알고리즘 스터디

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 요일 | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
|  | 이주연 | 박해미 | 황유진 | 박해미 | 황유진 | 김승미 | 김승미 |
|  | 김세린 | 김주영 | 이주연 |  | 김주영 | 김세린 |  |
|  |  |  |  |  |  |  |  |

**var, const, let의 차이점**

## 1. var

* ES6 이전의 변수 선언 방식이다.
* 함수 단위의 스코프를 가진다.
* 다른 언어와 다르게 유연한 방식으로 변수 선언이 가능하지만 이로 인해 다음과 같은 문제점들이 생긴다.

### 문제점 1. 스코프

var foo='hello';

if(true) {

var foo = 'hello if';

}

console.log(foo); // hello if

만약 위의 구문이 하나의 함수 안에 존재한다고 가정하면,  
if문 안의 foo와 if문 밖의 foo는 동일한 변수가 되며 값이 변경된다.

### 문제점 2. 여러 번 선언

var foo='hello1';

var foo='hello2';

console.log(foo); // hello2

같은 변수를 두 번 선언해도 오류가 나지 않는다.  
이러한 유연한 방식은 편리한 점도 있지만 많은 오류를 발생시킨다.

### 문제점 3. 선언하기 전 호출

console.log(foo); // undefined

var foo;

선언보다 호출이 먼저 있었음에도, 호이스팅 현상에 의해 정상적으로 코드가 실행된다.

## 2. const

* 상수를 선언한다.
* var와 다르게 변수 재선언 불가능이다.
* 블록 단위 스코프를 가진다.
* 참조형(배열, 객체, 함수)의 경우 const로 선언하는 것이 바람직하다.

const foo = [0, 1];

const bar = foo;

foo.push(2);

bar[0] = 10;

console.log(foo, bar) // [10, 1, 2] [10, 1, 2]

참조형은 const로 선언하더라도 멤버 값을 조작하는 것이 가능하다.

## 3. let

* 변수를 선언한다.
* var와 다르게 변수 재선언 불가능이다.
* 하지만 값을 재정의(재할당)는 할 수 있다.
* 블록 단위 스코프를 가진다.

## TDZ (Temporal Dead Zone)

호이스팅에 대한 이해가 요구된다.

* 변수가 선언되기 전까지는 액세스할 수 없는 현상을 말한다.
* let, const는 TDZ에 의해 제약을 받는다. 이는 var와의 차이점 중 하나이다.

console.log(foo); // Error: Uncaught ReferenceError: foo is not defined

let foo;

: var와 다르게 호출한 시점에서 변수가 선언되어 있지 않음을 알리는 에러가 발생한다.

let foo = 'bar1';

console.log(foo); // bar1

if (true) {

console.log(foo); // Uncaught ReferenceError: foo is not defined

let foo = 'bar2';

}

어떤 변수가 호출되었을 때 블록 안에 같은 이름의 변수가 없으면 상위 블록에서 선언된 같은 이름의 변수를 호출한다.

하지만 블록 안에서 let이나 const로 변수 선언이 있었다면 그 이름의 변수는 변수가 선언되기 이전까지 그 블록안에서는 정의되지 않은 변수로 간주된다.

## 결론

* ES6에서는 var보다 const와 let을 사용해 좀 더 명확한 코드를 만들어 내는 것을 권장한다.
* 참조형은 const로 선언한다.