JS에서 변수를 할당했을 때
컴퓨터 내부에서 정확히 어떻게 동작하는가?

1) JS에서의 원시타입은 선언되면 메모리의 어디에 선언될까요?

2) JS에서 참조타입은 선언되면 메모리의 어디에 선언될까요

3) 메모리 관점에서 얕은 복사와, 깊은 복사에 대해서 설명해주세요

4) JS의 가비지 컬렉팅(Garbage Collecting, GC)에 대해서 간단하게 설명해주세요

- GC는 왜 필요한가?
- GC는 메모리의 어느 부분에 쌓인 가비지를 처리하는 것인가?
- GC가 과연 모든 문제를 해결해주었을까 ?

원시 타입 primitive type

- (string, number, bigint, boolean, undefined, symbol)
- 정수, 실수, 문자, 논리 리터럴등의 **실제 데이터 값을 저장하는 타입**

참조 타입 reference type

- 원시타입이 아닌 모두

- object, array, function

- 참조 타입은 메모리에 직접 접근이 아닌 메모리의 **위치(주소)**에 대한 간접적인 참조를 통해 메모리에 접근하는 데이터 타입 // 값은 heap에 저장

원시타입 => 값
참조타입 => 주소

원시타입 변수 선언

아래와 같이 변수를 선언하면 메모리에선 어떤 일이 발생할까?

let a = 10;

자바스크립트는 내부적으로 let a;와 a= 10;으로 나누어 변수의 선언과 값의 할당을 수행한다.

a변수에는 10이 저장된 메모리의 주소값이 할당된다.

1. 변수 선언

: 자바스크립트는 소스 코드를 실행하기 전, 선언문(변수, 함수, 객체 등)을 먼저 스캔한다. (호이스팅)
: let a 선언문을 발견하고, a변수를 '실행 컨텍스트'라는 곳에 등록한다.

2. 초기화

: 스캔작업이 완료된 뒤, 소스코드를 위에서부터 한줄씩 실행하기 시작한다.
: let a; 구문을 만나면, 메모리에 공간을 확보하고 undefined를 할당하여 초기화 한다.
: 변수 a에는 해당 **메모리의 주소값을 저장한다.**

