CHAPTER 07 표준 내장 객체

자바스크립트 프로그래밍 입문 (2판)

한빛아카데미 Hanbit Academy, Inc.

Contents

학습목표

- 기본 자료형과 객체 자료형의 차이를 이해합니다.
- 자바스크립트의 표준 내장 객체를 이해합니다.

내용

- 기본 자료형과 객체 자료형의 차이
- Number 객체
- String 객체
- Date 객체
- Array 객체
- 조금 더 나아가기

1. 기본 자료형과 객체 자료형의 차이

- 자바스크립트는 다양한 객체를 제공

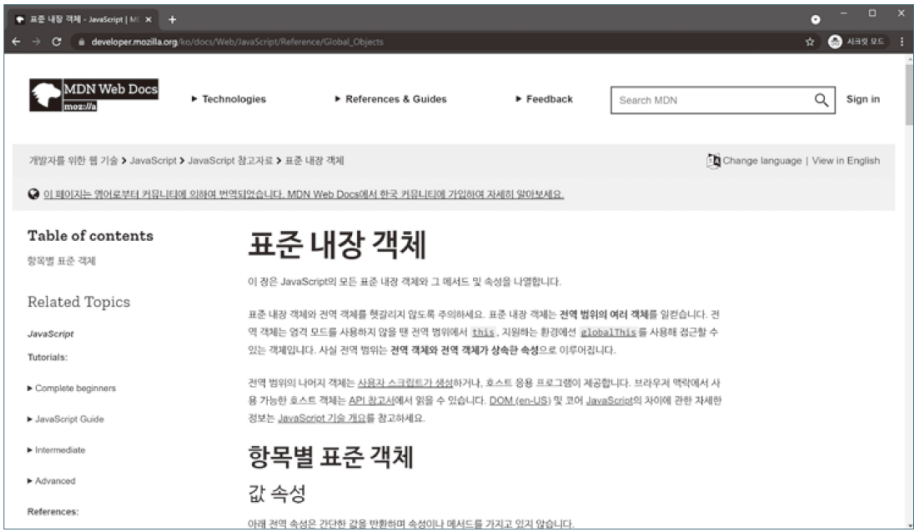

그림 7–1 모질라 레퍼런스 – 표준 내장 객체

1. 기본 자료형과 객체 자료형의 차이

- 통합 개발 환경에서 자동 완성 기능

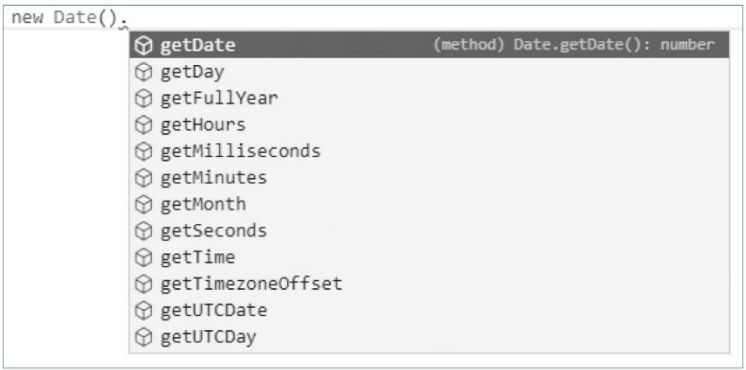

그림 7-2 통합 개발 환경에서 자동 완성 기능

1. 기본 자료형과 객체 자료형의 차이

- 기본 자료형 숫자, 문자열, 불

코드 7-1 기본 자료형 숫자, 문자열, 불 primitiveType.js

```
// 기본 자료형
let number = 273;
let string = '안녕하세요';
let boolean = true;

// 자료형을 출력합니다.
console.log(typeof number);
console.log(typeof string);
console.log(typeof boolean);
```

실행 결과

```
number
string
boolean
```

1. 기본 자료형과 객체 자료형의 차이

- 객체 숫자, 문자열, 불

코드 7-2 객체 숫자, 문자열, 불 boxedType.js

```
// 객체 자료형
let number = new Number(273);
let string = new String('안녕하세요');
let boolean = new Boolean(true);

// 자료형을 출력합니다.
console.log(typeof number);
console.log(typeof string);
console.log(typeof boolean);
```

