코어 JS - 1단원 답변

질문

1. '=='와 '==='의 차이점은?

- 공통점: 비교 연산자
- 차이점: 전자는 타입 변환을 한 후 비교함 후자는 타입 변환을 하지 않고 반환함

엄격한 일치 연산자인 '==='을 사용하는 것이

코드의 안정성과 명확성을 높이는 데 유리함

```
// 전자
1 == '1';   // true (숫자 1과 문자열 '1'은 타입 변환 후 같음)
true == 1;  // true (true는 1로 변환됨)
null == undefined; // true (null과 undefined는 동등하다고 간주됨)

//후자
1 === '1';  // false (숫자와 문자열은 타입이 다르므로 일치하지 않음)
true === 1; // false (불리언과 숫자는 타입이 다르므로 일치하지 않음)
null === undefined; // false (타입이 다름)
```

2. undefined와 null의 차이점은?

- 둘 다 '값이 없음'을 나타냄

	undefined	null
의미	변수나 값이 정의되지 않은 상태	명시적으로 "값이 없음"을 나타내는 상태
타입	`undefined` 타입	`객체` 타입
사용 용도	변수 선언 후 값이 할당 X, 함수 호출 시 인자가 전달 X, 객체의 프로퍼티가 존재 X 등 - 주로 시스템이 자동으로 할당하는 값으로 사용함	프로그래머가 명시적으로 "값이 없음"을 표현하기 위해 사용

```
// undefined
1. yes 선언 no 할당
let x;
console.log(x); // undefined

2. 함수 호출 시 인자가 전달되지 않은 경우
function myFunction(a) {
  console.log(a); // undefined
}
```

```
myFunction();

3. 객체 프로퍼티가 X
let obj = {};
console.log(obj.property); // undefined


// null의 타입 확인은 typeof 대신 다른 방식 필요
console.log(typeof null); // "object"
-> null 값을 정확하게 확인하려면 직접 값과 비교
```

3. var, let, const의 차이점은?

- 변수를 선언하는 방법
- 범위(scope), 재할당 가능성, 그리고 호이스팅(hoisting) 등에서 차이 O

var

- Scope
  - var로 선언된 변수는 함수 스코프를 가짐
  - 즉, 함수 내에서만 유효하며, 함수 외부에서는 접근 X

```
function example() {
  var x = 10;
  if (true) {
    var x = 20; // 같은 함수 내에서 x는 같은 변수로 간주됨
    console.log(x); // 20
  }
  console.log(x); // 20 (블록 스코프를 무시)
}
example();
```

- 호이스팅
  - 선언이 호이스팅되지만, 초기화는 호이스팅되지 않음
  - 즉, 선언이 함수의 최상위로 끌어올려짐 / 선언 단계 & 초기화 단계 한번에 진행

```
console.log(x); // undefined (변수 선언은 호이스팅됨)
var x = 10;
```

- 재선언 및 재할당
  - 같은 스코트 내에서 재선언 및 재할당 가능

```
var x = 10;
var x = 20; // 재선언 허용
x = 30; // 재할당 허용
```

let

- scope
  - 블록 스코프를 가짐. 즉, 블록({})내에서만 유효함

```
function example() {
  let x = 10;
  if (true) {
    let x = 20; // 블록 내에서 별도의 변수로 간주됨
    console.log(x); // 20
  }
  console.log(x); // 10
}
example();
```

- 호이스팅
  - 선언은 호이스팅되지만 초기화 전에는 사용이 불가능

```
console.log(x); // ReferenceError: Cannot access 'x' before initializat
let x = 10;
```

- 재선언 및 재할당
  - 같은 스코프 내에서 재선언 불가능, 재할당은 가능

```
let x = 10;
let x = 20; // SyntaxError: Identifier(식별자) 'x' has already been declare
x = 30; // 재할당 허용
```

const

- scope
  - 블록 스코프를 가짐
- 호이스팅
  - 선언은 호이스팅되지만, 초기화 전에는 사용이 불가능

```
console.log(x); // ReferenceError: Cannot access 'x' before initializatio
const x = 10;
```

- 재선언 및 재할당
  - 같은 스코프 내에서 재선언 불가능, 초기화 이후 재할당 불가능 (불변을 의미 x)
  - but, `const`로 선언된 객체나 배열의 내부 속성이나 요소 변경 가능
  - 선언 단계 & 초기화 단계 분리 돼서 진행 → 선언 / TDZ / 초기화 / 할당

```
const x = 10;
const x = 20; // SyntaxError: Identifier 'x' has already been declared
x = 30; // TypeError: Assignment to constant variable.

const person = { name: "Alice" };
person.name = "Bob"; // 객체의 속성 변경 가능
console.log(person.name); // "Bob"
```

정리

```
var
함수 스코프
선언이 호이스팅됨
재선언 및 재할당 가능

let
블록 스코프
선언이 호이스팅되지만 초기화 전 사용 불가 (일시적 사각지대) 선언/사각/초기화/할당
재선언 불가, 재할당 가능

const
블록 스코프
선언이 호이스팅되지만 초기화 전 사용 불가 (일시적 사각지대)
재선언 불가, 재할당 불가 (단, 객체나 배열의 내부 속성이나 요소는 변경 가능)

var 사용은 X
재할당이 필요한 경우에 한정해 let 사용 이 때 변수의 스코프는 최대한 좁게
재할당이 필요 없는 상수, 객체에는 const 사용을 권장 / 이후 let으로 바꿔도 늦지 않음
```

4. 불변값과 상수의 차이점은?

