# var, let, const



## 자바스크립트 var / let / const 차이점 5가지

1. 중복 선언 가능 여부
2. 재할당 가능 여부
3. 변수 스코프 유효범위
4. 변수 호이스팅 방식
5. 전역객체 프로퍼티 여부

---

### 1. 중복 선언 가능 여부

**var** : 중복해서 선언(+초기화)가 가능하다.

이 경우, 마지막에 할당된 값이 변수에 저장된다.

**기존에 선언해둔 변수의 존재를 까먹고, 값을 재할당하게 되는 등의 실수가 발생하기 쉽다.**

**const, let** : 중복 선언 불가능

이미 선언한 변수를 다시 선언할 경우, 에러가 발생한다.

var에 비해서 코드의 안정성을 높여줄 수 있는 방식으로, 다른 언어를 쓰던 사람들에게도 익숙할 것이다.

---

### 2. 재할당 가능 여부

**var, let** : 값의 재할당이 **가능**한 변수다.

변수 선언 및 초기화 이후에 반복해서 다른 값을 재할당 할 수 있다.

**const** : 값의 재할당이 **불가능**한 상수다.

const는 상수를 선언하는 키워드다.

처음에 선언 및 초기화하고 나면 다른 값을 재할당 할 수 없다.

- **var 또는 let과 달리, const 선언에서는 반드시 값을 선언과 동시에 정의해야 한다**
- 재할당이 필요없는 경우, const를 사용해 불필요한 변수의 재사용을 방지하고, 재할당이 필요한 경우 let을 사용하는 것이 좋음.

---

## 3. 변수 스코프 유효범위

스코프란 유효한 참조 범위를 말한다.

예를 들어, 함수 내부에서 선언된 변수는 함수 내부에서만 참조가 가능하다.

자바스크립트는 var로 선언한 변수의 스코프와 let, const로 선언한 변수의 스코프가 다르다.

**var** : 함수 레벨 스코프(function-level scope)

var는 함수 내부에 선언된 변수만 지역변수로 한정하며, 나머지는 모두 전역변수로 간주한다.

자바스크립트에서는 if문, for문, while문, try/catch 문 등의 코드 블럭{ ... } 내부에서 var로 선언된 변수를 전역 변수로 간주한다.

그래서 블럭 외부에서도 어디에서나 참조할 수 있다.

**let, const** : 블록 레벨 스코프(block-level scope)

let, const는 함수 내부는 물론, if문이나 for문 등의 코드 블럭{ ... } 에서 선언된 변수도 지역변수로 취급한다. 당연히 함수 내부에서 선언된 변수도 외부에서 참조할 수 없다.

> 💡 Tip
>
> - 정리하자면, var는 함수 내부에 선언된 변수만 지역 변수로 인정하는 함수 레벨 스코프이다. let, const는 모든 블록 내부에서 선언된 변수까지 지역변수로 인정하는 블록 레벨 스코프이다.

---

## 4. 변수 호이스팅 방식

자바스클립트는 코드를 실행하기 전, 일종의 '코드 평가 과정'을 거치는데,

이 때 '변수 선언문'을 미리 실행 해두기 때문에 뒤에서 선언된 변수도 앞의 코드에서 참조할 수 있게 된다.

이를 변수 호이스팅이라고 한다.

**var**: 변수 호이스팅이 발생한다.

```
console.log(a);  // undefined
var a = 10;
console.log(a);  // 10
```

뒤에서 선언된 변수 a가 앞에서 참조되었음에도 에러를 발생시키지 않는다.

코드 실행 전에 자바스크립트 엔진이 미리

1) 변수를 선언하고,

2) undefined로 초기화해 두었기 때문이다.

코드로 표현하면 이렇게 된다.

```
var a = undefined;
console.log(a);  // undefined
a = 10;
console.log(a);  // 10
```

이게 바로 var로 선언된 변수의 호이스팅이다.

**let, const**: 변수 호이스팅이 발생한다. 하지만 다른 방식으로 작동한다.

```
console.log(a);  // ReferenceError: a is not defined
let a = 10;
```

뒤에서 선언된 변수를 앞에서 참조하려 하니 에러가 발생한다.

마치 호이스팅이 발생하지 않는 것처럼 보인다.