3. 데이터 타입 체크

: a=10; 구분을 만나면,
: 10이라는 값이 **number** 타입임을 파악

4.새로운 메모리 확보 및 값 할당

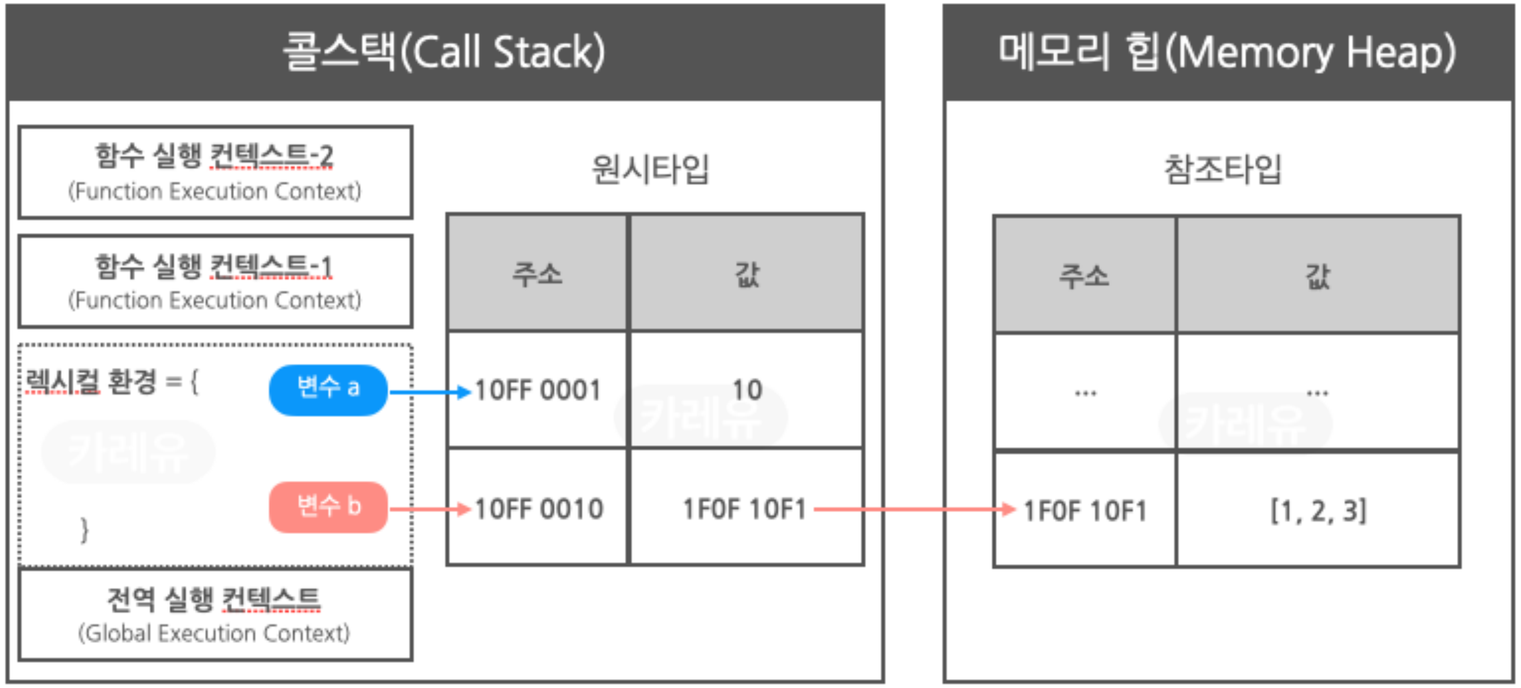
: number타입 크기인 8바이트(64비트)의 메모리 공간을 새로 확보하고,
: 10을 저장한다. * 기존에 할당된 메모리에 저장된 undefined를 10으로 변경하지 않음에 주의!

5. 주소값 저장

: a 변수에 10이 저장된 새로운 메모리의 첫번째 셀 주소값을 저장

콜스택, 메모리힙의 데이터 구조

원시타입은 콜스택에 저장되고, 참조타입은 메모리힙에 저장된다.



