

# I. 데이터 타입

## 1. 데이터 타입의 종류



Primitive type (원시[기본]형)

- Number
- String
- Boolean
- Null
- Undefined
- **Symbol (ES6)**



Reference type (참조형)

- **Object**
  - Array
  - Function
  - Date
  - RegExp (정규표현식)
  - **Map, WeakMap (ES6)**
  - **Set, WeakSet (ES6)**

---

할당이나 연산 시

- 원시형 : 복제 → 값이 담긴 주소값을 바로 복제
- 참조형 : 참조 → 값이 담긴 주소값들로 이루어진 묶음을 가리키는 주소값을 복제

원시형은 **불변성(immutability)**을 띈다.

## 2. 데이터 타입에 관한 배경지식

### 1. 메모리와 데이터

- 1byte = 8bit
  - JS : 숫자의 경우 8byte(64bit)를 확보 → 개발자가 형변환해야 하는 상황 줄어듦
- 각 **비트**는 고유한 식별자를 통해 위치를 확인할 수 있다.
  - 모든 데이터는 메모리 주소값을 통해 서로 구분하고 연결할 수 있다.

### 2. 식별자와 변수

- **변수(variable)** → 변할 수 있는 무언가
- **식별자(identifier)** → 변수명

## 3. 변수 선언과 데이터 할당

### 1. 변수 선언

```
| var a;
```

→ 이 데이터의 식별자는 a로 한다.


→

변수란 변경 가능한 데이터가 담길 수 있는 공간 또는 그릇.

## 2. 데이터 할당

```
var a;           // 변수 a 선언  
a = 'abc';       // 변수 a에 데이터 할당
```

```
var a = 'abc';   // 변수 선언과 할당을 한 문장으로 표현
```

 그림 1-4 데이터 할당에 대한 메모리 영역의 변화

변수 영역	주소	...	1002	1003	1004	1005	...
	데이터			이름: a <sup>①</sup> 값: @5004 <sup>③</sup>			
데이터 영역	주소	...	5002	5003	5004	5005	...
	데이터				'abc' <sup>②</sup>		

- ① 변수 영역에서 빈 공간(@1003)을 확보한다.
- ② 확보한 공간의 식별자를 a로 지정한다.
- ③ 데이터 영역의 빈 공간(@5004)에 문자열 'abc'를 저장한다.
- ④ 변수 영역에서 a라는 식별자를 검색한다(@1003).
- ⑤ 앞서 저장한 문자열의 주소(@5004)를 @1003의 공간에 대입한다.



500개의 변수를 생성해서 모든 변수에 숫자 5를 할당하는 상황.  
숫자형은 8byte가 필요함.

- 각 변수 공간마다 매번 숫자 5를 할당하려는 경우  
→  $500 * 8 = 4000\text{byte}$
- 숫자 5를 별도의 별도의 공간에 한 번만 저장하고 해당 주소만 입력하는 경우  
→ 주소 공간의 크기가 2byte라고 한다면  
→  $500 * 2 + 8 = 1008\text{byte}$
- **변수 영역**과 **데이터 영역**을 분리하면 중복된 데이터에 대한 처리 효율이 높아진다.

## 4. 기본형 데이터와 참조형 데이터

### 1. 불변값

- **변수 영역 메모리**
  - 변경 가능성을 기준
  - **변수(variable)** vs **상수(constant)**
  - 바꿀 수 있다 vs 바꿀 수 없다
- **데이터 영역 메모리**
  - 한 번 데이터 할당이 이뤄진 변수 공간에 다른 데이터를 **재할당** 할 수 있는지 여부를 기준
  - **불변성有** vs **불변성無**
- **불변성**의 성질
  - 변경은 새로 만드는 동작을 통해서만 이루어진다.
  - 한 번 만들어진 값은 가비지 컬렉팅을 당하지 않는 한 영원히 변하지 않는다.

### 2. 가변값

- **참조형 데이터**의 경우 기본적인 성질은 가변값인 경우가 많다.

**but** 설정에 따라 변경 불가능한 경우도 있고 아예 불변값으로 활용하는 방안도 있다.

- 참조형 데이터와 기본형 데이터의 차이는

**‘객체의 변수(프로퍼티) 영역’이 별도로 존재한다는 점이다.**

- 객체가 별도로 할애할 영역은 변수 영역일 뿐 ‘데이터 영역’은 기존의 메모리 공간을 그대로 활용할 수 있다.
- 데이터 영역에 저장된 값은 모두 불변값이다.

**but** 변수에는 다른 값을 얼마든지 대입할 수 있다.

→ 참조형 데이터는 불변하지 않다(가변값이다)

- 참조카운트가 0인 메모리 주소는 가비지 컬렉터의 수거 대상이 된다.

