



1주차 : 자바스크립트 기본 문법



매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일



[학습 목표]

1. 자바스크립트의 역사와 특징에 대해 이해할 수 있습니다.
2. 자바스크립트의 기본 문법인 변수, 상수, 데이터 타입, 연산자, 함수 등을 이해하고 적용할 수 있습니다.
3. 변수 선언과 할당, 산술, 논리, 비교, 삼항 연산자 등 자바스크립트에서의 표현식을 작성할 수 있습니다.
4. 조건문과 반복문을 이해하고 적용할 수 있습니다.
5. 배열과 객체의 기본 개념을 이해하고 활용할 수 있습니다.

1. JS(자바스크립트) 언어의 특징 그리고 역사



출처 : <https://www.iplocation.net/javascript>

▼ (1) JS의 역사

- 1995년 자바스크립트 탄생
 - 🌐 넷스케이프 커뮤니케이션(LiveScript → Javascript)
 - 🖥️ 브라우저 동작 스크립트 언어
- 1999년 자바스크립트 표준화(ECMA-262) 완료
- 2005년 AJAX 등장
 - 비동기 웹 애플리케이션 개발 가능
 - 폭발적인 UX 향상 🚀
- 2008년 V8 엔진 출시(google)
 - super fast(코드 실행 속도 상당부분 개선)
 - 🛠️ 컴파일러, 메모리관리 시스템 👍
- 2009년 Node.js 등장, 서버 개발 활성화
 - 서버개발의 지각변동 : PHP, Ruby, Python, Java → **Javascript(Node.js)**
 - 하나의 언어 → FrontEnd + BackEnd + DB(MongoDB) = FullStack
- 2015년 ECMAScript 6(ES6) 버전 출시 🚀
- 2016년 프론트엔드 프레임워크(React, Vue, Angular) 대중화, SPA 개발 활성화 ☀️
- 현재 자바스크립트는 프론트엔드 및 백엔드 개발 분야에서 널리 사용되며, Node.js를 이용한 서버 개발도 활발하게 이루어지고 있습니다.

▼ (2) JS 언어의 특징

▼ [1] 객체 지향 프로그래밍 지원

자바스크립트는 객체 지향 프로그래밍이 가능한 언어로서, 객체를 생성하고 관리하는 데에 강점이 있습니다. 이를 통해 복잡한 기능을 구현할 수 있습니다.



객체지향 vs 절차지향

절차지향 프로그래밍은 순서대로 실행되는 일련의 과정을 단계적으로 기술하여 프로그램을 만드는 것이고, 객체지향 프로그래밍은 데이터와 함수를 객체라는 그룹으로 묶어서 처리하는 방법입니다.



예를 들어, **절차지향 프로그래밍**에서는 첫 번째 일을 처리한 다음에 두 번째 일을 처리하고, 그 다음에 세 번째 일을 처리하는 순서대로 프로그램을 만듭니다.



반면 **객체지향 프로그래밍**에서는 데이터와 함수를 객체라는 그룹으로 묶어서 처리합니다. 객체는 상태와 행동을 가지고 있으며, 상태는 객체의 속성 (property)이라고도 합니다.

✓ JS에서 객체를 만들 때는?

자바스크립트에서 객체를 만들 때는 중괄호({})를 사용하여 객체를 만들고, 각 속성을 선표(,)로 구분하여 추가합니다. 속성은 이름과 값을 가지며, 이름은 문자열로 작성하고, 값은 다양한 데이터 타입을 사용할 수 있습니다. 객체 내부의 함수를 메소드(method)라고 부르며, 속성의 값으로 함수를 추가할 수 있습니다. 객체를 만들어서 필요한 속성과 메소드를 추가하여 사용합니다.

▼ [2] 동적 타이핑

자바스크립트는 **동적 타이핑**을 지원하는 언어로서, 변수를 **선언**할 때 타입을 지정하지 않습니다. 이것은 **런타임 시점**에 변수에 할당되는 값에 따라 자동으로 데이터 타입이 결정된다는 것을 의미합니다. 예를 들어, 아래와 같은 코드를 살펴보겠습니다.

선언과 **할당**이라는 단어에 주목해주세요! 🤖

```
let myVariable = "Hello, world!"; // 변수 선언 및 문자열 데이터 타입으로 할당
console.log(typeof myVariable); // "string" 출력

myVariable = 123; // 숫자 데이터 타입으로 할당
console.log(typeof myVariable); // "number" 출력
```

```
myVariable = true; // 불리언 데이터 타입으로 할당
console.log(typeof myVariable); // "boolean" 출력
```



런타임(Runtime)?

런타임 시점이란, 프로그램이 실행되는 동안의 시점을 의미합니다. 즉, 코드가 실행되는 동안에 발생하는 시점을 말합니다. 반대의 의미로는 컴파일 시점이 있어요.