이런 현상이 발생하는 이유는 let, const의 호이스팅 과정이 var와 다르게 진행되기 때문이다.

let/const로 변수를 선언하는 경우,

코드 실행 전에는

1) 변수 선언만 해두며,

2) 초기화는 코드 실행 과정에서 변수 선언문을 만났을 때 수행한다.

```
let a;  // 선언만 하고 정의는 하지 않음. var은 정의 하지않아도 undefined가 들어감.
console.log(a);  // ReferenceError: a is not defined
a = 10;
console.log(a);  // 10
```

그래서 호이스팅이 발생하기는 하지만, 값을 참조할 수 없어서 호이스팅이 발생하지 않는 것처럼 보이는 것이다.

그렇다면, 호이스팅이 발생하는걸 어떻게 확인할 수 있을까?

```
let a = 10;  // 전역변수 a선언
if(true){
  console.log(a); // 10
}
```

이 코드는 전역변수로 선언된 a의 값 10을 if문 블럭에서 참조하여 출력하고 있디.

```
let a = 10;  // 전역변수 a선언
if(true){
  console.log(a);  // ReferenceError: a is not defined
  let a = 20;  // 지역변수 a 선언
}
```

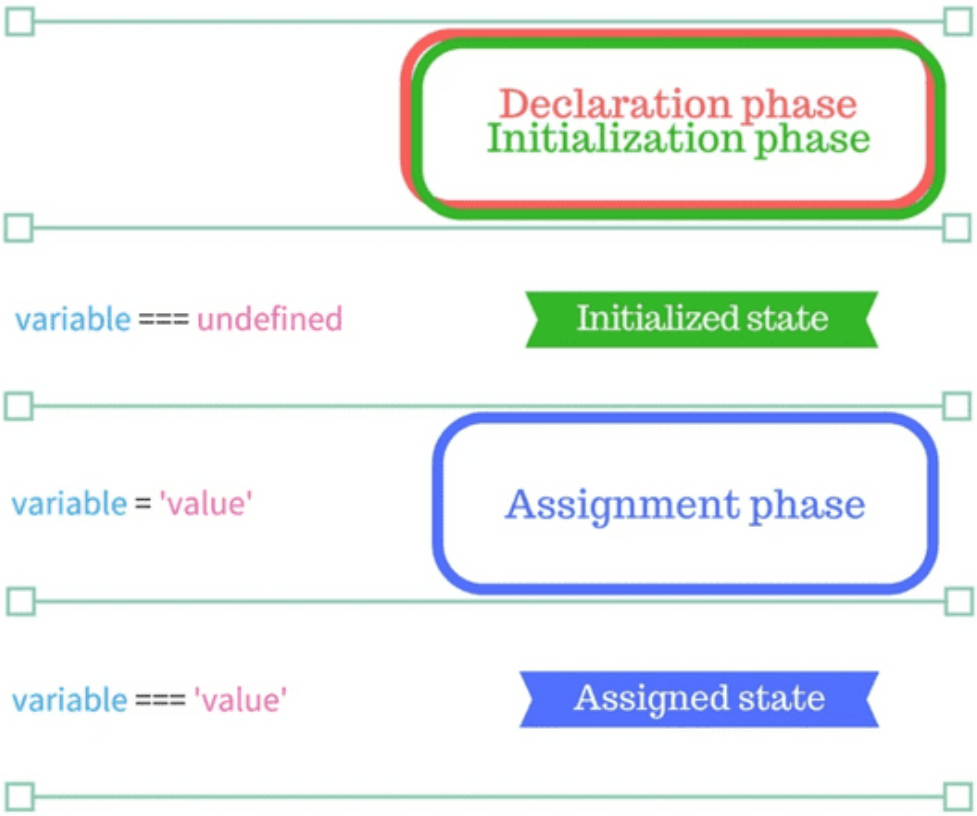
이 코드는 if문 블럭 내부에서 지역변수 a를 다시 선언했다.

