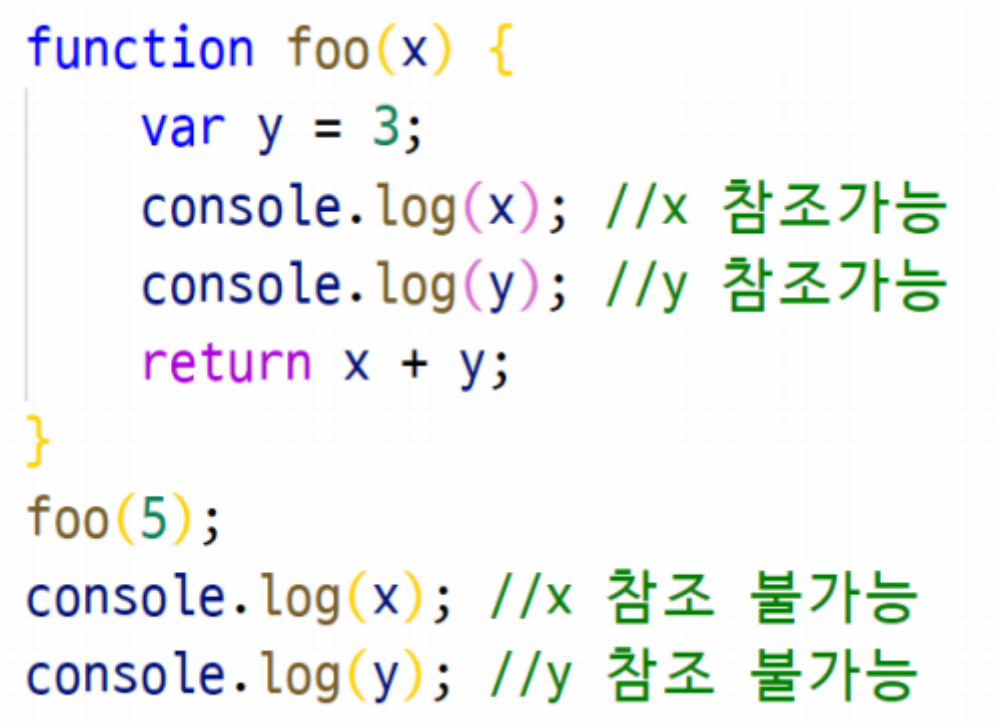
4강 - 스코프와 var키워드의 문제점

스코프(scope)란??

- 모든 식별자(변수, 함수, 클래스 등)는 자신이 선언된 위치에 의해 다른 코드가 자신을 참조할 수 있는 유효 범위가 결정되는데 이를 스코프라고 부릅니다.



전역 스코프와 전역 변수 (global scope)

- 코드의 가장 바깥쪽 영역에서 선언된 변수를 전역 변수라고 부르며, 전역 변수는 코드 전체
의 전역 스코프에서 참조가 가능합니다.

- 전역 변수는 코드 모든부분에서 사용이 가능합니다.
- 전역 변수의 수명은 프로그램의 수명과 같다.

```
var x = 'global x';
var y = 'global y';

function outer() {
    console.log(x);
    console.log(y);

    function inner() {
        console.log(x);
        console.log(y);
    }
    inner();
}
outer();
console.log(x);
console.log(y);
```

지역 스코프와 지역 변수 (local scope)

- 지역이란 함수 몸체 내부를 말합니다. 지역 변수는 자신이 선언된 지역과 하위 지역(중첩 함
  수)에서만 참조할 수 있습니다.
- 즉, 지역 변수는 자신의 지역 스코프와 하위 지역 스코프에서만 유효합니다.
- 전역 변수와 지역 변수의 이름이 같을 경우 함수 내부에선 지역 스코프를 먼저 참조합니다.

```
var x = 'global x';
function outer(x) {
    var y = 'outer local y';
    console.log(x);
    console.log(y);
    // console.log(z); //참조 불가능

    function inner() {
        var z = 'inner local z';
        console.log(x);
        console.log(y);
        console.log(z);
    }
    inner();
}
outer('outer local x');
// console.log(x);  //모두 참조 불가능
// console.log(y);
// console.log(z);
```

전역변수 사용은 최대한 자제해야 한다!!

1. 암묵적 결합

- 전역 변수를 선언한 의도는 모든 코드가 전역 변수를 참조하고 변경할 수 있는 암묵적 결합(Implicit coupling)을 허용하는 것입니다.
- 변수의 유효 범위가 클 수록 코드의 가독성이 나빠지고 의도치 않게 상태가 변경될 수 있는 위험성도 높아집니다.

2. 긴 생명 주기

- 전역 변수는 살아있는 시간이 길어서 메모리 리소스도 오랜 기간 소비합니다. 더욱이 var 키워드는 변수의 중복 선언을 암묵적으로 허용해주므로 변수 이름이 중복된 전역 변수는 의도치 않는 재할당으로 인해 변경이 일어납니다.

