LIKE LION

내장 함수, 내장 객체

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

주요 내장 함수

내장 객체 - Math

내장 객체 - JSON

{ 내장 객체 - Object }

내장 객체 - String

내장 객체 - RegExp

내장 객체 - Date

내장 객체 - Array

LIKE LION

ex07-01.ts 

▶ `parseInt(string: string, radix?: number): number`
- 지정한 문자열을 정수로 변환
- `string`: 숫자로 변환할 문자열
- `radix`: 2진수부터 36진수까지 변환할 진법. 기본은 10진수
- 리턴값: 변환된 숫자

▶ `parseFloat(string: string): number`
- 지정한 문자열을 부동소수점 방식의 숫자로 변환
- `string`: 숫자로 변환할 문자열
- 리턴값: 변환된 숫자

```
console.log(parseInt('100')); // 100
console.log(parseInt('100', 10)); // 100
console.log(parseInt('100', 2)); // 4
console.log(parseInt('100', 16)); // 256
console.log(parseInt('99.876')); // 99

console.log(parseFloat('99.876')); // 99.876
```

LIKE LION

ex07-02.ts 

▶ `setTimeout(handler: TimerHandler, timeout?: number, ...arguments): number`

- 지정된 시간이 지난 후 특정 코드를 한 번 실행하도록 예약
- 비동기적으로 호출되므로 setTimeout 이후의 코드가 먼저 실행
- delay 값이 0 이어도 현재 실행중인 함수의 코드가 먼저 실행된 후 실행됨
- handler: 지연 후 실행될 코드나 콜백 함수
  - `type TimerHandler = string | Function;`
- timeout: 대기 시간(밀리초 단위, 1초 = 1000ms)
- arguments: 콜백 함수에 전달될 추가 인자값들
- 리턴값: 예약을 중지할 때 사용하는 타이머 id(정수)

▶ `clearTimeout(id: number | undefined): void`

- 취소할 타이머 id (setTimeout()의 리턴값)
- id가 undefined일 경우 타이머 취소가 무시됨

```
console.log('시작');

const timerId = setTimeout(() => {
  console.log('3초 후 실행');
}, 1000*3); // 3초 후 실행

// clearTimeout(timerId);
console.log('종료');
```

LIKE LION

ex07-03.ts 

▶ `setInterval(handler: TimerHandler, timeout?: number, ...arguments): number`
- 지정된 시간 간격으로 특정 코드를 계속 실행하도록 예약
- 매개변수와 리턴값은 setTimeout과 동일

▶ `clearInterval(id: number | undefined): void`
- 취소할 타이머 id (setInterval()의 리턴값)
- id가 undefined일 경우 타이머 취소가 무시됨

```
console.log('시작');

const timerId = setInterval(() => {
  console.log('3초 간격으로 실행');
}, 1000*3); // 3초 간격

// clearInterval(timerId);
console.log('종료');
```

▶ 수학 관련 기능을 제공하는 내장 객체

▶ `Math.abs(x: number): number`
- 절대값

▶ `Math.round(x: number): number`
- 반올림

▶ `Math.ceil(x: number): number`
- 올림

▶ `Math.floor(x: number): number`
- 내림

▶ `Math.trunc(x: number): number`
- 소수 버림

▶ `Math.sign(x: number): number`
- 부호 반환, 양수: 1, 음수: -1, 0: 0

```
console.log(Math.abs(-5)); // 5 절대값
// 소수 첫째자리에서 반올림
console.log(Math.round(4.6)); // 5
// 소수 첫째자리에서 올림
console.log(Math.ceil(4.1)); // 5
// 소수 첫째자리에서 내림
console.log(Math.floor(4.9)); // 4
// 소수 첫째자리에서 내림
console.log(Math.floor(-4.9)); // -5
console.log(Math.trunc(4.9)); // 4 소수 버림
console.log(Math.trunc(-4.9)); // -4 소수 버림
// 부호, 양수: 1, 음수: -1, 0: 0
console.log(Math.sign(-10)); // -1
```

