 정보가
사라진 test함수의 scope객체를
z함수 정보의 상위 스코프 참조가
가리키고 있으므로
test scope 객체 는 사라지지 않고 활용 가능하다

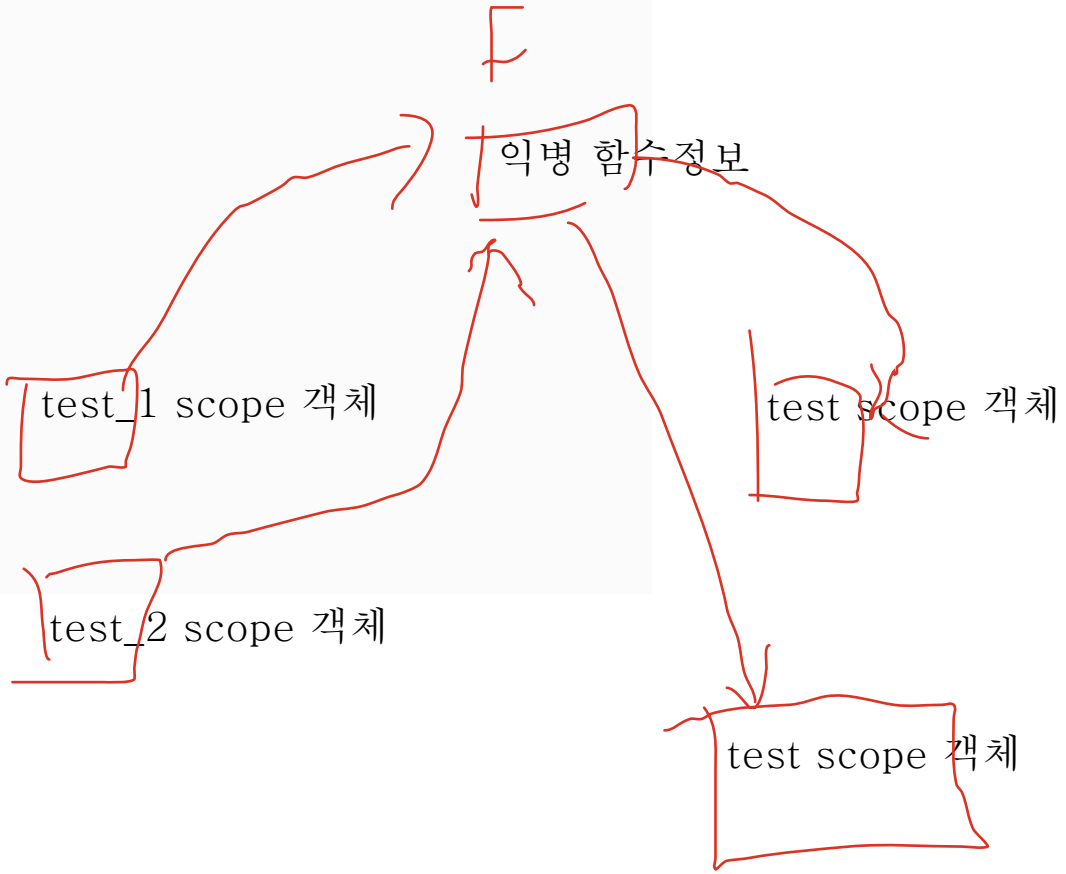
당연히 해당 스코프는 내부에서만 접근 가능하겠죠

클로저 정의

```
function test(name) {
    var output = 'Hello ' + name + '...!';

    return function() {
        console.log(output)
    }
}

var test_1 = test('Node');
var test_2 = test('Javascript');
test_1();
test_2();
```

반환되어

함수가 배정할 때마다 스코프 객체가 생기고 따로 배정된다.



클로저

```
function outer() {
    let value = 1234;
    function inner() {
        document write("value = " + value + "<br>");
    }
    return inner;
}

let outin = outer()
outin();
```

내부 함수에서
지역 변수를 참조한다.

그 내부 함수에 대한 참조를
외부에서 가지고 있다.

이 시점에서 outer가 아직 종료되어서는 안된다.

