자바스크립트
데이터 타입

임준혁

목차

데이터 타입?

데이터 타입?

데이터 타입이란?

변수나 상수에 저장되는 데이터의 종류를 정의하는 방법

데이터 타입?

원시 타입

참조 타입

원시 타입

원시 타입의 정의

원시 타입(Primitive Type)이란?

객체(Object)가 아니면서 메서드(Method)로 가지지 않는 데이터이고,
모든 원시 요소들은 불변성(Immutability)을 띔

원시 타입의 종류 – null , undefined , boolean

null

: 어떤 값이 의도적으로 비어 있음을 표현하는 타입

```
const _null = null; // null
const _null2 = Null; // 오류
const _null3 = NULL; // 오류
```

undefined

: 값을 할당하지 않은 변수를 가리키는 타입

```
let _undefined; // undefined
let _undefined2 = undefined; // undefined
let _undefined3 = Undefined; // 오류
```

Boolean

: 논리형 데이터 타입인 true, false 값을 표현하는 타입

```
const boolean = true; // true
const boolean2 = false; // false
console.log(typeof(boolean, boolean2)) // boolean
```

원시 타입의 종류 – String , Number

String

: 문자형 데이터를 나타내는 타입

```
const string = 'string'; // 'string'
const string2 = '홍길동'; // '홍길동'
const string3 = '1541'; // '1541'
```

Number

: 1, -1, 1.2같은 숫자를 표현하고 다루는 타입
$(-2^{53} - 1) \sim (2^{53} - 1)$ 범위 안의 데이터를 안정적으로 저장
해당 안전 범위를 벗어나면 근사치를 저장

```
const number = 1; // 1
const number2 = -1; // -1
const number3 = 1.2; // 1.2
console.log(typeof(number)); // number
console.log(typeof(number2)); // number
console.log(typeof(number3)); // number
```

원시 타입의 종류 – BigInt , Symbol

BigInt

: Number의 값이 안정적으로 나타낼 수 있는 최대치인 $2^{53} - 1$ 보다 큰 정수를 안정적으로 표현하는 타입

```
const bigint = BigInt(1234); // 1234n
const bigint2 = BigInt(1234n); // 1234n
const bigint3 = BigInt(-1234n); // -1234n
const bigint4 = bigint(); // 오류
const bigint5 = Bigint(); // 오류
console.log(typeof(bigint)); // bigint
console.log(typeof(bigint2)); // bigint
console.log(typeof(bigint3)); // bigint
```

Symbol

: 유일한(unique)한 식별자를 생성하기 위한 타입

```
const _symbol = Symbol(); // Symbol()
const _symbol2 = Symbol('심볼id'); // Symbol(심볼id)
const _symbol3 = symbol(); // 오류
console.log(typeof(_symbol)); // symbol
```

원시 타입의 특징

예시

```
let bar = 'primitive value';

bar.toUpperCase();
console.log(bar); //

// ex2)
bar = bar.toUpperCase();
console.log(bar); //


let foo = 2;

function addTwo(num) {
  return num += 2;
};

addTwo(foo);
console.log(foo);

foo = addTwo(foo)
console.log(foo);
```

참조 타입

참조 타입(Reference Type)이란?

Unlike Primitive, 즉 원시 자료형이 아닌 모든 것
참조 타입은 변수에 값을 직접 저장하지 않음
변수에 저장되는 것은 메모리 공간 주소를 가리킴

참조 타입의 종류 – Array, Object

Array

: 이름과 인덱스로 참조되는 정렬된 값의 집합
  즉, 하나의 변수에 1개 이상의 값을 저장하는 타입

```
const array = [1, '2', true]; // [1, '2', true]
console.log(typeof(array)); // object
```

Object

: 이름과 값으로 구성된 프로퍼티의 정렬되지 않은 집합

```
const object = {              // {
  name: '임준혁',             //   name: '임준혁',
  address: '안산',            //   address: '안산',
  accounts: ['기업', '국민']  //   accounts: ['기업', '국민']
};                            // }
console.log(object);
console.log(typeof(object)); // object
```

참조 타입의 종류 – Function

Function

: 하나의 특별한 목적의 작업을 수행하도록 설계된 독립적인 블록

```
// 함수 표현식
function func () {
  return '어떤걸 반환할까';
};
// 함수 선언식
const def = function () {
  return '어떤걸 반환할까';
};
// 화살표 함수
const func2 = () => {
  return '어떤걸 반환할까'
};
console.log(typeof(func)); // function
console.log(typeof(func2)); // function
console.log(typeof(def)); // function
```

참조 타입의 종류 – 날짜(Date) , 정규표현식(RegularExpression)

날짜(Date)