LIKE LION

ex07-04.ts, 04-02 

- `Math.max(...values: number[]): number`
  - 인자값들 중 최댓값
- `Math.min(...values: number[]): number`
  - 인자값들 중 최솟값
- `Math.random(): number`
  - 0이상 1미만 사이의 난수
- `Math.pow(x: number, y: number): number`
  - 거듭제곱
- `Math.sqrt(x: number): number`
  - 제곱근
- `Math.PI: number`
  - 파이값, 3.141592653589793
- 삼각함수
  - Math.sin(), Math.cos(), Math.tan()

```ts
// 0 이상 1 미만 사이의 난수
console.log(Math.random());
// 인자값들 중 최대값
console.log(Math.max(1, 5, 3)); // 5
// 인자값들 중 최소값
console.log(Math.min(1, 5, 3)); // 1
console.log(Math.pow(2, 3)); // 8 거듭제곱
console.log(Math.sqrt(9)); // 3 제곱근
console.log(Math.PI); // 3.141592653589793
```

▶ 객체를 문자열로 변환하거나, 문자열을 다시 객체로 변환할 때 사용하는 내장 객체

▶ `JSON.stringify(value, replacer?, space?): string`

- JavaScript 값이나 객체를 JSON 문자열로 변환하는 함수
- 객체를 네트워크로 전송하거나 파일에 저장할 때 문자열로 변환할 필요가 있음(직렬화)

- 매개변수
  - value: 변환할 값(객체, 배열, 숫자, 문자열 등)
  - replacer
    - 함수: 변환하기 전에 값을 수정할 수 있음
    - 배열: 변환에 포함할 속성만 지정 가능
  - space: 들여쓰기 설정(가독성을 위해 공백 추가)

- 반환값
  - 변환된 JSON 형식의 문자열

LIKE LION

ex07-05.ts 

▶ JSON.parse(text, reviver?)

- JSON 문자열의 구문을 분석하고, 그 결과에서 JavaScript 값이나 객체를 생성
- 선택적으로, reviver 함수를 인수로 전달할 경우, 결과를 반환하기 전에 변형할 수 있음

- 매개변수
  - text: 객체로 변환할 JSON 형식의 문자열
  - reviver: 변환할 각 속성에 대해 호출되며 reviver가 반환한 값이 모여서 최종 parse()의 반환값이 됨

```
const haru = { name: '하루', age: 5 };
const strHaru = JSON.stringify(haru);
const objHaru = JSON.parse(strHaru);
console.log(typeof haru, haru); // object { name: '하루', age: 5 }
console.log(typeof strHaru, strHaru); // string {"name":"하루","age":5}
console.log(typeof objHaru, objHaru); // object { name: '하루', age: 5 }
```

모든 객체의 부모 역할을 하는 내장 객체

- 객체를 생성하거나 객체 관련 작업을 할 때 사용
- static 메서드 들과 모든 객체에서 프로토타입 체인에 의해 호출 할 수 있는 메서드로 구성

`Object.keys(o: object): string[]`

- 객체의 모든 키를 배열로 반환

`Object.values(o: object): any[])`

- 객체의 모든 값을 배열로 반환

`Object.entries(o: object): [string, any][])`

- 객체의 모든 속성을 [키, 값] 쌍의 배열로 반환

`Object.assign(target: object, ...sources): any`

- sources 객체들의 속성을 복사해서 target 객체에 추가(객체의 속성을 병합)
- 매개변수
  - target: 복사된 속성을 저장할 객체
  - sources: 복사할 객체들. 이곳에 지정한 객체들의 속성이 target에 추가됨
- 리턴값: sources 속성들이 복사된 target 객체

LIKE LION

ex07-06.ts

```ts
const haru = { name: '하루', age: 5 };

console.log(Object.keys(haru)); // ['name', 'age']
console.log(Object.values(haru)); // ['하루', 5]
console.log(Object.entries(haru)); // [['name', '하루'], ['age', 5]]

const newUser = Object.fromEntries([['name', '나무'], ['age', 8]]);
console.log(newUser); // { name: '나무', age: 8 }

const haru2 = haru;
haru.age++;
console.log(haru.age, haru2.age); // 6 6

const haru3 = Object.assign({}, haru);
haru.age++;
console.log(haru.age, haru3.age); // 7 6
```