실행 결과

```
object
object
object
```

1. 기본 자료형과 객체 자료형의 차이

- 기본 자료형의 속성 또는 메소드를 사용할 때 기본 자료형이 자동으로
- 객체로 변환이 됨. 즉, 기본 자료형과 객체 자료형 모두 속성과 메소드를 사용함

코드 7-3 기본 자료형과 객체 자료형의 메소드 사용 — primitivesMethod.js

```
let string = '과자¦1500원';
console.log(string.split('¦'));
```

```
let string = new String('과자¦1500원');
console.log(string.split('¦'))
```

실행 결과

```
[ '과자', '1500원' ]
```

1. 기본 자료형과 객체 자료형의 차이

- 차이점 : 기본 자료형은 객체가 아니므로 속성과 메소드를 추가할 수 없음

코드 7-4 기본 자료형에 속성 또는 메소드 추가 methodToPrimitives.js

```
// 변수를 생성합니다.
let primitiveNumber = 273;

// 메소드를 추가합니다.
primitiveNumber.method = function () {
    return 'Primitive Method';
};

// 메소드를 실행합니다.
console.log(primitiveNumber.method());
```

실행 결과

```
C:\example\methodToPrimitives.js:10
console.log(primitiveNumber.method());
                            ^

TypeError: primitiveNumber.method is not a function
    at Object.<anonymous> (C:\example\methodToPrimitives.js:10:29)
[90m    at Module._compile (internal/modules/cjs/loader.js:1085:14)[39m
[90m    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)[39m
[90m    at Module.load (internal/modules/cjs/loader.js:950:32)[39m
[90m    at Function.Module._load (internal/modules/cjs/loader.js:790:14)[39m
[90m    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:
76:12)[39m
[90m    at internal/main/run_main_module.js:17:47[39m
```

1. 기본 자료형과 객체 자료형의 차이

- 기본 자료형에 프로토타입으로 메소드 추가

코드 7-5 프로토타입에 메소드 추가 methodToObject.js

```
// 변수를 생성합니다.
let primitiveNumber = 273;

// 메소드를 추가합니다.
Number.prototype.method = function () {
    return 'Primitive Method';
};

// 메소드를 실행합니다.
console.log(primitiveNumber.method());
```

실행 결과

```
Primitive Method
```

2. Number 객체

- 자바스크립트에서 숫자를 표현할 때 사용
- Number 객체 생성

코드 7-6 숫자 생성

```
let numberFromLiteral = 273;
let numberFromConstructor = new Number(273);
```

메소드

표 7-1 Number 객체의 메소드

메소드	설명
toExponential()	숫자를 지수 표시로 나타낸 문자열을 리턴합니다.
toFixed()	숫자를 고정소수점 표시로 나타낸 문자열을 리턴합니다.
toPrecision()	숫자를 길이에 따라 지수 표시 또는 고정소수점 표시로 나타낸 문자열을 리턴합니다.

2. Number 객체

- [예제 7-1] Number 객체의 메소드
  - toFixed( ) 메소드를 사용해 소수점 자릿수를 자르는 방법

코드 7-7 Number 객체의 메소드 numberMethod.js

```
// 변수를 선언합니다.
let number = 273.5210332;

// 출력합니다.
console.log(number.toFixed(1));
console.log(number.toFixed(4));
```

실행 결과

```
273.5
273.5210
```

2. Number 객체

생성자 함수의 속성

- 생성자 함수에 속성과 메소드 추가

코드 7-8 클래스에 속성과 메소드 추가 constructor.js

```
// 클래스를 생성하고, 속성과 메소드를 추가합니다.
class Test {}
Test.property = 273;
Test.method = function () { };

// 출력합니다.
console.log(Test.property)
console.log(Test.method)
```

실행 결과

```
273
[Function (anonymous)]
```

표 7-2 Number 클래스의 속성

속성	설명
MAX_VALUE	자바스크립트의 숫자가 나타낼 수 있는 최대 숫자
MIN_VALUE	자바스크립트의 숫자가 나타낼 수 있는 최소 숫자
NaN	자바스크립트의 숫자로 나타낼 수 없는 숫자
POSITIVE_INFINITY	양의 무한대 숫자
NEGATIVE_INFINITY	음의 무한대 숫자

2. Number 객체

- [예제 7-2] Number 생성자 함수의 속성

코드 7-9 Number 클래스의 MAX_VALUE 속성 numberStaticMethod.js

```
// 변수를 선언합니다.
let numberA = Number.MAX_VALUE;
let numberB = Number.MAX_VALUE + 1;

// 출력합니다.
console.log(numberA);
console.log(numberB);
```