13_closure.js

```
function outer() {
    var value = 1234;

    function inner() {
        console.log("value = " + value);
    }

    inner();
}

outer();
```

13_closure2.js

```
function outer() {
    var value = 1234;
}

outer();

console.log("value = " + value);
```

13_closure3.js

```
function outer() {
    var value = 1234;

    function inner() {
        console.log("value = " + value);
    }

    return inner;
}

var outin = outer();
outin();
```

13_closure4.js

```
function outcount() {
    var count = 0;

    setInterval(function() {
        count++;
        console.log(count + "초 지났습니다.");
    }, 1000);
}
outcount();
```

```
function outer() {
    let value;
    이벤트 등록(function() {
        value 사용;
    });
}
```

함수 고급

재귀호출

- 자기 자신을 호출하는 함수

callee

- arguments의 속성
  - 해당 함수를 호출한 함수에 대한 참조
  - 익명 함수의 재귀호출에 사용
    - 함수의 이름이 없으로 함수명으로 호출 불가

14_recursive.js

```
function fact(n) {
    if (n == 1) {
        return 1;
    } else {
        return n * fact(n-1);
    }
}

console.log("5! = " + fact(5));
```

15_callee.js

```
console.log("5! = " + function(n) {
    if (n == 1) {
        return 1;
    } else {
        return n * arguments.callee(n-1);
    }
}(5));
```

화살표 함수

일반 함수

```
const greeting1 = function(name) {
  return 'hello ' + name;
}
```

화살표 함수

```
const greeting2 = (name) => {
  return 'hello ' + name;
}

const greeting3 = name => {
  return 'hello ' + name;
}

const greeting4 = () => 'hello';
```

KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

화살표 함수

[KB] IT's Your Life

KB 국민은행

화살표 함수

ex01.js

```
const greeting1 = function (name) {
  return 'hello ' + name;
};

console.log(greeting1('hong'));

const greeting2 = (name) => {
  return 'hello ' + name;
};

console.log(greeting2('kim'));

const greeting3 = (name) => {
  return 'hello ' + name;
};

console.log(greeting3('park'));

const greeting4 = () => 'hello';
console.log(greeting4());
```

```
hello hong
hello kim
hello park
hello
```

암시적 반환

return 키워드 생략

- 1줄 함수인 경우 해당 표현식의 값을 자동으로 리턴

```
const greeting1 = (name) =>'hello' + name;

const greeting2 = (name) =>`hello ${name}`;
```

- 객체 리터럴의 암시적 반환
  - 객체 리터럴을 ()로 감싸야 함

```
const race = '100m dash';
const runners = ['Usain Bolt', 'Justin Gatlin', 'Asafa Powell'];
const results = runners.map((runner, i) => ({name: runner, race, place: i + 1}));
console. log(results);
```

화살표 함수

ex02.js

```
const greeting1 = (name) => 'hello' + name;
console.log('hong');

const greeting2 = (name) => `hello ${name}`;
console.log('kim');

const race = '100m dash';
const runners = ['Usain Bolt', 'Justin Gatlin', 'Asafa Powell'];
const results = runners.map((runner, i) => ({
  name: runner,
  race,
  place: i + 1,
}));
console.log(results);
```

```
hong
kim
[
  { name: 'Usain Bolt', race: '100m dash', place: 1 },
  { name: 'Justin Gatlin', race: '100m dash', place: 2 },
  { name: 'Asafa Powell', race: '100m dash', place: 3 }
]
```

화살표 함수

화살표 함수는 익명 함수

- 참조할 이름이 필요하다면 함수를 변수에 할당하여 사용

```
const greeting = (name) =>`hello ${name}`;
greeting('Tom');
```

화살표 함수

- **일반 함수 내의 this**
  - 해당 함수를 호출한 객체
  - 또는 call(), apply()를 이용하여 임의의 객체로 설정 가능

- **화살표 함수 내의 this**
  - 상위 스코프에서 상속

ex03.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>

    <style>
      .opening {
        background-color: red;
      }
    </style>
  </head>

