자바스크립트 자료형

자료형 (Data Types)

1. 기본형 (Primitive Types)
   기본형 데이터는 값 자체가 변수에 저장됩니다. 이들은 변경 불가능(immutable)하며, 다음과 같은 타입들이 있습니다.

타입	설명	예제
Number	모든 정수 및 부동 소수점 숫자를 포함합니다. JavaScript에서 가장 널리 사용되는 숫자 타입으로, 매우 큰 정수는 제외됩니다.	100, 3.14
BigInt	매우 큰 정수를 다룰 때 사용합니다. 일반적인 Number 타입으로 표현할 수 없는 크기의 정수를 나타내며, 값 끝에 'n'을 붙여 표기합니다.	9007199254740991n
String	텍스트 데이터를 저장하는 데 사용됩니다. 문자열은 큰따옴표, 작은따옴표, 또는 백틱으로 둘러싸여 표현됩니다.	'Hello, World!'
Boolean	논리적 값인 'true'와 'false'를 나타냅니다. 주로 조건문이나 논리 연산에서 사용됩니다.	true, false
Symbol	유일하고 변경 불가능한 기본값으로, 주로 객체의 고유한 속성 키로 사용됩니다. 충돌을 방지하는데 유용합니다.	Symbol('id')
Undefined	변수가 선언되었으나 아직 값이 할당되지 않았음을 나타냅니다.	let a;
Null	변수에 값이 의도적으로 '없음' 또는 '무효'임을 나타내는 값입니다. 개발자가 명시적으로 할당합니다.	let a = null;

2. 참조형 (Reference Types)
   참조형 데이터는 객체의 메모리 주소를 참조하는 데이터 타입을 말합니다. 이는 변수가 실제 값이 아닌 메모리에 저장된 객체의 위치를 참조한다는 의미입니다. 참조형 데이터는 복잡한 데이터 구조를 효율적으로 다루기 위해 사용되며, 변수 간에 객체의 주소가 공유될 수 있습니다.

타입	설명	예제
Object	여러 속성(키-값 쌍)을 포함할 수 있는 구조로, 일반 객체, 배열, 함수 등이 포함됩니다.	`let person = { name: 'Alice', age: 30 };`
Array	순서가 있는 데이터 모음으로, 각 항목에 인덱스를 통해 접근할 수 있으며, 서로 다른 타입의 데이터를 포함할 수 있습니다.	`let numbers = [1, 2, 3, 4, 5];`
Function	실행 가능한 코드 블록으로, 호출될 때 특정 작업을 수행합니다. 함수는 일급 객체로, 다른 객체와 마찬가지로 속성과 메서드를 가질 수 있습니다.	`function greet() { console.log('Hello'); }`
Date	날짜와 시간을 표현하는 데 사용되며, `Date` 객체는 시간에 관련된 여러 메서드를 제공합니다.	`let currentDate = new Date();`
RegExp	정규 표현식을 나타내며, 문자열에 대한 패턴 매칭 및 검색-치환 작업을 수행합니다.	`let pattern = /\d+/g;`

3. 복합형 (Composite Types)
   복합형은 데이터와 기능이 결합된 복잡한 데이터 구조를 나타냅니다. 참조형 데이터의 한 분류로, 여러 값이나 서로 다른 타입의 데이터를 단일 구조로 그룹화할 수 있습니다.

타입	설명	예제
Object	데이터와 메서드의 집합으로서의 객체입니다. 인스턴스와 기능을 포함합니다.	`let person = { name: 'Alice', greet() { console.log('Hello'); } };`
Array	다양한 데이터 타입을 포함할 수 있는 순서가 있는 컬렉션입니다.	`let mixed = [1, 'Hello', { a: 1 }, [1, 2, 3]];`
Function	호출되었을 때 특정 작업을 수행하는 코드입니다. 종종 객체의 메서드로 사용됩니다.	`let add = (x, y) => x + y;`

문자열 (string)

문자열(String)은 문자의 시퀀스(순서가 있는 배열)를 나타내는 데이터 타입입니다. JavaScript에서 문자열은 작은따옴표(''), 큰따옴표(""), 또는 백틱(``)으로 둘러싸여 표현됩니다.

