2강
변수와 연산자
&
데이터 타입

2강

변수

JavaScript

➢ 변수란?

- 데이터 저장 공간
- 변수는 **이름표가 붙은 그릇**

  그릇 안에는 물, 밥, 커피 등 어떤 걸 담든 자유롭게 **바꿀 수 있다**.

- **value(값) => 10**
- **variable(변수, 즉 값이 들어있는 그릇) => a**

Ex) var a = 10;

같다 X

JavaScript

변수 이름 작성 규칙

1. **영문자, 숫자, 언더스코어(_), 달러($)만 사용 가능**
   예: user1, _count, $value

2. **숫자로 시작할 수 없음**
   1user(X) user1(O)

3. **공백(띄어쓰기) 불가능**
   user name(X) username(O)

4. **대소문자 구분**
   user 와 User 는 다른 변수로 취급됨.

5. **예약어 사용 금지**
   예: var, let, const, if, for 등은 변수명으로 쓸 수 없음.

camelCase 사용: userName, totalCount
의미 있는 이름 사용: x 대신 score, age 등

변수에 데이터 할당 하는 방법

JavaScript

➢ 변수 선언 + 데이터 할당

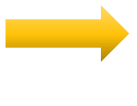
```
var a = 10;
let b = "Hello";
const c = true;
```

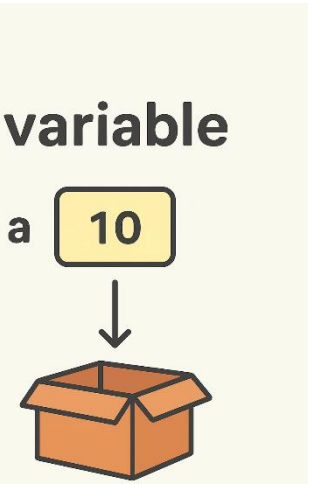
var, let, const → 변수를 만드는 키워드

= (할당 연산자) → 오른쪽 값을 왼쪽 변수에 저장

10, "Hello", true → 실제 데이터(값)

변수에 데이터를 할당하는 방법은 = 기호를 사용해서 변수에 값을 담는 것이고, 값은 숫자, 문자열, 배열, 객체, 함수 등 어떤 데이터든 들어갈 수 있다.





JavaScript

➤ 변수

- 자바스크립트의 데이터 저장 공간

```
var score;                  // 변수 score 선언
var year, month, day;       // year, month, day의 3 개의 변수 선언
var address = "서울시";     // address 변수를 선언하고 "서울시"로 초기화
```

```
age = 21;          // var 없이, 변수 age를 선언하고 21로 초기화
```

- 자바스트립트는 변수 타입을 선언하지 않는다.

```
var score; // 정상적인 변수 선언
int score; // 오류. 변수 타입 int 없음
```

- 변수에 저장되는 값에 대한 제약이 없다.

```
score = 66.8; // 실수도 저장 가능
score = "high"; // 문자열로 저장 가능
```

JavaScript

➢ 변수 선언 키워드를 이용한 변수 선언 3가지 방식

1.var

var문에서 변수에 초기 값을 지정하지 않는다면, 변수는 값이 설정될 때까지 **undefined** 값을 갖게 된다.

```
var i;  // 선언, "undefined"가 저장됨
var sum = 0;  // 선언과 초기화

var i, sum; // 한 번에 여러 개의 변수를 함께 선언할 수 있음

var i=0, sum=10, message="Hello";  // 선언과 초기화를 동시에 해줄 수 있음

name = "javascript";  // 선언되지 않은 변수는 전역 변수가 됨
```

재선언

```
var title = 'book';
console.log(title); // book

var title = 'movie';
console.log(title); //movie

title = 'music';
console.log(title);//music
```

- **var** 는 원조 변수선언방식으로, 옆 코드와 같이 선언한 변수가 동일한 이름으로 **중복 선언이** 가능하다. 즉, 마지막에 할당된 값이 최종 변수에 저장된다.
- 변수를 유연하게 사용할 수 있지만, 기존에 선언해 둔 변수의 존재를 잊고 재선언 하는 경우 문제가 발생할 수 있다.
- 특히 간단한 코드가 아닌, 길고 복잡한 코드에서 같은 이름의 변수가 여러번 선언되어 사용되면 어떤 부분에서 값이 변경되고 문제가 발생하는지 파악하기 힘들다.
- 이를 보완하기 위해 ES6 부터 추가된 변수 선언 방식이 let 과 const 이다.

JavaScript

➢ 변수 선언 3가지 방식

2. let

```
let title = 'book';
console.log(title); // book

let title = 'movie';
console.log(title);
//Uncaught SyntaxError: Identifier 'title' has already been declared

title = 'music';
console.log(title);//music
```

- let은 var와 달리 중복 선언 시, 해당 변수는 이미 선언되었다는 에러 메시지를 뱉는다.
- 즉, **중복 선언이 불가**하다. 하지만 변수에 값을 **재할당하는 것은 가능**하다.

= 재선언 X

JavaScript

➢ 변수 선언 3가지 방식

3. const (상수 변수) : 반드시 선언과 할당을 동시에 지정한다.

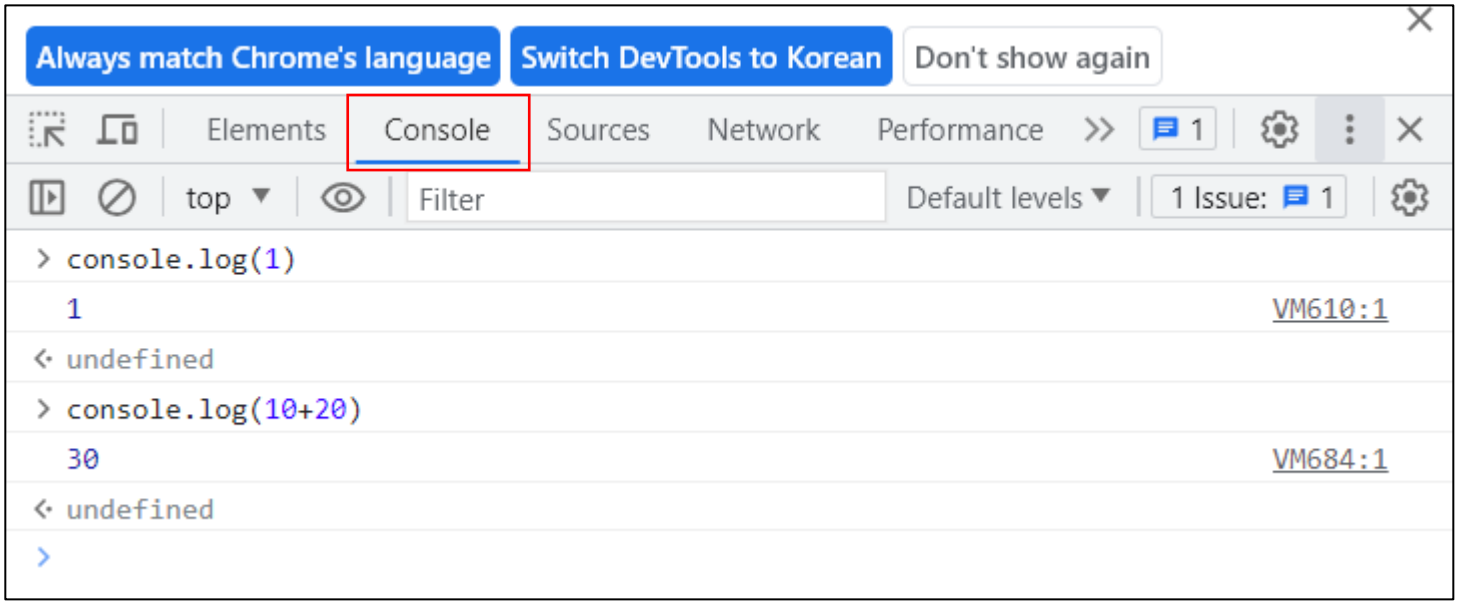
```
const title = 'book';
console.log(title); // book

const title = 'movie';
console.log(title);
//Uncaught SyntaxError: Identifier 'title' has already been declared

title = 'music';
console.log(title);
//Uncaught TypeError: Assignment to constant variable
```

