코딩 테스트 개요 및 문제 풀이를 위한 JavaScript 문법

2) 문제 풀이를 위한 JavaScript 핵심 문법 알아보기

코딩 테스트 알아보기 | JavaScript 핵심 문법 알아보기

강사 나동빈

코딩 테스트 개요 및 문제 풀이를 위한 JavaScript 문법

2) 문제 풀이를 위한 JavaScript 핵심 문법 알아보기

알고리즘 코딩 테스트 문제의 입출력 형식

- 알고리즘 문제에서는 적절한(약속된) 입출력 양식이 주어진다.

1. 첫 번째 단계는 데이터를 **입력 받거나 생성**하는 것이다.
2. 이후에 적절한 알고리즘을 사용하여 **도출된 정답을 정확한 형식으로 출력**한다.

- **예시)** N명의 학생의 성적 데이터가 주어졌을 때, 이를 내림차순 정렬한 결과를 출력하여라.

알고리즘 코딩 테스트 문제의 입출력 형식

- **예시)** N명의 학생의 성적 데이터가 주어졌을 때, 내림차순 정렬한 결과를 출력하여라.
- **입력 형식**: 첫째 줄에는 학생의 수 N이 자연수로 주어지고, 둘째 줄에 공백을 기준으로 하여 N명의 학생에 대한 성적이 정수 형태로 주어진다. $(2 \leq N \leq 1{,}000,\ 0 \leq \text{성적} \leq 100)$
- **출력 형식**: N명의 학생의 성적을 내림차순 정렬한 결과를 첫째 줄에 공백을 기준으로 구분하여 출력하여라.

알고리즘 코딩 테스트 문제의 입출력 형식

- **예시)** N명의 학생의 성적 데이터가 주어졌을 때, 내림차순 정렬한 결과를 출력하여라.

[입력 예시]

```
5
17 88 53 100 73
```

[출력 예시]

```
100 88 73 53 17
```

코딩 테스트용 JavaScript 기본 출력

- 일반적인 알고리즘 문제를 풀 때, 표준 출력으로 console.log()를 이용한다.

```
// 단순히 문자열을 출력합니다.
console.log('Hello World!');

result = 35;
// 템플릿 리터럴을 사용해 문자열 내부에 변수를 포함합니다. (백틱 문자 사용)
console.log(`정답은 ${result}입니다.`);
```

코딩 테스트용 JavaScript 기본 사칙 연산

- JavaScript 프로그래밍 언어에서는 기본적인 사칙 연산 기능을 제공한다.

```
a = 7;
b = 3;

console.log(a + b);
console.log(a - b);
console.log(a * b);
console.log(parseInt(a / b)); // 몫만 남기기
console.log(a % b);
```

코딩 테스트용 JavaScript 빠른 출력

- JavaScript로 코딩 테스트 문제를 풀 때, 출력 과정만으로 시간 초과를 받을 때가 있다.
- **출력 시간을 단축**하기 위해 다음과 같은 방법을 유용하게 사용할 수 있다.

```
let answer = '';

/*
여러 출력 결과를 한 줄에 하나씩 출력할 때 매 번 console.log()를 실행하지 않고,
하나의 문자열에 결과를 저장해서 한꺼번에 출력하는 것이 대개 더 빠르게 수행됩니다.
*/
for (let i = 1; i <= 100; i++) {
  answer += i + '\n'; // 문자열로 변환하여 기록
}

console.log(answer);
```

코딩 테스트용 JavaScript 기본 입력 – fs 모듈

- 입력 데이터가 텍스트 파일 형태로 주어지는 경우, **파일 시스템 모듈**을 사용한다.
- 예를 들어 /dev/stdin 파일에 적힌 텍스트를 읽어오는 경우, 다음과 같이 코드를 작성한다.
- **기능**: 전체 텍스트를 읽어 온 뒤에, 줄바꿈 기호를 기준으로 구분하여 리스트로 변환하기

```
// readline 모듈보다는 fs를 이용해 파일 전체를 읽어 들여 처리하기
let fs = require('fs');
let input = fs.readFileSync('/dev/stdin').toString().split('\n');
// let input = fs.readFileSync('input.txt').toString().split('\n');

console.log(input);
```

코딩 테스트용 JavaScript 기본 입력 – fs 모듈

- **기능**: 전체 텍스트를 읽어 온 뒤에, 줄바꿈 기호를 기준으로 구분하여 리스트로 변환하기

```
// readline 모듈보다는 fs를 이용해 파일 전체를 읽어 들여 처리하기
let fs = require('fs');
let input = fs.readFileSync('input.txt').toString().split('\n');

console.log(input);
```

input.txt

```
123
456
789 1000
```

[출력 예시]

```
['123', '456', '789 1000']
```

코딩 테스트용 JavaScript 기본 입력 – readline 모듈

- 한 줄씩 입력을 받아서, 처리하여 정답을 출력할 때는 **readline 모듈**을 사용할 수 있다.

```
const rl = require('readline').createInterface({
  input: process.stdin,
  output: process.stdout
});

let input = [];
rl.on('line', function(line) {
  // 콘솔 입력 창에서 줄바꿈(Enter)를 입력할 때마다 호출
  input.push(line);
}).on('close', function() {
  // 콘솔 입력 창에서 Ctrl + C 혹은 Ctrl + D를 입력하면 호출(입력의 종료)
  console.log(input);
  process.exit();
});
```