LIKE LION

ex07-07.ts 

문자열 데이터를 다루기 위한 래퍼 객체

- string 값을 감싸 다양한 속성과 메서드를 사용할 수 있게함
- string 원시형 타입에서 바로 메서드를 호출하면 임시 String 객체를 생성해서 해당 메서드 실행 후 바로 삭제
- 유사 배열 객체: length 속성이 있고 0부터 시작해서 증가하는 index에 글자가 하나씩 저장되어 있음
- 모든 메서드는 기존의 문자열을 수정하지 않고 새로운 문자열을 만들어서 반환

`trim(): string`

- 문자열 앞뒤의 공백 제거

`charAt(index: number): string`

- 문자열에서 `index` 위치에 있는 한 글자를 반환

`charCodeAt(index: number): number`

- 문자열에서 `index` 위치에 있는 문자의
- 유니코드(UTF-16) 값을 반환

```
let msg = '  Hello World ';
console.log(`[${msg}]`);

msg = msg.trim();

console.log(`[${msg}]`); // [Hello World]

console.log(msg.charAt(8)); // r

console.log(msg.charCodeAt(0)); // 72
```

LIKE LION

ex07-07.ts 

▶ `indexOf(searchString: string, position?: number): number`
- 문자열에서 특정 문자열이나 문자를 왼쪽부터 검색해서 처음 발견된 위치를 반환
- `position:` 검색을 시작할 index (기본값 0)

▶ `lastIndexOf(searchString: string, position?: number): number`
- 문자열에서 특정 문자열이나 문자를 오른쪽부터 검색해서 처음 발견된 위치를 반환
- `position:` 검색을 시작할 index (기본값 length-1)

▶ `includes(searchString: string, position?: number): boolean`
- 문자열이 특정 문자나 부분 문자열을 포함하고 있는지 여부를 반환

```
let msg = 'Hello World';

console.log(msg.indexOf('o')); // 4

console.log(msg.lastIndexOf('o')); // 7

console.log(msg.includes('llo')); // true
```

LIKE LION

ex07-07.ts 

- `startsWith(searchString: string, position?: number): boolean`
  - 문자열이 특정 문자열이나 문자로 시작하는지 여부를 반환
- `endsWith(searchString: string, endPosition?: number): boolean`
  - 문자열이 특정 문자열이나 문자로 끝나는지 여부를 반환
- `concat(...strings: string[]): string`
  - 문자열에 `strings` 문자열들을 합쳐서 반환 (+ 연산자와 같은 기능)

```
let msg = 'Hello World';

console.log(msg.startsWith('H')); // true

console.log(msg.endsWith('d')); // true

console.log(msg.concat(' and', ' JavaScript')); // Hello World and JavaScript
```

LIKE LION

ex07-07.ts

▶ replace(searchValue: string | RegExp, replaceValue: string): string
- 문자열의 처음부터 searchValue를 하나 찾아서 replaceValue로 교체한 문자열을 반환

▶ replaceAll(searchValue: string | RegExp, replaceValue: string): string
- 문자열에서 searchValue를 모두 찾아서 replaceValue로 교체한 문자열을 반환

▶ repeat(count: number): string
- 문자열을 count 횟수만큼 반복해서 결합한 문자열을 반환

```
let msg = 'Hello World';

console.log(msg.replace('Hello', 'Hi')); // Hi World

console.log(msg.replaceAll('o', 'O')); // HellO WOrld

console.log(msg.repeat(3)); // Hello WorldHello WorldHello World
```

LIKE LION

ex07-07.ts 

▶ `split(separator: string | RegExp, limit?: number): string[]`
- 문자열을 `separator` 구분자를 기준으로 나눠 배열로 반환. `limit` 는 반환할 배열의 최대 길이

