코어 자바스크립트

I. 데이터 타입

1. 데이터 타입의 종류

```
Primitive type (기본형)
- Number
- String
- Boolean
- null
- Undefined
- Symbol

Reference type (참조형)
- Object (객체)
    - Array
    - Function
    - Date
    - RegExp
    - Map, WeakMap
    - Set, Weakset
```

How 기본형과 참조형 구분?

- 기본형: 값이 담긴 주솟값을 바로 복제
- 참조형: 값이 담긴 주솟값들로 이루어진 묶음을 가리키는 주솟값을 복제
- 기본형은 `불변성`을 띔 —> 메모리와 데이터에 대한 지식 必

II. 데이터 타입에 관한 배경지식

1. 메모리와 데이터

- 비트: 0 or 1만 표현할 수 있는 하나의 메모리 조각
- 각 비트는 **고유한 식별자**(unique identifier)를 통해 위치 확인 가능
- 적정한 공간을 묶어 검색 시간 ↓, 표현할 수 있는 데이터의 개수 ↑ —> 바이트
- 1 byte = 8 bit
- 모든 데이터는 바이트 단위의 식별자, 즉 **메모리 주솟값**(memory address)을 통해 서로 구분하고 연결할 수 있음

2. 식별자와 변수

- 변수(variable): 변할 수 있는 데이터
- 식별자(identifier): 데이터를 식별하는데 사용하는 이름, 즉 변수명

III. 변수 선언과 데이터 할당

1. 변수 선언

```
var a;
- 변할 수 있는 데이터를 만든다. 이 데이터의 식별자는 a로 한다
∴ 변수 == 변경 가능한 데이터가 담길 수 있는 공간 또는 그릇
```

주소	...	1003	1004	...
데이터		이름: a 값:		

2. 데이터 할당

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

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

변수 영역

주소	…	1003	1004	…
데이터		이름: a 값: @5002		

데이터 영역

주소	…	5002	5003	…
데이터		‘abc’		

Why 변수 영역에 값을 직접 대입 X ?

∴ **데이터 변환**을 **자유롭게** 할 수 있게 함 **+ 메모리**를 더욱 **효율적**으로 관리하기 위해

- 기존 문자열에 어떤 변환을 가하든 상관 없이 무조건 새로 만들어 별도의 공간에 저장함

변수 영역

주소	…	1003	1004	…
데이터		이름: a 값: @5003		

데이터 영역

주소	…	5002	5003	…
데이터		‘abc’	‘abcdef’	

기존(@5002) 데이터는 자신의 주소를 저장하는 변수가 하나도 없게 되면

가비지 컬렉터의 수거 대상이 됨

IV. 기본형 데이터와 참조형 데이터

1. 불변값

- 변수(o) vs 상수(x): 변경 가능성
  - 변수/상수를 구분 짓는 변경 가능성의 대상은 **변수 영역**의 메모리
- 불변성 여부를 구분할 때의 변경 가능성의 대상은 **데이터 영역**의 메모리
- 불변값의 성질

- 한 번 만든 값을 바꿀 수 X
- 한 번 만들어진 값은 가비지 컬렉팅을 당할 때까지 영원히 변하지 X

2. 가변값

- 기본형 데이터는 모두 불변값
- 참조형 데이터는 가변값인 경우 多

  but 설정에 따라 변경 불가능한 경우 / 불변값으로 활용하는 방안도 있음
- 기본형 데이터와의 차이는 '**객체의 변수(프로퍼티) 영역**'이 별도로 존재한다는 점
- 데이터 영역에 저장된 값은 모두 불변값

변수 영역

주소	1001	1002	1003	...
데이터		이름: obj1 값: @5001		

데이터 영역

주소	5001	5002	5003	...
데이터	@7103 ~ ?	1	'bbb'	

