자료구조와 자료형

숫자형

- 일반적인 숫자(number 타입)는 **배정밀도 부동소수점 숫자(double precision floating point number)로 알려진 64비트 형식의 IEEE-754**에 저장
  - 자바스크립트는 **정수, 실수 타입의 구분이 없음** (모두 number 타입)
- 값이 매우 큰 정수는 BigInt 타입의 숫자로 저장

과학적 표기법을 이용한 큰, 작은 수 표현

```
// 10억, 1과 9개의 0
let billion = 1e9; // 1 * 10^9

// 73억 (7,300,000,000)
alert( 7.3e9 ); // 7.3 * 10^9

// 0.000001
let ms = 1e-6; // 1 * 10^-6
```

어림수 구하기

- 내림, 올림, 반올림 적용하기

Math.floor

소수점 첫째 자리에서 내림 (3.1 => 3)

Math.ceil

소수점 첫째 자리에서 올림 (3.1 => 4)

Math.round

소수점 첫째 자리에서 반올림 (3.1 => 3, 3.6 => 4)

부정확한 계산

- 숫자가 너무 커서 **64비트 저장 범위를 초과한 경우** 값을 Infinity로 처리함을 유의

```
// Infinity
alert( 1e500 );
```

- 소수점 값을 연산하는 과정에서 **정밀도 손실(loss of precision) 현상 발생 가능**

```
// false
alert( 0.1 + 0.2 == 0.3 );
// 0.30000000000000004 (미세한 오차 발생)
console.log( 0.1 + 0.2 );
```

- 정밀도 손실 현상 해결하기 위해서 toFixed 메서드 사용 가능

```
let sum = 0.1 + 0.2;
alert( sum.toFixed(1) ); // "0.3" 출력
alert( +sum.toFixed(1) ); // 0.3 출력
```

- 두 소수점 값을 비교하는 방법
    - https://stackoverflow.com/questions/3343623/javascript-comparing-two-float-values

해결법 1) toFixed 메서드 사용

```
let c = 1.2;
let r = c * 3;
// false
console.log(r == 3.6);
// true (단, toFixed의 결과는 문자열이므로 문자열 비교가 된다는 점을 주의!)
console.log(r.toFixed(1) == (3.6).toFixed(1));
```

해결법 2) 입실론 값을 사용

```
let abs = Math.abs(r - 3.6);
console.log(abs);
console.log(abs < 0.0000001);
// 임의의 아주 작은 값(입실론 값)을 대입
const DELTA = 0.0000001;
// 두 값을 뺀 차가 앞서 정의한 입실론 값보다 적으면 값이 같다고 취급하기
if(Math.abs(r - 3.6) < DELTA) {
    console.log("Math.abs(r - 3.6) < DELTA == true");
}
```

isNaN과 isFinite

- 특수한 숫자값
    - Infinity와 -Infinity
        - 그 어떤 숫자보다 큰 혹은 작은 특수 숫자 값
    - NaN
        - 숫자 연산 중 발생한 에러를 나타내는 값
- 다음과 같이 **비교 연산자를 이용하여 NaN 값 직접 비교 불가**

```
// ==, === 연산자 모두 비교 불가!
alert( NaN == NaN ); // false
alert( NaN === NaN ); // false

// a는 NaN
let a = "asdf" / 100;
if(a === NaN) {
    alert("a === NaN");
} else {
    // "else"가 출력됨을 유의!
    alert("else");
}
```

- NaN 값을 구분하기 위해서 isNaN 함수를 사용

```
// true
isNaN(NaN);
// 전달한 인수를 먼저 숫자로 변환한 후에 NaN 여부를 확인하므로 다음 결과값도 true
isNaN("str");
```

- Infinity의 경우 비교 연산자를 이용해 값 비교가 가능

```
// true
Infinity == Infinity
// true
Infinity === Infinity
```