백틱(``)을 사용하는 경우에는 템플릿 리터럴(template literal)을 사용하여 문자열 내에서 변수나 표현식을 삽입할 수 있습니다.

```
let coffee = "커피";
let chocoBread = '"초코"빵';
let set = `${coffee}와 ${chocoBread} 세트`;
console.log(set);
```

문자열은 문자의 시퀀스이므로 각 문자는 문자열의 인덱스를 사용하여 접근할 수 있습니다. 예를 들어, 문자열에서 특정 위치에 있는 문자를 접근하려면 다음과 같이 인덱스를 사용합니다.

```
let str = 'Hello';
console.log(str[0]); // 'H'를 출력합니다.
console.log(str[1]); // 'e'를 출력합니다.
```

문자열은 변경 불가능한(immutable) 데이터 타입이기 때문에 한 번 생성되면 내용을 변경할 수 없습니다. 하지만 문자열 연산을 통해 새로운 문자열을 생성할 수는 있습니다.

```
let str1 = 'Hello';
let str2 = 'World';
let combined = str1 + ' ' + str2; // 'Hello World'
```

타입 체크 (typeof)

JavaScript에서는 typeof 연산자를 사용하여 특정 변수 또는 값의 데이터 타입을 확인할 수 있습니다.

```
// 타입 체크
let coffee = 100;              // 변수에 숫자형 값 할당
console.log(typeof(coffee));  // 결과: 'number'

coffee = "아메리카노";            // 변수 값을 문자열로 변경
console.log(typeof(coffee));  // 결과: 'string'

// 타입 변환
let donut = "20";              // 변수에 문자열 값 할당
console.log(typeof donut);    // 결과: 'string'

let choco = Number(donut);    // 문자열 'donut'을 숫자형으로 변환하여 'choco'에 할
당
console.log(typeof choco);    // 결과: 'number'
```

객체 (object)

JavaScript에서 객체(Object)는 중요한 데이터 유형 중 하나로, 키-값 쌍의 컬렉션입니다. 객체는 다양한 속성(Properties)과 메서드(Methods)를 가질 수 있으며, 다음과 같은 특징을 가지고 있습니다.

1. 객체는 속성을 저장하는데, 각 속성은 키(key)와 값(value)으로 이루어져 있습니다. 키는 문자열이나 심볼(Symbol)이고, 값은 어떠한 JavaScript 데이터 타입도 될 수 있습니다.
2. 객체는 참조 타입(Reference Type)입니다. 이는 객체를 변수에 할당할 때 실제 데이터가 아닌 메모리 주소를 참조한다는 의미입니다.
3. 객체는 리터럴 문법({}), Object 생성자 함수, Object.create() 메서드 등 다양한 방법으로 생성할 수 있습니다.
4. 객체의 속성은 언제든지 추가, 삭제, 수정할 수 있습니다. 이는 JavaScript가 동적인 언어라는 것을 반영합니다.
5. 객체의 속성 중에서 함수인 경우 이를 메서드라고 부릅니다. 메서드를 통해 객체의 데이터에 접근하고 조작할 수 있습니다.

객체 종류	설명	예시
일반 객체	가장 기본적인 객체 형태, 키-값 쌍을 포함	`{name: 'Alice', age: 25}`
배열	순서가 있는 값의 집합	`['apple', 'banana', 'cherry']`
함수	실행 가능한 코드 블록을 포함하는 객체	`function greet(name) { return 'Hello ' + name; }`

객체 종류	설명	예시
날짜 (Date)	날짜와 시간을 나타냄	`new Date()`
정규 표현식 (RegExp)	문자열에서 패턴 검색 및 처리를 위해 사용	`/[a-zA-Z]+/g`
에러 (Error)	실행 중에 발생하는 오류를 나타냄	`new Error('Something went wrong')`
내장 객체	자바스크립트에 내장된 객체, 고유한 기능과 속성을 가짐	`Math.random()`, `Number.parseInt('10')`
전역 객체	코드 어디에서나 접근할 수 있는 객체	`globalThis` (브라우저), `globalThis` (Node.js)

```
// 일반 객체
const person = { name: 'Alice', age: 25, isOpen: true };
person.house = '서울';
person.isOpen = false;    // 속성 값 변경
console.log(person);      // 객체의 전체 데이터 확인
console.log(person.name); // Alice

// 배열
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits[1]); // banana

// 함수
function greet(name) {
  return 'Hello ' + name;
}
console.log(greet('Alice')); // Hello Alice

// 날짜
const studyDay = new Date();
const studyString = studyDay.toISOString().split('T')[0];
console.log(studyString);
console.log(studyDay);
console.log(studyDay.toString());
console.log(studyDay.getFullYear(), studyDay.getMonth(),
studyDay.getDate())

// 정규표현식
const pattern = /[a-zA-Z]+/g;
const result = pattern.test('Hello');
console.log(result); // true

// 에러
try {
  throw new Error('Something went wrong');
} catch (e) {
  console.error(e.message); // Something went wrong
}

// 내장 객체
```

```
console.log(Math.random()); // 0과 1 사이의 랜덤한 숫자
console.log(Number.parseInt('10', 10)); // 10

// 전역 객체
console.log(globalThis.location.href); // 브라우저 환경
console.log(globalThis.process.version); // Node.js 환경
```