원시타입의 변수 데이터

변수 a, b 식별자는 실행컨텍스트의 렉시컬 환경에 저장

원시타입의 데이터 값은 콜스택에 저장되고,

데이터 값이 저장된 콜스택의 주소값은 변수 a, b 에 각각 저장됨

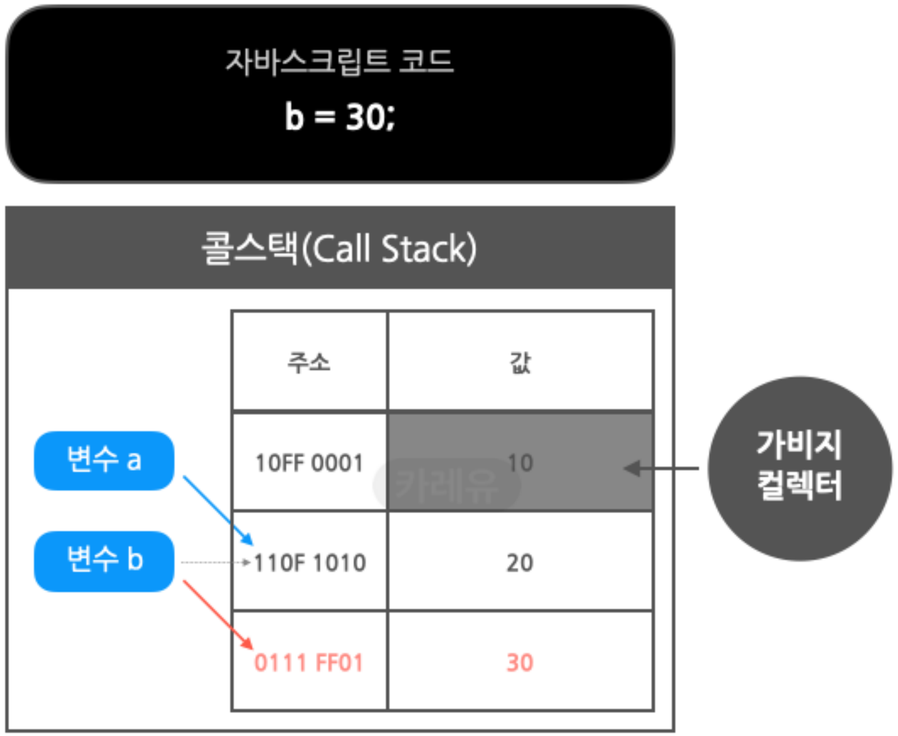
2. 원시타입 재할당 - Case1

변수 a에 20을 재할당하면,
본인의 메모리에 있는 값을 변경하는게 아니라,
기존에 20을 저장하고 있는 메모리의 주소값으로 교체한다.
a에 저장된 주소값은 20을 가리키고 있던
b에 저장된 주소값과 동일해진다.

3. 원시타입 재할당 - Case2

변수 b에 30을 재할당하면,
변수 b의 주소값이 가리키는 메모리에 저장된 20을 30으로 교체하는게 아니라,
새로운 메모리를 확보하여 30을 저장하고,
변수 b에 저장된 주소값을 해당 주소값으로 교체한다.

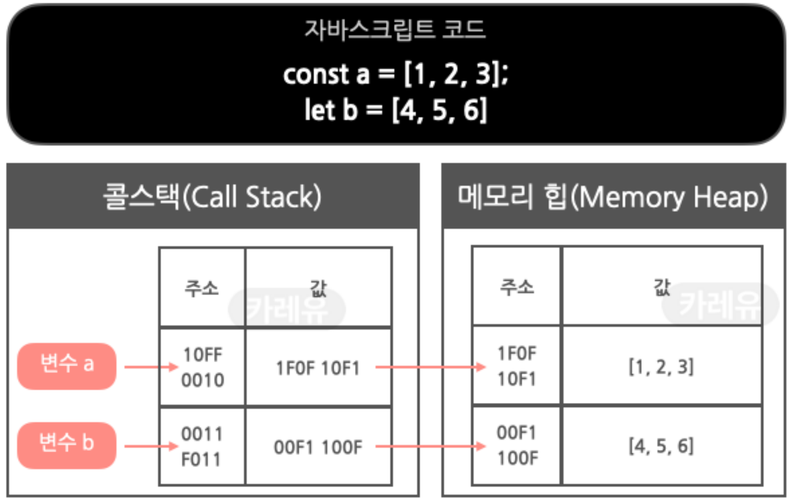
가비지 컬렉터



더이상 참조되지 않는 데이터는

가비지 컬렉터에 의해 적절한 시점에 메모리에서 해제된다.