▶ `slice(start?: number, end?: number): string`
- 문자열의 `start` 인덱스부터 `end` 인덱스 앞까지 문자를 반환
- 음수를 지정하면 뒤에서부터 계산
- end를 생략하면 문자열의 끝까지 추출 (length - 1)
- start도 생략하면 문자열의 시작부터 끝까지 복사해서 반환

```
let msg = 'Hello World';

console.log(msg.split(' ')); // ['Hello', 'World']

console.log(msg.slice(0, 3)); // Hel(0번째 위치부터 3번째 위치 이전까지 반환(0~2))
console.log(msg.slice(7, 9)); // or(7번째 위치부터 9번째 위치 이전까지 반환(7~8))
console.log(msg.slice(-3)); // rld(뒤에서 3번째 위치부터 끝까지 반환)
console.log(msg.slice(7, -1)); // orld(7번째 위치부터 뒤에서 1번째 위치 이전까지 반환)
```

LIKE LION

ex07-07.ts

- `toLowerCase(): string`
  - 문자열을 모두 소문자로 변환해서 반환

- `toUpperCase(): string`
  - 문자열을 모두 대문자로 변환해서 반환

```
let msg = 'Hello World';

console.log(msg.toLowerCase()); // hello world

console.log(msg.toUpperCase()); // HELLO WORLD
```

LIKE LION

ex07-08.ts 

정규표현식(Regular Expression)

- 문자열에서 특정 패턴을 검색하거나 치환할 때 사용하는 문법
- 주로 문자열 검증, 찾기/바꾸기, 추출 등에 사용

RegExp

- 자바스크립트에서 정규표현식을 다루는 생성자 함수
  - `const regex = new RegExp('pattern', 'flag');`
- 정규표현식 리터럴
  - `const regex = /pattern/flag;`
- 예시: 영문자와 숫자의 조합만으로 1글자 이상인지 확인
  - `const regex = /^[a-z0-9]+$/i;`

예시 설명

- ^ 문자열의 시작
- [] 문자 집합
  - [abc] -> a, b, c 중 하나
  - [a-z0-9] -> a부터 z 까지 또는 0에서 9까지의 문자
- + 1회 이상 반복
- $ 문자열의 끝
- i 대소문자 구별 안함

▶ 정규표현식 주요 패턴

패턴	의미	예시	설명
.	임의의 한 문자	/a.b/ → aab, acb 등	임의의 한 문자를 의미하며, 개별 문자를 하나씩 대체함
₩d	숫자 (0~9)	/₩d{3}/ → 123, 456 등	숫자 하나를 의미하며, `{}`를 사용해 반복횟수 설정 가능
₩w	알파벳, 숫자, 밑줄	/₩w+/ → abc123_	영문자, 숫자, 밑줄을 포함하는 문자들
₩s	공백 문자	/₩s+/ → " " 공백	공백, 탭, 줄바꿈 등 공백 문자를 의미
^	문자열의 시작	/^hi/ → hi로 시작	문자열의 시작 부분이 일치하는지 확인
$	문자열의 끝	/end$/ → end로 끝남	문자열의 끝 부분이 일치하는지 확인
[]	문자 집합	/[abc]/ → a, b, c 중 하나 /[a-z]/ → a ~ z 사이의 문자	대괄호 안의 문자들 중 하나에 일치
{n,m}	n~m회 반복	/a{2,4}/ → aa, aaa, aaaa	앞 문자가 n번에서 m번 반복되는 경우
()	그룹	/(ab)+/ → ab, abab, ababab	그룹화하여 패턴을 묶고 반복할 수 있음
|	또는 (OR 조건)	/cat|dog/ → cat 또는 dog	OR 조건으로 둘 중 하나에 일치
+	1회 이상 반복	/a+/ → a, aa, aaa, ...	앞 문자가 1회 이상 반복되는 경우
*	0회 이상 반복	/a*/ → '', a, aa, aaa, ...	앞 문자가 0회 이상 반복되는 경우
?	0회 또는 1회 반복	/a?/ → '', a	앞 문자가 0회 또는 1회 반복되는 경우