- NaN/Infinity/-Infinity가 아닌 **일반 숫자인지 여부를 반환**하는 isFinite 함수가 존재

```
// isFinite 내부에서 먼저 숫자 변환을 시도하므로, 그리고 변화한 결과값이 숫자 15이므로
true를 반환
alert( isFinite("15") );
// false 반환, 변환 결과가 NaN이기 때문
alert( isFinite("str") );
// false 반환, 반환 결과가 Infinity이기 때문
alert( isFinite(Infinity) );
```

- isFinite 함수는 **문자열의 내용이 일반 숫자인지 검증**하기 위해서 주로 사용

```
// prompt의 반환값은 문자열
let num = +prompt("숫자를 입력하세요.", '');

// 숫자가 아닌 값을 입력하거나 Infinity, -Infinity를 입력하면 false가 출력
alert( isFinite(num) );
```

- 빈 문자열이나 공백만 있는 문자열은 isFinite를 포함한 모든 숫자 관련 내장 함수에서 0으로 취급

```
// 빈 문자열이므로 0으로 취급하여 true 반환
isFinite("");
// 공백문자(스페이스, 개행, 탭 문자)만 포함하므로 0으로 취급하여 true 반환
isFinite(" \n\t");
```

parseInt와 parseFloat

- + 단항 연산자나 Number를 이용하여 숫자로 변환 시에는 적용되는 규칙이 엄격함
  - **숫자 혹은 숫자 표현과 관련되지 않은 다른 내용이 존재하면 변환 실패**
    - 단, 과학적 표기법을 이용한 "1e5"와 같은 문자열은 잘 변환됨 (숫자 표현과 연관된 내용)
- parseInt, parseFloat 변환 함수는 **숫자가 포함된 영역까지는 변환을 시도**한다는 차이 존재
  - parseInt => 정수 변환 함수
  - parseFloat => 실수 변환 함수

```
// "100"까지 읽고 p부터 실패하므로 문자열 "100"을 재료로 해서 값 변환 시도
alert( parseInt('100px') );
// a부터 실패하므로 123 반환
alert( parseInt("123abc456") );
// "12.5"까지 읽고 e부터 실패하므로 문자열 "12.5"를 재료로 해서 값 변환 시도
alert( parseFloat('12.5em') );
// 과학적 표기법은 문제 없이 변환 가능
alert( parseFloat('12.5e2') ); // 1250

// parseInt 함수를 사용하여 변환했으므로 정수 부분만 반환
alert( parseInt('12.3') );
// 두 번째 점은 무시되므로 12.3 반환
alert( parseFloat('12.3.4') );

// 시작하자마자 a를 만나 변환을 실패하므로 NaN 반환
alert( parseInt('a123') );
```

- parseInt의 두 번째 매개 변수(radix, 진법)는 선택적으로 사용 가능
  - 값을 전달하지 않을 경우 10진법을 이용하여 변환

```
// 16진법 표기법 숫자를 변환 (두 번째 매개 변수로 16전달)
alert( parseInt('0xff', 16) );
// 0x 없어도 잘 동작
alert( parseInt('ff', 16) );
```

기타 수학 함수

Math.random

- 0부터 1을 제외한 범위의 난수 반환

```
alert( Math.random() ); // 0 ~ 0.99999999999999999 사이(1은 제외)의 무작위 수
```

- 1부터 특정 정수를 포함한 범위의 수 구하기

```
// 1 ~ 10(포함)까지 범위 사이의 수 반환
let r = Math.floor(Math.random() * 10) + 1;
console.log(r);
```

- 특정 범위 사이의 수 구하기

```
let max = 100;
let min = 1;
// 1 ~ 100(포함)까지 범위 사이의 수 반환
let r = Math.floor(Math.random() * (max - min + 1)) + min;
```

- 함수 만들기

```
function randInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
};
```

Math.max / Math.min

- **가변 인수**를 전달받아 최대, 최소값을 반환
    - **배열을 전달받지 않는다는 점에 유의**

```
// 가변 인수를 전달받으므로 값 개수는 자유롭게 전달 가능
alert( Math.max(3, 5, -10, 0, 1) ); // 5
alert( Math.min(1, 2) ); // 1

// 배열 전달 X, NaN 출력
alert( Math.max([3, 5, -10, 0, 1]) );
```

Math.pow(n, power)