▼ [3] 함수형 프로그래밍 지원

자바스크립트는 함수형 프로그래밍을 지원하는 언어로서, 함수를 일급 객체로 취급하고, 고차 함수를 지원합니다. 이를 통해 코드의 재사용성과 가독성을 높일 수 있습니다.



일급객체란?(2주차에서 좀 더 자세히 배울 예정이에요!)

일급 객체(first-class object)란, **함수를 일반 값과 마찬가지로** 변수에 할당하거나, 함수의 인자로 전달하거나, 함수의 반환값으로 사용할 수 있는 객체를 의미합니다.

👉 코드예시

```
// 함수를 변수에 할당
const add = function(a, b) {
  return a + b;
}

// 함수를 인자로 받는 함수
function calculate(func, a, b) {
  return func(a, b);
}

// 함수를 반환하는 함수
function getAddFunction() {
  return add;
}

// 함수를 일급 객체로 다루어 코드의 재사용성을 높임
console.log(calculate(add, 2, 3)); // 5
console.log(getAddFunction()(2, 3)); // 5
```



고차함수란?

고차 함수(higher-order function)란, 함수를 인자로 받거나, 함수를 반환하는 함수를 의미합니다.

```
// 고차 함수 예시: 함수를 인자로 받는 함수
function operate(func, a, b) {
  return func(a, b);
}

function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  return a * b;
}

console.log(operate(add, 2, 3)); // 5
console.log(operate(multiply, 2, 3)); // 6
```

▼ [4] 비동기 처리

비동기 처리는 작업을 순차적으로 기다리지 않고, 병렬로 처리할 수 있도록 하는 방식이에요. 지금은 이해하기 어려우니, 이후 과정을 통해 익히도록 합시다 😊

▼ [5] 클라이언트 측 및 서버 측 모두에서 사용 가능

자바스크립트는 클라이언트 측에서만 사용되는 것이 아니라, Node.js를 이용하여 서버 측에서도 사용됩니다. 이를 통해 웹 개발 전반에 걸쳐 자바스크립트를 활용할 수 있습니다.

2. 기본 문법

▼ (1) 변수와 상수



변수가 무엇이고, 왜 필요할까요?

모든 프로그래밍 언어는 기억하고 싶은 값을 메모리에 저장하고, 저장된 값을 읽어 들여 **재사용**

하기 위해 변수라는 메커니즘을 이용합니다.

만약, 변수를 이용해서 재사용을 하지 않는다면, 우리는 “10 + 20 + 30 = 60”이라는 식에서 60이라는 결과값을 얻기 위해서 항상 저렇게 긴 식을 써줘야 하겠죠. 변수를 사용한다면 60이라는 결과만 변수에 담아놓고, 계속해서 재사용하기만 하면 돼요 😊

(자세한 매커니즘은 3주차에 배우게 됩니다)



변수의 5가지 주요 개념

변수 이름 : 저장된 값의 고유 이름

변수 값 : 변수에 저장된 값

변수 할당 : 변수에 값을 저장하는 행위

변수 선언 : 변수를 사용하기 위해 컴퓨터에 알리는 행위

변수 참조 : 변수에 할당된 값을 읽어오는 것

자바스크립트에서 변수는 **var**, **let**, **const** 세 가지 방법으로 선언할 수 있습니다. **var**은 예전부터 사용되던 방법입니다. **let**과 **const**는 ECMAScript 6(ES6)에서 새로 도입된 방법입니다.

```
// var로 변수 선언
var myVar = "Hello World";
console.log(myVar); // "Hello World"

// let으로 변수 선언
let myLet = "Hello World";
console.log(myLet); // "Hello World"

// const로 상수 선언
const myConst = "Hello World";
console.log(myConst); // "Hello World"
```

var는 같은 이름의 변수를 여러 번 선언해도 오류가 발생하지 않고, 가장 마지막에 선언한 값으로 변수가 덮어씌워집니다. **let**과 **const**는 같은 이름의 변수를 두 번 선언하면

오류가 발생합니다. `const` 는 선언 후에 값을 변경할 수 없는 상수를 선언할 때 사용합니다.

```
// var로 변수 덮어쓰기
var myVar = "Hello";
var myVar = "World";
console.log(myVar); // "World"

// let으로 변수 덮어쓰기
let myLet = "Hello";
myLet = "World"; // 기존 값을 덮어쓰기
console.log(myLet); // "World"

// const로 상수 선언 후 값 변경하기
const myConst = "Hello";
myConst = "World"; // 오류 발생
console.log(myConst);
```

▼ (2) 데이터 타입과 형 변환

▼ [1] 데이터 타입

1. 숫자(Number)

a. 정수형 숫자(Integer)

```
let num1 = 10;
console.log(num1); // 10
console.log(typeof num1); // "number"
```

b. 실수형 숫자(Float)

```
let num2 = 3.14;
console.log(num2); // 3.14
console.log(typeof num2); // "number"
```

c. 지수형 숫자(Exponential)

```
let num3 = 2.5e5; // 2.5 x 10^5
console.log(num3); // 250000
console.log(typeof num3); // "number"
```

d. NaN(Not a Number)

```
let num4 = "Hello" / 2;
console.log(num4); // NaN
console.log(typeof num4); // "number"
```