3. 스코프 체인 상 종점에 존재

- 변수를 검색할 때 지역 스코프부터 검색하므로 전역 변수는 검색속도가 가장 느립니다.

var 키워드의 한계

변수 중복 선언 허용

```
var x = 1;
var x = 100;

console.log(x); // 100
```

- var 키워드로 변수를 선언하면 동일한 이름으로 중복 선언시 var 키워드를 안 붙인 것처럼 동작합니다.
- 위와 같이 동일 이름 변수가 선언된 지 모르고 변수를 중복 선언하면 의도치 않게 변수의 값이 변경되는 부작용이 발생합니다.

블록 레벨 스코프를 지원하지 않는다!

```
var x = 1;

if (true) {
    var x = 10;
}
console.log(x);  // 10
```

- var 키워드 변수는 오로지 함수의 영역만을 지역 스코프로 인정합니다.
- 따라서 함수가 아닌 블록들에서는 모두 전역 변수로 일괄 적용됩니다.
- 이는 전역 변수를 남발할 가능성이 커지는 문제가 생깁니다.

변수 호이스팅 문제

```
y = 100;
console.log(`y: ${y}`);

var y;
```

- var 키워드로 변수를 선언하면 변수 호이스팅에 의해 변수 선언문이 자동으로 맨 위로 끌어올려진 것처럼 동작합니다.
- 이는 프로그램의 흐름을 방해하여 코드의 가독성과 유지보수성을 현격하게 떨어뜨립니다.

var의 문제를 해결해주는 let과 const

let 키워드

변수 중복선언 불허용

```
let x = 3;
let x = 4; // SyntaxError!
```

```
SyntaxError: Identifier 'x' has already been declared
```

- ES6에서는 var의 단점을 보완하기 위해 새로운 변수 선언 키워드 let과 const가 등장했습니다.
- let은 var와 달리 변수의 중복 선언을 허용하지 않고 에러를 발생시켜 안전성을 제공합니다.

블록레벨 스코프를 지원!

```
let y = 10;
if (true) {
    let y = 20;
    console.log(`if내부 y: ${y}`);
}
console.log(`if외부 y: ${y}`);
```

```
if내부 y: 20
if외부 y: 10
```

- let 키워드로 선언한 변수는 모든 코드 블록(함수, if문, for문, while문, try/catch문 등)을 각각의 지역 스코프로 인정하는 블록 레벨 스코프를 지원합니다.
- 이는 전역 변수의 사용을 제어할 수 있는 좋은 방법으로 쓰일 수 있습니다.

변수 호이스팅이 일어나지 않는다

```
z = 10;
console.log(z); //ReferenceError

let z;
```

```
ReferenceError: Cannot access 'z' before initialization
```

- let 키워드로 선언된 변수는 ‘선언 단계’와 ‘초기화 단계’를 엄격하게 구분합니다.
- 따라서 암묵적으로 호이스팅이 일어나지 않아 안전한 코딩을 할 수 있게 도와줍니다.

const 키워드

const 변수는 반드시 선언과 동시에 초기화해야 한다!

```
const x; //SyntaxError
```

```
SyntaxError: Missing initializer in const declaration
```

- const 키워드는 상수를 선언하기 위해 사용합니다.
- const 키워드로 선언한 변수는 let과 마찬가지로 블록레벨 스코프를 지원합니다.
- const는 let과 달리 반드시 초기에 값을 할당하는 초기화 작업을 필수로 해야 합니다.

상수란??

```
//세율
const TAX_RATE = 0.1;
//세전 가격
let preTaxPrice = 100;
//세후 가격
let afterTaxPrice = preTaxPrice + (preTaxPrice * TAX_RATE);
```

- 상수란 변수의 반대 개념으로 값 변경이 금지된 변수를 말합니다.
- 상수는 값의 상태 유지와 가독성, 유지 보수의 편의를 위해 적극적으로 사용해야 합니다.
- 상수 이름은 대문자로 지정하여 상수임을 관례적으로 알립니다.

객체의 불변성을 지키기 위한 const의 사용

객체의 불변성이란?

- 객체의 **불변성**은 객체가 생성된 이후 해당 객체의 상태를 변경할 수 없는 성질을 의미합니다.
- 이런 성질을 적용해서 코딩을 하면, 원본 객체의 데이터가 변경되거나 훼손되는 것을 방지할 수 있다.
- 따라서 변수를 선언할 때는 const를 우선적으로 사용하고 변경이 필요한 부분에서만 부분적으로 let을 사용하는 것이 좋다!

1. ES6부터는 var 키워드는 사용하지 마세요!

2. 재할당이 필요한 경우에만 한정하여 let을 사용하세요! 이 때, 변수의 스코프는 최대한 좁게 만드세요.

3. 변경이 발생하지 않고 읽기 전용으로 사용하는 기본 타입의 값과 객체에는 const를 사용하세요.