  <body>
    <div class="box opening">This is a box</div>
```

화살표 함수

ex03.html

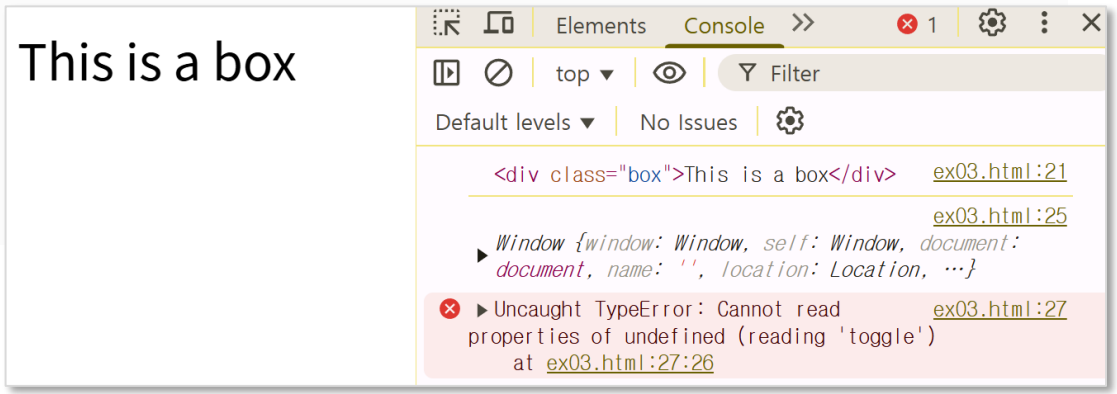
```
    <script>
      // box 클래스를 가진 div를 가져온다
      const box = document.querySelector('.box');

      // click 이벤트 핸들러를 등록
      box.addEventListener('click', function () {
        console.log(this); // 이벤트를 발생시킨 객체 - box
        // div에 opening 클래스를 토글
        this.classList.toggle('opening');

        setTimeout(function () {
          console.log(this); // setTimeout의 콜백은 window 객체가 호출 - window
          // 클래스를 다시 토글
          this.classList.toggle('opening'); // 에러 발생
        }, 500);
      });
    </script>
  </body>
</html>
```

이벤트 콜백 시 콜백함수는
click을 당한 태그요소가 this가 된다

setTimeOut을 처리하는 것은 윈도우 객체가 하기 때문에. 등록된 콜백함수의 this는 window객체가 된다

나는 this를 상위 스코프에서 상속받게금 하고 싶은데?

화살표 함수로 ㄱㄱ



화살표 함수

ex04.html

```
    <script>
      // box 클래스를 가진 div를 가져온다
      const box = document.querySelector('.box');

      // click 이벤트 핸들러를 등록
      box.addEventListener('click', function () {
        console.log(this); // 이벤트를 발생시킨 객체 - box
        // div에 opening 클래스를 토글
        this.classList.toggle('opening');