NaN(Not a Number)은 자바스크립트에서 숫자가 아님을 나타내는 값입니다. 보통 수학적으로 정의되지 않는 계산을 수행하거나, 숫자가 아닌 값을 숫자로 변환하려고 할 때 발생합니다.

e. Infinity

```
let num5 = 1 / 0;
console.log(num5); // Infinity
console.log(typeof num5); // "number"
```

f. Infinity

```
let num6 = -1 / 0;
console.log(num6); // -Infinity
console.log(typeof num6); // "number"
```

2. 문자열(String)

문자열은 문자의 나열입니다. 작은 따옴표(')나 큰 따옴표(")로 감싸서 표현합니다.

```
let name = 'Alice';
let message = "Hello, world!";
```

1. 문자열 길이(length) 확인하기

```
let str = "Hello, world!";
console.log(str.length); // 13
```

2. 문자열 결합(concatenation)

```
let str1 = "Hello, ";
let str2 = "world!";
```



```
let result = str1.concat(str2);
console.log(result); // "Hello, world!"
```

3. 문자열 자르기(substr, slice)

```
let str = "Hello, world!";
console.log(str.substr(7, 5)); // "world"
console.log(str.slice(7, 12)); // "world"
```

4. 문자열 검색(search)

```
let str = "Hello, world!";
console.log(str.search("world")); // 7
```

5. 문자열 대체(replace)

```
let str = "Hello, world!";
let result = str.replace("world", "JavaScript");
console.log(result); // "Hello, JavaScript!"
```

6. 문자열 분할(split)

```
let str = "apple, banana, kiwi";
let result = str.split(",");
console.log(result); // ["apple", " banana", " kiwi"]
```

3. 불리언(Boolean)

불리언은 참(true)과 거짓(false)을 나타내는 데이터 타입입니다.

```
let bool1 = true;
console.log(bool1); // true
console.log(typeof bool1); // "boolean"
```

```
let bool2 = false;
console.log(bool2); // false
console.log(typeof bool2); // "boolean"
```

불리언 데이터 타입은 조건문(if, else, switch 등)과 논리 연산자(&&, ||, !)와 함께 많이 사용됩니다. 예를 들어, 다음과 같은 코드를 작성할 수 있습니다.

```

let x = 10;
let y = 5;

if (x > y) {
  console.log("x is greater than y");
} else {
  console.log("x is less than or equal to y");
}

let a = true;
let b = false;

console.log(a && b); // false
console.log(a || b); // true
console.log(!a); // false

```

위 코드에서는 if 조건문을 사용하여 x가 y보다 큰 경우에는 "x is greater than y"를 출력하고, 그렇지 않은 경우에는 "x is less than or equal to y"를 출력합니다. 또한, 논리 연산자를 사용하여 a와 b의 논리적인 AND(&&)와 OR(||) 연산을 수행하고, NOT(!) 연산을 수행합니다.

4. undefined

undefined는 값이 할당되지 않은 변수를 의미합니다.

```

let x;
console.log(x); // undefined

```

5. null

null은 값이 존재하지 않음을 의미합니다. undefined와는 다릅니다.

```

let y = null;

```

6. 객체(Object)

자바스크립트에서는 객체가 매우 중요한 역할을 합니다. 객체는 속성과 메소드를 가지는 컨테이너입니다. 중괄호({})를 사용하여 객체를 생성합니다. 1주차 말미에 자세히 배웁니다 😊

```

let person = { name: 'Alice', age: 20 };
person.name // 'Alice'
person.age // 20

```

7. 배열(Array)

배열은 여러 개의 데이터를 순서대로 저장하는 데이터 타입입니다. 대괄호([])를 사용하여 배열을 생성합니다. 역시 1주차 말미에 자세히 배워요 😊

```
let numbers = [1, 2, 3, 4, 5];
let fruits = ['apple', 'banana', 'orange'];
```

▼ [2] 형 변환

자바스크립트에서는 다양한 자료형을 다룰 수 있습니다. 그리고 이 자료형들은 서로 형변환이 가능합니다. 이번에는 자바스크립트의 형 변환에 대해서 알아보겠습니다.

[2]-1. 암시적 형 변환(implicit coercion)

암시적 형 변환은 자바스크립트에서 자동으로 수행되는 형 변환이며, 일반적으로 연산자를 사용할 때 발생합니다.

[2]-1-1. 문자열 변환

```
console.log(1 + "2");    // "12"
console.log("1" + true); // "1true"
console.log("1" + {});   // "1[object Object]"
console.log("1" + null);  // "1null"
console.log("1" + undefined); // "1undefined"
```

위의 예제에서는 문자열과 다른 자료형이 연산자로 결합되어 있습니다. 이 경우에는 자바스크립트는 다른 자료형을 문자열로 변환한 후 연산을 수행합니다.