LIKE LION

ex07-08.ts

▶ 정규표현식 주요 플래그

패턴	의미	예시	설명
g	전역 검색	/ab/g → "ab"가 여러 번 나온 경우 모두 선택	전체 문자열에서 일치하는 모든 결과를 찾기
i	대소문자 구별 없음	/abc/i → "ABC"나 "aBc" 모두 일치	대소문자에 상관없이 일치 여부를 확인

```
const emailExp = /^[\w.-]+@[\w.-]+\.\w{2,}$/;
// 영문, 숫자 조합 8자리 이상
const passwordExp = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/;
const nicknameExp = /^[가-힣a-zA-Z0-9]{2,10}$/; // 한글, 영문, 숫자 2~10자리
const phoneExp = /^01\d{1}-\d{3,4}-\d{4}$/; // 01로 시작하고 3자리-3~4자리-4자리 숫자

console.log(emailExp.test('uzoolove@gmail.com')); // true
console.log(passwordExp.test('ab123456')); // true
console.log(nicknameExp.test('하루')); // true
console.log(phoneExp.test('010-1234-5678')); // true
```

LIKE LION

ex07-09.ts, 09-02, 09-03.js

▶ 날짜와 시간을 다룰 때 사용

```
const today = new Date(); // 현재 시간
console.log(today); // 현재 시간 출력(UTC)

console.log(today.getFullYear()); // 4자리 년도
console.log(today.getMonth()+1); // 월(0부터 시작)
console.log(today.getDate()); // 일
console.log(today.getDay()); // 요일 (0: 일요일 ~ 6: 토요일)

console.log(today.getHours()); // 시
console.log(today.getMinutes()); // 분
console.log(today.getSeconds()); // 초

console.log(today.getTime()); // 1970년 1월 1일 00:00:00부터 현재까지의 밀리초
```

LIKE LION

ex07-10.ts

▶ push(...items): number
- 배열의 마지막 위치에 items 요소들을 추가하고 새로운 배열 길이를 반환

▶ pop(): string | undefined
- 배열의 마지막 요소를 제거하고 반환

```
const fruits = ['사과', '바나나'];

const newLength = fruits.push('오렌지');
console.log(newLength, fruits);    // 3 ['사과', '바나나', '오렌지']

fruits.push('딸기', '포도');
console.log(fruits);    // ['사과', '바나나', '오렌지', '딸기', '포도']

let lastFruit = fruits.pop();
console.log(lastFruit, fruits);   // 포도 ['사과', '바나나', '오렌지', '딸기']

lastFruit = fruits.pop();
console.log(lastFruit, fruits);   // 딸기 ['사과', '바나나', '오렌지']
```

LIKE LION

ex07-11.ts

▶ `unshift(...items): number`
- 배열의 맨앞에 items 요소들을 삽입하고 새로운 배열 길이를 반환

▶ `shift(): string | undefined`
- 배열의 첫 번째 요소를 제거하고 반환

```
const fruits = ['사과', '바나나'];

const newLength = fruits.unshift('딸기');
console.log(newLength, fruits); // 3 ['딸기', '사과', '바나나']

fruits.unshift('딸기', '포도');
console.log(fruits); // ['딸기', '포도', '딸기', '사과', '바나나']

let firstFruit = fruits.shift();
console.log(firstFruit, fruits); // 딸기 ['포도', '딸기', '사과', '바나나']

firstFruit = fruits.shift();
console.log(firstFruit,  fruits); // 포도 ['딸기', '사과', '바나나']
```

LIKE LION

ex07-12.ts

▶ `indexOf(searchElement, fromIndex?: number): number`
- 배열의 요소 중 `searchElement`와 일치하는 첫 번째 요소의 인덱스를 반환. 일치하는 요소가 없으면 -1 반환
- `fromIndex`에 지정한 인덱스부터 탐색을 시작 (기본값 0)