- let와 const의 차이는 **immutable(재할당)가능여부**이다. 재할당은 가능한 let과 달리 const는 재할당 또한 불가하다.

재선언 / 재할당 X

JavaScript

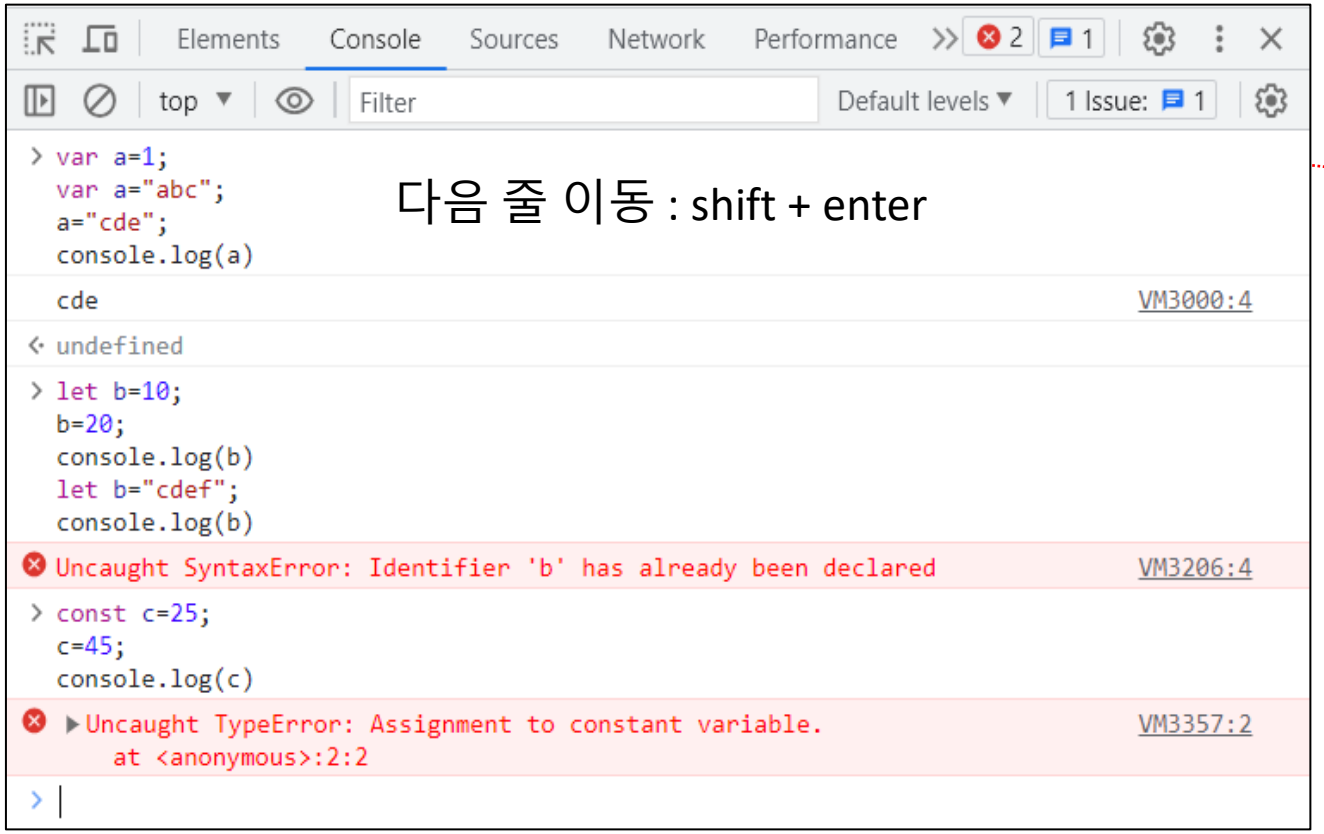
Console.log



- Console.log는 Java의 System.out.println("javascript")와 같이 데이터를 console 창에 출력하는 명령어이다.

JavaScript

➢ Var, let, const 변수 선언 키워드 비교



다음 줄 이동 : shift + enter

키워드	설명
var	재선언O, 재할당O
let	재선언X, 재할당O
const	재선언X, 재할당X

JavaScript

➢ 변수 선언과 메모리

자바스크립트 엔진이 변수 a라는 이름표(label)를 만들고,
메모리 어딘가에 10이라는 값을 저장한 뒤, 그 위치(주소)를 a와 연결해 둔다.
즉, 변수는 값 자체가 아니라 값이 있는 메모리 주소를 가리키는 이름표라고 볼 수 있다.

1. Stack
    - num과 str 같은 기본형(primitive type) 값은 스택에 바로 저장된다.
    - obj와 arr 같은 참조형(reference type) 값은 스택에 주소(참조 값)만 저장된다.
2. Heap
    - 실제 객체와 배열 데이터는 힙에 저장된다.
    - obj가 가리키는 { name: "Tom" }, arr가 가리키는 [1,2,3]가 힙에 존재한다.

3. 요약

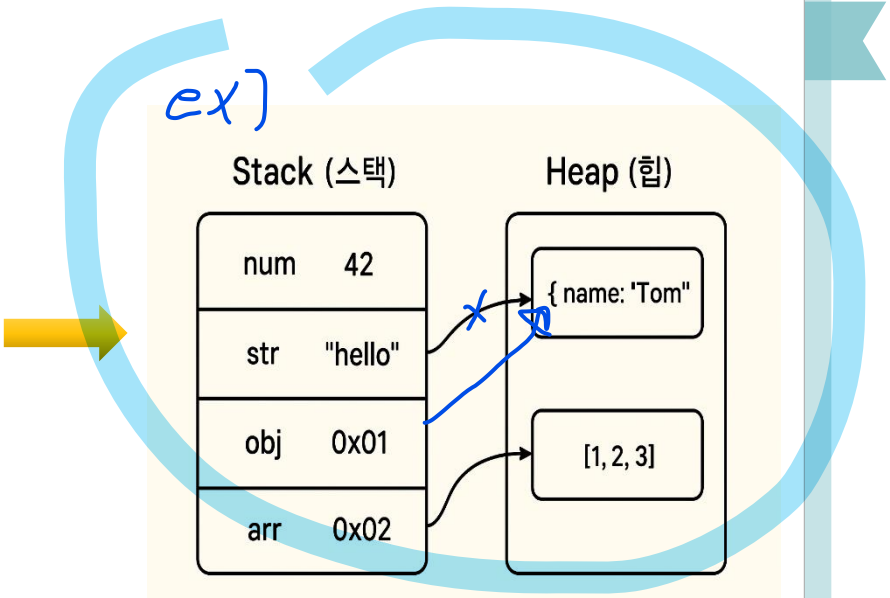
- 스택: 변수 이름과 실제 값(기본형) / 참조값(참조형) 저장
- 힙: 참조형 데이터 실제 저장