실행 결과

```
1.7976931348623157e+308
1.7976931348623157e+308
```

2. Number 객체

- [예제 7-2] Number 생성자 함수의 속성
  - 1을 추가했지만 결국 출력되는 값이 같음
  - 자바스크립트가 너무 큰 수를 다룰 때는 부동소수점 형식으로 숫자를 다루기 때문에 작은 수는 무시함

코드 7-10 Infinity로 변환되는 숫자

```
// 변수를 선언합니다.
let addNumber = Number('0.0000000000000001e+308');
let number = Number.MAX_VALUE + addNumber;

// 출력합니다.
console.log(number);
```

3. String 객체

- String 객체 생성

코드 7-11 String 객체 생성

```
let stringFromLiteral = '안녕하세요';
let stringFromConstructor = new String('안녕하세요');
```

속성과 메소드

표 7-3 String 객체의 속성

속성	설명
length	문자열의 길이를 나타냅니다.

- String 객체의 메소드는 변경된 값을 리턴함

3. String 객체

표 7-4 String 객체의 메소드

메소드	설명
charAt(position)	position에 위치하는 문자를 리턴합니다.
charCodeAt(position)	position에 위치하는 문자의 유니코드 번호를 리턴합니다.
concat(args)	매개 변수로 입력한 문자열을 이어 리턴합니다.
indexOf(searchString, position)	앞에서부터 일치하는 문자열의 위치를 리턴합니다.
lastIndexOf(searchString, position)	뒤에서부터 일치하는 문자열의 위치를 리턴합니다.
match(regExp)	문자열 안에 regExp가 있는지 확인합니다.
replace(regExp, replacement)	regExp를 replacement로 바꾼 후 리턴합니다.
search(regExp)	regExp와 일치하는 문자열의 위치를 리턴합니다.
slice(start, end)	특정 위치의 문자열을 추출해 리턴합니다.
split(separator, limit)	separator로 문자열을 잘라 배열을 리턴합니다.
substr(start, count)	start부터 count만큼 문자열을 잘라서 리턴합니다.
substring(start, end)	start부터 end까지 문자열을 잘라서 리턴합니다.
toLowerCase()	문자열을 소문자로 바꾸어 리턴합니다.
toUpperCase()	문자열을 대문자로 바꾸어 리턴합니다.

3. String 객체

- 잘못된 String 객체의 메소드 사용

코드 7-12 잘못된 String 객체의 메소드 사용 stringMistake.js

```
// 변수를 선언합니다.
let string = 'abcdefg';

// 출력합니다.
string.toUpperCase();
console.log(string);
```

실행 결과

```
abcdefg
```

- 자기 자신을 변경하지 않고 리턴하는 것뿐이므로 소문자 상태로 출력

3. String 객체

코드 7-13 올바른 String 객체의 메소드 사용 solveStringMistake.js

```
// 변수를 선언합니다.
let string = 'abcdefg';

// 출력합니다.
string = string.toUpperCase();
console.log(string);
```

실행 결과

```
ABCDEFG
```

3. String 객체

메소드 활용

- [예제 7-3] 문자열 포함
  - indexOf( ) 메소드 : 특정 문자열이 있는지 확인, 위치를 리턴함
  - 문자열이 포함되어 있지 않을 때는 -1을 리턴

코드 7-16 indexOf() 메소드 stringIndexOf.js

```
// 변수를 선언합니다.
let string = '안녕하세요. 좋은 아침입니다.';

// 문자열 내부에 ¨아침¨이라는 문자열이 있는지 확인합니다.
if (string.indexOf('아침') >= 0) {
    console.log('좋은 아침이에요...!');
}
```

실행 결과

```
좋은 아침이에요...!
```

3. String 객체

- [예제 7-4] 문자열 분해
  - split ( ) 메소드를 사용하면 특정한 기호를 기반으로 문자열을 분해함

코드 7-17 split() 메소드 stringSplit.js

```
// 변수를 선언합니다.
let string = '감자,고구마,바나나,사과';

// 문자열을 쉼표로 자르고 출력합니다.
let array = string.split(',');
console.log(array);
```

실행 결과

```
[ '감자', '고구마', '바나나', '사과' ]
```

4. Date 객체

표 7-5 Date 객체 생성 방법

클래스	설명
new Date()	현재 시간으로 Date 객체를 생성합니다.
new Date(유닉스_타임)	유닉스 타임(1970년 1월 1일 00시 00분 00초부터 경과한 밀리초)으로 Date 객체를 생성합니다.
new Date(시간_문자열)	문자열로 Date 객체를 생성합니다.
new Date(연, 월 - 1, 일, 시간, 분, 초, 밀리초)	시간 요소(연, 월 – 1, 일, 시간, 분, 초, 밀리초)를 기반으로 Date 객체를 생성합니다.