- n을 power번 거듭제곱한 결과값을 반환

```
// 2의 10제곱 = 1024
alert( Math.pow(2, 10) );
```

문자열

- 자바스크립트엔 글자 하나만 저장할 수 있는 별도의 자료형(ex: char)이 없음
    - 즉, **길이에 상관없이 모두 문자열로 저장**
- 자바스크립트에서 문자열은 페이지의 인코딩 방식과 상관없이 내부적으로 UTF-16 방식으로 저장
- 문자열은 작은따옴표(')나 큰따옴표("), 백틱(`)으로 감쌀 수 있음

```
let single = '작은따옴표';
let double = "큰따옴표";
let backticks = `백틱`;
```

- **표현식을 ${ ... }로 감싸고 이를 백틱으로 감싼 문자열 중간에 넣어주면** 해당 표현식의 결괏값을 문자열 내부에 삽입 가능 => **템플릿 리터럴(template literal)**

```
function sum(a, b) {
    return a + b;
}

// 1 + 2 = 3.
alert(`1 + 2 = ${sum(1, 2)}.`);
```

- 백틱을 사용하면 문자열을 **여러 줄에 걸쳐 작성**할 수 있음
    - 따로 개행문자(\n)를 삽입하지 않아도 알아서 엔터를 개행 문자로 인식

```
let guestList = `손님:
 * John
 * Pete
 * Mary
`;

alert(guestList);
```

- 백틱은 **템플릿 함수(template function)에서도 사용**됨
    - func`string` 같이 첫 번째 백틱 바로 앞에 함수 이름(func)을 써주면, 이 함수는 백틱 안의 문자열 조각이나 표현식 평가 결과를 인수로 받아 자동으로 호출
        - 이런 기능을 태그드 템플릿(tagged template)이라 부르는데, 태그드 템플릿을 사용하면 사용자 지정 템플릿에 맞는 문자열을 쉽게 만들 수 있음

태그드 템플릿 활용 코드

```
let person = 'Mike';
let age = 28;

function myTag(strings, personExp, ageExp) {
    let str0 = strings[0]; // "That "
    let str1 = strings[1]; // " is a "
    let str2 = strings[2]; // "."

    let ageStr;
    if (ageExp > 99){
        ageStr = 'centenarian';
    } else {
        ageStr = 'youngster';
    }

    // We can even return a string built using a template literal
    return `${str0}${personExp}${str1}${ageStr}${str2}`;
}

// 백틱 앞에 함수 이름 쓰기
let output = myTag`That ${ person } is a ${ age }.`;

console.log(output);
// That Mike is a youngster.
```

특수 기호

- 모든 특수 문자(**이스케이프 문자(escape character)**라고도 불리움)는 **역슬래시(\)(backslash character)**로 시작함

특수 문자	설명
`\n`	줄 바꿈
`\r`	캐리지 리턴(carriage return). Windows에선 캐리지 리턴과 줄 바꿈 특수 문자를 조합(`\r\n`)해 줄을 바꿉니다. 캐리지 리턴을 단독으론 사용하는 경우는 없습니다.
`\'`, `\"`	따옴표
`\\`	역슬래시
`\t`	탭
`\b`, `\f`, `\v`	각각 백스페이스(Backspace), 폼 피드(Form Feed), 세로 탭(Vertical Tab)을 나타냅니다. 호환성 유지를 위해 남아있는 기호로 요즘엔 사용하지 않습니다.
`\xXX`	16진수 유니코드 `XX`로 표현한 유니코드 글자입니다(예시: 알파벳 `'z'`는 `'\x7A'`와 동일함).
`\uXXXX`	UTF-16 인코딩 규칙을 사용하는 16진수 코드 `XXXX`로 표현한 유니코드 기호입니다. `XXXX`는 반드시 네 개의 16진수로 구성되어야 합니다(예시: `\u00A9`는 저작권 기호 `©`의 유니코드임).
`\u{X…XXXXXX}` (한 개에서 여섯 개 사이의 16진수 글자)	UTF-32로 표현한 유니코드 기호입니다. 몇몇 특수한 글자는 두 개의 유니코드 기호를 사용해 인코딩되므로 4바이트를 차지합니다. 이 방법을 사용하면 긴 코드를 삽입할 수 있습니다.