JavaScript

➢ 메모리구조 (Stack & Heap)

```
let num = 42;              // 기본형 값
let str = "hello";         // 기본형 값
let obj = { name: "Tom" }; // 참조형 값
let arr = [1, 2, 3];       // 참조형 값
```

```
Stack (스택)                Heap (힙)
------------------          ------------------------
num  ->  42                  obj -> { name: "Tom" }
str  -> "hello"              arr -> [1, 2, 3]
obj  ->  (주소) 0x01
arr  ->  (주소) 0x02
```



JavaScript

➢ Stack이란?

1. Stack(스택)

정의: 컴퓨터 메모리에서 작고 빠른 메모리 영역

용도: 원시 값(Primitive data type)이나 함수 호출 정보를 저장

2. 특징:

LIFO(Last In First Out) 구조 → 나중에 들어간 데이터가 먼저 나옴

크기가 작지만 접근 속도가 빠름

메모리 할당과 해제가 자동적임

3. 저장되는 것:

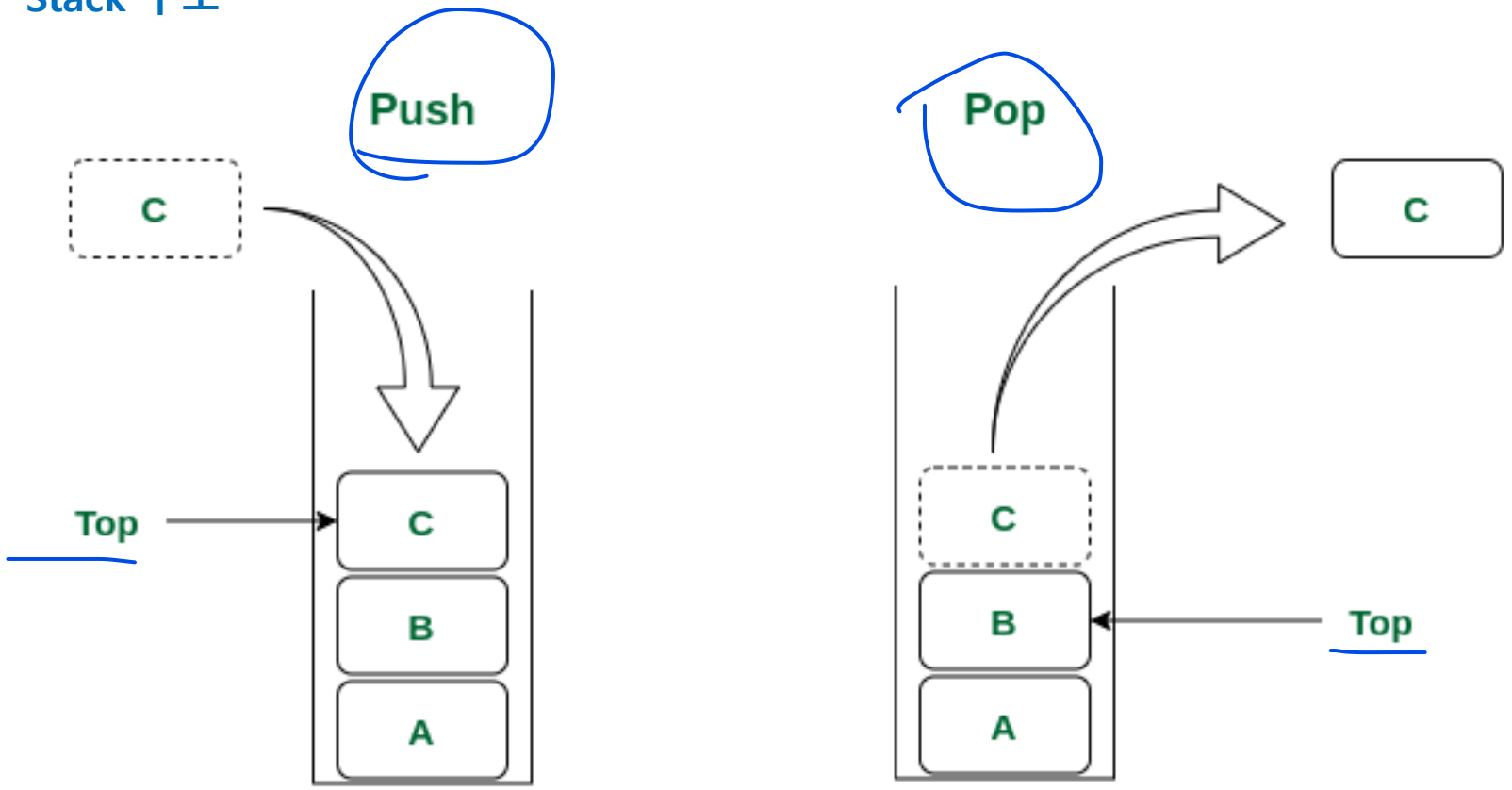
숫자(Number), 문자열(String, 짧은 경우), 불리언(Boolean), null, undefined, 심볼(Symbol)

함수 호출 시 생성되는 지역 변수

F/T

JavaScript

➢ Stack 구조



JavaScript

➢ Heap이란?

1. Heap(힙)

정의: 상대적으로 큰 메모리 영역, 구조화된 데이터를 저장

용도: 객체(Object), 배열(Array), 함수(Function) 등의 참조형 데이터 저장

2. 특징:

크기가 크고 동적(Dynamic) 할당 가능

접근 속도는 스택보다 느림

자바스크립트 엔진이 Garbage Collector를 사용해 더 이상 사용하지 않는 객체를 자동으로 정리

3. 저장되는 것:

객체, 배열, 함수등 참조형 데이터

☞ 여기서 잠깐 !!

1] 자바스크립트에서 변수 a에 숫자 10을 저장하려면 어떤 문장이 올바른가?
A) let a == 10;
B) let a = 10;
C) a : 10;
D) let 10 = a;

2] 다음 중 자바스크립트에서 변수 이름으로 사용할 수 없는 것은?
A) myVariable
B) _name
C) 2ndNumber
D) $price

3] const로 선언한 변수와 관련 없는 설명은?
A) 재할당이 불가능하다
B) 선언과 동시에 초기화해야 한다
C) 객체 내부 속성도 변경할 수 없다
D) 블록 스코프를 가진다

4] var, let, const 중 호이스팅이 발생하지만 값을 재할당할 수 있는 변수는?
A) var
B) let
C) const
D) 모두 해당 없음

☞ 여기서 잠깐 !!

