01

스코프(Scope)

학습내용

01 스코프(Scope)란

02 스코프(Scope)의 종류

03 함수 레벨 스코프(var)

04 렉시컬 스코프(Lexical Scope)

05 스코프(Scope) 체인

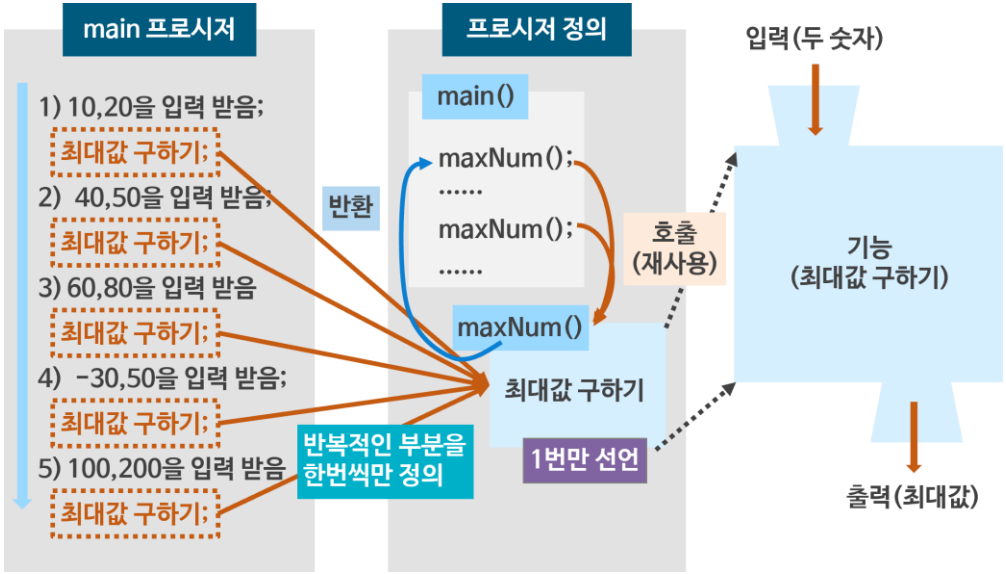
학습목표

- 스코프의 개념을 이해하고,
  변수의 유효 범위를 설명할 수 있다.
- 전역 스코프, 함수 스코프, 블록 스코프의
  차이를 구분할 수 있다.
- 함수 레벨 스코프와 렉시컬 스코프의 특징을
  예제를 통해 설명할 수 있다.
- 스코프 개념을 적용하여 변수 충돌을
  방지하는 코드를 작성할 수 있다.
- 스코프 체인의 작동 원리와 중첩 함수에서의
  변수 접근 흐름을 설명할 수 있다.

지난 주차 복습

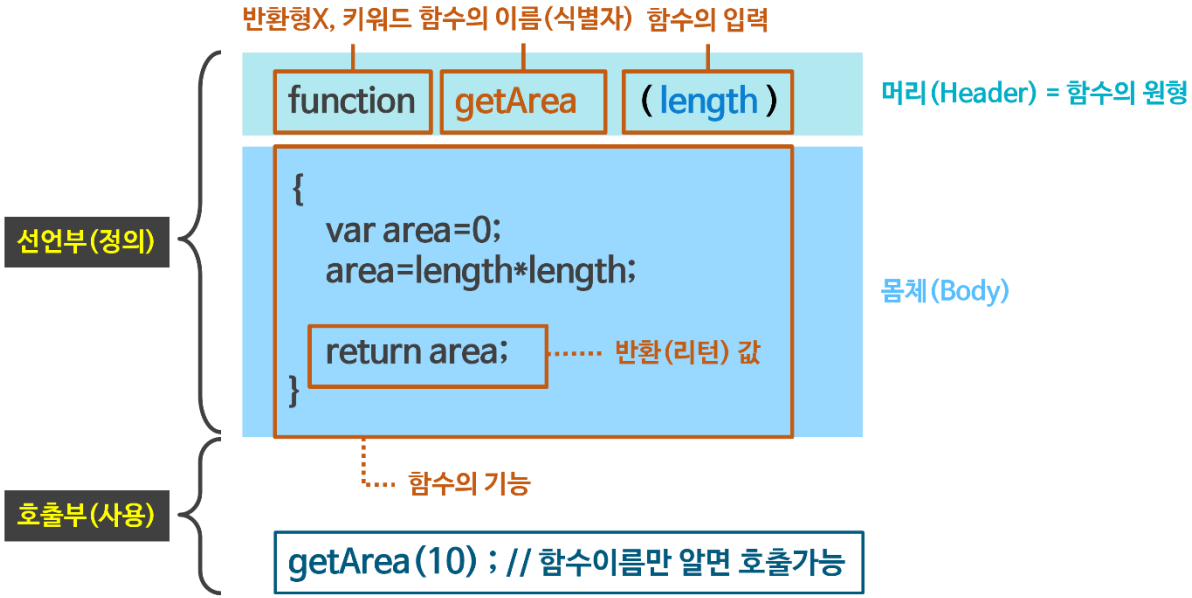
함수란

- 1번의 선언 + 여러 번의 호출로 코드를 재사용함



지난 주차 복습

함수의 정의



생각 해보기

Q JavaScript는 다음과 같이 멀티 패러다임을 지원하는 언어입니다.
이러한 특징은 다양한 문제에 적합한 프로그래밍 방식을 다양하게 선택 가능하다는 점에서 장점이 있습니다. 다음 프로젝트 예시(Node.js 기반 채팅 서버, 실시간 주가 데이터 대시보드) 중 하나를 선택해 적절한 패러다임을 적용해 보세요. 프로젝트 특성에 따라 패러다임을 복합적으로 선택할 수 있습니다.