- Month를 나타내는 '월'은 0부터 시작,
- 0 → 1월, 11 → 12월

4. Date 객체

- [예제 7-5] Date 객체 생성

코드 7-18 Date 객체 생성 newDate.js

```
// 현재 시간을 기반으로 Date 객체를 생성합니다.
let dateA = new Date();
console.log(dateA);
// 유닉스 타임(1970년 1월 1일 00시 00분 00초부터 경과한 밀리초(ms)를 정수 형태로 나타냅니다.)
let dateB = new Date(692281800000);
console.log(dateB);

// 문자열을 기반으로 Date 객체를 생성합니다.
let dateC = new Date("December 9, 1991 21:30:00")
console.log(dateC);

// 시간 요소(연, 월 - 1, 일, 시간, 분, 초, 밀리초)를 기반으로 Date 객체를 생성합니다.
let dateD = new Date(1991, 12 - 1, 9, 21, 30, 0, 0);
console.log(dateD);
```

실행 결과

```
2021-10-05T07:31:09.877Z
1991-12-09T12:30:00.000Z
1991-12-09T12:30:00.000Z
1991-12-09T12:30:00.000Z
```

4. Date 객체

메소드 활용

- Date 객체
  - get○○( ) 형태 메소드, set○○( ) 형태 메소드 : FullYear, Month, Day, Hours, Minutes, Seconds 등 사용
  - <모질라 레퍼런스 – Date>
  - https ://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Date

4. Date 객체

- [예제 7-6] 시간 더하기
  - 현재 시간에 1년, 11개월, 7일을 더해 출력. (현재 시간 : 2016년 8월 16일)

코드 7-19 시간 더하기 dateAdd.js

```
// 현재 시간을 구합니다.
let date = new Date();

// 출력1
console.log(date);

// 시간을 더합니다.
date.setFullYear(date.getFullYear() + 1);
date.setMonth(date.getMonth() + 11);
date.setDate(date.getDate() + 3);

// 출력2
console.log(date);
```

실행 결과

```
2021-10-05T07:42:58.249Z
2023-09-08T07:42:58.249Z
```

4. Date 객체

- [예제 7-7] 시간 간격 구하기
  - getTime( ) 메소드 : 유닉스 타임
  - 2개의 시간을 빼고, 일 단위(1000밀리초 * 60초 * 60분 * 24시간)로 나누어 시간 간격을 구함

코드 7-20 시간 간격 구하기 dateInterval.js

```
// 변수를 선언합니다.
let now = new Date();
let before = new Date('December 9, 1991');

// 시간 간격을 구합니다.
let interval = now.getTime() - before.getTime();
interval = Math.floor(interval / (1000 * 60 * 60 * 24));

// 출력합니다.
console.log(`태어나고 ${interval}일 지났습니다.`)
```

실행 결과

```
태어나고 10886일 지났습니다.
```

5. Array 객체

Array 객체의 기본 메소드

- 대부분 파괴적 메소드로 자기 자신을 변경

표 7-6 Array 객체의 메소드

메소드	설명
concat()	매개 변수로 입력한 배열의 요소를 모두 합쳐 배열을 만들어 리턴합니다.
join()	배열 안의 모든 요소를 문자열로 만들어 리턴합니다.
pop()*	배열의 마지막 요소를 제거하고 리턴합니다.
push()*	배열의 마지막 부분에 새로운 요소를 추가합니다.
reverse()*	배열의 요소 순서를 뒤집습니다.
slice()	배열 요소의 지정한 부분을 리턴합니다.
sort()*	배열의 요소를 정렬합니다.
splice()*	배열 요소의 지정한 부분을 삭제하고 삭제한 요소를 리턴합니다.

* 표시된 메소드는 자기 자신을 변화시킵니다.

