자바스크립트 변수 (Variable)

변수 선언

JavaScript에서 변수를 선언하는 데 사용되는 주요 키워드는 var, let, const 세 가지

변수 종류	설명
`var`	가장 오래된 변수 선언 키워드, 함수 범위 또는 전역 범위, 재선언 및 재할당 가능 (비추천)
`let`	ES6(ES2015)에서 도입, 블록 범위, 재할당 가능하지만 재선언 불가능
`const`	ES6에서 도입, 블록 범위, 상수(constant), 재할당 및 재선언 불가능, 초기값 필요

변수 명명규칙

```
var abc = 'abc';
let creamBread = 'a';
const chocoBread = 'b';
```

구분	설명
변수 이름	문자, 숫자, $, _ 사용 가능
규칙	첫 글자는 숫자가 될 수 없음, 예약어 사용 불가 (예: `implements, interface, let, package, private, protected, public, static, yield` 등)
변수 표기법	`let camelCase;` 카멜케이스 `let snake_case;` 스네이크케이스 `let PascalCase;` 파스칼케이스 `let $case1;` $ 달러 `let _case2;` _ 밑줄

연산자 (Operator)

연산자 종류	연산자	설명
할당 연산자	=	값 할당
비교 연산자	==, ===, !=, !==, >, <	값 비교
산술 연산자	%, ++, −, +	수학적 계산

연산자 종류	연산자	설명
비트 연산자	`&`, `|`, `^`	비트 단위 연산
논리 연산자	`&&`, `||`, `!`	논리적 조건 판단
문자열 연산자	`“”`, `‘’`	문자열 결합
조건 (삼항) 연산자	`조건 ? 값1 : 값2`	조건에 따른 값 선택
쉼표 연산자	`,`	여러 표현식 평가
단항 연산자	`delete`, `typeof`, `void`	단일 값에 대한 연산
관계 연산자	`in`, `instanceof`	객체 속성 및 인스턴스 검사
옵셔널체이닝	`?.`	연결된 객체 체인 검사
널 병합 연산자	`??`	null 또는 undefined 대체

- 비트 연산자
  비트 연산자는 숫자를 32비트 이진수로 처리하고, 이진수의 각 비트(bit) 단위로 연산을 수행하는 JavaScript 연산자입니다. 비트 연산자는 주로 성능 최적화, 암호화 연산 등에서 사용됩니다.
  - 비트 AND (&): 두 이진수의 각 비트를 비교하여 둘 다 1이면 결과는 1이고, 그렇지 않으면 0입니다.
  - 비트 OR (|): 두 이진수의 각 비트를 비교하여 하나라도 1이면 결과는 1이고, 둘 다 0이면 결과는 0입니다.
  - 비트 XOR (^): 두 이진수의 각 비트를 비교하여 같으면 결과는 0, 다르면 결과는 1입니다.
  - 비트 NOT (~): 단일 이진수의 각 비트를 반전시킵니다. 즉, 1은 0으로, 0은 1로 바뀝니다.

```
// 비트 연산을 위한 변수 정의
let a = 0b1100; // 이진수 1100 (십진수로 12)
let b = 0b0110; // 이진수 0110 (십진수로 6)

// AND 연산
let AND = a & b; // 같은 자리수가 모두 1인 경우
console.log('AND:', AND); // AND: 4 (0b0100)

// OR 연산
let OR = a | b; // 하나라도 1인 경우
console.log('OR:', OR); // OR: 14 (0b1110)
```

▶ 비트연산자 설명

정수를 이진수 단위로 연산합니다.
예시: a & b
a = 0b1100는 이진수로 1100 (십진수로 12) 입니다.
b = 0b0110는 이진수로 0110 (십진수로 6) 입니다.
a & b = 0b0100는 이진수로 0100 (십진수로 4) 입니다.
비트 OR (|) 연산자: 두 정수의 이진 표현에서 적어도 하나의 비트가 1이면 결과의 해당 비트를 1로 설정합니다.
예시: a | b
a = 0b1100는 이진수로 1100 (십진수로 12) 입니다.

> b = 0b0110는 이진수로 0110 (십진수로 6) 입니다.
> a | b = 0b1110는 이진수로 1110 (십진수로 14) 입니다.

값 할당, 변수 호출

프로그래밍 언어는 아래 방식으로 데이터를 저장하고
저장된 데이터를 다시 사용할 수 있는 메커니즘을 제공합니다.

