5. 자바스크립트 핵심개념

var,let,const 차이

JavaScript에서 `var`, `let`, `const`의 차이점과 스코프, 호이스팅에 대한 개념 정리

`var`, `let`, `const`의 차이점

- `var`: 재선언과 재할당이 가능하며, 함수 스코프를 가짐. 호이스팅 시 `undefined`로 초기화됨.
- `let`: 재선언이 불가능하고, 재할당은 가능하며 블록 스코프를 가짐. 호이스팅 시 선언 전에 접근하면 `ReferenceError` 발생.
- `const`: 재선언과 재할당이 불가능하며 블록 스코프를 가짐. 선언 시 반드시 초기화되어야 하며, 호이스팅 시 `ReferenceError` 발생.

스코프(Scope)란?

- 변수나 함수가 어디에서 접근할 수 있는지를 결정하는 규칙으로, 세 가지 종류가 있음:
  1. **전역 스코프(Global Scope)**: 함수나 블록 외부에서 선언된 변수. 프로그램 전체에서 접근 가능.
  2. **함수 스코프(Function Scope)**: 함수 내부에서 선언된 변수로, 함수 내부에서만 접근 가능.
  3. **블록 스코프(Block Scope)**: `{}`로 감싸진 블록(`if`, `while`, `for` 등) 내부에서 선언된 변수. 블록 내부에서만 접근 가능.

스코프의 작동 예시 코드

```
var a = 1; // 전역 스코프
```

```
function hello() {
  var b = 2; // 함수 스코프
  if (true) {
    let c = 3; // 블록 스코프
    console.log(a); // 1
    console.log(b); // 2
    console.log(c); // 3
  }
  console.log(a); // 1
  console.log(b); // 2
  // console.log(c); // ReferenceError: c is not defined
}

hello();
```

- 전역 스코프인 `a`는 어디서든 접근 가능.
- 함수 스코프인 `b`는 함수 내부에서만 접근 가능.
- 블록 스코프인 `c`는 해당 블록 내에서만 접근 가능하며, 블록을 벗어나면 `ReferenceError`가 발생.

호이스팅 함수

1. 함수 선언식 (Function Declaration)

함수 선언식은 `function` 키워드를 사용해 함수를 선언하는 방식입니다. 함수 이름을 명시적으로 지정하고 작성되며, **호이스팅 시 함수 전체가 스코프의 최상단으로 이동**됩니다. 이 때문에 함수 선언 이전에 호출해도 동작합니다.

예시:

```
function greet() {
  console.log("Hello, world!");
}

greet(); // "Hello, world!" 출력
```

호이스팅의 특징:

- 함수 선언식은 선언 이전에 호출이 가능합니다.

```
greet(); // "Hello, world!" 출력

function greet() {
  console.log("Hello, world!");
}
```

2. 함수 표현식 (Function Expression)

함수 표현식은 변수에 함수를 할당하는 방식입니다. 이때 `const`나 `let`, `var`를 사용해 변수에 익명 함수(또는 명명된 함수)를 할당합니다.

예시 (`const`를 사용한 경우):

```
const greet = function() {
  console.log("Hello, world!");
};

greet(); // "Hello, world!" 출력
```

호이스팅의 특징:

- 변수 `greet`는 호이스팅되지만 초기화되지 않기 때문에 함수 표현식을 선언하기 전에 호출하려고 하면 `ReferenceError`가 발생합니다.

```
greet(); // ReferenceError: Cannot access 'greet' before in
itialization

const greet = function() {
  console.log("Hello, world!");
};
```

차이점 요약

특성	함수 선언식	함수 표현식 (`const`)
호이스팅	함수 전체가 호이스팅됨	변수만 호이스팅, 초기화되지 않음
선언 이전 호출	가능	불가능 (선언 전에 `ReferenceError`)

변경 가능 여부	함수 이름 고정	`const`일 경우 변경 불가

결론

- **함수 선언식**은 함수 전체가 호이스팅되기 때문에 선언 이전에 호출이 가능합니다.
- **함수 표현식**에서 `const`로 선언된 경우, 변수 자체는 호이스팅되지만 초기화되지 않으므로 선언 이전에 호출하려고 하면 `ReferenceError`가 발생합니다.

실제 개발에서는 함수 표현식에 `const`를 사용하는 것이 일반적이며, 이는 재할당을 방지하고 코드의 안정성을 높여줍니다.

**

```
console.log(a);
var a = 20;
```

이렇게 작성하면 undefined 뜨는이유?

`console.log(a);`를 실행했을 때 `undefined`가 출력되는 이유는 **호이스팅(hoisting)** 때문입니다. 자바스크립트의 호이스팅은 변수 선언을 해당 스코프의 최상단으로 끌어올리는 것처럼 동작하지만, 초기화는 끌어올리지 않기 때문입니다.

호이스팅의 동작 방식

1. **선언 단계**: 자바스크립트 엔진이 코드를 실행하기 전에 변수 선언을 스코프의 최상단으로 끌어올립니다.
2. **초기화 단계**: `var`로 선언된 변수는 호이스팅 시 자동으로 `undefined`로 초기화됩니다.
3. **할당 단계**: 코드가 실행될 때 비로소 변수에 할당된 값이 적용됩니다.

예시 코드의 실제 동작 과정

```
console.log(a); // 출력: undefined
var a = 20;
```

위 코드는 자바스크립트 엔진에 의해 다음과 같이 해석됩니다:

```
var a;          // 선언이 호이스팅되어 최상단으로 이동, 초기값은 und
efined
console.log(a); // 현재 a는 undefined
a = 20;         // 여기서 실제로 값이 할당됨
```

핵심 요점

- `var`로 선언된 변수는 **선언 단계**에서 스코프의 최상단으로 호이스팅되며 `undefined`로 초기화됩니다.
- 따라서 `console.log(a);`를 실행할 때 `a`는 이미 선언되어 있지만, 아직 값이 할당되지 않았기 때문에 `undefined`가 출력됩니다.

참고: `let`과 `const`의 경우

- `let`과 `const`도 호이스팅되지만, 선언과 초기화가 동시에 이루어지지 않고, 초기화는 실제 코드에서 해당 선언에 도달했을 때 수행됩니다. 이 때문에 `let`과 `const`로 선언된 변수를 선언 전에 사용하려고 하면 `ReferenceError`가 발생합니다.