- 콘솔에서 찍어본 일반 객체

```
> const person = { name: 'Alice', age: 25, isOpen: true };
< undefined
> person
< ▼ {name: 'Alice', age: 25, isOpen: true} i
    age: 25
    isOpen: true
    name: "Alice"
  ▶ [[Prototype]]: Object
> person.house = '서울';
< '서울'
> person
< ▼ {name: 'Alice', age: 25, isOpen: true, house: '서울'} i
    age: 25
    house: "서울"
    isOpen: true
    name: "Alice"
  ▶ [[Prototype]]: Object
```

객체 메서드 종류

객체 메서드	설명
`Object.entries()`	객체의 [키, 값]을 담은 배열을 반환합니다.
`Object.keys()`	객체의 키만 담은 배열을 반환합니다.
`Object.values()`	객체의 값만 담은 배열을 반환합니다.
`Object.freeze()`	객체를 동결하여 수정할 수 없도록 만듭니다.
`Object.seal()`	객체를 밀봉하여 새 속성 추가는 금지하지만, 기존 속성의 수정은 허용합니다.

객체 메서드	설명
`Object.is()`	두 값이 같은지를 확인합니다. NaN을 NaN과 같은 값으로 간주하지 않습니다.
`Object.assign()`	하나 이상의 소스 객체로부터 대상 객체로 속성을 복사합니다.

```
// 빵 가게 정보 객체
const bakery = {
  saltBread: "소금빵",
  chocoBread: "초코빵",
  cheeseBread: "치즈빵"
};

// Object.entries()
const bakeryEntries = Object.entries(bakery);
console.log("Object.entries():", bakeryEntries);
// 출력: Object.entries(): [ [ 'saltBread', '소금빵' ], [ 'chocoBread', '초코
빵' ], [ 'cheeseBread', '치즈빵' ] ]

// Object.keys()
const bakeryKeys = Object.keys(bakery);
console.log("Object.keys():", bakeryKeys);
// 출력: Object.keys(): [ 'saltBread', 'chocoBread', 'cheeseBread' ]

// Object.values()
const bakeryValues = Object.values(bakery);
console.log("Object.values():", bakeryValues);
// 출력: Object.values(): [ '소금빵', '초코빵', '치즈빵' ]

// Object.freeze()
Object.freeze(bakery);
bakery.cheeseBread = "Baguettes"; // 변경이 적용되지 않음
console.log("Object.freeze():", bakery);
// 출력: Object.freeze(): { saltBread: '소금빵', chocoBread: '초코빵',
cheeseBread: '치즈빵' }

// Object.seal()
Object.seal(bakery);
bakery.chocoBread = "456 Bread Avenue"; // 변경이 허용됨
console.log("Object.seal():", bakery);
// 출력: Object.seal(): { saltBread: '소금빵', chocoBread: '456 Bread
Avenue', cheeseBread: '치즈빵' }

// Object.is()
console.log("Object.is(10, 10):", Object.is(10, 10));     // true
console.log("Object.is(10, '10'):", Object.is(10, '10')); // false

// Object.assign()
const target = { a: 1, b: 2 };
const source = { b: 3, c: 4 };
const mergedObject = Object.assign({}, target, source);
```

```
console.log("Object.assign():", mergedObject);
// 출력: Object.assign(): { a: 1, b: 3, c: 4 }
```

배열 (array)

배열(Array)은 JavaScript에서 가장 일반적으로 사용되는 데이터 구조 중 하나입니다. 배열은 순서가 있는 요소들의 모음으로, 각 요소는 인덱스(일반적으로 0부터 시작하는 정수)로 식별됩니다.

JavaScript 배열은 여러 가지 데이터 유형을 포함할 수 있으며, 크기가 동적으로 조정될 수 있습니다. 즉, 배열에 요소를 추가하거나 제거할 수 있습니다.