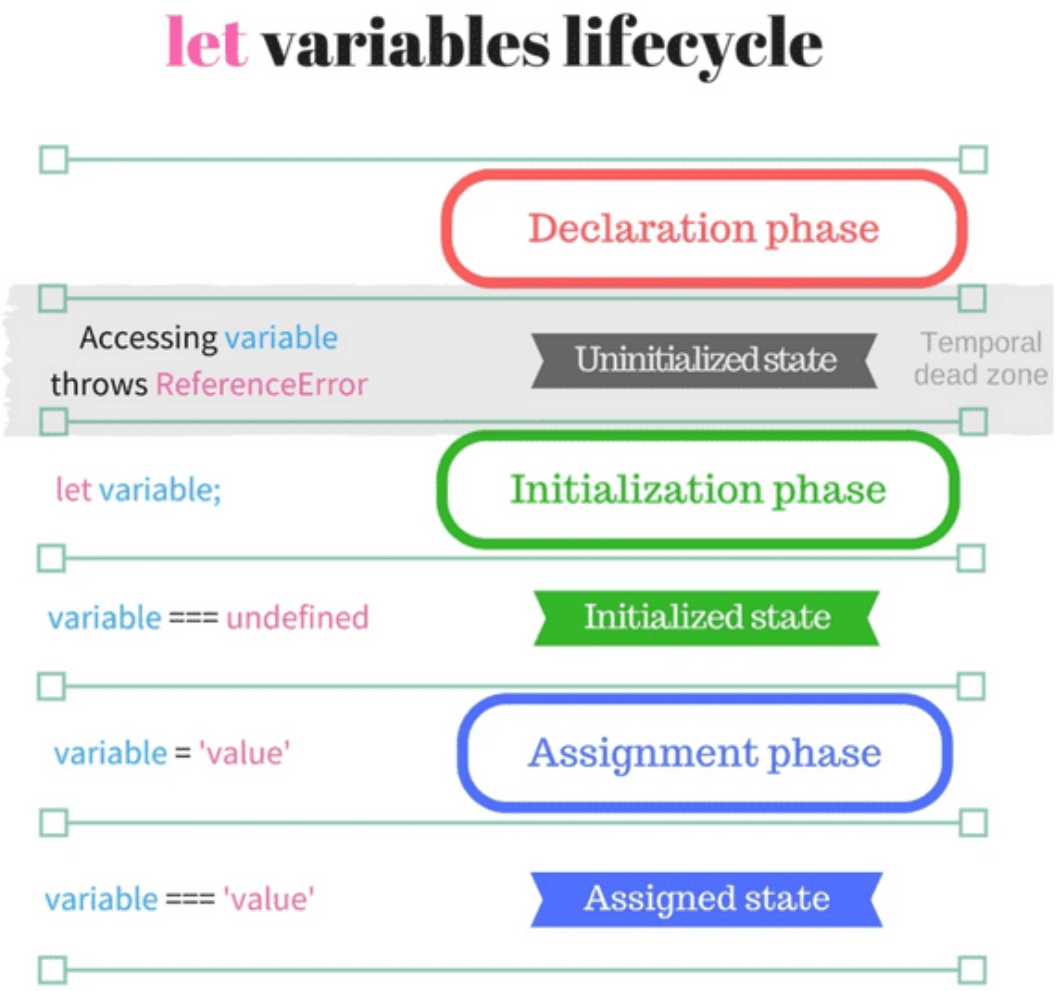
이 경우, 지역변수 a 앞에서 console.log()로 참조시 에러가 발생한다.(전역 변수 a가 있음에도!!)

왜냐하면 지역변수 a가 호이스팅되면서 TDZ 구간이 만들어졌기 때문이다.

- 변수의 선언과 초기화 사이에 일시적으로 변수 값을 참조할 수 없는 구간을 TDZ(Temporal Dead Zone)라고 한다.