[2]-1-2. 숫자 변환

```
console.log(1 - "2");    // -1
console.log("2" * "3");  // 6
console.log(4 + +"5");   // 9
```

위의 예제에서는 연산자를 사용할 때, 문자열을 숫자로 변환합니다. 이때, 빈 문자열("")이나 공백 문자열(" ")은 0으로 변환됩니다.

[2]-1-3. 불리언 변환

```
console.log(Boolean(0)); // false
console.log(Boolean("")); // false
console.log(Boolean(null)); // false
console.log(Boolean(undefined)); // false
```

```
console.log(Boolean(NaN)); // false
console.log(Boolean("false")); // true
console.log(Boolean({})); // true
```

위의 예제에서는 Boolean() 함수를 사용하여 불리언 값으로 변환합니다. 이때, 0, 빈 문자열(""), null, undefined, NaN은 false로 변환됩니다. 그 외의 값은 true로 변환됩니다.

[2]-2. 명시적 형 변환(explicit coercion)

명시적 형 변환은 **개발자가 직접** 자료형을 변환하는 것을 말합니다.

[2]-2-1. 문자열 변환

```
console.log(String(123)); // "123"
console.log(String(true)); // "true"
console.log(String(false)); // "false"
console.log(String(null)); // "null"
console.log(String(undefined)); // "undefined"
console.log(String({})); // "[object Object]"
```

위의 예제에서는 String() 함수를 사용하여 다른 자료형을 문자열로 변환합니다.

[2]-2-2. 숫자 변환

```
console.log(Number("123")); // 123
console.log(Number("")); // 0
console.log(Number(" ")); // 0
console.log(Number(true)); // 1
console.log(Number(false)); // 0
```

▼ (3) 연산자

자바스크립트에서는 다양한 연산자를 제공하여 변수나 상수를 다양한 방법으로 조작할 수 있습니다. 이번에는 자바스크립트의 연산자에 대해서 알아보겠습니다.

[1] 산술 연산자(arithmetic operators)

[1]-1. 더하기 연산자(+)

```
console.log(2 + 3); // 5
console.log("2" + "3"); // "23"
console.log("2" + 3); // "23"
console.log(2 + "3"); // "23"
```

위의 예제에서는 더하기 연산자를 사용하여 숫자나 문자열을 더할 수 있습니다. 이때, 더하기 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 문자열로 변환합니다.

[1]-2. 빼기 연산자(-)

```
console.log(5 - 2);           // 3
console.log("5" - "2");       // 3
console.log("5" - 2);         // 3
console.log(5 - "2");         // 3
console.log("five" - 2);      // NaN
```

위의 예제에서는 빼기 연산자를 사용하여 숫자를 뺄 수 있습니다. 이때, 빼기 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[1]-3. 곱하기 연산자(*)

```
console.log(2 * 3);           // 6
console.log("2" * "3");       // 6
console.log("2" * 3);         // 6
console.log(2 * "3");         // 6
console.log("two" * 3);       // NaN
```

위의 예제에서는 곱하기 연산자를 사용하여 숫자를 곱할 수 있습니다. 이때, 곱하기 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[1]-4. 나누기 연산자(/)

```
console.log(6 / 3);           // 2
console.log("6" / "3");       // 2
console.log("6" / 3);         // 2
console.log(6 / "3");         // 2
console.log("six" / 3);       // NaN
```

위의 예제에서는 나누기 연산자를 사용하여 숫자를 나눌 수 있습니다. 이때, 나누기 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[1]-5. 나누기 연산자(%)

```
console.log(7 % 3);           // 1
console.log("7" % "3");       // 1
console.log("7" % 3);         // 1
console.log(7 % "3");         // 1
console.log("seven" % 3);     // NaN
```

위의 예제에서는 나머지 연산자를 사용하여 나눗셈의 나머지를 구할 수 있습니다. 이때, 나머지 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[2] 할당 연산자(assignment operators)

[2]-1. 등호 연산자(=)

```
let x = 10;
console.log(x);    // 10

x = 5;
console.log(x);    // 5
```

위의 예제에서는 등호 연산자를 사용하여 변수에 값을 할당할 수 있습니다.

[2]-2. 더하기 등호 연산자(+=)

```
let x = 10;
console.log(x);    // 10

x += 5;
console.log(x);    // 15
```

위의 예제에서는 더하기 등호 연산자를 사용하여 변수에 값을 더할 수 있습니다.

[2]-3. 빼기 등호 연산자(-=)

```
let x = 10;
console.log(x);    // 10

x -= 5;
console.log(x);    // 5
```

위의 예제에서는 빼기 등호 연산자를 사용하여 변수에서 값을 뺄 수 있습니다.

[2]-4. 빼기 등호 연산자(-=)

```
let x = 10;
console.log(x);    // 10

x *= 5;
console.log(x);    // 50
```

위의 예제에서는 곱하기 등호 연산자를 사용하여 변수에 값을 곱할 수 있습니다.

