삼육대학교 SW융합교육원 김진호

자바스크립트 기초교육

<class>
</body>
<head>
<title>

06. scope

6-1. scope

6-2. let and const

6. scope

6-1. scope

6-1-1. global and local scope (전역과 지역 스코프)

- 전역은 코드의 가장 바깥 영역을 말하며 전역은 전역 스코프를 만든다.
  - 전역에 변수를 선언하면 전역 스코프를 갖는 전역 변수가 되며 전역 변수는 어디서든지 참조할 수 있다.

- 지역이란 함수 몸체 내부를 말하며 지역은 지역 스코프를 만든다.
  - 지역에 변수를 선언하면 지역 스코프를 갖는 지역 변수가 되며 자신의 지역 스코프와 하위 지역 스코프에서 유효하다.

6. scope

6-1. scope

6-1-1. global and local scope (전역과 지역 스코프)

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

function outer() {
    var z = "outer's local z";

    console.log(x);             // global x
    console.log(y);             // global y
    console.log(z);             // outer's local z

    function inner() {
        var x = "inner's local x";

        console.log(x);         // inner's local x
        console.log(y);         // global y
        console.log(z);         // outer's local z
    }

    inner();
}

outer();

console.log(x);                 // global x
// console.log(z);              // ReferenceError: z is not defined
```

6. scope

6-1. scope

6-1-2. function level scope (함수 레벨 스코프)

```
// 함수 밖에서 var 키워드로 선언 된 변수는 전역 변수이다.
var i = 0;

// for 코드 블록 내부에서 i라는 변수를 선언한다.
// 이는 전역 변수로 이미 선언 된 전역 변수 i가 있어 중복 선언된다.
for(var i = 0; i < 10; i++) {}

// 의도와 달리 for 코드 블록 내부에서의 값 변화가 반영된다.
console.log(i);
```

- C, 자바 등 대부분의 프로그래밍 언어는 함수 몸체만이 아니라 모든 코드 블록(if, for, while, try/catch 등)이 지역 스코프를 만드는 블록 레벨 스코프(block level scope)를 가진다.
- 하지만 var 키워드로 선언 된 변수는 오로지 함수의 코드 블록(함수 몸체)만을 지역 스코프로 인정하는 함수 레벨 스코프(function level scope)를 가진다.
- ES6에서 도입된 let, const 키워드는 블록 레벨 스코프를 지원한다.

6. scope

6-2. let and const

6-2-1. var

ES5까지 변수를 선언할 수 있는 유일한 방법은 var 키워드를 사용하는 것이었는데
이는 몇 가지 문제를 야기한다.

- **변수 중복 선언 허용**
  - var 키워드로 선언 된 변수는 같은 스코프 내에서 중복 선언이 허용된다.
  - 초기화 문이 있는 변수 선언문은 자바스크립트 엔진에 의해 var 키워드가 없는 것처럼 동작한다.
  - 초기화문이 없는 변수 선언문은 무시된다.

```
var msg = '안녕하세요';
console.log(msg);
var msg = '안녕히가세요';
console.log(msg);
var msg;
console.log(msg);
```

6. scope

6-2. let and const

6-2-1. var

- **함수 레벨 스코프**
  - 함수 외부에서 var 키워드로 선언한 변수는 코드 블록 내에서 선언해도 모두 전역 변수가 된다.

```
var i = 0;
for(var i = 0; i < 10; i++) {}
console.log(i); // 의도치 않게 값이 0에서 10으로 변경되었다.
```

6. scope

6-2. let and const

6-2-1. var

- **변수 호이스팅**
  - var 키워드로 변수를 선언하면 변수 호이스팅에 의해 변수 선언문이 스코프의 선두로 끌어올려진 것처럼 동작한다.
  - 즉, 변수 선언문 이전에 참조할 수 있다.
  - 실행 시 오류가 발생하지는 않지만 이는 프로그램의 흐름에 맞지 않고 가독성을 떨어트리며 오류를 만들 여지가 있다.

```
console.log(test);  // 변수는 이미 선언되었고 undefined로 초기화 되었다.
test = '반갑습니다';
console.log(test);
var test;   // 변수 선언은 런타임 이전에 암묵적으로 실행 된다.
```

6. scope

6-2. let and const

6-2-2. let

var 키워드의 단점을 보완하기 위해 ES6에서는 새로운 변수 선언 키워드인 let, const를 도입했다.

■ **변수 중복 선언 금지**

▫ let이나 const 키워드로 선언 된 변수는 같은 스코프 내에서 중복 선언을 허용하지 않는다.

```
let msg = '안녕하세요';
let msg = '안녕히가세요';    // Syntax Error
```

6. scope

6-2. let and const

6-2-2. let

var 키워드의 단점을 보완하기 위해 ES6에서는 새로운 변수 선언 키워드인 let, const를 도입했다.

- **블록 레벨 스코프**
  - let 키워드로 선언한 변수는 모든 코드 블록(함수, if문, for문, while문, try/catch문)을 지역 스코프로 인정한다.

```
let i = 0;
for(let i = 0; i < 10; i++) {
    console.log(`지역 변수 i : ${i}`)
}
console.log(`전역 변수 i : ${i}`);
```

6. scope

6-2. let and const

6-2-2. let

- **변수 호이스팅**
  - let 키워드로 선언한 변수는 변수 호이스팅이 발생하지 않는 것처럼 동작한다.
  - var 키워드는 선언 단계와 초기화 단계가 함께 진행되지만 let 키워드는 선언 단계와 초기화 단계가 분리되어 진행된다.
  - 따라서 선언은 되었지만 초기화가 되지 않아 참조 시 오류가 발생한다.

```
// console.log(x);   // ReferenceError: Cannot access 'x' before initialization
let x;

// 하지만 변수 호이스팅이 발생하지 않는 것은 아니다.
let y = 1;
if(true){
    // 변수 호이스팅으로 아래의 지역변수 y 선언이 먼저 일어났기 때문에 전역 변수 y를 참조하지 않고 오류가 발생한다.
    // console.log(y); // ReferenceError: Cannot access 'y' before initialization
    let y = 2;
}
```

6. scope

6-2. let and const

6-2-3. const

const 키워드는 상수(constant)를 선언하기 위해 사용한다. let 키워드와 마찬가지로 블록 레벨 스코프를 가지며 변수 호이스팅이 발생하지 않는 것처럼 동작한다. 아래에서는 let 키워드와 다른 점을 중심으로 살펴본다.

```
// const x;          // Syntax Error
const x = 1;

// x = 2; // TypeError: Assignment to constant variable.
```

- const 키워드로 선언한 변수는 반드시 선언과 동시에 초기화 해야 한다.
- const 키워드로 선언한 변수는 재할당이 금지된다.

※ 정리하기

ES6를 사용한다면 var 키워드는 사용하지 않는다.

재할당이 필용한 경우에 한정해 let 키워드를 사용하며 변수의 스코프는 최대한 좁게 만든다.

변경이 발생하지 않고 읽기 전용으로 사용하는 원시 값과 객체에는 const 키워드를 사용한다.

const 키워드는 재할당을 금지하므로 var, let 키워드보다 안전한다.