# var variables lifecycle





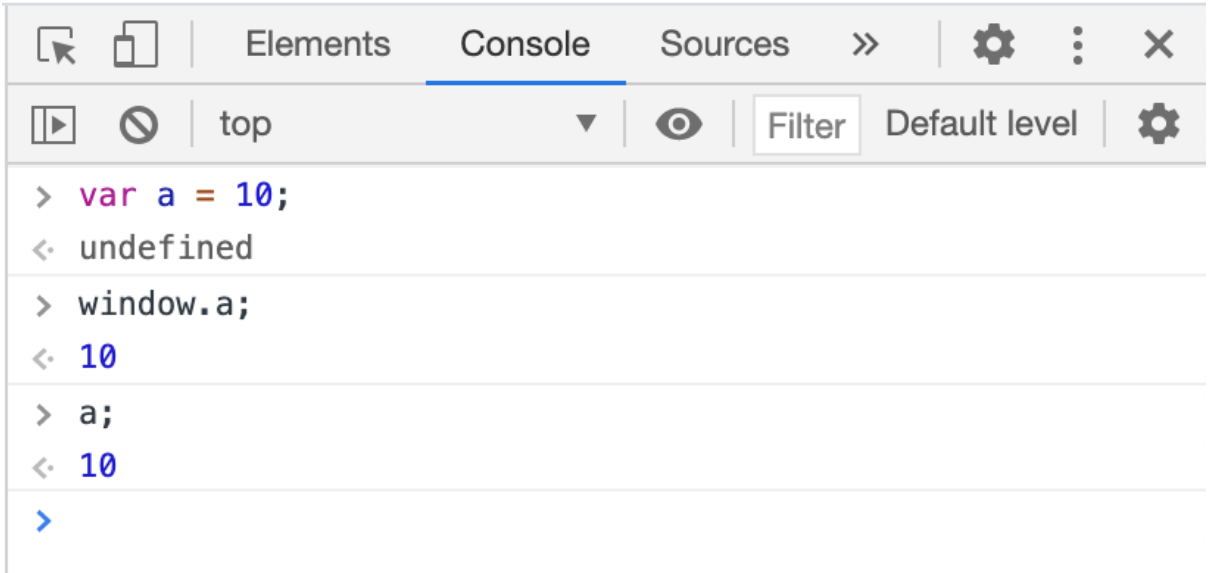

## 5. 전역객체 프로퍼티 여부

| keyword | const | let | var |
| --- | --- | --- | --- |
| **global scope** | NO | NO | YES |
| **function scope** | YES | YES | YES |
| **block scope** | YES | YES | NO |
| **can be reassigned** | NO | YES | YES |

**var** : var로 선언된 변수는 전역객체(브라우저 환경의 경우 window)의 프로퍼티다.

```
var a = 10;
console.log(window.a);  // 10
console.log(a);  // 10
```

브라우저 환경(크롬 콘솔 등)에서 위 코드 실행 시, var로 선언한 변수 a는 브라우저 전역객체인 window의 프로퍼티로 할당된다.

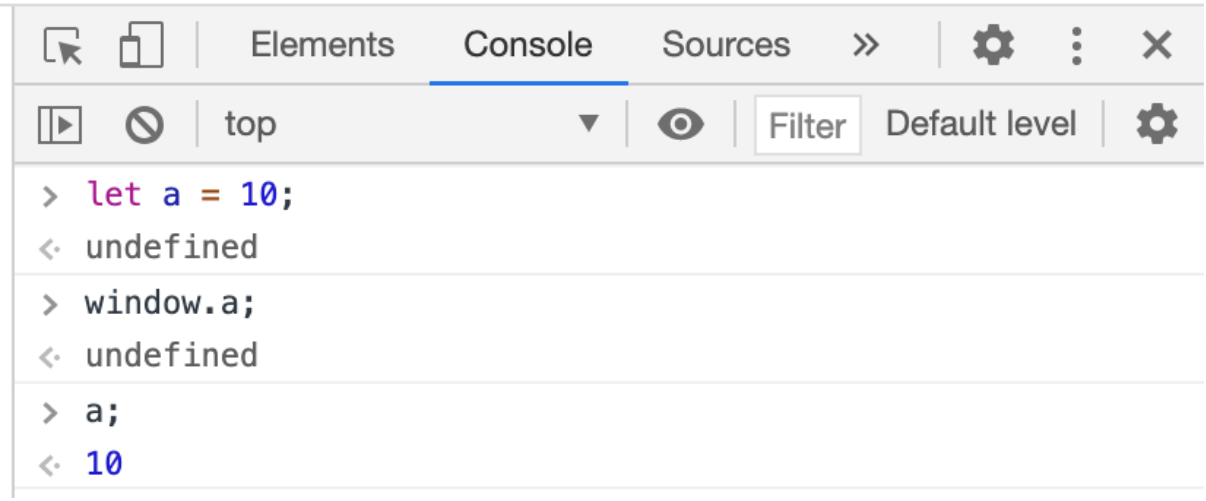



**let, const** : let/const 로 선언된 변수는 전역객체 프로퍼티가 아니다.

```
let a = 10;
console.log(window.a);  // undefined
console.log(a);  // 10
```

브라우저 환경(크롬 콘솔 등)에서 위 코드 실행 시,

let이나 const로 선언한 변수 a는 브라우저 전역객체인 window의 프로퍼티로 할당되지 않았음을 알 수 있다.




---

💡 **1. const와 let을 이용해서 변수를 선언하라.**

**2. 값을 재 할당하는 경우가 아니라면, const를 default로 사용하라.**

**3. var을 남용하지 말라**