For in 반복문

-객체의 모든 열거(enumerable) 가능한 속성(property)에 대한 반복

var obj = { a: 1, b: 2, c: 3 };

for (var prop in obj) { console.log(prop, obj[prop])}; // a 1, b 2, c 3

\*for in은 prototype으로 상위에 추가된(Array 객체) 객체들까지도 나타냄. 이러한 단점으로 인해 for of 등장

-for of 반복문(주로 배열에 사용)

-for of 구문을 사용하기 위해선 컬렉션 객체가 [Symbol.iterator] 속성을 가지고 있어야만 한다.

-> String, Map, Array, Set, Dom컬렉션(HTMLCollection, NodeList) 등

var iterable = [10, 20, 30];

for (var value of iterable) { console.log(value);} // 10, 20, 30

for in 단점 예)

Object.prototype.objCustom = function () {}; // object 프로퍼티에 객체 생성

Array.prototype.arrCustom = function () {}; // array 프로퍼티에 객체 생성

var iterable = [3, 5, 7];

iterable.foo = "hello";

for (var key in iterable) { console.log(key);} // 0, 1, 2, "foo", "arrCustom", "objCustom"

for (var value of iterable) { console.log(value);} // 3, 5, 7

호이스팅(hoisting) : 코드에 선언된 변수 및 함수를 코드 상단으로 끌어올리는 것

-함수 내에서 선언한 함수 범위(function scope)의 변수는 해당 함수의 최상위로

-함수 밖에서 선언한 전역 범위(global scope)의 전역 변수는 스크립트 단위의 최상위로

1. 함수 호이스팅

함수 선언식의 경우 호이스팅이 발생한다.

noDefine();

function noDefine(){

    console.log("not defined :" + name); // not defined :undefined

    var name="javascript";

    console.log("defined :" + name); // defined :ojava

}

함수 표현식의 경우 호이스팅이 발생하지 않는다.

noDefine();

var example = function noDefine(){

    console.log("not defined :" + name); // not defined :undefined

    var name="javascript";

    console.log("defined :" + name); // defined :ojava

}

결과값

noDefine();

^

ReferenceError: noDefine is not defined

1. 변수 호이스팅.

console.log(a); // undefined

var a="A";

console.log(a); // A

let, const의 경우 변수 선언과 초기화가 분리되어 실행(변수가 사용되기 전까지 초기화 X)

console.log(a);

let a = "A";

console.log(a);

결과값

console.log(a);

            ^

ReferenceError: Cannot access 'a' before initialization