배열은 대괄호 `[ ]` 를 사용하여 생성되며, 각 요소는 쉼표로 구분됩니다

```
// 빈 배열 생성
const breads = [];

// 배열에 요소 추가
breads.push("소금빵", "초코빵", "치즈빵");
console.log(breads.length); // 3

// 원하는 인덱스에 새 요소 추가
breads[5] = "식빵";
console.log(breads[5]); // '식빵'
console.log(Object.keys(breads)); // ['0', '1', '2', '5']
console.log(breads.length); // 6

// 빈 슬롯 추가로 배열 확장
breads.length = 10;
console.log(breads); // ['소금빵', '초코빵', '치즈빵', empty x 2, '식빵', empty x
4]
console.log(Object.keys(breads)); // ['0', '1', '2', '5']
console.log(breads.length); // 10
console.log(breads[8]); // undefined

// 길이를 줄이면 요소가 삭제됨
breads.length = 2;
console.log(Object.keys(breads)); // ['0', '1']
console.log(breads.length); // 2
console.log(breads); // ['소금빵', '초코빵']
```

- 콘솔에서 찍어본 배열 결과

```
> breads[5] = "식빵"
< '식빵'
> Object.keys(breads)
< ▼ (4) ['0', '1', '2', '5'] i
    0: "0"
    1: "1"
    2: "2"
    3: "5"
    length: 4
  ▶ [[Prototype]]: Array(0)
> breads
< ▼ (6) ['소금빵', '초코빵', '치즈빵', 비어 있음 × 2, '식빵'] i
    0: "소금빵"
    1: "초코빵"
    2: "치즈빵"
    5: "식빵"
    length: 6
  ▶ [[Prototype]]: Array(0)
```

배열 메서드 (Array prototype method)

- 주로 많이 사용되는 array 메서드

메서드	설명
concat()	두 개 이상의 배열을 병합하는 데 사용, 새 배열을 반환
filter()	주어진 배열의 일부에 대한 얕은 복사본을 생성하고, 주어진 함수에 의해 구현된 테스트를 통과한 요소로만 필터링
map()	배열 내의 모든 요소 각각에 대하여 주어진 함수를 호출한 결과를 모아 새로운 배열을 반환
slice()	어떤 배열의 `begin` 부터 `end` 까지(`end` 미포함)에 대한 얕은 복사본을 새로운 배열 객체로 반환
sort()	배열의 요소를 적절한 위치에 정렬한 후 그 배열을 반환
splice()	배열의 기존 요소를 삭제 또는 교체하거나 새 요소를 추가하여 배열의 내용을 변경
forEach()	각 배열 요소에 대해 제공된 함수를 한 번씩 실행
find()	제공된 테스트 함수를 만족하는 첫 번째 요소를 반환
join()	배열의 모든 요소를 쉼표나 지정된 구분 문자열로 구분하여 연결한 새 문자열을 만들어 반환
includes()	배열의 항목에 특정 값이 포함되어 있는지를 판단하여 적절히 `true` 또는 `false`를 반환

메서드	설명
reduce()	배열의 각 요소에 대해 함수를 실행하고 단일 출력 값을 반환
push()	배열의 끝에 하나 이상의 요소를 추가하고, 배열의 새 길이를 반환
pop()	배열의 마지막 요소를 제거하고 그 요소를 반환
flatMap()	배열의 각 요소에 주어진 콜백 함수를 적용한 다음 그 결과를 한 단계씩 평탄화하여 형성된 새 배열을 반환
indexOf()	배열에서 주어진 요소를 찾을 수 있는 첫 번째 인덱스를 반환, 없으면 -1을 반환