불변값

- 한 번 생성된 후에는 변경할 수 없는 **값**을 의미
- 변수/상수를 구분 짓는 변경 가능성의 대상은 **변수 영역**의 메모리
- 불변성 여부를 구분할 때의 변경 가능성의 대상은 **데이터 영역** 메모리

- 기본형 데이터(primitive type)는 불변값

```
let str = "Hello";
str[0] = "J"; // 문자열은 불변이므로 변경되지 않음
console.log(str); // "Hello"

let num = 42;
num = num + 1; // 새로운 값을 할당하는 것은 가능하지만, 원래의 숫자 값은 불변
console.log(num); // 43
```

상수

- 값이 한 번 할당되면 재할당할 수 없는 **변수**를 의미
- **const** 키워드를 사용하여 상수를 선언함
- 변수에 할당된 객체의 속성은 변경할 수 있음

```
const PI = 3.14159;
PI = 3.14; // 오류: const 변수는 재할당할 수 없음

const person = { name: "Alice" };
person.name = "Bob"; // 객체의 속성은 변경 가능
console.log(person.name); // "Bob"
```

- **불변값 (Immutable Value)**:
  - 값 자체가 변경되지 않음.
  - 원시 타입(숫자, 문자열, 불리언 등)은 불변값
  - 객체의 상태가 생성된 후 변경될 수 없는 것 → 데이터 무결성을 보장하는 데 유용
  - 한 번 생성된 후 내부 값을 변경할 수 없음.
- **상수 (Constant)**:
  - 변수에 할당된 값이 변경되지 않음.
  - `const` 키워드를 사용하여 선언.
  - 객체나 배열을 상수로 선언해도 내부 속성이나 요소는 변경 가능.
  - 재할당이 불가능하지만, 할당된 객체나 배열의 내부 상태는 변경 가능.

5. 작동 원리

```
// 결과 값과 메모리에서의 검색 과정 설명

var obj = {
    x: 3,
    y: [ 3, 4, 5 ]
};
console.log(obj.y[1])
```

변수 영역

주소	1001	1002	1003	...
데이터		이름: obj 값: @5001		

데이터 영역

주소	5001	5002	5003	5004	5005
데이터	@7103 ~ ?	3	@8104 ~?	4	5

객체 @5001의 변수 영역

주소	7103	7104	7105	...
데이터	이름: x 값: @5002	이름: y 값: @5003		

객체 @5003의 변수 영역

주소	8104	8105	8106	...
데이터	이름: 0 값: @5002	이름: 1 값: @5004	이름: 2 값: @5005	

1. 컴퓨터는 우선 변수 영역의 빈 공간(@1002)을 확보하고, 그 주소의 이름을 obj로 지정합니

2. 임의의 데이터 저장 공간(@5001)에 데이터를 저장하려는데,
   이 데이터는 여러 개의 변수와 값들을 모아놓은 그룹(객체)입니다.
   이 그룹의 각 변수(프로퍼티)들을 저장하기 위해 별도의 변수 영역을 마련하고(@7103 ~?),
   그 영역의 주소를 @5001에 저장합니다.

3. @7103에 이름 x를, @7104에 이를 y를 지정합니다.

4. 데이터 영역에서 숫자 3을 검색합니다. 없으므로 임의로 @5002에 저장하고,
   이 주소를 @7103에 저장합니다.

5. @7104에 저장할 값은 배열로서 역시 데이터 그룹입니다. 이 그룹 내부의 프로퍼티들을 저장
   위해 별도의 변수 영역을 마련하고(@8104 ~?), 그 영역의 주소 정보(@8104 ~?)를 @500
   다음, @5003을 @7104에 저장합니다.

6. 배열의 요소가 총 3개이므로 3개의 변수 공간을 확보하고 각각 인덱스를 부여합니다.(0, 1,

7. 데이터 영역에서 숫자 3을 검색해서(@5002) 그 주소를 @8104에 저장합니다.

8. 데이터 영역에 숫자 4가 없으므로 @5004에 저장하고, 이 주소를 @8105에 저장합니다.

9. 데이터 영역에 숫자 5가 없으므로 @5005에 저장하고, 이 주소를 @8106에 저장합니다.