5] 자바스크립트의 스택(Stack) 자료구조 특징으로 올바른 것은?
A) FIFO 구조를 따른다
B) LIFO 구조를 따른다
C) 임의 위치에 삽입이 가능하다
D) 자동으로 메모리가 해제되지 않는다

6] 다음 중 힙(Heap)에 저장되는 데이터는 무엇인가?
A) 숫자, 문자열 등 원시 값
B) 객체, 배열, 함수
C) 상수
D) 변수 이름

7] 자바스크립트 변수의 데이터 타입이 아닌 것은?
A) number
B) string
C) boolean
D) char

8] let a = 10; 후 a = 20;을 수행하면 메모리에서 a의 값은?
A) 10
B) 20
C) 오류 발생
D) undefined

2강

연산자

JavaScript

➤ 연산자 종류 & 증감 연산자

연산 종류	연산자	연산 종류	연산자
산술	+ - * / %	대입	= *= /= += -= &= ^= |= <<= >>= >>>=
증감	++ --	비교	> < >= <= == !=
비트	& | ^ ~	논리	&& || !
시프트	>> << >>>	조건	? :

(a) 전위연산자

```
a = 1;
b = ++ a;
```

① a=2
② b=2

(b) 후위연산자

```
a = 1;
b = a ++;
```

① a=2
② b=1

a++의 연산은 증가 전의 값 1을 반환하여 b의 값은 1이 됨

연산자	내용	연산자	내용
a++	a를 1 증가하고 증가 전의 값 반환	++a	a를 1 증가하고 증가된 값 반환
a--	a를 1 감소하고 감소 전의 값 반환	--a	a를 1 감소하고 감소된 값 반환

JavaScript

➢ 논리 연산자 종류

연산자	별칭	내용
a && b	논리 AND 연산	a, b 모두 true일 때 true 리턴
a || b	논리 OR 연산	a, b 중 하나라도 true이면 true 리턴
!a	논리 NOT 연산	a가 true이면 false 값을, false이면 true 값 리턴

➢ 조건 연산

- condition ? expT : expF

  condition이 true이면 전체 결과는 expT의 계산 값

  false이면 expF의 계산 값

```
var x=5, y=3;
var big = (x>y) ? x : y; // (x>y)가 true이므로 x 값 5가 big에 대입된다.
```

2강

데이터타입

JavaScript

➢ 데이터 타입

- **숫자 타입(Number)** : 정수, 실수 (예 : 42, 3.14)
- **Boolean 타입** : 참, 거짓 (예 : true, false)
- **문자열 타입(String)** : 예) "좋은 세상", "a". "365", "2+4"
- **객체 레퍼런스 타입(Object)** : 객체를 가리킴 , C언의 포인터와 유사
- **null** : 값이 없음을 표시하는 특수 키워드, Null, NULL과는 다름
- **자바스크립트는 문자 타입 없고, 문자열로 표현**
- **Undefined** : 선언 이후 값을 할당하지 않은 변수는 undefined 값을 가진다. 즉, 선언은 되었지만 값을 할당하지 않은 변수에 접근하거나 존재하지 않는 객체 프로퍼티에 접근할 경우 undefined가 반환된다
- **Symbol** : 심볼은 주로 이름의 충돌 위험이 없는 유일한 객체의 프로퍼티 키(property key)를 만들기 위해 사용한다. 심볼은 Symbol 함수를 호출해 생성한다.
- **NaN** : 산술 연산 불가(not-a-number)(더하기는 제외)

JavaScript

➤ 문자열 데이터 타입: String

```
> console.log("Hello")
  Hello                                          VM487:1
< undefined
> console.log('World')
  World                                          VM548:1
< undefined
> console.log('Hello'+'World')
  HelloWorld                                     VM643:1
< undefined
> console.log('Hello '+ 'World')
  Hello World                                    VM733:1
< undefined
>
```

JavaScript

➤ 숫자 데이터 타입: Number

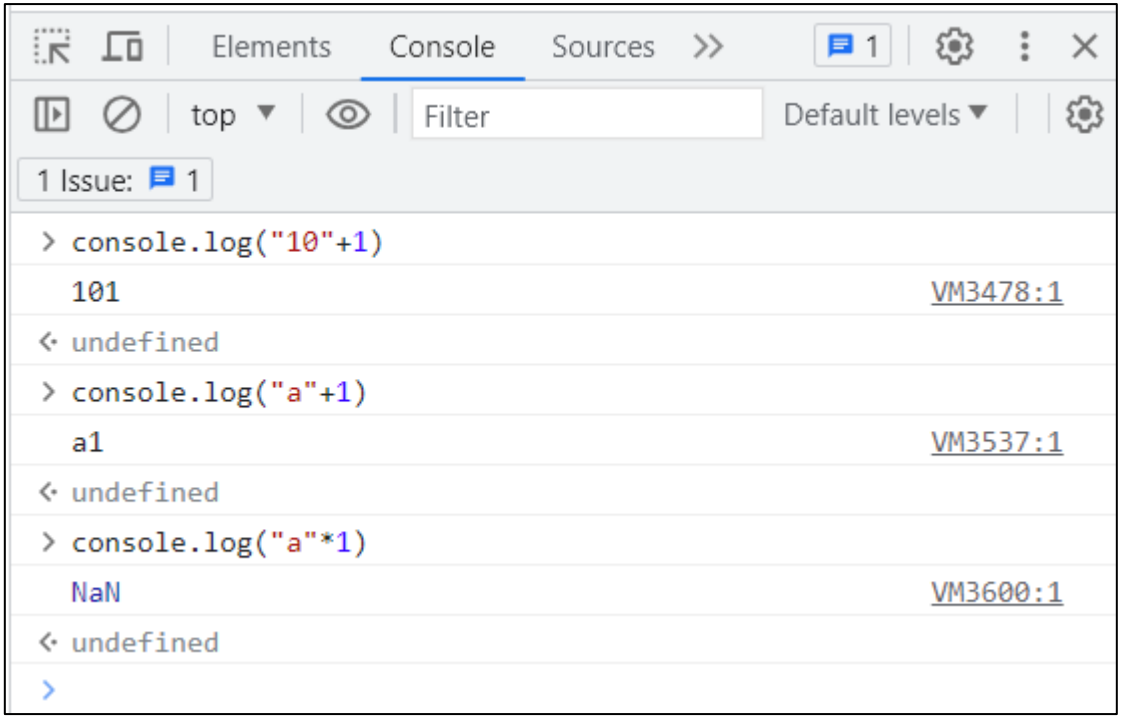
```
> console.log(1+2)
  3                                   VM791:1
< undefined
> console.log(3-1)
  2                                   VM847:1
< undefined
> console.log(10/2)
  5                                   VM909:1
< undefined
> console.log(3*5)
  15                                  VM969:1
< undefined
> console.log('a'+1)
  a1                                  VM1030:1
```

```
> console.log('1' + 1)
  11                                  VM1155:1
< undefined
> console.log(1 + '')
  1                                   VM1220:1
< undefined
> console.log('a' * 1)
  NaN                                 VM1292:1
< undefined
> console.log('2'*2)
  4                                   VM1353:1
```

NaN => Not A Number의 약어이다.