[2]-5. 나누기 등호 연산자(/=)

```
let x = 10;
console.log(x);    // 10

x /= 5;
console.log(x);    // 2
```

위의 예제에서는 나누기 등호 연산자를 사용하여 변수에서 값을 나눌 수 있습니다.

[2]-6. 나머지 등호 연산자(%)

```
let x = 10;
console.log(x);    // 10

x %= 3;
console.log(x);    // 1
```

위의 예제에서는 나머지 등호 연산자를 사용하여 변수에서 값을 나눈 나머지를 구할 수 있습니다.

[3] 비교 연산자(comparison operators)

[3]-1. 일치 연산자(===)

```
console.log(2 === 2);    // true
console.log("2" === 2);  // false
console.log(2 === "2");  // false
```

위의 예제에서는 일치 연산자를 사용하여 두 값이 같은지 비교할 수 있습니다. 이때, 일치 연산자는 자료형까지 비교합니다.

[3]-2. 불일치 연산자(!==)

```
console.log(2 !== 2);    // false
console.log("2" !== 2);  // true
console.log(2 !== "2");  // true
```

위의 예제에서는 불일치 연산자를 사용하여 두 값이 다른지 비교할 수 있습니다. 이때, 불일치 연산자는 자료형까지 비교합니다.

[3]-3. 작다(<) 연산자

```
console.log(2 < 3);    // true
console.log(2 < "3");  // true
console.log("2" < 3);  // true
```

위의 예제에서는 작다 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 작다 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[3]-4. 크다(>) 연산자

```
console.log(2 > 3);    // false
console.log(2 > "3");  // false
console.log("2" > 3);  // false
```

위의 예제에서는 크다 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 크다 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[3]-5. 작거나 같다(<=) 연산자

```
console.log(2 <= 3);   // true
console.log(2 <= "3"); // true
console.log("2" <= 3); // true
console.log(2 <= 2);   // true
```

위의 예제에서는 작거나 같다 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 작거나 같다 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[3]-6. 크거나 같다(>=) 연산자

```
console.log(2 >= 3);   // false
console.log(2 >= "3"); // false
console.log("2" >= 3); // false
console.log(2 >= 2);   // true
```

위의 예제에서는 크거나 같다 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 크거나 같다 연산자는 숫자와 문자열을 함께 사용할 경우, 자동으로 숫자로 변환합니다.

[4] 논리 연산자(logical operators)

[4]-1. 논리곱(&&) 연산자


```
console.log(true && true);    // true
console.log(true && false);   // false
console.log(false && true);   // false
console.log(false && false);  // false
```

위의 예제에서는 논리곱 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 논리곱 연산자는 두 값이 모두 true일 경우에만 true를 반환합니다.

[4]-2. 논리합(||) 연산자

```
console.log(true || true);   // true
console.log(true || false);  // true
console.log(false || true);  // true
console.log(false || false); // false
```

위의 예제에서는 논리합 연산자를 사용하여 두 값을 비교할 수 있습니다. 이때, 논리합 연산자는 두 값 중 하나라도 true일 경우 true를 반환합니다.

[4]-3. 논리부정(!) 연산자

```
console.log(!true);    // false
console.log(!false);   // true
console.log(!(2 > 1));  // false
```

위의 예제에서는 논리부정 연산자를 사용하여 값을 반대로 바꿀 수 있습니다. 이때, 논리부정 연산자는 true를 false로, false를 true로 바꿉니다.

[5] 삼항 연산자(ternary operator)

[5]-1. 삼항 연산자(?:)

```
let x = 10;
let result = (x > 5) ? "크다" : "작다";
console.log(result);    // "크다"
```

위의 예제에서는 삼항 연산자를 사용하여 조건에 따라 값을 선택할 수 있습니다. 이때, 삼항 연산자는 조건식 ? true일 때의 값 : false일 때의 값 형태로 사용합니다.

[6] 타입 연산자(type operators)

[6]-1. typeof 연산자

```
console.log(typeof 123);    // "number"
console.log(typeof "123");  // "string"
console.log(typeof true);   // "boolean"
console.log(typeof undefined); // "undefined"
console.log(typeof null);   // "object"
console.log(typeof {});     // "object"
console.log(typeof []);     // "object"
console.log(typeof function(){}); // "function"
```

위의 예제에서는 typeof 연산자를 사용하여 값의 자료형을 확인할 수 있습니다. 이때, typeof 연산자는 원시 자료형의 경우, 해당 자료형의 이름을, 객체나 함수의 경우, "object" 또는 "function"을 반환합니다. typeof null의 경우 "object"를 반환하는 버그가 있습니다.

▼ (4) 함수

자바스크립트에서는 함수를 정의하여 코드의 재사용성을 높일 수 있습니다. 이번에는 자바스크립트의 함수에 대해서 알아보겠습니다.