▶ `lastIndexOf(searchElement, fromIndex?: number): number`
- 배열의 요소 중 `searchElement`와 일치하는 마지막 요소의 인덱스를 반환. 일치하는 요소가 없으면 -1 반환
- `fromIndex`에 지정한 인덱스부터 탐색을 시작 (기본값 0)

```
const arr = [1, 3, 5, 8, 9, 3, 4, 5];
console.log('첫번째 3의 위치', arr.indexOf(3));
console.log('마지막 3의 위치', arr.lastIndexOf(3));

console.log(arr.find(num => num % 2 === 0));
console.log(arr.findIndex(num => num % 2 === 0));
```

LIKE LION

ex07-13.ts

- `includes(searchElement, fromIndex?: number): boolean`
  - 배열의 요소 중 `searchElement` 값이 있는지 여부를 반환
  - `fromIndex`에 지정한 인덱스부터 탐색을 시작 (기본값 0)

- `concat(...items)`
  - `items` 배열들을 병합한 새로운 배열을 반환

```
const arr = ['오렌지', '딸기', '레몬'];
console.log(arr.includes('레몬')); // true
console.log(arr.includes('사과')); // false

const arr2 = arr.concat(['사과', '바나나'], ['포도']);
console.log(arr2.includes('사과')); // true

console.log(arr); // [ '오렌지', '딸기', '레몬' ]
console.log(arr2); // [ '오렌지', '딸기', '레몬', '사과', '바나나', '포도' ]
```

LIKE LION

ex07-14.ts 

▶ `splice(start: number, deleteCount?: number, ...items)`

- 배열에서 요소를 추가, 제거 또는 교체 한다.
- `start:` 시작 인덱스
- `deleteCount:` 제거할 요소 수(기본값 length-1)
- `items:` 삽입할 요소 목록

```
const arr1 = ['한놈', '두식이', '석삼', '너구리', '오징어', '육개장', '칠뜨기'];

let arr2 = arr1.splice(1, 2); // 인덱스 1부터 2개 추출
console.log(arr1, arr2); // ['한놈','너구리','오징어','육개장','칠뜨기'] ['두식이','석삼']

arr2 = arr1.splice(2, 2); // 인덱스 2부터 2개 추출
console.log(arr1, arr2); // ['한놈', '너구리', '칠뜨기'] ['오징어', '육개장']

arr2 = arr1.splice(2); // 인덱스 2부터 끝까지 추출
console.log(arr1, arr2); // ['한놈', '너구리'] ['칠뜨기']

arr2 = arr1.splice(1, 1, '두식이', '석삼'); // 인덱스 1부터 1개 추출하고 두식이, 석삼 추가
console.log(arr1, arr2); // ['한놈', '두식이', '석삼'] ['너구리']
```

LIKE LION

ex07-15.ts

▶ slice(start?: number, end?: number)

- 배열의 지정한 범위를 복사해서 새 배열로 반환
- start: 시작 인덱스 (기본값 0)
- end: 종료 인덱스 (기본값 length-1)

```
const arr1 = ['한놈', '두식이', '석삼', '너구리', '오징어', '육개장', '칠뜨기'];

let arr2 = arr1.slice(1, 3); // 인덱스 1부터 3 앞까지 복사
console.log(arr2); // ['두식이', '석삼']

arr2 = arr1.slice(2, 2); // 인덱스 2부터 2 앞까지 복사
console.log(arr2); // []

arr2 = arr1.slice(5); // 인덱스 5부터 끝까지 복사
console.log(arr2); // ['육개장', '칠뜨기']

arr2 = arr1.slice(-2); // 인덱스 -2부터 끝까지 복사
console.log(arr2); // ['육개장', '칠뜨기']

console.log(arr1);
```

LIKE LION

ex07-16.ts 

▶ forEach(callbackFn(currentValue, index, array), thisArg?)