JavaScript

➤ 데이터 타입 : NaN



JavaScript

➢ 데이터 타입 : 배열(Array)

- 순서가 있는 데이터 컬렉션을 저장할 때 사용한다.
- 데이터 컬렉션이 논리적으로 정의 된 규칙에 의해 나열된 것이다.
- 자바스크립트에서 배열(array)은 이름과 인덱스로 참조되는 정렬된 값의 집합으로 정의된다.
- 배열을 구성하는 각각의 값을 배열 요소(**element**)라고 하며, 배열에서의 위치를 가리키는 숫자를 인덱스(**index**)라고 한다.
- 배열 요소의 타입이 고정되어 있지 않으므로, 같은 배열에 있는 **배열 요소끼리의 타입이 서로 다를 수도 있다.**
- 배열 요소의 인덱스가 연속적이지 않아도 되며, 따라서 특정 배열 요소가 비어 있을 수도 있다.
- 자바스크립트에서 배열은 Array 객체로 다뤄진다.

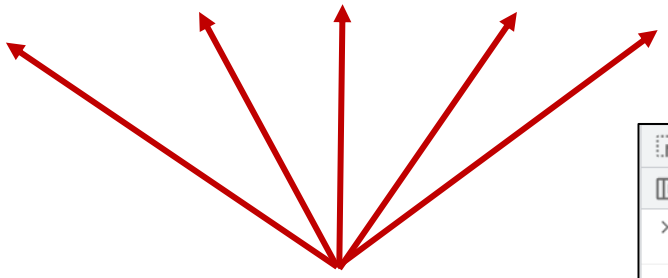
문법

1. *var* arr = [배열요소1, 배열요소2,...]; *// 배열 리터럴을 이용하는 방법*
2. *var* arr = Array(배열요소1, 배열요소2,...); *// Array 객체의 생성자를 이용하는 방법*
3. *var* arr = new Array(배열요소1, 배열요소2,...); *// new 연산자를 이용한 Array 객체 생성 방법*

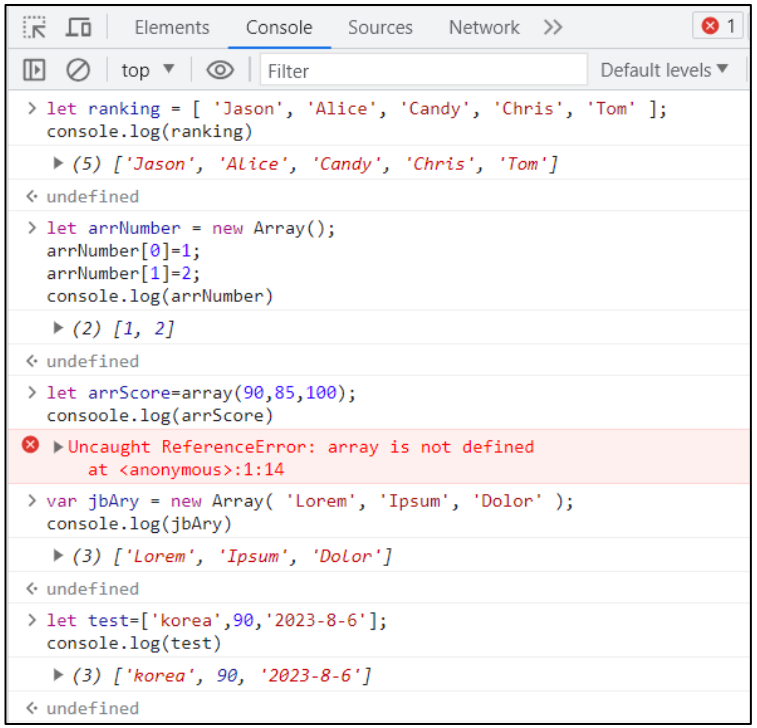
JavaScript

➢ 데이터 타입 : 배열(Array)

let ranking = [“Jason”, ”Alice”, ”Candy”, ”Chris”, ”Tom”]



요소 (Element)



JavaScript

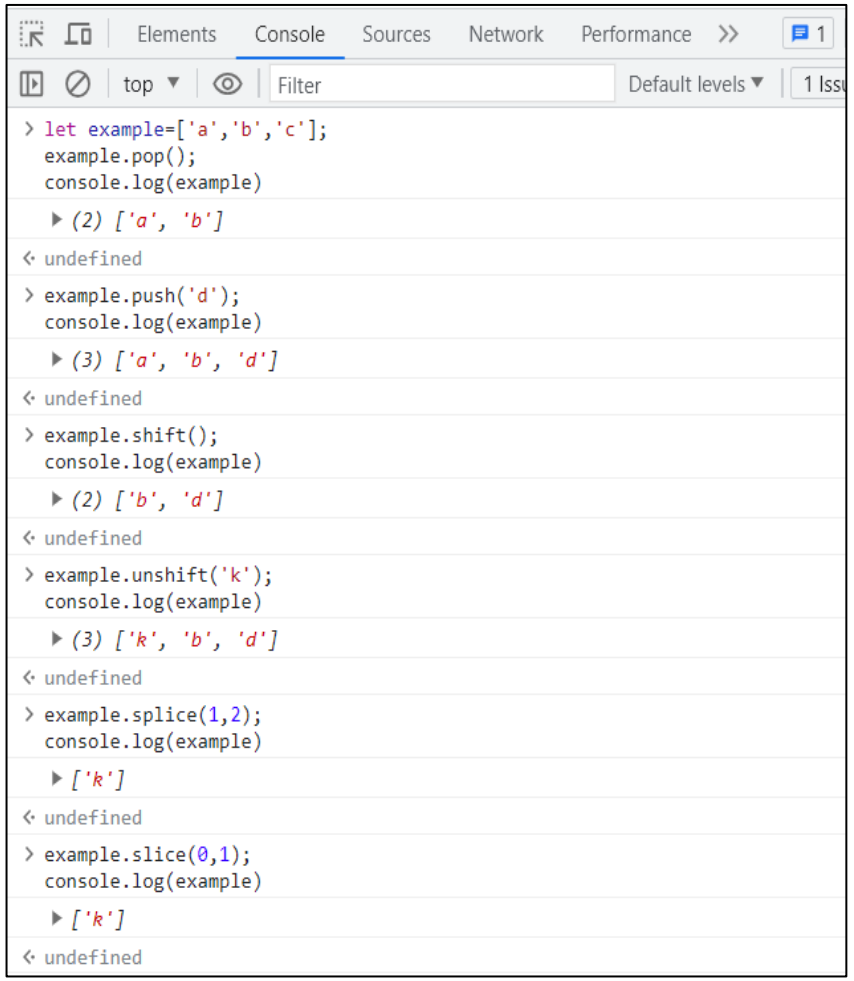
➢ 데이터 타입 : 배열(Array)의 property 및 함수(Method)

- Property : 데이터 타입 마다 가지고 있는 고유한 속성들
  -> Array.length : 배열 내의 요소 개수를 나타내는 property
- Method : 기능을 가지고 있는 명령어

