JS-중급1

태그	
날짜	@2023년 5월 30일

변수(variable)

```
let //es6 이후로 생김
const //es6 이후로 생김
var

var는 한번 선언된 변수를 다시 선언할 수 있다.

var name = 'Mike';
console.log(name); // Mike

var name = 'Jane';
console.log(name); // Jane

name이 두번 선언되었지만 아무 문제 없다.

let name = 'Mike';
console.log(name); // Mike

let name = 'Jane';
console.log(name); // error!

---------------------------------------------------

var는 선언하기 전에 사용할 수 있다.

console.log(name); // undefined
var name = 'Mike';

name이 나오기전 사용했는데도 에러가 발생하지 않았다.

var는

var name;
console.log(name); // undefined
name = 'Mike';

이렇게 동작한다.
var로 선언한 모든 변수는 코드가 실제로 이동하진 않지만
최상위로 끌어 올려진 것 처럼 동작한다.
```

```
이를 호이스팅(hoisting)이라고 한다.

여기서 콘솔은 undefined를 찍는다.
이유는 선언은 호이스팅이 되지만 할당은 호이스팅이
되지 않기 때문이다.
name 이라는 변수만 올려진 것이고, 'Mike'는 올려지지 않았다.

같은 상황에서의 let
console.log(name); // ReferenceError
let name = 'Mike';

호이스팅이 되지 않는걸까? 아니다.
let과 const도 호이스팅이 된다.
```

호이스팅(Hoisting)

스코프 내부 어디서든 변수 선언은 최상위에 선언된 것처럼 행동

Temporal Dead Zone(TDZ)


```
TDZ영역에 있는 변수들은 사용할 수 없다.
let과 const는 TDZ의 영향을 받는다.
할당하기 전에는 사용할 수 없다.
이는 코드를 예측 가능하게 하고 잠재적인 버그를 줄일 수 있다.


let age = 30;
function showAge(){
```

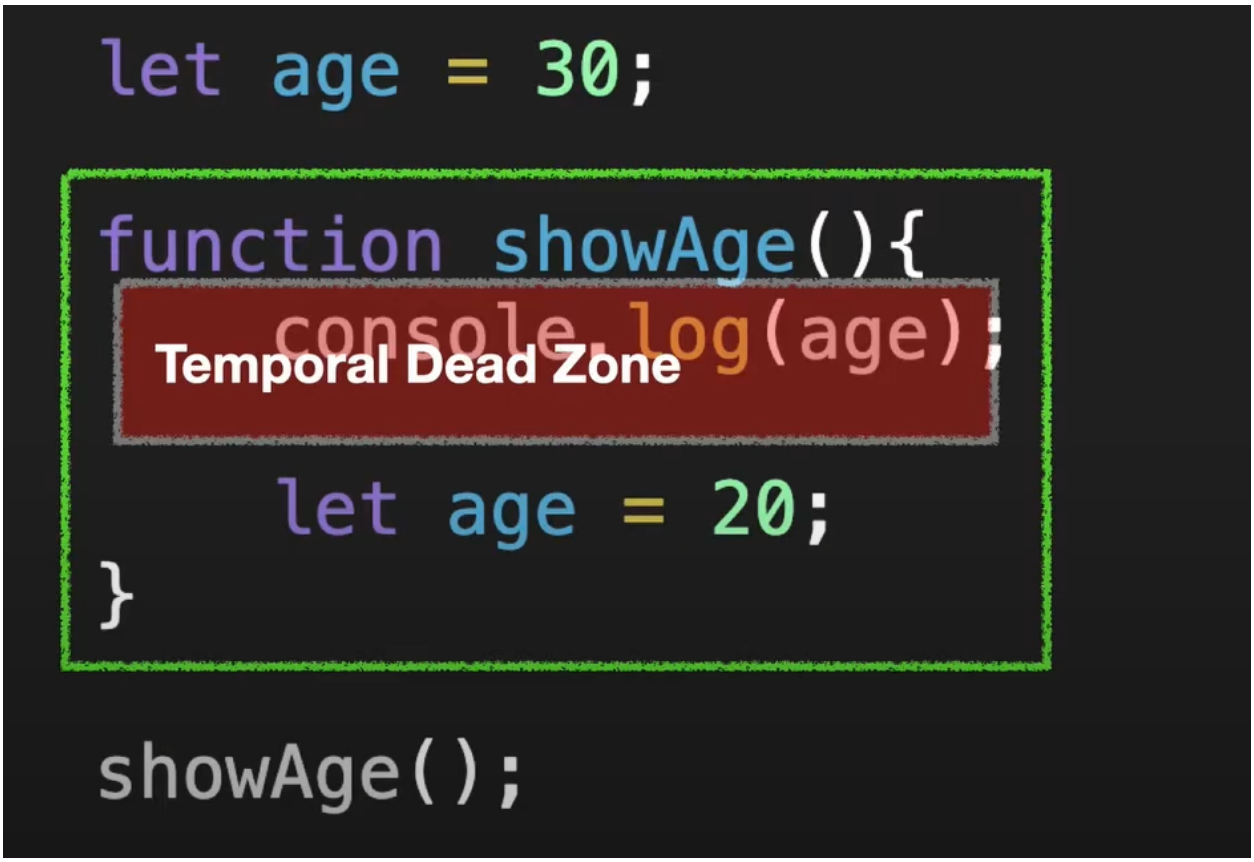
```
  console.log(age);
}
showAge();
```