구분	설명
값 할당	할당 연산자인 =를 사용해 변수에 데이터를 저장합니다. `let number = 5;`는 '5'를 'number' 변수에 할당합니다. 할당은 우측 값에서 좌측 변수로 진행됩니다.
호출	변수 이름으로 저장된 값을 사용합니다. `console.log(number);`는 'number'에 저장된 값을 콘솔에 출력합니다. 변수의 이름을 사용하여 변수와 연결된 메모리 공간에 접근 후 저장된 데이터를 가져옵니다.

```
// 변수를 선언하고 연산자를 사용하여 값 할당
var message1 = "coffee";   // 재할당 가능
let message2 = "cake";     // 재할당 가능
const message3 = "bread";  // 재할당 불가

// 값을 다시 재할당 했을 때 값이 변하는 변수(var, let)
message1 = "iceCoffee"    // var
message2 = "cheeseCake"   // let

// 주의) 기존에 변수가 선언되어 있지만 다시 선언이 가능하기 때문에 프로그램이 원하는대로 동작하지
않을 가능성이 높음
var message1 = "hotCoffee"       // 같은 변수명으로 재선언 가능
// let message2 = "chocoCake"    // 에러(같은 변수명으로 재선언 불가)

// 변수를 호출해서 변수에 담긴 값 확인
console.log(`var : ${message1}, let : ${message2}, const : ${message3}`);
console.log(message1)
console.log('변수 입니다 : ', message1)
console.log(`변수 입니다 : ${message1}`)
```

변수에 다른 변수 값 할당

```
let ice;                    // 변수만 선언
let coffee = "아메리카노";    // 변수 선언 후 값 할당

ice = coffee;               // coffee 변수의 값을 복사해서 ice 변수에 할당
```

```
console.log(`coffee : ${coffee}, ice : ${ice}`);
```

저수준 프로그래밍 언어와 고수준 프로그래밍 언어

유형	설명	종류
저수준 프로그래밍 언어 (Low-level programming language)	일반적으로 기계어와 어셈블리어에 해당하며, 실행 속도가 매우 빠르지만 러닝커브가 높고 유지보수가 힘든 것이 단점입니다.	기계어, 어셈블리 언어, C
고수준 프로그래밍 언어 (High-level programming language)	사람이 쉽게 이해할 수 있도록 설계되어 가독성이 높고 다루기 간단한 장점이 있으며, 컴파일러나 인터프리터에 의해 저수준 프로그래밍 언어로 번역되어 실행됩니다.	Python, Java, JavaScript, C#, Ruby

---

자바스크립트 메모리 관리 (Memory management, Garbage collection)

메모리 생존주기

메모리 생존주기는 대부분의 프로그래밍 언어에서 사용합니다.

단계	설명	비고
1. 할당	필요할 때 메모리를 할당합니다.	고수준 언어에서는 대부분 암묵적으로 할당됩니다.
2. 사용	할당된 메모리를 사용합니다 (읽기, 쓰기).	모든 프로그래밍 언어에서 명시적으로 사용됩니다.
3. 해제	더 이상 필요하지 않을 때 메모리를 해제합니다.	고수준 언어에서는 대부분 암묵적으로 해제됩니다.

자바스크립트 메모리

자바스크립트 메모리 관리 는 가비지 컬렉션에 의해 관리됩니다. 가비지 컬렉션은 JavaScript는 객체가 생성되었을 때 자동으로 메모리를 할당하고 더 이상 필요하지 않을 때 자동으로 해제합니다.

- 자바스크립트 메모리 할당 예제

```
// 기본형 데이터(정수, 문자열) 할당
const count = 123;  // 정수 할당
```

```
const coffee = "커피";  // 문자열 할당

// 객체 할당 - 객체와 그 속성들을 위한 메모리 할당
const cafe = { starbucks: 1, ediya: null };

// 배열 할당 - 배열과 그 요소들을 위한 메모리 할당
const tea = [1, null, "밀크티"];

// 함수 할당 - 함수는 호출 가능한 객체로, 메모리에 할당됨
function bread(a) {
  return a + 2;
}

// 익명 함수 할당 - 이벤트 리스너에서 사용될 익명 함수를 메모리에 할당
someElement.addEventListener(
  "click",
  () => { someElement.style.backgroundColor = "blue"; },
  false
);

// Date 객체 할당 - Date 객체 인스턴스를 위한 메모리 할당
const studyDay = new Date();

// DOM 엘리먼트 할당 - 새로운 div 엘리먼트를 생성하고 메모리에 할당
const selectDiv = document.createElement("div");
```