[1] 함수 정의하기

[1]-1. 함수 선언문(function declaration)

```
function add(x, y) {
  return x + y;
}

console.log(add(2, 3)); // 5
```

위의 예제에서는 function 키워드를 사용하여 add라는 함수를 선언하였습니다. 함수 선언문을 사용하면 함수를 미리 정의해두고, 필요할 때 호출할 수 있습니다.

[1]-2. 함수 표현식(function expression)

```
let add = function(x, y) {
  return x + y;
}

console.log(add(2, 3)); // 5
```

위의 예제에서는 function 키워드를 사용하여 add라는 변수에 함수를 할당하였습니다. 함수 표현식을 사용하면 함수를 변수에 할당하여 익명 함수를 생성할 수 있습니다.

[2] 함수 호출하기

[2]-1. 함수 호출하기

```
function add(x, y) {  
  return x + y;  
}  
  
console.log(add(2, 3)); // 5
```

위의 예제에서는 add라는 함수를 호출하여 결과값을 반환합니다. 함수를 호출할 때는 함수 이름 뒤에 괄호를 사용합니다.

[3] 함수 매개변수와 반환값

[3]-1. 함수 매개변수

```
function add(x, y) {  
  return x + y;  
}  
  
console.log(add(2, 3)); // 5
```



위의 예제에서는 add라는 함수가 x와 y라는 두 개의 매개변수를 받아들입니다. 함수를 호출할 때는 매개변수에 값을 전달합니다.

[3]-2. 함수 반환값

```
function add(x, y) {  
  return x + y;  
}  
  
let result = add(2, 3);  
console.log(result); // 5
```

위의 예제에서는 add라는 함수가 x와 y라는 두 개의 매개변수를 받아들이고, 이를 더한 값을 반환합니다. 함수를 호출한 결과값을 변수에 할당하여 사용할 수 있습니다.

[4] 함수 스코프

- scope** 미국식 [skoup]  영국식 [skeup]  ★
1. [명사] (무엇을 하거나 이룰 수 있는) 기회[여지/능력] (=potential)
 2. [명사] (주제조직활동 등이 다루는) 범위
 3. [동사][비격식] 살살이[자세히] 살피다

[4]-1. 전역 스코프(global scope)

```
let x = 10;

function printX() {
  console.log(x);
}

printX(); // 10
```

위의 예제에서는 전역 스코프에서 변수 x를 선언하고, 함수 printX에서 변수 x를 참조합니다. 전역 스코프에서 선언된 변수는 어디에서든지 참조할 수 있습니다.

[4]-2. 지역 스코프(local scope)

```
function printX() {
  let x = 10;
  console.log(x);
}

printX(); //
```

위의 예제에서는 지역 스코프에서 변수 x를 선언하고, 함수 printX에서 변수 x를 참조합니다. 지역 스코프에서 선언된 변수는 해당 함수 내에서만 참조할 수 있습니다.

[4]-3. 블록 스코프(block scope)

```
if (true) {
  let x = 10;
  console.log(x);
}

console.log(x); // ReferenceError: x is not defined
```

위의 예제에서는 if문 내에서 변수 x를 선언하고, 이를 출력합니다. if문 내에서 선언된 변수는 해당 블록 내에서만 참조할 수 있습니다.

[5] 화살표 함수

[5]-1. 기본적인 화살표 함수

```
let add = (x, y) => {  
  return x + y;  
}  
  
console.log(add(2, 3)); // 5
```

위의 예제에서는 화살표 함수를 사용하여 add라는 함수를 선언하였습니다. 화살표 함수를 사용하면 함수의 선언이 간결해집니다.

[5]-2. 한 줄로 된 화살표 함수

```
let add = (x, y) => x + y;  
  
console.log(add(2, 3)); // 5
```

위의 예제에서는 한 줄로 된 화살표 함수를 사용하여 add라는 함수를 선언하였습니다. 함수 내부에 return문이 한 줄로 작성될 경우, 중괄호와 return 키워드를 생략할 수 있습니다.

[5]-3. 매개변수가 하나인 화살표 함수

```
let square = x => x * x;  
  
console.log(square(3)); // 9
```

위의 예제에서는 매개변수가 하나인 화살표 함수를 사용하여 square라는 함수를 선언하였습니다. 매개변수가 하나일 경우에는 괄호를 생략할 수 있습니다.

3. 문

▼ (1) 조건문(if, else if, else, switch)

자바스크립트에서는 조건문을 사용하여 특정 조건을 만족하는 경우에만 코드를 실행할 수 있습니다. 이번에는 자바스크립트의 조건문에 대해서 알아보겠습니다.

1. if문

1-1. 기본적인 if문

```
let x = 10;

if (x > 0) {
  console.log("x는 양수입니다.");
}
```

위의 예제에서는 변수 x가 양수인지를 판별하여, 양수인 경우 "x는 양수입니다."라는 메시지를 출력합니다. if문은 조건이 참인 경우에만 코드를 실행합니다.