이 코드는 문제가 없다.

```
let age = 30;
function showAge(){
  console.log(age);

  let age = 20;
}
showAge();
```

이 코드는 문제가 발생한다.
여기서 오해 할 수 있는데 let은 호이스팅 되지 않는구나
생각할 수 있따.

호이스팅은 스코프 단위로 일어난다.
여기서 스코프는



함수 내부이다.

```
let으로 선언한 두번째 변수가 호이스팅을 일으킨다.
만약 호이스팅이 도지 않았다면 함수 바깥에서 선언한 age가
정상적으로 찍혀야 한다.
```

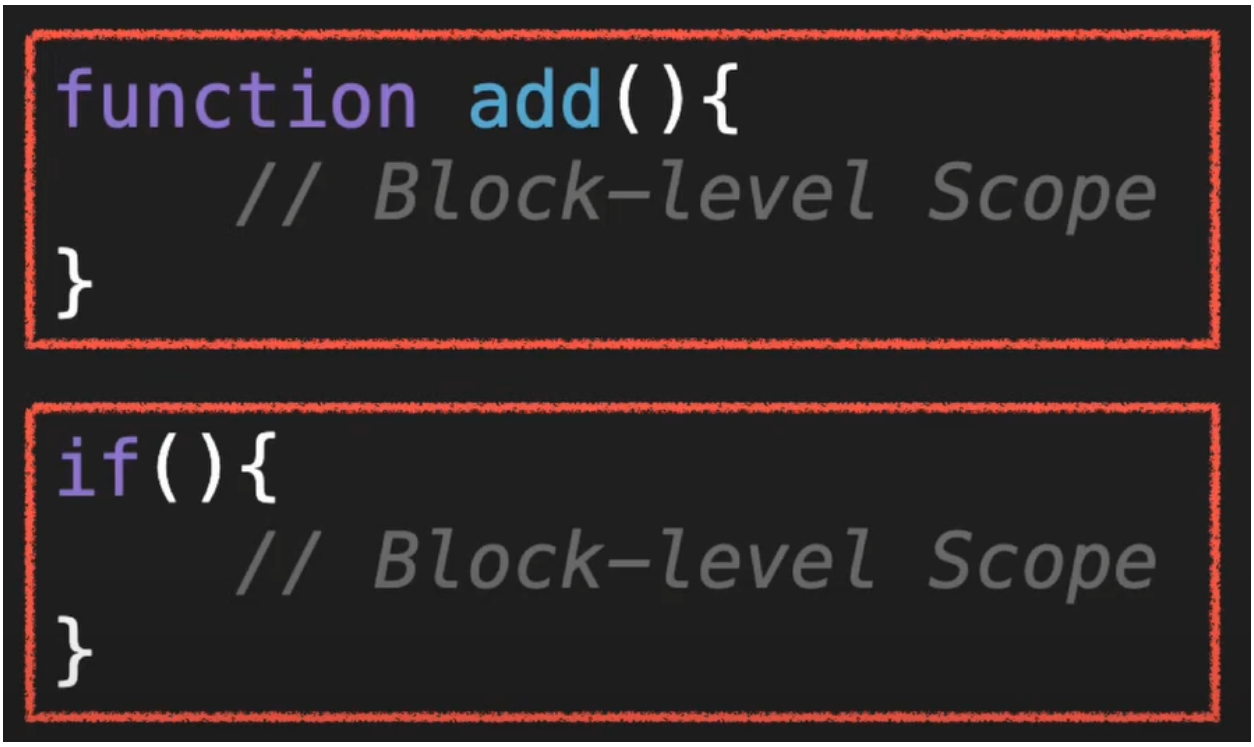
변수의 생성과정

1. 선언단계
2. 초기화 단계
3. 할당 단계

```
var 1. 선언 및 초기화 단계
    2. 할당 단계
선언과 초기화가 동시에 된다.
그래서 할당 전에 호출하면 에러를 내지 않고 undefined가
나온다.

let 1. 선언 단계
    2. 초기화 단계
    3. 할당 단계

let은 선언단계와 초기화 단계가 분리되어서 진행된다.
호이스팅 되면서 선언단계가 이루어 지지만, 선언단계는
실제 코드에 도달했을 때 되기 때문에 레퍼런스 에러가 발생한다.

const 1. 선언 + 초기화 + 할당
const는 선언과 할당이 동시에 되어야 한다.
let과 var는 선언만 해두고 나중에 할당하는 것을 허용한다.
생각해보면 let과 var는 값을 바꿀 수 있기 때문에 어떻게 보면
가능하다.

let name;
name = 'Mike';

var age;
age = 30;

const gender;
gender = 'male';

name과 age는 괜찮지만 const로 선언한 gender부분에서
에러가 발생한다.
선언하면서 할당을 바로 하지 않아서 이다.
```

스코프

```
scope란 우리말로 번역하면 '범위'라는 뜻을 가지고 있따.
즉, 스코프(scope)란 '변수에 접근할 수 있는 범위' 라고 할
수 있다.

자바스크립트에서 스코프는 2가지 타입이 있다.