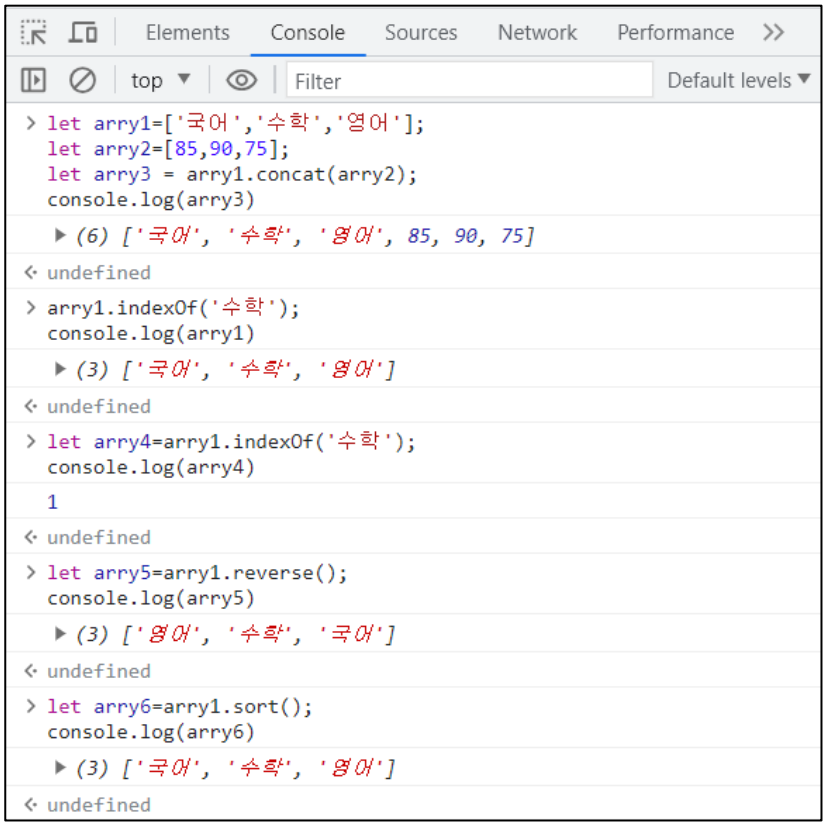
함수	설명
pop()	배열의 맨 마지막 요소를 반환 및 제거합니다
push(new data)	배열의 맨 마지막 인덱스에 새로운 데이터를 추가합니다
shift()	배열의 맨 앞 에있는 요소를 반환 및 제거합니다.
unshift(new data)	배열의 맨 앞에 인덱스에 새로운 데이터를 추가합니다.
join(연결할 문자)	배열 객체의 요소들을 인자로 넘긴 문자로 연결한 새로운 데이터를 반환합니다.
reverse()	배열 객체의 데이터의 순서를 바꾼 새로운 배열을 반환합니다.
sort()	배열 객체 데이터들을 오름차순으로 정렬합니다.
slice(start Index, end Index)	인수로 넘긴 인덱스 구간만큼 잘라서 새로운 배열 객체를 만들어 반환합니다.
splice()	배열 객체의 지정한 데이터를 삭제하고 새로운 데이터를 삽입합니다.
concat()	2개의 배열객체를 하나로 합친 배열 객체를 반환합니다.
indexof()	주어진 값과 일치하는 값이 있는 배열요소의 첫 인덱스를 찾는다.
Includes()	배열에 특정 값이 존재하면 true, 아니면 false를 반환합니다.

JavaScript

➤ 데이터 타입 : 배열(Array)의 함수(Method)



```
> let example=['a','b','c'];
  example.pop();
  console.log(example)
  ▶ (2) ['a', 'b']
< undefined
> example.push('d');
  console.log(example)
  ▶ (3) ['a', 'b', 'd']
< undefined
> example.shift();
  console.log(example)
  ▶ (2) ['b', 'd']
< undefined
> example.unshift('k');
  console.log(example)
  ▶ (3) ['k', 'b', 'd']
< undefined
> example.splice(1,2);
  console.log(example)
  ▶ ['k']
< undefined
> example.slice(0,1);
  console.log(example)
  ▶ ['k']
< undefined
```

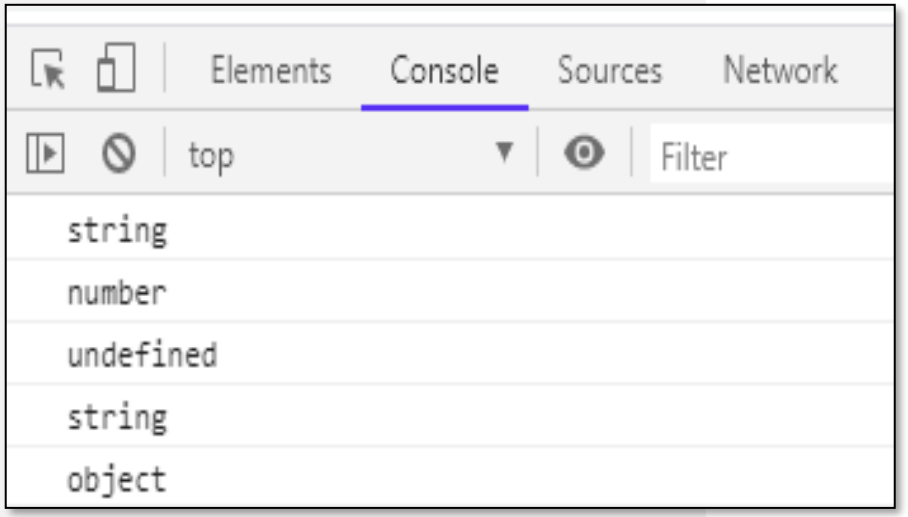


```
> let arry1=['국어','수학','영어'];
  let arry2=[85,90,75];
  let arry3 = arry1.concat(arry2);
  console.log(arry3)
  ▶ (6) ['국어', '수학', '영어', 85, 90, 75]
< undefined
> arry1.indexOf('수학');
  console.log(arry1)
  ▶ (3) ['국어', '수학', '영어']
< undefined
> let arry4=arry1.indexOf('수학');
  console.log(arry4)
  1
< undefined
> let arry5=arry1.reverse();
  console.log(arry5)
  ▶ (3) ['영어', '수학', '국어']
< undefined
> let arry6=arry1.sort();
  console.log(arry6)
  ▶ (3) ['국어', '수학', '영어']
< undefined
```

JavaScript

➢ 데이터 타입 : typeof 연산자(자료형 확인)

```
<script>
   var fruit = "apple";
   var birthyear = 1990;
   var age;
   var k = "";
   var g = null;
   console.log(typeof fruit);
   console.log(typeof birthyear);
   console.log(typeof age);
   console.log(typeof k);
   console.log(typeof g);
</script>
```



null이란 object 타입이며, 아직 '값'이 정해지지 않은 것을 의미한다.
Undefined 는 자료형이 정의되지 않았을 때의 상태이다.

자바스크립트는 자료형은 가진다. 다만, 타입을 먼저 선언하고 타입에 맞는 데이터를 할당하지 않는다.