1-2. if-else문

```
let x = -10;

if (x > 0) {
  console.log("x는 양수입니다.");
} else {
  console.log("x는 음수입니다.");
}
```

위의 예제에서는 변수 x가 양수인지를 판별하여, 양수인 경우 "x는 양수입니다."라는 메시지를 출력하고, 그렇지 않은 경우 "x는 음수입니다."라는 메시지를 출력합니다. if-else 문은 조건이 참인 경우와 거짓인 경우 각각 다른 코드를 실행합니다.

1-3. if-else if-else문

```
let x = 0;

if (x > 0) {
  console.log("x는 양수입니다.");
} else if (x < 0) {
  console.log("x는 음수입니다.");
} else {
  console.log("x는 0입니다.");
}
```

위의 예제에서는 변수 x가 양수인지, 음수인지를 판별하여, 0인 경우를 포함해 각각 다른 메시지를 출력합니다. if-else if-else문은 여러 개의 조건을 순서대로 비교하여, 해당하는 조건에 맞는 코드를 실행합니다.

2. switch문

```
let fruit = "사과";

switch (fruit) {
  case "사과":
    console.log("사과는 빨간색입니다.");
    break;
  case "바나나":
    console.log("바나나는 노란색입니다.");
    break;
  case "오렌지":
    console.log("오렌지는 주황색입니다.");
    break;
  default:
    console.log("해당하는 과일이 없습니다.");
    break;
}
```

위의 예제에서는 switch문을 사용하여 과일의 종류에 따라 색상을 출력합니다. switch문은 변수의 값에 따라 여러 개의 경우(case) 중 하나를 선택하여 해당하는 코드를 실행합니다. default는 모든 경우에 맞지 않는 경우에 실행되는 코드를 작성합니다.

3. 삼항 연산자

```
let age = 20;
let message = (age >= 18) ? "성인입니다." : "미성년자입니다.";
console.log(message); // "성인입니다."
```

위의 예제에서는 삼항 연산자를 사용하여 변수 age가 18세 이상인 경우 "성인입니다.", 그렇지 않은 경우 "미성년자입니다."라는 메시지를 출력합니다. 삼항 연산자는 if문과 비슷한 역할을 하며, 조건이 참인 경우와 거짓인 경우 각각 다른 값을 반환합니다.

4. 조건문의 중첩

```
let age = 20;
let gender = "여성";

if (age >= 18) {
  if (gender === "남성") {
    console.log("성인 남성입니다.");
  } else {
    console.log("성인 여성입니다.");
  }
} else {
  console.log("미성년자입니다.");
}
```

```
console.log("미성년자입니다.");  
}
```

위의 예제에서는 중첩된 if문을 사용하여 성별에 따라 성인 여부를 판별합니다. 조건문 안에 또 다른 조건문을 사용하여 복잡한 조건을 판별할 수 있습니다.

5. 조건부 실행

```
let x = 10;  
  
(x > 0) && console.log("x는 양수입니다.");
```

위의 예제에서는 조건부 실행을 사용하여 변수 x가 양수인 경우에만 "x는 양수입니다."라는 메시지를 출력합니다. && 연산자를 사용하여 조건부 실행을 할 수 있습니다.

6. 삼항 연산자와 단축 평가

```
let x;  
let y = x || 10;  
  
console.log(y); // 10
```

위의 예제에서는 삼항 연산자를 사용하여 변수 x가 존재하지 않는 경우 기본값으로 10을 사용합니다. || 연산자를 사용하여 단축 평가(short-circuit evaluation)를 할 수 있습니다. 변수 x가 존재하지 않는 경우, || 연산자는 false 값을 반환하고, 기본값으로 지정한 10을 반환합니다.

7. falsy한 값과 truthy한 값

```
if (0) {  
  console.log("이 코드는 실행되지 않습니다.");  
}  
  
if ("") {  
  console.log("이 코드는 실행되지 않습니다.");  
}  
  
if (null) {  
  console.log("이 코드는 실행되지 않습니다.");  
}  
  
if (undefined) {  
  console.log("이 코드는 실행되지 않습니다.");  
}
```



```

}

if (NaN) {
  console.log("이 코드는 실행되지 않습니다.");
}

if (false) {
  console.log("이 코드는 실행되지 않습니다.");
}

```

위의 예제에서는 falsy한 값들을 사용하여 if문의 조건을 만족시키지 못하도록 합니다. 0, 빈 문자열, null, undefined, NaN, false는 falsy한 값으로, if문의 조건을 만족시키지 못합니다. 그 외의 값들은 모두 truthy한 값으로, if문의 조건을 만족시킵니다.

▼ (2) 반복문

자바스크립트에서는 반복문을 사용하여 특정 코드를 반복해서 실행할 수 있습니다. 이번에는 자바스크립트의 반복문에 대해서 알아보겠습니다.