global(전역)
local(지역)

전역 스코프(Global Scope)는 말그대로 전역에 선언되어 있어
어느 곳에서든지 해당 변수에 접근할 수 있다는 의미이고,
지역 스코프(Local Scope)는 해당 지역에서만 접근할 수 있어
지역을 벗어난 곳에선 접근할 수 없다는 의미이다.

자바스크립트에서 함수를 선언하면 함수를 선언할 때마다
새로운 스코프를 생성하게 된다.
그러므로 함수 몸체에 선언한 변수는 해당 함수 몸체 안에서만
접근할 수 있는다.

이걸 함수 스코프(function-scoped)라고 한다.
함수 스코프가 바로 지역 스코프의 예라고 할 수 있다.


var : 함수 스코프 (function-scoped)
let, const : 블록 스코프(Block-scoped)

블록 스코프는 모든 코드 블록 내에서 선언된 변수는 코드 블록
내에서만 유효하며 외부에서는 접근할 수 없다는 의미이다.
코드 블록 내에서 선언한 변수는 지역 변수 인것이다.

여기서 말하는 코드 블록은 함수, if문, for문, while문,
try/catch등을 의미한다.
```



반면 함수 스코프는 함수 내에서만 그 지역 변수가 되는것

```
const age = 30;
if (age > 19){
  var txt = '성인';
}

console.log(txt); //'성인'
```

if문 안에서 var로 선언한 변수는 if문 밖에서도 사용이 가능

하지만 let과 const는 이렇게 사용할 수 없다.
if문 중괄호 내부에서만 사용 가능하다.
이를 블록 스코프라고 한다.

var

```
function add(num1, num2){
    var result = num1 + num2;
}

add(2,3);

console.log(result);
```

```
▶Uncaught ReferenceError: result
is not defined
```

var도 이렇게 함수 내에서 선언되면 함수 밖에서 사용할 수 없다.
유일하게 벗어날 수 없는 스코프가 함수라고 생각하면 된다.