- **length 속성**을 통해 문자열 길이 읽어오기 (속성이고, 메서드가 아님을 유의!)

```
// 3 출력
// \n은 '특수 문자' 한 개로 취급
alert( `My\n`.length );
```

- 문자열 내 특정 위치에 있는 글자에 접근하려면 대괄호를 이용하거나, charAt 메서드를 호출 (위치는 0부터 시작)

```
let str = "Hello";

// 첫 번째 글자 H 출력
alert( str[0] );
// charAt 메소드 사용
alert( str.charAt(0) );

// 마지막 글자 o 출력
alert( str[str.length - 1] );
```

- 대괄호와 charAt 차이 => 해당 위치에 글자를 찾을 수 없는 경우 반환값
    - 보통 charAt 보다는 대괄호를 많이 사용함

```
let str = "Hello";

// 대괄호 => 위치에 글자가 없을 경우 undefined 반환
alert( str[1000] );
// charAt => 위치에 글자가 없을 경우 빈 문자열('') 반환
alert( str.charAt(1000) );
```

- **for .. of 반복문**을 사용하여 문자열 순회 가능
    - for .. in 반복문이 객체의 key 값을 순회하기 위해서 사용된다면, **for .. of 반복문은 배열을 순회하기 위해서 사용**됨 (정확히 말하면 iterable 객체 (ex: 배열, 문자열 등))

```
for(let char of "Hello") {
    alert(char); // H,e,l,l,o (char는 순차적으로 H, e, l, l, o가 됩니다.)
}

let lst = [1, 2, 3];
// 1, 2, 3 출력
for(let item of lst) {
    console.log(item);
}
```

문자열의 불변성

- 문자열은 불변값이므로 **문자열의 값을 직접 수정할 수 없음**

```
let str = 'Hi';

// Error: Cannot assign to read only property '0' of string 'Hi'
str[0] = 'A';

// 변경되지 않은 채로, "Hi" 출력
alert( str[0] );
```

- 문자열 내용을 수정하기 위해서는 **완전히 새로운 문자열을 하나 만든 다음, 이 문자열을 할당**해야 함

```
let str = 'Hi';

// 문자열 전체를 교체함
str = 'h' + str[1];
// 혹은 바로 값을 대입
// str = 'hi';

// "hi" 출력
alert( str );
```

부분 문자열 찾기

str.indexOf(substr, pos) 메서드

문자열 str의 pos에서부터 시작해, 부분 문자열 substr이 어디에 위치하는지를 찾아주는 메소드

```
let str = 'Widget with id';

alert( str.indexOf('Widget') ); // 0, str은 'Widget'으로 시작함
alert( str.indexOf('widget') ); // -1, indexOf는 대·소문자를 따지므로 원하는 문자열을 찾지 못함

alert( str.indexOf("id") ); // 1, "id"는 첫 번째 위치에서 발견됨 (Widget에서 id)
```

- str.indexOf(substr, pos)의 두 번째 매개변수 pos는 선택적으로 사용할 수 있는데, 이를 명시하면 검색이 해당 위치부터 시작

```
let str = 'Widget with id';

alert( str.indexOf('id', 2) ) // 12
```

모든 "as"의 위치 찾기 코드

```
let str = 'As sly as a fox, as strong as an ox';
let target = 'as';
let pos = 0;

while (true) {
    let foundPos = str.indexOf(target, pos);
    if (foundPos == -1) break;

    alert( `위치: ${foundPos}` );
    // 다음 위치를 기준으로 검색을 이어갑니다.
    pos = foundPos + 1;
}
```

짧게 줄인 코드

```
let str = "As sly as a fox, as strong as an ox";
let target = "as";

let pos = -1;
while ((pos = str.indexOf(target, pos + 1)) != -1) {
  alert( `위치: ${pos}` );
}
```