JavaScript 문법 – 조건문

- 조건에 따라서 프로그램의 흐름을 결정할 때 사용할 수 있는 문법이다.

```
/*
condition: 참 혹은 거짓을 반환하는 조건식
statement1: condition1이 참일 때 실행되는 구문
statement2: condition1이 거짓이고, condition2가 참일 때 실행되는 구문
statement3: condition1과 condition2가 거짓이고, condition3이 참일 때 실행되는 구문
statementN: 앞의 모든 조건문이 거짓일 때 실행되는 구문
*/

if (condition1)
  statement1
else if (condition2)
  statement2
else if (condition3)
  statement3
...
else
  statementN
```

JavaScript 문법 – for 반복문

- **조건에 따라서** 특정한 코드를 반복하고자 할 때 사용할 수 있는 코드다.

```
/*
- 초기문이 존재한다면 초기문이 먼저 실행됩니다.
- 조건문이 참이라면 블록 내부 코드가 실행되고, 거짓이면 반복문이 종료됩니다.
- 블록 내부 코드가 실행된 뒤에 증감문이 실행됩니다.
*/
for (초기문; 조건문; 증감문) {
  // statements
}

// 1부터 100까지의 합 계산
let summary = 0;
for (let i = 1; i <= 100; i++) {
  summary += i;
}
console.log(summary);
```

JavaScript 문법 – while 반복문

- **조건에 따라서** 특정한 코드를 반복하고자 할 때 사용할 수 있는 코드다.

```
/*
- while문은 조건문이 참일 때 실행되는 반복문입니다.
- 블록 내부의 코드가 실행되기 전에 참 혹은 거짓을 판단합니다.
*/
while (조건문) {
    // 실행되는 코드 부분
}
```

JavaScript 문법 – Number와 String 형태 변환

- 수(number) 데이터와 **문자열(string)** 데이터간의 상호 변환이 필요하다.

```
/*
Int -> String
*/
let a = "777";
let b = Number(a);
console.log(b); // 777
```

```
/*
String -> Int
*/
let a = 777;
let b = String(a);
console.log(b); // "777"
```

JavaScript 문법 – Array.prototype.reduce()

- 배열의 모든 원소에 대해 특정한 연산을 순차적으로 적용할 때 reduce()를 사용한다.

```
/*
reduce() 메서드는 배열의 각 요소에 대해 reducer 함수를 실행한 뒤에 하나의 결과를 반환합니다.
reducer의 형태: (accumulator, currentValue) => (반환값)
- 배열의 각 원소를 하나씩 확인하며, 각 원소는 currentValue에 해당합니다.
- 반환값은 그 이후의 원소에 대하여 accumulator에 저장됩니다.
*/

let data = [5, 2, 9, 8, 4];

// minValue 구하기 예제
let minValue = data.reduce((a, b) => Math.min(a, b));
console.log(minValue); // 2

// 원소의 합계 구하기 예제
let summary = data.reduce((a, b) => a + b);
console.log(summary); // 28
```

JavaScript 문법 – 배열 초기화 방법

- 알고리즘 문제를 풀 때 자주 사용되는 배열 초기화 방식은 다음과 같다.

```
// 직접 값을 설정하여 초기화
let arr = [8, 1, 4, 5, 7];
```

```
// 길이가 5이고 모든 원소의 값이 0인 배열 초기화
let arr = new Array(5).fill(0);
```

JavaScript 문법 – 집합 자료형

- 특정한 **원소의 등장 여부**를 파악할 때 집합 자료형을 효과적으로 사용할 수 있다.

```
let mySet = new Set(); // 집합 객체 생성

// 여러 개의 원소 삽입
mySet.add(3);
mySet.add(5);
mySet.add(7);
mySet.add(3);

console.log(`원소의 개수: ${mySet.size}`);
console.log(`원소 7을 포함하고 있는가? ${mySet.has(7)}`);

// 원소 5 제거
mySet.delete(5);

// 원소를 하나씩 출력
for (let item of mySet) console.log(item);
```

JavaScript 문법 – 소수점 아래 특정 자리에서 반올림

- 실수를 출력할 때 소수점 아래 특정 자리에서 반올림할 수 있다.

```
// 특정 실수에 대하여 toFixed()를 이용해 소수점 아래 둘째 자리까지 출력
let x = 123.456;
console.log(x.toFixed(2));
```

이스케이프 시퀀스(Escape Sequence)

- **예약 문자** 혹은 **특수 문자**를 출력하기 위하여 이스케이프 시퀀스를 사용할 수 있다.

시퀀스	문자
₩t	탭
₩₩	역 슬래시
₩"	큰 따옴표
₩'	작은 따옴표

```
console.log("그는 \"비범함\"을 보였다.");
```