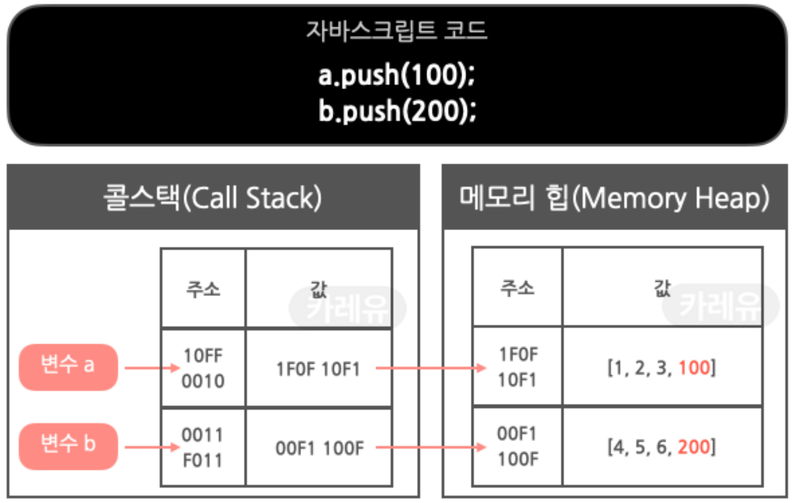
참조타입의 변수 데이터



1. 참조타입의 변수 생성

배열과 같은 참조타입 데이터는 메모리 힙에 저장된다.

메모리 힙의 주소값은 콜 스택에 저장되고, a와 b에는 해당 콜스택의 주소값이 각각 저장된다.



2. 참조타입 데이터 값 변경

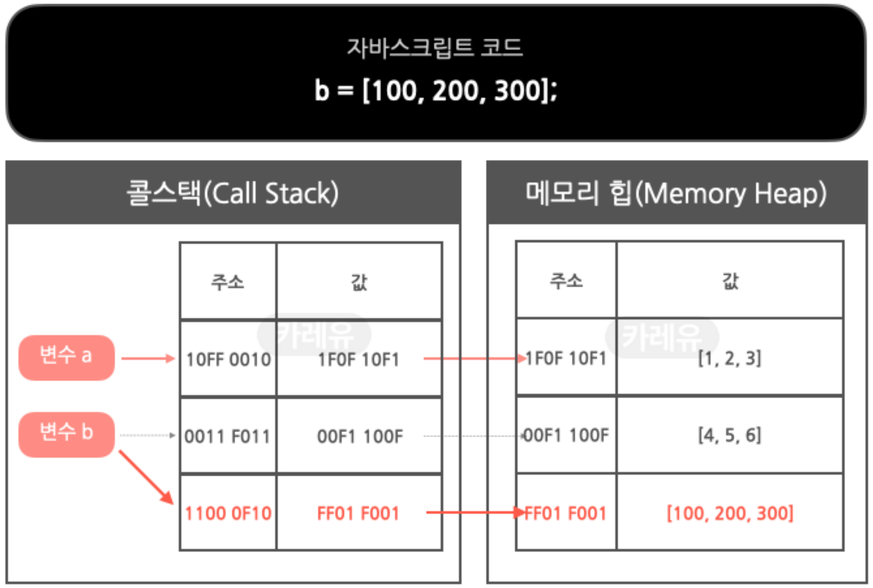
변수에 값을 재할당한게 아니라, 변수에 저장된 데이터를 수정한 것임에 주의하자.

메모리힙에 저장된 배열의 값을 변경하더라도, 배열이 저장된 **메모리힙의 주소는 그대로다.**

즉, 콜스택에 저장된 메모리힙의 주소는 변경되지 않는다.

따라서 변수 a와 b에 저장된 주소값도 변하지 않는다.