- 그 외 array 메서드

메서드	설명
copyWithin()	배열의 일부를 같은 배열의 다른 위치로 얕게 복사하며, 배열의 길이를 수정하지 않고 해당 배열을 반환
every()	배열의 모든 요소가 제공된 함수로 구현된 테스트를 통과하는지 테스트합니다. 이 메서드는 불리언 값을 반환
flat()	모든 하위 배열 요소가 지정된 깊이까지 재귀적으로 연결된 새 배열을 생성
shift()	배열의 첫 번째 요소를 제거하고, 그 요소를 반환
unshift()	배열의 시작에 하나 이상의 요소를 추가하고, 새 길이를 반환
lastIndexOf()	배열에서 주어진 값과 일치하는 마지막 요소의 인덱스를 반환, 없으면 -1을 반환
reduceRight()	배열의 각 요소에 대해 함수를 오른쪽에서 왼쪽으로 적용하고 단일 출력 값을 반환
fill()	배열의 지정된 범위를 정해진 값으로 채움
from()	유사 배열 또는 반복 가능한 객체를 얕은 복사하여 새 배열 객체 생성

```
// concat()
const breads = ['바게트', '치아바타'];
const pastries = ['크루아상', '데니쉬'];
const menuItems = breads.concat(pastries);
console.log(menuItems); // ['바게트', '치아바타', '크루아상', '데니쉬']

// filter()
const coffeePrices = [3.5, 4.0, 2.5, 3.0, 4.5];
const affordableCoffees = coffeePrices.filter(price => price < 4);
console.log(affordableCoffees); // [3.5, 2.5, 3.0]

// map()
const coffees = ['에스프레소', '라떼', '카푸치노'];
const menuDisplay = coffees.map(coffee => coffee.toUpperCase());
console.log(menuDisplay); // ['에스프레소', '라떼', '카푸치노']

// slice()
const cafeMenu = ['커피', '차', '샌드위치', '샐러드', '케이크'];
const beverageMenu = cafeMenu.slice(0, 2);
console.log(beverageMenu); // ['커피', '차']
```

```
// sort()
const pastryTypes = ['크루아상', '데니쉬', '에클레어', '스콘'];
pastryTypes.sort();
console.log(pastryTypes); // ['크루아상', '데니쉬', '에클레어', '스콘']

// splice()
const menu = ['커피', '차', '주스'];
menu.splice(2, 0, '에스프레소', '라떼');
console.log(menu); // ['커피', '차', '에스프레소', '라떼', '주스']

// forEach()
const bakeryItems = ['커피', '빵', '케이크'];
bakeryItems.forEach(item => {
  console.log(item);
});
// 커피
// 빵
// 케이크

// find()
const temperatures = [72, 68, 75, 80, 77];
const idealTemp = temperatures.find(temp => temp === 75);
console.log('커피 온도는 :' + idealTemp); // 75

// join()
const coffeeTypes = ['에스프레소', '라떼', '카푸치노'];
console.log(coffeeTypes.join(', ')); // '에스프레소, 라떼, 카푸치노'

// reduce()
const orderTotals = [4.25, 3.50, 2.75];
const total = orderTotals.reduce((total, current) => total + current, 0);
console.log(total); // 10.50

// push()
const cakes = ['초콜릿', '바닐라'];
const updatedLength = cakes.push('레드 벨벳');
console.log(cakes); // ['초콜릿', '바닐라', '레드 벨벳']
console.log(updatedLength); // 3

// pop()
const teaFlavors = ['얼그레이', '녹차', '페퍼민트', '모카라떼'];
const lastTea = teaFlavors.pop();
console.log(teaFlavors); // ['얼그레이', '녹차', '페퍼민트']
console.log(lastTea); // '모카라떼'
```

배열 구조 분해 할당

- 구조 분해 할당 : 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식

```
// 기본 예제
// 배열의 요소들을 변수에 할당합니다.
const bakery = ['소금빵', '치즈빵', '초코빵'];
const [saltBread, cheeseBread, chocoBread] = bakery;
console.log(saltBread);   // '소금빵'
console.log(cheeseBread); // '치즈빵'
console.log(chocoBread);  // '초코빵'

// 일부 요소만 선택하여 할당
// 특정 요소들만 변수에 할당하고 나머지는 무시합니다.
const numbers = [1, 2, 3, 4, 5];
const [first, , third, , fifth] = numbers;
console.log(first); // 1
console.log(third); // 3
console.log(fifth); // 5

// 기본값 설정
// 할당할 배열의 요소가 부족할 때 기본값을 사용합니다.
const [a, b, c = '버터빵'] = ['모카빵', '앙금빵'];
console.log(a); // '모카빵'
console.log(b); // '앙금빵'
console.log(c); // '버터빵' <- 기본 값

// 나머지 요소들을 새로운 배열로 할당
// 나머지 요소들을 다른 배열로 모읍니다.
const bread = ['초코빵', '크림빵', '소보로빵', '팥빵'];
const [chocoBread, creamBread, ...others] = bread;
console.log(chocoBread); // '초코빵'
console.log(creamBread); // '크림빵'
console.log(others); // ['소보로빵', '팥빵']

// 함수에서 여러 값을 반환하고 구조분해할당 사용
// 함수에서 배열을 반환하고, 구조분해할당을 사용하여 결과를 변수에 할당합니다.
function getBread() {
  return ['모카', '빵'];
}

const [firstName, lastName] = getBread();
console.log(firstName); // '모카'
console.log(lastName); // '빵'

// ...rest 연산자
// 나머지 요소들을 새로운 배열로 할당
const cafe = ['아메리카노', '카페라떼', '카푸치노', '에스프레소'];
const [americano, cafeLatte, ...others] = cafe;

console.log(americano); // '아메리카노'
console.log(cafeLatte); // '카페라떼'
console.log(others); // ['카푸치노', '에스프레소']
```

템플릿 리터럴 : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals
Function : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Function
Array : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array
RegExp : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/RegExp
Error : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Error