객체 @5001의 변수 영역

주소	7103	7104	7105	...
데이터	이름: a 값: @5002	이름: b 값: @5003		

3.변수 복사 비교

```
// 기본형 데이터와 참조형 데이터의 차이 확인

var a = 10;
var b = a;

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

변수 영역

주소	1001	1002	1003	1004
데이터	이름: a 값: @5001	이름: b 값: @5001	이름: obj1 값: @5002	이름: obj2 값: @5002

데이터 영역

주소	5001	5002	5003	5004
데이터	10	@7103 ~?	'ddd'	

객체 @5002의 변수 영역

주소	7103	7104	7105	...
데이터	이름: c 값: @5001	이름: d 값: @5003		

- 복사 과정은 동일 but 데이터 할당 과정에서 차이 O → 변수 복사 이후 동작에 큰 차이 O
- **어떤 데이터 타입이든** 변수에 할당하기 위해서는 **주솟값을 복사**해야 함

  → 엄밀히 따지면 JS의 모든 데이터 타입은 참조형 데이터일 수밖에 없음

  → 다만 기본형은 주솟값을 복사하는 과정이 한 번만 이루어지고

  참조형은 한 단계를 더 거치게 된다는 차이가 있음
- 참조형 데이터의 '**가변값**'의 가변

  → 참조형 데이터 자체를 변경 X, 그 내부의 프로퍼티를 변경할 때 성립

V. 불변 객체

1. 불변 객체를 만드는 간단한 방법

```
// 기존 정보를 복사해서 새로운 객체를 반환하는 함수 (얕은 복사)

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

```
            result[prop] = target[prop];
        }
        return result;
};

var user = {
    name: 'Jaenam',
    gender: 'male'
};

var user2 = copyObject(user);
user2.name = 'Jung'

console.log(user.name, user2.name);
console.log(user === user2);
```

불변 객체가 필요한 상황?

- 값으로 전달 받은 객체에 변경을 가하더라도 원본 객체는 변하지 않아야 하는 경우가 종종 발생함 → 이럴 때 불변 객체가 필요함
- 프로퍼티 변경을 할 수 없게끔 시스템적 제약을 거는 법
    - immutable.js, baobab.js 등의 라이브러리
    - JS 내장 객체가 아닌 라이브러리 자체에서 불변성을 지닌 별도의 데이터 타입 & 메서드 제공

2. 얕은 복사와 깊은 복사

- 얕은 복사: 바로 아래 단계의 값만 복사하는 방법
- 깊은 복사: 내부의 모든 값들을 하나하나 찾아서 전부 복사하는 방법
- 기본형 데이터일 경우에는 그대로 복사하면 되지만 참조형 데이터는 다시 그 내부의 프로퍼티들을 복사해야 함
- hasOwnProperty 메서드 → 프로토타입 체이닝 → 프로퍼티를 복사하지 않게끔 가능
- ES 5의 getter/setter를 복사하는 방법은 ES6의 Object.getOwnPropertyDescriptor or ES2017의 Object.getOwnPropertyDescriptors로 가능

```
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;
}

var obj = {
    a: 1,
    b: {
        c: null,
        d: [1, 2]
    }
};

var obj2 = copyObjectDeep(obj);

console.log(obj);  // 원본 객체 출력
console.log(obj2); // 깊은 복사된 객체 출력
console.log(obj === obj2); // false, 서로 다른 객체
console.log(obj.b === obj2.b); // false, 중첩된 객체도 서로 다른 ㅈ
console.log(obj.b.d === obj2.b.d); // false, 중첩된 배열도 서로 ㄷ
```

VI. undefined와 null

- '없음'을 나타내는 값 두 가지
  - undefined
  - null

- JS 엔진이 자동으로 undefined를 부여하는 경우

- 데이터 영역의 메모리 주소를 지정하지 않은 식별자에 접근할 때
- 객체 내부의 존재하지 않는 프로퍼티에 접근하려고 할 때
- return 문이 없거나 호출되지 않는 함수의 실행 결과

- ‘비어있는 요소’와 ‘undefined를 할당한 요소’는 출력 결과부터 다름
  - 전자는 순회와 관련된 많은 배열 메서드들의 순회 대상에서 제외됨

- ‘비어있음’을 명시적으로 나타내고 싶을 때는 undefined가 아닌 null을 써야 혼돈 x

  ∴ undefined는 어떤 변수에 값이 존재하지 않을 경우를 의미하고

  null은 사용자가 명시적으로 ‘없음’을 표현하기 위해 대입한 값