5. Array 객체

- [예제 7-8] 배열 가공

코드 7-21 Array 객체의 메소드 arrayMethod.js

```
// 배열을 선언합니다.
let array = [{
    name: '고구마',
    price: 1000
}, {
    name: '감자',
    price: 500
}, {
    name: '바나나',
    price: 1500
}];

// 배열의 요소를 꺼냅니다.
let popped = array.pop();
console.log('- 배열에서 꺼낸 요소');
console.log(popped);
console.log('- pop() 메소드를 호출한 이후의 배열');
console.log(array);
```

5. Array 객체

- [예제 7-8] 배열 가공

```
// 배열에 요소를 넣습니다.
array.push(popped);
array.push({
    name: '사과',
    price: 2000
});
console.log('- push() 메소드를 호출한 이후의 배열');
console.log(array);
```

5. Array 객체

실행 결과

```
- 배열에서 꺼낸 요소
{ name: '바나나', price: 1500 }
- pop() 메소드를 호출한 이후의 배열
[ { name: '고구마', price: 1000 }, { name: '감자', price: 500 } ]
- push() 메소드를 호출한 이후의 배열
[
  { name: '고구마', price: 1000 },
  { name: '감자', price: 500 },
  { name: '바나나', price: 1500 },
  { name: '사과', price: 2000 }
]
```

5. Array 객체

- [예제 7-9] 배열 정렬 - sort ( ) 메소드 : 배열 정렬

코드 7-22 sort() 메소드 arraySort.js

```
// 배열을 선언합니다.
let arrayA = ['고구마', '감자', '바나나'];
let arrayB = [{
    name: '고구마',
    price: 1000
}, {
    name: '감자',
    price: 500
}, {
    name: '바나나',
    price: 400
}];
```

5. Array 객체

- [예제 7-9] 배열 정렬 - sort ( ) 메소드 : 배열 정렬

```
// 기본 배열을 정렬하고 출력합니다.
arrayA.sort();
console.log('- 문자열로 정렬');
console.log(arrayA);
console.log();
console.log('- 문자열로 정렬(역순)');
console.log(arrayA.reverse());
console.log();
```

```
// 객체 내부의 숫자로 정렬하고 출력합니다.
arrayB.sort((itemA, itemB) => {
    return itemA.price - itemB.price;
});
console.log('- 객체 내부의 숫자로 정렬')
console.log(arrayB);
console.log();
```

```
// 객체 내부의 문자열로 정렬하고 출력합니다.
arrayB.sort((itemA, itemB) => {
    if (itemA.name < itemB.name) {
        return -1;
    } else if (itemA.name > itemB.name) {
        return 1;
    } else {
        return 0;
    }
});
console.log('- 객체 내부의 문자열로 정렬')
console.log(arrayB);
```

5. Array 객체

실행 결과

```
- 문자열로 정렬
[ '감자', '고구마', '바나나' ]

- 문자열로 정렬(역순)
[ '바나나', '고구마', '감자' ]
```

```
- 객체 내부의 숫자로 정렬
[
  { name: '바나나', price: 400 },
  { name: '감자', price: 500 },
  { name: '고구마', price: 1000 }
]

- 객체 내부의 문자열로 정렬
[
  { name: '감자', price: 500 },
  { name: '고구마', price: 1000 },
  { name: '바나나', price: 400 }
]
```

5. Array 객체

ECMAScript5에서 추가된 메소드

- 모질라 레퍼런스 - Array

https ://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array

표 7-7 Array 객체의 메소드

메소드	설명
forEach()	배열의 요소를 하나씩 뽑아 반복을 돌립니다.
map()	콜백 함수에서 리턴하는 것을 기반으로 새로운 배열을 만듭니다.
filter()	콜백 함수에서 true를 리턴하는 것으로만 새로운 배열을 만들어 리턴합니다.

코드 7-23 Array 객체의 메소드 콜백 함수 형태

```
[52, 273, 32].forEach((item, index) => {

});
```

5. Array 객체

- [예제 7-10] ECMAScript5에서 추가된 Array 객체의 메소드

코드 7-24 Array 객체의 메소드 newArrayMethod.js

```
// 배열을 선언합니다.
let array = [52, 273, 32, 64, 72];

// forEach() 메소드
console.log('--- forEach() 메소드 ---');
array.forEach((item, index) => {
    console.log(`- ${index}번째 요소는 ${item}입니다.`);
});

// map() 메소드
console.log();
console.log('--- map() 메소드 ---');
let outputA = array.map((item, index) => {
    // 배열의 모든 요소를 제곱해서 새로운 배열을 만듭니다.
    return item * item;
});
console.log(outputA);

// filter() 메소드
console.log();
console.log('--- filter() 메소드 ---');
let outputB = array.filter((item, index) => {
    // 짝수만 추출합니다.
    return item % 2 == 0;
});
console.log(outputB);
```