제시된 프로젝트 중 선택	Node.js 기반 채팅 서버, 실시간 주가 데이터 대시보드
정답 예시	관리자용 ERP 시스템(Angular) = 객체지향 + 비동기

패러다임의 종류	지원 방식	예시
절차지향(Procedural)	순서대로 명령을 실행하는 전통적인 방식	for, if, switch 등
객체지향(Object-Oriented)	객체, 클래스, 상속, 캡슐화 등 지원	class, constructor, extends
함수형(Functional)	일급 함수, 고차 함수, 불변성 등 지원	map, reduce, 클로저, 순수 함수
이벤트 기반(Event-Driven)	UI, 서버 등에서 이벤트에 반응하는 구조	onclick, addEventListener, Node.js
비동기/반응형 (Asynchronous/Reactive)	Promise, async/await, RxJS 등	fetch(), await, Observable

01

스코프(Scope)란

스코프(Scope)

변수에 접근할 수 있는 범위(유효 범위)

JavaScript 정적 스코프 (Lexical Scope)의 원칙을 준수

변수의 접근 가능 여부(= 스코프)는 “실행되는 위치”가 아니라
“변수가 선언된 위치”에 의해 결정(코드 작성 시 결정)됨

2) JavaScript의 스코프

- 전역 스코프와 지역(함수,블록) 스코프
- 함수 밖/안에서 변수 선언과 호출 테스트를 통해 유효 범위 확인 가능

```
var x = 10;  //전역 변수

function showX() {
  var y=20;  //지역 변수
  console.log(x); // 접근 가능
}

showX();
console.log(x); // 접근 가능
// console.log(y); // 접근 불가능
```

x는 전역 변수,
y는 지역 변수
(해당 지역에서만
접근 가능)로 작동
➡ 선언된 위치에
따라 접근 가능
여부가 결정됨

02

스코프(Scope)의 종류

1) 스코프의 종류

전역 스코프	코드 어디에서든 접근 가능 예 let x = 1;
함수 스코프	함수 내부에서만 접근 가능 ➡ var 로 선언 예 { var y = 2; }
블록 스코프	블록 내에서만 접근 가능 ➡ let, const로 선언 예 if (true) { let z = 3; }

2) 스코프 차이


```
function test() {
  if (true) {
    var a = "var";
    let b = "let";
    console.log(a); // var
    console.log(b); // let
  }
  console.log(a);  // var (함수 스코프)
  // console.log(b); // ✕ 오류 (블록 스코프)
}
test();
```

let: 블록 내부에서만 접근 가능

ReferenceError: b is not defined

03

함수 레벨 스코프(var)

1) var선언(함수 레벨 스코프)의 특징

var는 블록 단위가 아니라 함수 단위 스코프를 가짐

함수내 블록(if, for, {})에서 선언해도 함수 전체에서 접근 가능함

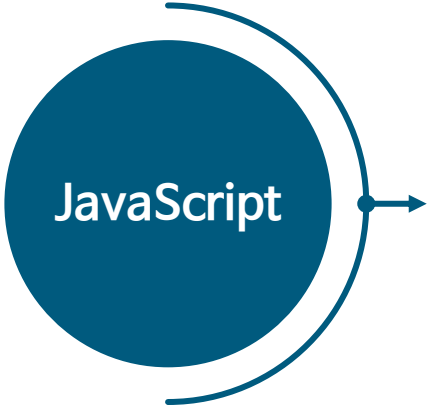
```
function test() {
  if (true) {
    var x = 5; // let과 비교
  }
  console.log(x); // 5 (var는 함수 레벨 스코프)
}
test();
```

- var 와 let의 스코프 차이 확인
- 블록 내 변수 선언과 외부 접근 여부 비교

04

렉시컬 스코프 (Lexical Scope)

1) 렉시컬 스코프(Lexical Scope)의 특징



“ 렉시컬 스코프 기반 언어 ”

