2023.03.05

A ✓

1. const, let, var
    a. const

    상수(constants)의 약자로 변하지 않는 값을 선언할 때 사용하는 키워드입니다. 선언과 동시에 초기화해야 하며, 블록 내에서 유효합니다.

    b. let

    블록 내에서 지역변수를 선언할 때 사용하는 키워드로, 선언과 동시에 초기화할 수 있으며 할당된 값을 변경할 수 있습니다.

    > TMI로, `let`이라는 키워드의 유래는 수학에서 자주 사용하는 구문인 'x를 임의의 실수라 가정하자(let x be arbitrary real number)'에서 나온 구문이라고 합니다.

    c. var

    함수 내에서 변수를 선언할 때 사용하는 키워드로, 선언과 동시에 초기화할 수 있고, 값을 변경할 수도 있으며 변수 자체를 다시 선언할 수도 있습니다.

    - `let`의 스코프는 자신을 선언한 블록이지만 `var`의 스코프는 자신의 선언을 포함하는 블록 전체(함수)라는 점에서 차이가 있습니다.
        - var

        ```
        console.log(name); // ReferenceError: name is not defined
        function test() {
          var name = "Kim";
          console.log(name); // Kim
          if (true) {
            var name = "Lee";
            console.log(name); // Lee
          }
          console.log(name); // Lee
        }
        test();
        ```

        - let

```
function test() {
  let name = "Kim";
  console.log(name); // Kim
  if (true) {
    let name = "Lee";
    console.log(name); // Lee
  }
  console.log(name); // Kim
}
test();
```

- 최상위에서 선언하면 전역 객체인 window에 property로서 추가됩니다.

```
var name = "Kim";
console.log(name); // Kim
console.log(window.name); // Kim
```

2. Hoisting

- Hoist는 직역하면 '끌어올리다' 라는 뜻입니다. 자바스크립트에서는 인터프리터가 **변수와 함수의 메모리 공간을 미리 할당해 두는 것**을 의미합니다. 변수나 함수를 블록의 어떤 부분에서 선언하든 미리 공간이 할당되기 때문에, 마치 변수나 함수의 선언이 블록의 최상단으로 '끌어 올려지는' 것과 같다고 볼 수 있어 Hoisting이라고 합니다.
- 이렇게 변수나 함수가 최상단으로 끌어 올려지기 때문에, 변수나 함수를 호출하는 부분이 선언하는 부분보다 먼저 나와도 정상적으로 실행됩니다. 단, **선언만 Hoisting의 대상**입니다.
  - `var` 키워드로 선언된 변수는 Hoisting될 때 `undefined`로 초기화됩니다. 따라서 실제 코드에서 선언 전에 접근이 가능합니다.

```
console.log(name); // undefined
var name = "Kim";
console.log(name); // Kim
```

  - `let`, `const` 키워드로 선언된 변수는 선언이 Hoisting되지만 **초기화되지 않습니다.** 따라서 선언 전에 접근하면 Reference Error가 발생합니다.

```
console.log(name); // ReferenceError: name is not defined
const name = "Kim";
console.log(name); // Kim
```

- 변수 선언 키워드를 사용하지 않으면 초기화만 이루어지므로 Hoisting되지 않습니다. 따라서 `let`, `const` 와 마찬가지로 초기화 전에 접근하면 Reference Error가 발생합니다.

```
console.log(name); // ReferenceError: name is not defined
name = "Kim";
console.log(name); // Kim
```

- 함수의 선언 역시 Hoisting의 대상이므로 함수를 정의하기 전에 호출해도 정상적으로 실행됩니다.

```
sayHou(); // holla
function sayHolla() { console.log('holla') }
```

3. TDZ(Temporal Dead Zone)

- 위 1,2번 내용에서 언급한 것처럼, `let` 키워드로 변수를 선언하는 경우, **스코프의 최상단**으로 선언이 hoisting되지만 초기화 시점은 **해당 변수가 선언된 라인**입니다. 이 간격을 **시간 상의 사각지대(Temporal Dead Zone)**이라고 합니다.
- '시간 상'의 사각지대인 이유는, 코드의 작성 순서가 아닌 코드의 실행 순서에 따라 TDZ가 형성되기 때문입니다. TDZ 안에서 해당 변수에 접근하려고 하면 Reference Error가 발생합니다.

```
function studyTDZ() {
  /* letVar의 Temporal Dead Zone Start */

  const func = () => {
    console.log(letVar);
  }

  func(); // Reference Error: Cannot access 'letVar' before initialization

  let letVar = "Hello"; /* letVar의 Temporal Dead Zone End */

  func(); // Hello
}
```