부분 문자열 여부를 검사하려면 아래와 같이 -1과 비교

```
let str = "Widget with id";

// 0번 위치에 있는 경우 0이 false로 평가되므로 코드 실행이 안됨!
// if (str.indexOf("Widget")) {
// -1과 비교 필요
if (str.indexOf("Widget") != -1) {
    alert("찾았다!"); // 의도한 대로 동작합니다.
}
```

includes, startsWith, endsWith

includes : 부분 문자열의 위치 정보는 필요하지 않고 포함 여부만 알고 싶을 때 적합한 메서드

```
alert( "Widget with id".includes("Widget") ); // true
alert( "Hello".includes("Bye") ); // false
```

startsWith, endsWith : 메서드 이름 그대로 문자열 str이 특정 문자열로 시작하는지(start with) 여부와 특정 문자열로 끝나는지(end with) 여부를 확인

```
alert( "Widget".startsWith("Wid") ); // true, "Widget"은 "Wid"로 시작합니다.
alert( "Widget".endsWith("get") ); // true, "Widget"은 "get"으로 끝납니다.
```

부분 문자열 추출하기

str.slice(start [, end])

```
let str = "stringify";
alert( str.slice(0, 5) ); // 'strin', 0번째부터 5번째 위치까지(5번째 위치의 글자는 포함하지 않음)
alert( str.slice(0, 1) ); // 's', 0번째부터 1번째 위치까지(1번째 위치의 자는 포함하지 않음)
```

- 두 번째 인수가 생략된 경우엔, 명시한 위치부터 문자열 끝까지를 반환

```
let str = "stringify";
// ringify, 2번째부터 끝까지
alert( str.slice(2) );
```

- start와 end는 음수가 될 수도 있습니다. 음수를 넘기면 문자열 끝에서부터 카운팅을 시작

```
let str = "stringify";
// 끝에서 4번째부터 시작해 끝에서 1번째 위치까지
alert( str.slice(-4, -1) ); // gif
```

str.substring(start [, end])

substring은 slice와 아주 유사하지만 start가 end보다 커도 괜찮다는 데 차이

```
let str = "stringify";

// 동일한 부분 문자열을 반환합니다.
alert( str.substring(2, 6) ); // "ring"
alert( str.substring(6, 2) ); // "ring"

// slice를 사용하면 결과가 다릅니다.
alert( str.slice(2, 6) ); // "ring" (같음)
alert( str.slice(6, 2) ); // "" (빈 문자열)
```

- substring은 음수 인수를 허용하지 않습니다. 음수는 0으로 처리

str.substr(start [, length])

start에서부터 시작해 length 개의 글자를 반환 (끝 위치 대신에 **길이를 기준으로 문자열을 추출**한다는 점에서 substring과 slice와 차이)

```
let str = "stringify";
alert( str.substr(2, 4) ); // ring, 두 번째부터 글자 네 개
```

- 첫 번째 인수가 음수면 뒤에서부터 개수

```
let str = "stringify";
alert( str.substr(-4, 2) ); // gi, 끝에서 네 번째 위치부터 글자 두 개
```

메서드	추출할 부분 문자열	음수 허용 여부(인수)
`slice(start, end)`	`start`부터 `end`까지(`end`는 미포함)	음수 허용
`substring(start, end)`	`start`와 `end` 사이	음수는 `0`으로 취급함
`substr(start, length)`	`start`부터 `length`개의 글자	음수 허용

문자열 비교하기

소문자는 대문자보다 항상 큽니다.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
alert( 'a' > 'Z' ); // true
```

https://unicode-table.com/en/

str.codePointAt(pos)

특정 글자의 코드포인트(유니코드의 위치) 반환

```
// 글자는 같지만 케이스는 다르므로 반환되는 코드가 다릅니다.
alert( "z".codePointAt(0) ); // 122
alert( "Z".codePointAt(0) ); // 90
alert( "Ö".codePointAt(0) ); // 214
alert( "가".codePointAt(0) ); // 44032
```

String.fromCodePoint(code)

코드포인트로부터 글자 반환

```
let c = String.fromCodePoint(90)
alert( c ); // Z
c = String.fromCodePoint(44032)
alert( c ); // 가
```