변수의 접근 가능 여부는 변수가 선언된 위치에 의해 미리 결정됨

실행 위치 X

1) 렉시컬 스코프(Lexical Scope)의 특징

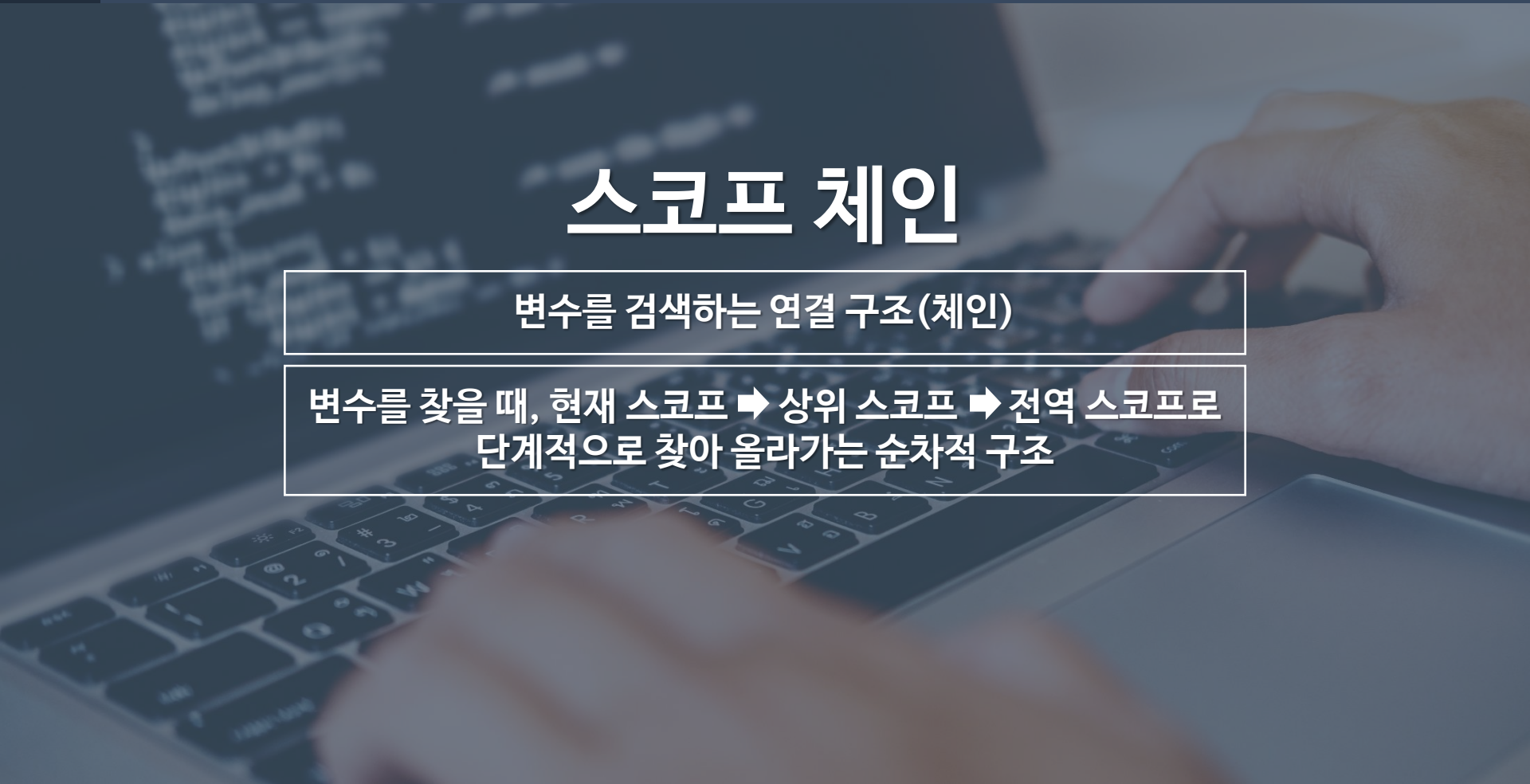

```
var msg = "전역";

function outer() {
  var msg = "outer";
  function inner() {
    console.log(msg); // "outer" — 정의된 위치 기준
  }
  return inner;
}
const fn = outer();  // inner 함수 값이 반환됨 ➡ 클로저
fn(); // "outer"
```

inner는 outer 안에서 선언되었기 때문에, msg는 렉시컬 스코프 기준으로 'outer'를 참조 ➡ 실행 위치와 상관없이 inner는 자신이 정의된 위치를 기준으로 변수 탐색

반환된 inner 함수는 outer의 실행 컨텍스트가 끝난 뒤에도 그 환경(스코프)에 대한 참조 유지

05

스코프(Scope) 체인



스코프 체인

변수를 검색하는 연결 구조(체인)

변수를 찾을 때, 현재 스코프 ➡ 상위 스코프 ➡ 전역 스코프로
단계적으로 찾아 올라가는 순차적 구조

2) 중첩 함수에서의 변수 참조 범위


```
let a = “global”;

function outer() {
  let b = “outer”;

  function inner() {
    let c = “inner”;
    console.log(a); // “global”
    console.log(b); // “outer”
    console.log(c); // “inner”
  }

  inner();
}

outer();
```

각 스코프의 위치에 따라
어떤 변수를 참조하는지
예측해 보기

➡ 스코프 체인을 따라
“가장 가까운” 변수 탐색

세종사이버대학교

05주. 나만의 기능 만들기 : 여러 가지 함수

02

고급 함수의 활용