호이스팅

호이스팅?

코드를 실행하기 전 변수 및 함수 선언을 해당 스코프의 최상단으로 끌어올리는 것이 **아닙니다**. 호이스팅은 코드가 실행하기 전 변수 및 함수 선언이 해당 스코프의 최상단으로 **끌어 올려진 것 같은 현상**을 말합니다.

JavaScript 엔진은 코드를 실행하기 전 **실행 가능한 코드를 형상화하고 구분하는 과정(실행 컨텍스트를 위한 과정)**을 거친다. 이 과정에서 **모든 선언(var, let, const, function, class)을 스코프에 등록**한다.

- 실행 컨텍스트란?
    - 실행 가능한 코드가 실행되기 위해 필요한 환경을 의미한다.

코드 실행 전 이미 변수 및 함수 선언이 저장되어 있기 때문에 선언문보다 참조/호출이 빨라도 오류 없이 동작한다. (정확히는 var 키워드로 선언한 변수와 함수 선언문일 경우 오류 없이 동작한다. 이는 선언이 파일의 맨 위로 끌어올려진 것처럼 보이게 한다.)

변수 호이스팅

자바스크립트의 모든 선언에는 호이스팅이 일어납니다. 하지만 let, const, class를 이용한 선언문은 호이스팅이 발생하지 않는 것**처럼** 동작합니다.

예를 들어, var 키워드로 선언된 변수와 달리 **let 키워드로 선언된 변수를 이전에 참조하면 참조 에러(ReferenceError) 가 발생**합니다. 이는 let 키워드로 선언된 변수는 **스코프의 시작에서 변수의 선언까지 일시적 사각지대(TDZ) 에 빠지기 때문**입니다.

여기서 중요한 것은 이 호이스팅이라는 용어가 **‘선언이 먼저 메모리에 저장되었다.’** 는 것을 의미하기 때문에 모든 선언은 호이스팅이 일어난다는 말은 참이 된다는 것입니다. 즉, **let 키워드에서 참조 에러가 나온다고 해서 호이스팅이 일어나지 않은 것이 아니라는 것**입니다. 선언은 분명히 코드 실행 전에 메모리에 저장된 것은 맞으나, var의 경우 선언과 함께 undefined로 초기화되어 메모리에 저장되지만 let, const는 초기화되지 않은 상태로 선언만 메모리에 저장되기 때문입니다. 초기화 되지 않은 변수는 참조할 수 없기 때문에 에러가 나는 것입니다.

변수의 생성 및 호이스팅 단계

변수는 총 3단계에 걸쳐 생성됩니다.

1단계 : 선언 단계

- 변수를 실행 컨텍스트의 변수 객체에 등록합니다.
- 이 변수 객체는 스코프가 참조하는 대상이 됩니다.

2단계 : 초기화 단계

- 변수 객체에 등록된 변수를 위한 공간을 메모리에 확보합니다.
- 이 단계에서 변수는 undefined로 초기화 됩니다.

3단계 : 할당 단계

- undefined로 초기화된 변수에 실제 값을 할당합니다.

var 키워드의 경우 선언 및 초기화 단계가 한 번에 이뤄집니다. 따라서 변수 선언문 이전에 변수에 접근해도 undefined를 반환할 뿐입니다.

하지만 **let 키워드로 선언된 변수는 선언 단계와 초기화 단계가 분리되어 진행됩니다**. 즉, 스코프에 변수를 등록(선언) 하지만 초기화 단계는 변수 선언문에 도달했을 때(코드 실행 후) 이뤄집니다. 그렇기 때문에 참조 에러도 발생하고요. 즉, 변수를 위한 메모리 공간이 아직 확보되지 않았다는 것입니다. 따라서 **스코프의 시작 지점부터 초기화 시작 지점까지는 변수를 참조할 수 없고** 이 구간을 **일시적 사각지대 즉, TDZ** 라고 부릅니다.

const

let과 달리 선언 단계와 초기화 단계가 동시에 진행됩니다. 다만 선언과 초기화가 **무조건** 동시에 일어나야 하기 때문에 런타임 이전에는 실행될 수 없다.

```
console.log(name) // Error : Cannot access 'name' before initialization

const name = 'yjh'
const good // Error
```

함수 호이스팅

함수의 경우 함수 표현식은 호이스팅이 적용되지 않으나 일반 함수 선언문은 함수 호이스팅이 적용된다.

```
// 1. 함수 선언문
// 함수 이름 생략 불가능
add(1, 2) // 호이스팅 일어남

function add(x, y) {
  return x+y;
}

// 2. 함수 표현식
// 함수 이름 생략 가능

add(1, 2) // 호이스팅 일어나지 않음
var add = function(x, y) {
```

```
  return x+y;
}

// 3. Function 생성자 함수
add(1, 2) // 호이스팅 일어나지 않음
var add = new Function('x', 'y', 'return x+y')

// 4. 화살표 함수
add(1, 2) // 호이스팅 일어나지 않음
var add = (x, y) => x + y
```

```
console.dir(add) // f add(x, y)
console.dir(sub) // undefined // 이 경우 sub가 var 로 선언되었고, 함수 실행도 아니기에 undefined

console.log(add(2, 5)) // 7
console.log(sub(2, 5)) // Uncaught TypeError

// 함수 선언문
function add(x, y) {
  return x+y
}

// 함수 표현식
var sub = function(x, y) {
  return x+y
}
```

우선순위

변수 선언이 함수 선언보다 위로 끌어올려지는 것처럼 보입니다. 단, 값이 할당되어 있지 않은 변수와 값이 할당된 변수 중에서는 값이 할당된 변수가 호이스팅 됩니다.

```
var myName = 'hi'
var yourName

function myName() {
 console.log("jehyeok")
}

function yourName() {
  console.log("zz")
}
```

```
console.log(typeof myName) // string
console.log(typeof yourName) // function
```