JavaScript

➢ 데이터 타입 : 객체(Object) 종류

- 자바스크립트는 객체 기반 언어
  자바스크립트는 객체 지향 언어 아님
- 자바스크립트 객체의 유형

1. 코어 객체(내장 객체)
   자바스크립트 언어가 실행되는 어디서나 사용 가능한 기본 객체
   기본 객체로 표준 객체
   Array, Date, String, Math 타입 등
   웹 페이지 자바스크립트 코드에서 혹은 서버에서 사용 가능
2. HTML DOM 객체
   HTML 문서에 작성된 각 HTML 태그들을 객체화 한 것들
   HTML 문서의 내용과 모양을 제어하기 위한 목적
   W3C의 표준 객체
   Document Object Model 의 약어이다.
3. 브라우저 객체
   자바스크립트로 브라우저를 제어하기 위해 제공되는 객체
   BOM(Brower Object Model)에 따르는 객체들
   비표준 객체

JavaScript

➢ 프로토타입(Prototype) 이란?

- 객체의 모양을 가진 틀
- 붕어빵은 객체이고, 붕어빵을 찍어내는 틀은 프로토타입
- C++, Java에서는 프로토타입을 클래스라고 부름
- Array, Date, String : 자바스크립트에서 제공하는 프로토타입
- 객체 생성시 'new 프로토타입' 이용

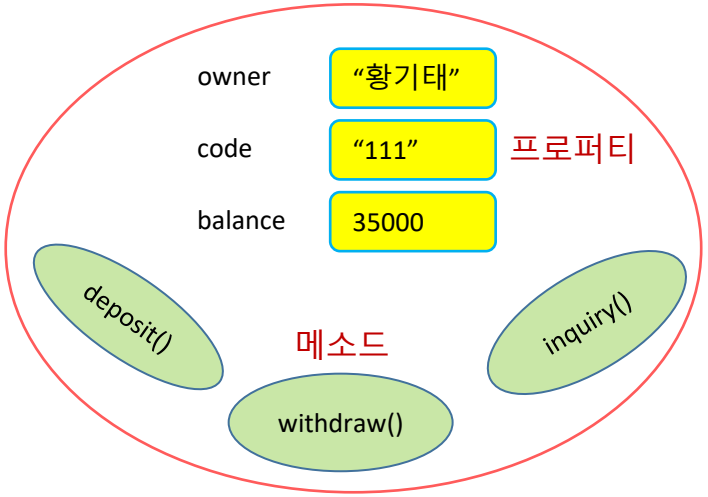
```
var week = new Array(7); // Array는 프로토타입임
var hello = new String("hello"); // String은 프로토타입임
```

- 웹 문서에 있는 요소를 프로그램에서 사용하려면 객체 형태여야 한다. 예를 들어 웹 문서에 있는 이미지를 자바스크립트에서 다룰 때는 image 객체를 사용한다.
- Image 객체는 모든 웹 이미지가 공통으로 가지는 속성과 기능을 모아 놓은 것이다. 웹 이미지를 만들기 위한 기본 틀이다. 이런 틀을 프로토타입(prototype)이라고 부른다.
- 만약 웹 문서에 3개의 이미지를 포함 시켜야 한다면 image 객체를 사용해서 똑같은 모양의 객체3개를 찍어 낸 다음 객체마다 원하는 이미지를 담으면 된다. 이렇게 프로토타입(prototype)을 사용해 만들어낸 객체는 '인스턴스(instance)'라고 한다. 새 인스턴스는 모두 image 객체의 속성과 함수를 그대로 사용한다.

JavaScript

➢ 객체(Object) 구성

- 여러 개의 프로퍼티(property)와 메소드로 구성
  - 프로퍼티 : 객체의 고유한 속성(변수)
  - 메소드(method) : 함수



자바스크립트 객체 account

```
var account = {
   owner       : "황기태",
   code        : "111",
   balance     : 35000,
   deposit     : function() { … },
   withdraw    : function() { … },
   inquiry     : function() { … }
};
```

account 객체를 만드는 자바스크립트 코드

JavaScript

➢ 객체(Object) 구성

- 객체(object)란 실생활에서 우리가 인식할 수 있는 사물로 이해할 수 있습니다.
- 하나의 변수에 다양한 정보를 담기 위해 사용하는 자료형이 객체이다.
- 객체는 여러 프로퍼티(property)나 메소드(method)를 같은 이름으로 묶어놓은 일종의 집합체입니다.

[객체구조]

Var 변수이름 = { 키(key) : 값(value) .....}

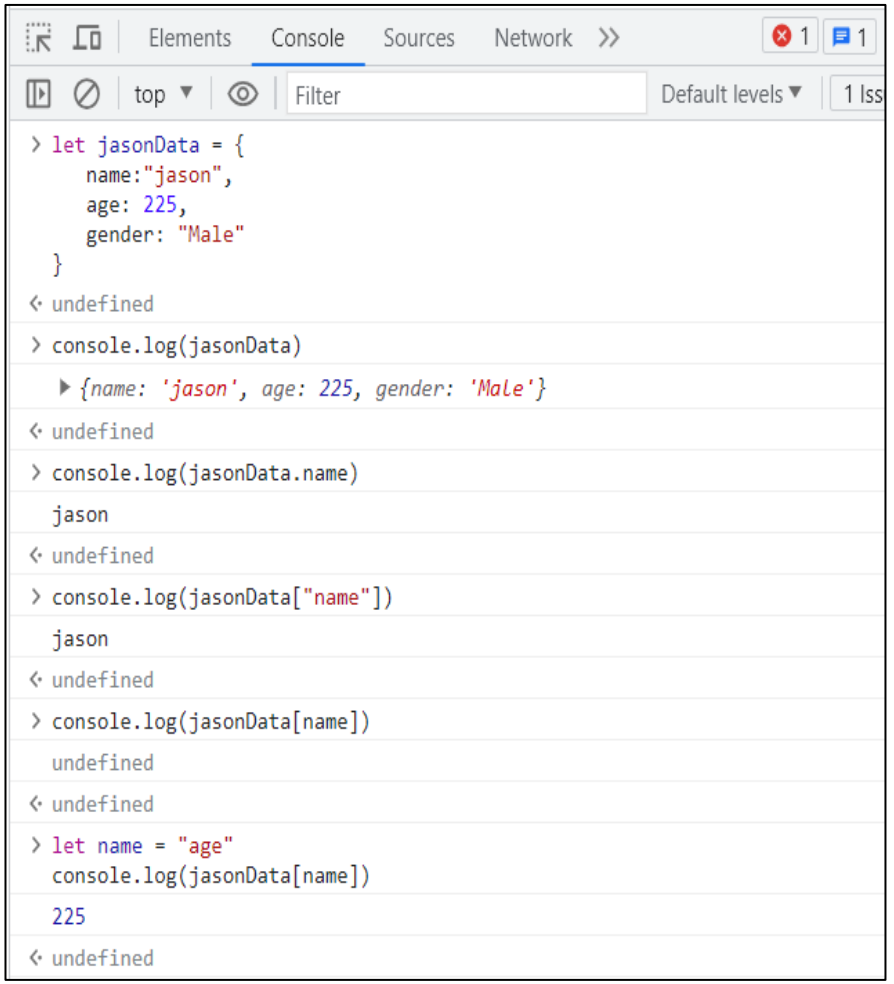
```
var kim ={
    firstname : “john”,
    lastname : “kim”,
    age : 35,
    address : “seoul”,
    list :  information( )
}
```

프로퍼티(property)

메소드(method)

JavaScript

➤ 객체(Object) 구성



JavaScript

➢ 데이터 타입 : 객체(Object)의 함수(Method)

- **Object.assign()**
하나 이상의 원본 객체들로부터 모든 열거 가능한 속성들을 대상 **객체로 복사**합니다.