: 시간의 한 점을 플랫폼에 종속되지 않는 형태로 나타냄
즉, 시간(날짜)을 저장하는 타입

```
const date = new Date(); // 현재 날짜, 시간 출력
console.log(date); // ex) 2023-04-16T050:43:07.666Z
console.log(typeof(date)); // object
```

정규표현식(RegularExpression)

: 문자열에서 특정 문자 조합을 찾기 위한 패턴

```
const string = '정규식으로 값을 찾거나 변환 할 문자열';
const regex = new RegExp(/정규식/);
regex.test(string); // true false 반환
regex.replace(string); // 정규식에 맞는 문자열 치환
// ... 등등 많음
console.log(typeof(regex)); // object
```

참조 타입의 종류 – Set, WeakSet

Set

: 자료형에 관계 없이 원시 값과 객체 참조 모두 유일한 값을 저장할 수 있는 타입
즉, 배열과 유사하지만 배열 내부 데이터가 중복되지 않는 타입

```
const set = new Set(); // Set(1) {
                       //   add한 값이 들어옴
                       // }
set.add(1); // 1을 추가
set.delete(1) // 1을 삭제
// ... 등등
```

WeakSet

: Set과 유사하지만 WeakSet는 참조를 가지고 있는 객체만 저장 가능 (원시 타입 불가)

```
const set2 = new WeakSet();
const arr = [1,2];
set2.add(arr);
set2.add(1) // 오류
```

참조 타입의 종류 – Map, WeakMap

Map

: Key와 Value로 이루어진 타입
Key는 고유한 값을 가져야 함

```
const map = new Map();      // Map(안에들어가있는요소의개수) {}
map.set(1, 1);              // Map(1) { 1 => 1 }
map.get(1);                 // 1
```

WeakMap

: Map과 유사하지만 Key는 반드시 객체임 (원시 타입 불가)

```
const map = new WeakMap();
const obj = {};
map.set(obj, 'set obj');
map.get(obj); // 'set obj'
map.set(1, 1); // 오류
```

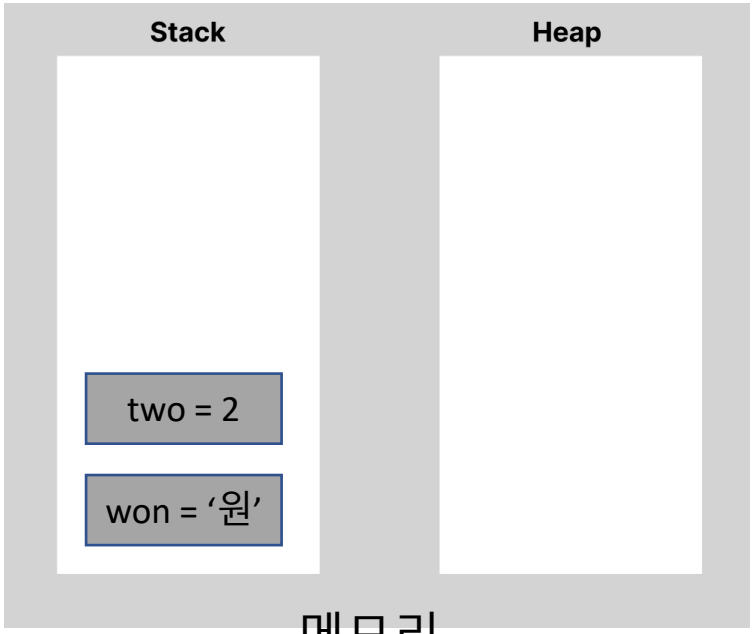
참조 타입 특징

예시

```
const array = [1, 2, 3, 4, 5] // 참조타입
// array는 [1,2,3,4,5]가 들어간 메모리 주소를 변수(array)에 저장

function hi () {
  console.log('func');
}
// hi는 함수 내부 코드블록을 메모리 주소를 함수(hi)에 저장

const object = { // 참조타입
  name: '임준혁', // 원시타입
  address: '안산', // 원시타입
  accounts: ['기업', '국민'] // 참조타입
}
// name과 address는 원시타입이라 변수에 원시 값이 할당된 반면에,
// object.accounts는 배열(참조타입)이라
// ['기업','국민']의 값이 저장된 메모리주소를 accounts에 할당함

const copy = object; // copy에 object가 가르키는 메모리주소 값을 복사
console.log(copy === object); //
copy.name = '홍길동';
console.log(copy.name, object.name); //

const obj_1 = { name: 'obj_1' };
const obj_2 = { name: 'obj_2' };
console.log(obj_1 === obj_2); //
```

원시, 참조 타입 저장

```
let won = '원';
let two = 2;
```




```
const object = { // 참조타입
  name: '임준혁', // 원시타입
  address: '안산', // 원시타입
  accounts: ['기업', '국민'] // 참조타입
}
```