        setTimeout(() => {
          console.log(this); // 상위 스코프에서 상속 - box
          // 클래스를 다시 토글
          this.classList.toggle('opening');
        }, 500);
      });
    </script>
  </body>
</html>
```

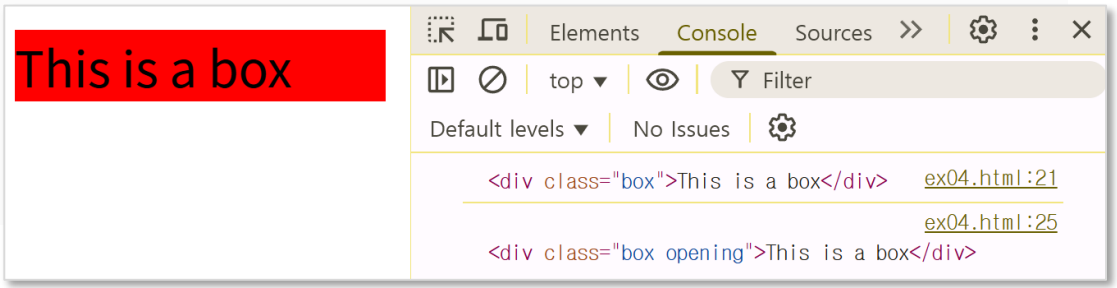
뎃네

근데 나는 이벤트 헨들러의 콜백함수도 상위 스코프의 this를 상속받게 하고 싶은데?

하지만 이 경우 화살표 함수로 바꾸면 상위 스코프는 window객체 스코프이기때문에

태그 요소가 아닌 window객체이기 때문에 의도적으로 작동하지 않을 수 있다.

이 경우 해당 함수를 화살표 함수로 바꾸면 안된다.



화살표 함수

화살표 함수를 피해야 하는 경우

top-level에서 사용하는 경우

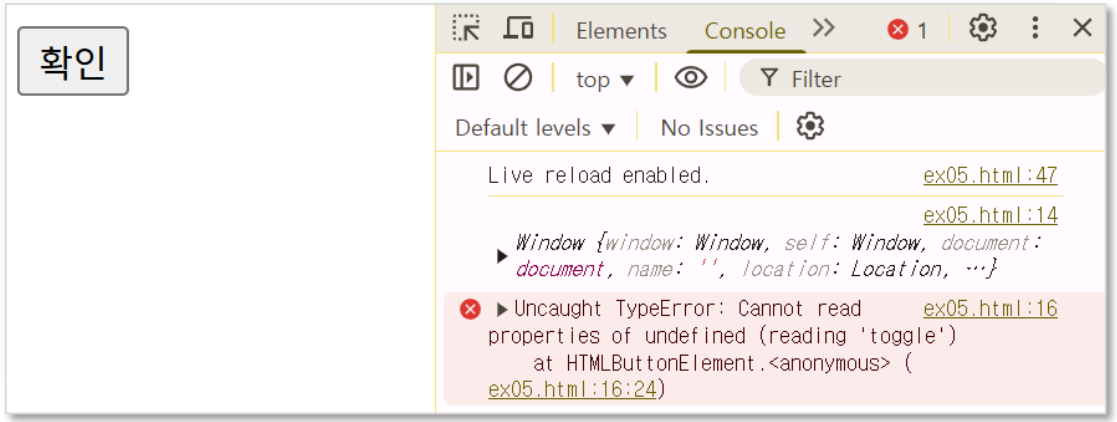
- this는 window 또는 global 객체가 됨

객체 리터럴의 메서드 정의에서 사용하는 경우

화살표 함수

ex05.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <button>확인</button>

    <script>
      const button = document.querySelector('button');
      button.addEventListener('click', () => {
        console.log(this);
        // 오류: 여기서는 this는 window 객체
        this.classList.toggle('on');
      });
    </script>
  </body>
</html>
```



화살표 함수

ex06.js

```
const person1 = {
  age: 10,
  grow: function () {
    this.age++; // this는 person1
    console.log(this.age);
  },
};

person1.grow();

const person2 = {
  age: 10,

  grow: () => {
    this.age++; // 오류: 여기서 this는 global 객체를 가리킴
    console.log(this.age);
  },
};

person2.grow();
```

```
11
NaN
```

화살표 함수

arguments

일반 함수

- 해당 함수에 전달된 인수의 값을 가지는 유사 배열 객체

화살표 함수

- 브라우저에서는 arguments가 없음
- node에서는 실행 정보를 가지는 전역 객체가 됨
- 동일한 효과를 가지려면 ... 나머지 연산자로 처리

화살표 함수

ex07.js

```
function example() {
  console.log(arguments);
  console.log(arguments[0]);
}

example(1, 2, 3);
```

```
[Arguments] { '0': 1, '1': 2, '2': 3 }
1
```

화살표 함수

ex08-1.js

```
const showWinner = () => {
  const winner = arguments[0];
  console.log(`${winner} was the winner`);
};

showWinner('Usain Bolt', 'Justin Gatlin', 'Asafa Powell');
```

```
[object Object] was the winner
```

ex08-2.js

```
const showWinner = (...args) => {
  console.log(args);
  const winner = args[0];
  console.log(`${winner} was the winner`);
};

showWinner('Usain Bolt', 'Justin Gatlin', 'Asafa Powell');
```

```
[ 'Usain Bolt', 'Justin Gatlin', 'Asafa Powell' ]
Usain Bolt was the winner
```