5. Array 객체

실행 결과

```
--- forEach() 메소드 ---
- 0번째 요소는 52입니다.
- 1번째 요소는 273입니다.
- 2번째 요소는 32입니다.
- 3번째 요소는 64입니다.
- 4번째 요소는 72입니다.

--- map() 메소드 ---
[ 2704, 74529, 1024, 4096, 5184 ]

--- filter() 메소드 ---
[ 52, 32, 64, 72 ]
```

6. 조금 더 나아가기

lodash 라이브러리

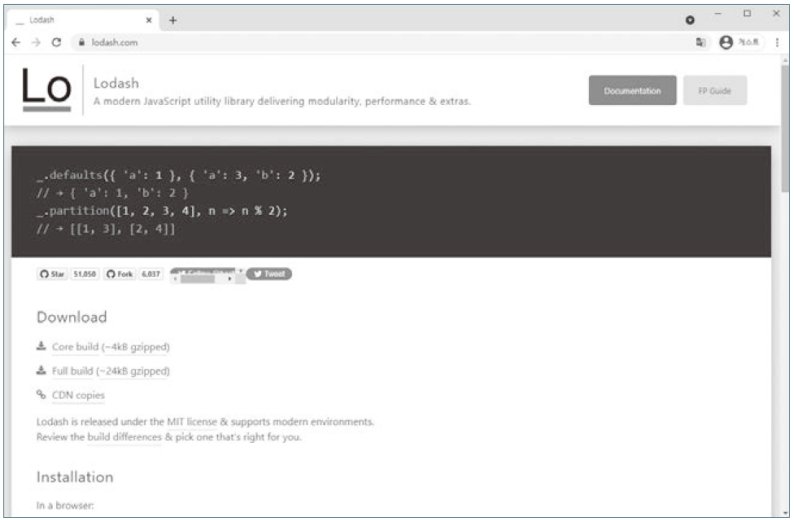

그림 7-3 lodash 라이브러리(https://lodash.com/)

6. 조금 더 나아가기

```
> npm install lodash
```

```
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

코드 7-25 lodash 라이브러리를 사용한 정렬

```
// 모듈을 추출합니다.
const _ = require('lodash');

// 변수를 선언합니다.
const arra-=y = [{
    name: '고구마',
    price: 1000
}, {
    name: '감자',
    price: 500
}, {
    name: '바나나',
    price: 1500
}];

// 1번 형태
const outputA = _.sortBy(array, (item) => item.price)
console.log(outputA);

// 2번 형태
const outputB = _(array).sortBy((item) => item.name);
console.log(outputB);
```

6. 조금 더 나아가기

표 7-8 JSON 객체의 메소드

메소드	설명
JSON.stringify(객체, 변환_함수, 공백_개수)	자바스크립트 객체를 문자로 만듭니다.
JSON.parse(문자열)	문자열을 자바스크립트 객체로 파싱합니다.

코드 7-26 JSON 객체 활용 jsonObject.js

```
// 변수를 선언합니다.
const javascriptObject = [{
    name: '윤인성',
    region: '서울'
}, {
    name: '윤명월',
    region: '도쿄'
}];

// JSON.stringify() 메소드로 자바스크립트 객체를 JSON 문자열로 변경합니다.
const outputA = JSON.stringify(javascriptObject);
const outputB = JSON.stringify(javascriptObject, null, 2);
console.log(typeof(outputA));
console.log(outputA);
console.log(outputB);
console.log('--------------------');

// JSON.parse() 메소드로 JSON 문자열을 자바스크립트 객체로 변경합니다.
const outputC = JSON.parse(outputB)
console.log(typeof (outputC));
console.log(outputC);
```

6. 조금 더 나아가기

실행 결과

```
string
[{"name":"윤인성","region":"서울"},{"name":"윤명월","region":"도쿄"}]
[
  {
    "name": "윤인성",
    "region": "서울"
  },
  {
    "name": "윤명월",
    "region": "도쿄"
  }
]
------------------
object
[ { name: '윤인성', region: '서울' }, { name: '윤명월', region: '도쿄' } ]
```


THANK
YOU!