- 배열의 각 요소에 대해 callbackFn 함수를 실행한다.
- 콜백 함수의 currentValue에는 배열의 요소가, index에는 전달되는 요소의 인덱스가, array에는 원본 배열이 전달된다.
- thisArg는 콜백 함수에서 this로 사용할 객체를 전달

```
const arr = [10, 20, 30];

let newArr: number[] = [];
// 배열의 각 요소를 순회하며 실행
arr.forEach((elem, i) => {
  newArr.push(elem ** 2);
});
console.log('forEach', newArr); // [100, 400, 900]
```

LIKE LION

ex07-16.ts

▶ map(callbackfn(currentValue, index, array), thisArg?)

- forEach와 동일
- forEach는 리턴값이 없지만 map은 콜백 함수에서 리턴한 값을 새로운 배열로 만들어서 반환한다.

```
const arr = [10, 20, 30];

// 배열의 각 요소를 순회하며 반환받은 값으로 새로운 배열 생성
const newArr = arr.map(function(elem, i) {
  // 요소의 제곱값 반환
  return elem ** 2;
});

console.log('map', newArr); // [100, 400, 900]
```

LIKE LION

ex07-17.ts

▶ `find(callbackFn, thisArg?): any | undefined`
- 배열의 각 요소에 대해 `callbackFn` 함수가 호출된다.
- true를 반환한 첫 콜백 함수에 전달된 엘리먼트가 `find`의 결과로 반환된다.
- true를 반환한 콜백 함수가 없을 경우 `undefined`가 반환된다.

▶ `findIndex(callbackFn, thisArg?): number`
- 배열의 각 요소에 대해 `callbackFn` 함수가 호출된다.
- true를 반환한 첫 콜백 함수에 전달된 인덱스가 `find`의 결과로 반환된다.
- true를 반환한 콜백 함수가 없을 경우 0이 반환된다.

▶ `filter(callbackFn, thisArg?): any | undefined`
- 배열의 각 요소에 대해 `callbackFn` 함수가 호출된다.
- true를 반환한 콜백 함수에 전달된 요소만 모아서 새로운 배열로 반환
- true를 반환한 콜백 함수가 없을 경우 `undefined`가 반환된다.

```
const arr = [1, 3, 5, 8, 9, 3, 4, 5];
console.log(arr.find(num => num % 2 === 0)); // 8
console.log(arr.findIndex(num => num % 2 === 0)); // 3
console.log(arr.filter(n => n % 2 === 0)); // [8, 4]
```

LIKE LION

ex07-18.ts

▶ `some(callbackFn, thisArg?): boolean`
- 배열의 각 요소에 대해 `callbackFn` 함수가 호출된다.
- 콜백 함수 중 하나라도 true를 반환하면 some은 true를 반환한다.
- 콜백 함수 전부 true를 반환하지 않으면 some은 false를 반환한다.

▶ `every(callbackFn, thisArg?): boolean`
- 배열의 각 요소에 대해 `callbackFn` 함수가 호출된다.
- 콜백 함수 전부 true를 반환하면 every는 true를 반환한다.
- 콜백 함수 중 하나라도 true를 반환하지 않으면 every는 false를 반환한다.

```
const arr = [1, 2, 3];
const hasEven = arr.some(n => n % 2 === 0); // true
const isAllEven = arr.every(n => n % 2 === 0); // false

console.log(hasEven, isAllEven);
```

LIKE LION

ex07-19.ts

reduce()

- 배열의 각 요소에 대해 제공한 리듀서 함수를 실행
- 이전 리듀서의 반환값이 다음 리듀서의 인자값으로 전달되며 최종적으로 하나의 결과값을 반환
- 리듀서가 처음 실행되면 "이전 리듀서의 반환값"이 없으므로 reduce 함수의 두번째 인자로 전달하는 값을 사용하거나 두번째 인자가 생략될 경우 배열의 index 0 값이 지정되고 배열의 두번째 요소부터 리듀서가 실행

```
const arr = [1, 2, 3, 4];
const initialValue = 0;
const sum = arr.reduce(function(accumulator, currentValue){
  return accumulator + currentValue
}, initialValue);

console.log(sum); // 0 + 1 + 2 + 3 + 4
```