- **Object.keys()**
객체의 **키만 담은 배열을 반환**합니다.

- **Object.values()**
객체의 **값만 담은 배열을 반환**합니다.

- **Object.entries()**
[키, 값] 쌍을 담은 배열을 반환합니다.

- **Object.is()**
두 값이 같은지를 비교합니다.

- **toString() 메소드**
이 메소드를 호출한 **객체의 값을 문자열로 반환**합니다.

JavaScript

➤ 데이터 타입 : 객체(Object)의 함수(Method)

Elements Console Sources Network Pe

top Filter

```
> var user={
    name:"Tom",
    age: 30
  }
  var userCopy=user
  console.log(userCopy)
  ▶ {name: 'Tom', age: 30}
undefined
> var userCopy2 = Object.assign({},user)
  console.log(userCopy2)
  ▶ {name: 'Tom', age: 30}
undefined
> var userCopy3= Object.keys
  console.log(userCopy2)
  ▶ {name: 'Tom', age: 30}
undefined
> userCopy3=Object.keys(user)
  console.log(userCopy3)
  ▶ (2) ['name', 'age']
undefined
> var userCopy4=Object.values(user)
  console.log(userCopy4)
  ▶ (2) ['Tom', 30]
```

```
> var userCopy5=Object.entries(user)
  console.log(userCopy5)
  ▼ (2) [Array(2), Array(2)]
    ▶ 0: (2) ['name', 'Tom']
    ▶ 1: (2) ['age', 30]
      length: 2
    ▶ [[Prototype]]: Array(0)
undefined
> var userCopy6=Object.is('Tom',30)
  console.log(userCopy6)
  false
undefined
> var userCopy7=Object.toString(user)
  console.log(userCopy7)
  function Object() { [native code] }
undefined
> var userCopy7=user.toString()
  console.log(userCopy7)
  [object Object]
undefined
> console.log(userCopy3.includes("name")
Uncaught SyntaxError: missing ) after argument list
> console.log(userCopy3.includes("name"))
  true
undefined
```

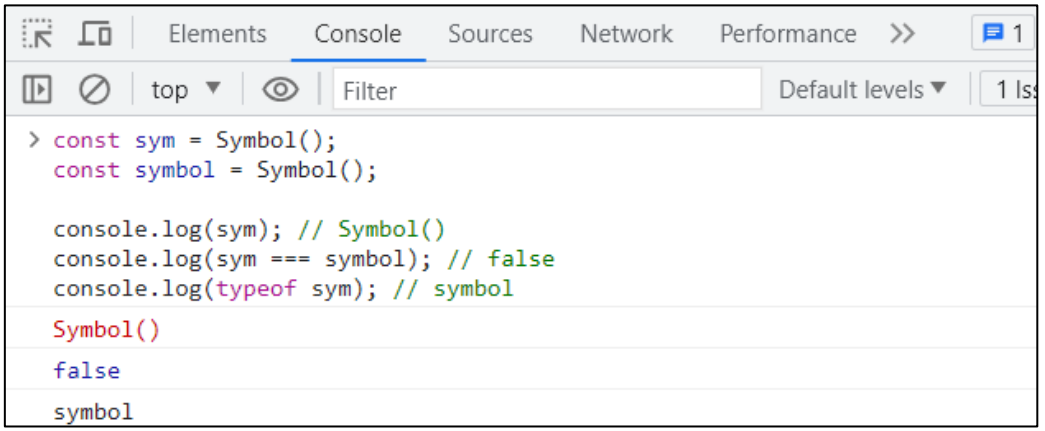
JavaScript

➢ 데이터 타입 : 심볼(Symbol)이란 ?

심볼이란 ES6에서 새롭게 추가된 원시 타입 중 하나로써, 객체 프로퍼티에 대한 식별자로 사용된다. 모든 심볼 값은 고유하고 중복이 되지 않으며 한번 생성되면 그 값은 변경되지 않기 때문에 충돌 위험이 없는 타입이다. 심볼은 new 연산자를 사용한 문법을 지원하지 않기 때문에 생성자 측면에선 불완전한 내장 객체 클래스라고 볼 수 있다.

➢ 심볼(Symbol) 생성

심볼은 Symbol() 함수로 생성 할 수 있으며 이 함수로 생성된 심볼의 타입은 객체가 아닌 Symbol 이다. 또한 변경 불가능하고 고유한 원시 타입 답게 이 함수는 호출이 될 때마다 새로운 값을 생성한다.



JavaScript

➢ 심볼(Symbol) 생성

Symbol() 함수는 인자에 문자열을 전달할 수 있으나 이 문자열은 해당 심볼에 대한 설명을 담기 위한 주석인 description 속성만을 의미할 뿐 심볼 자체에는 아무 영향을 주지 않는다. 따라서 같은 문자열을 전달하여 심볼 두 개를 생성하더라도 해당 심볼들은 서로 다른 값을 담고 있다.

```
> const sym = Symbol("moon");
  const symbol = Symbol("moon");

  console.log(sym.description); // "moon"
  console.log(symbol.description); // "moon"

  console.log(sym === symbol); // false
  moon
  moon
  false
```

```
> let jason = {};
  let sym = Symbol("moon");
  let sym=Symbol("earth");
  console.log(sym)
  Uncaught SyntaxError: Identifier 'sym' has already been declared
```

JavaScript

➢ 심볼(Symbol) 활용 – 고유한 프로퍼티 키 생성

객체의 프로퍼티 키를 생성할 때 심볼을 사용하면 다른 어떠한 프로퍼티와도 충돌하지 않는 키를 생성할 수 있다.

```
> const obj = {};
  const sym = Symbol("moon");

  obj[sym] = "moon";

  console.log(obj); // { [Symbol(moon)]: "moon"}
  console.log(obj[sym]); // "moon"
```

```
▼ {Symbol(moon): 'moon'} i
    Symbol(moon): "moon"
  ▶ [[Prototype]]: Object
```

```
moon
```

JavaScript

➢ 심볼(Symbol) 활용 – 고유한 프로퍼티 키 생성

심볼로 만들어진 키는 Object.getOwnPropertySymbols( )를 통해서만 가져올 수 있다.

```
> const jason = {};
  const sym = Symbol("moon");

  jason[sym] = "moon";
  jason["str"] = "string";
  jason.hello = "hello";

  console.log(Object.getOwnPropertySymbols(jason));
▼ [Symbol(moon)]
    0: Symbol(moon)
    length: 1
  ▶ [[Prototype]]: Array(0)
< undefined
> console.log(jason)
▼ {str: 'string', hello: 'hello', Symbol(moon): 'moon'}
    hello: "hello"
    str: "string"
    Symbol(moon): "moon"
  ▶ [[Prototype]]: Object
< undefined
```