참조타입의 변수 데이터

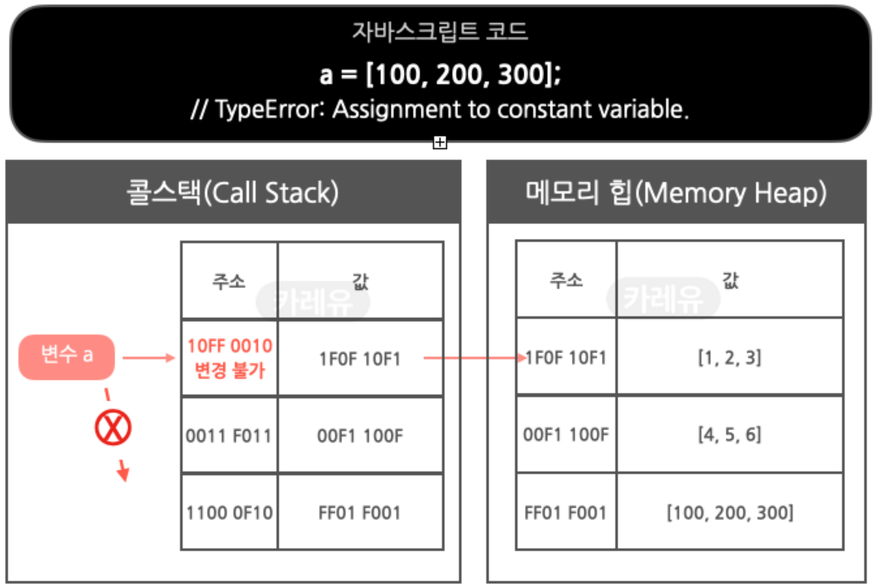


1. let으로 선언된 참조타입의 재할당

let으로 선언된 b에 다른 배열을 할당
이 경우, 메모리 **힙에 새로운 주소가 확보**되고, 새로운 배열이 저장된다.
콜스택에 저장되는 메모리힙의 주소값도 **새로운 주소로 변경**이 필요하다.

이때 콜스택도 기존에 저장되어 있던 주소값을 바꾸는게 아니라,
새로운 메모리를 확보해서 **새로운 메모리힙의 주소값을 저장**한다.

따라서 변수 **b에 저장되는 콜스택의 주소값도 변경**된다
* 참조타입은 동일한 구성의 객체를 생성하더라도, 매번 새 메모리를 확보하여 새 객체를 생성한다



2. 참조타입 데이터 값 변경

const로 선언된 a에 다른 배열을 할당을 할당하면 에러가 난다

const로 선언된 변수는 본인에게 할당된 콜스택 주소값의 변경을 허용하지 않는다. => error

* 객체와 같은 참조타입 데이터는 동적으로 내부 요소가 변경될 수 있는 관계로, 동일한 요소의 객체를 생성하더라도 메모리힙에 각각 별도의 메모리 공간을 확보하여 저장된다.

가비지 컬렉터

메모리 관리는 왜 필요한가?

대부분의 언어에서 메모리 라이프 사이클은 메모리 할당 → 메모리 사용 → 메모리 해제의 단계를 거친다. C같은 low-level 언어의 경우 이 라이프 사이클을 개발자가 `malloc()`이나 `free()`를 사용하여 직접 관리를 해주어야 하지만 자바스크립트와 같은 high-level 언어는 대부분 Garbage Collection이라는 자동 메모리 관리를 사용하기 때문에 개발자가 별도의 신경을 쓰지 않는다.

가비지컬렉션(Garbage Collection)이란?

가비지컬렉션은 더이상 사용하지 않는 메모리를 발견하고 이를 해제해주는 역할을 한다. 가비지컬렉션이 자동으로 메모리 관리를 해준다고해서 개발자가 완전히 신경을 쓰지 않는다면 메모리 누수(memory-leak)가 발생할 수도 있다. 메모리 누수란, 더이상 어플리케이션에서 사용하지 않는도 불구하고 메모리 해제가 되지 않는 상태를 말한다.

주요 알고리즘

- Reference-counting

- Mark-and-sweep

깊은 복사

- primitive data 복사 시
- 새로운 메모리 공간을 확보해 **완전히 복사**하는 것을 의미

얇은 복사

- reference type data 복사 시
- 참조 타입 데이터가 저장한 '**메모리 주소 값'을 복사**한 것을 의미

=> 아래와 같이 얕은 복사 시 원본이 변경되지 않도록 주의해야 한다.

참조문헌

https://curryyou.tistory.com/275

https://curryyou.tistory.com/276

https://sustainable-dev.tistory.com/158#recentEntries