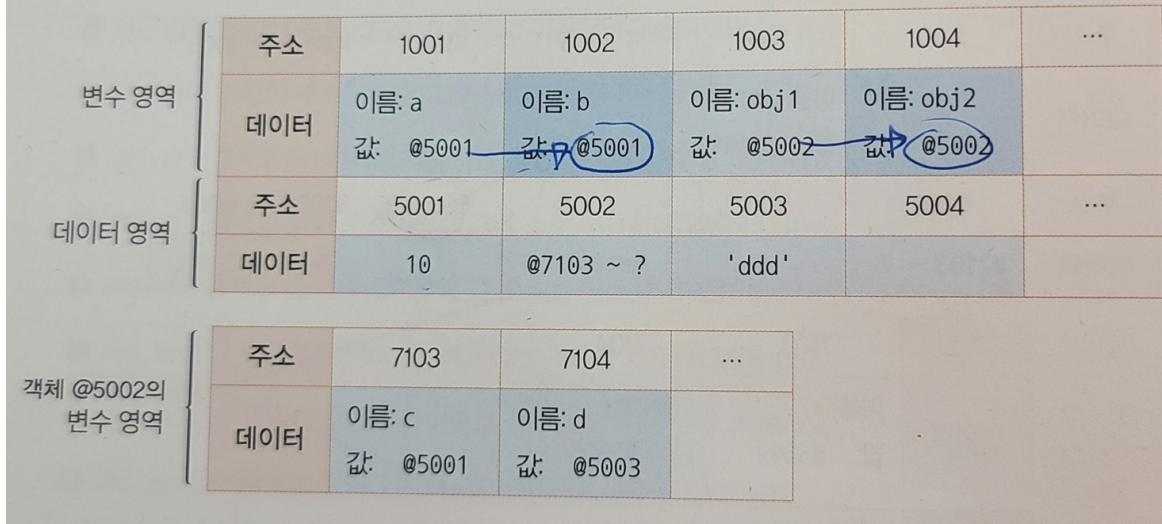
### 3. 변수 복사 비교

- 원시형(기본형) 데이터와 참조형 데이터의 차이

```
var a = 10;
var b = a;

var obj1 = { c : 10, d : 'ddd' };
var obj2 = obj1;
```

그림 1-10 변수 복사 비교



- 복사 과정은 동일하지만 데이터 할당 과정에서 이미 차이가 있기 때문에 변수 복사 이후의 동작에도 큰 차이가 발생한다.



사실은 어떠한 데이터 타입이든 변수에 할당하기 위해서는 주솟값을 복사해야한다. 엄밀히 따지면 자바스크립트의 모든 데이터 타입은 참조형 데이터일 수밖에 없다.



다만 기본형은 주솟값을 복사하는 과정이 한번만 이루어지고, 참조형은 한 단계를 더 거치게 된다는 차이가 있다.

즉, 참조형 데이터가 '가변값'이라고 설명할 때의 '가변'은 참조형 데이터 자체를 변경할 경우가 아니라 그 내부의 프로퍼티(객체)를 변경할 때만 성립한다.

## 5. 불변 객체

### 1. 불변 객체를 만드는 간단한 방법

```
var copyObject = function (target) {
  var result = {};
  for (var prop in target) {
    result[prop] = target[prop];
  }
  return result;
}
```

- 불변성 확보가 필요한 경우 → 객체의 가변성으로 방해를 받음 → 불변 객체를 만들자!!
- `copyObject` 함수는 result 객체에 target 객체의 프로퍼티들을 복사하는 함수이다.
  - 프로토타입 체이닝 상의 모든 프로퍼티를 복사하며, getter, setter는 복사하지 않는다.

## 2. 얕은 복사와 깊은 복사

- 얕은 복사 : 바로 아래 단계의 값만 복사하는 방법
- 깊은 복사 : 내부의 모든 값들을 하나하나 찾아서 전부 복사하는 방법

```
var copyObjectDeep = function(target) {
  var result = {};
  if (typeof target === 'object' && target !== null) {
    for (var prop in target) {
      result[prop] = copyObjectDeep(target[prop]);
    }
  } else {
    result = target;
  }
  return result;
};
```

## 6. undefined와 null

- '없음'을 나타내는 값
  - `undefined` : 어떤 변수에 값이 존재하지 않을 경우를 의미
  - `null` : 사용자가 명시적으로 '없음'을 표현하기 위해 대입한 값

사용자가 응당 어떤 값을 지정할 것이라고 예상되는 상황임에도 실제로 그렇게 하지 않았을 때 `undefined`를 반환한다.

1. 값을 대입하지 않은 변수, 즉 데이터 영역의 메모리 주소를 지정하지 않은 식별자에 접근할 때
2. 객체 내부의 존재하지 않는 프로퍼티에 접근하려고 할 때
3. `return` 문이 없거나 호출되지 않는 함수의 실행 결과

- 배열의 경우 조금 특이한 동작

```
var arr1 = [];  
  
arr1.length = 3;  
console.log(arr1); // [empty × 3]
```

- `undefined`와 `null`의 비교

```
var n = null;  
console.log(typeof n); // object  
  
console.log(n == undefined); // true  
console.log(n == null); // true  
  
console.log(n === undefined); // false  
console.log(n === null); // true
```