1. for문

1-1. 기본적인 for문

```

for (let i = 0; i < 10; i++) {
  console.log(i);
}

```

위의 예제에서는 for문을 사용하여 0부터 9까지의 숫자를 출력합니다. for문은 초기값, 조건식, 증감식을 사용하여 반복 횟수를 제어합니다.

1-2. 배열과 함께 사용하는 for문

```

let numbers = [1, 2, 3, 4, 5];

for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}

```

위의 예제에서는 배열 numbers와 함께 for문을 사용하여 배열의 요소를 출력합니다. 배열의 요소 개수만큼 반복하여 실행합니다.

1-3. for...in문

```
let person = { name: "John", age: 30, gender: "male" };

for (let key in person) {
  console.log(key + ": " + person[key]);
}
```

위의 예제에서는 for...in문을 사용하여 객체 person의 프로퍼티를 출력합니다. for...in문은 객체의 프로퍼티를 순서대로 접근할 수 있습니다.

2. while문

```
let i = 0;

while (i < 10) {
  console.log(i);
  i++;
}
```

위의 예제에서는 while문을 사용하여 0부터 9까지의 숫자를 출력합니다. while문은 조건식이 참인 경우에만 코드를 반복해서 실행합니다.

3. do...while문

```
let i = 0;

do {
  console.log(i);
  i++;
} while (i < 10);
```

위의 예제에서는 do...while문을 사용하여 0부터 9까지의 숫자를 출력합니다. do...while문은 일단 한 번은 코드를 실행하고, 그 후에 조건식을 체크하여 반복 여부를 결정합니다.

4. break문과 continue문

4-1. break문

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break;
  }
}
```

```
console.log(i);  
}
```

위의 예제에서는 for문과 함께 break문을 사용하여 0부터 4까지의 숫자만 출력합니다. break문은 반복문을 종료합니다.

4-2. continue문

```
for (let i = 0; i < 10; i++) {  
  if (i === 5) {  
    continue;  
  }  
  console.log(i);  
}
```

위의 예제에서는 for문과 함께 continue문을 사용하여 5를 제외한 0부터 9까지의 숫자를 출력합니다.

4. 배열, 객체 기초

▼ 객체와 객체 메소드

자바스크립트에서는 객체(Object)를 사용하여 여러 개의 값을 하나의 변수에 담고 관리할 수 있습니다. 이번에는 자바스크립트의 객체와 객체 메소드에 대해서 알아보겠습니다.

1. 객체 생성

1-1. 기본적인 객체 생성

```
let person = {  
  name: "홍길동",  
  age: 30,  
  gender: "남자"  
};
```

위의 예제에서는 객체 person을 생성합니다. 객체를 만들 때는 중괄호({})를 사용하며, 속성과 값을 콜론(:)으로 구분하여 작성합니다. 각 속성과 값은 쉼표(,)로 구분합니다.

1-2. 생성자 함수를 사용한 객체 생성

```
function Person(name, age, gender) {
  this.name = name;
  this.age = age;
  this.gender = gender;
}

let person1 = new Person("홍길동", 30, "남자");
let person2 = new Person("홍길순", 25, "여자");
```

위의 예제에서는 생성자 함수 Person()을 사용하여 객체 person1과 person2를 생성합니다. 생성자 함수를 사용하면 객체를 일괄적으로 생성할 수 있습니다.

2. 객체 속성 접근

```
let person = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

console.log(person.name); // "홍길동"
console.log(person.age); // 30
console.log(person.gender); // "남자"
```

위의 예제에서는 객체 person의 속성에 접근하여 값을 출력합니다. 객체의 속성에 접근할 때는 점(.)을 사용하여 속성 이름을 입력합니다.

3. 객체 메소드

3-1. Object.keys() 메소드

```
let person = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

let keys = Object.keys(person);

console.log(keys); // ["name", "age", "gender"]
```

위의 예제에서는 Object.keys() 메소드를 사용하여 객체 person의 속성 이름을 배열로 반환합니다. Object.keys() 메소드는 객체의 속성 이름을 배열로 반환합니다.

3-2. Object.values() 메소드

```
let person = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

let values = Object.values(person);

console.log(values);    // ["홍길동", 30, "남자"]
```

위의 예제에서는 Object.values() 메소드를 사용하여 객체 person의 속성 값들을 배열로 반환합니다. Object.values() 메소드는 객체의 속성 값들을 배열로 반환합니다.

3-3. Object.entries() 메소드

```
let person = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

let entries = Object.entries(person);

console.log(entries);
```

위의 예제에서는 Object.entries() 메소드를 사용하여 객체 person의 속성 이름과 속성 값들을 2차원 배열로 반환합니다. Object.entries() 메소드는 객체의 속성 이름과 속성 값들을 2차원 배열로 반환합니다.

3-4. Object.assign() 메소드

```
let person = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

let newPerson = Object.assign({}, person, { age: 35 });

console.log(newPerson);    // { name: "홍길동", age: 35, gender: "남자" }
```

위의 예제에서는 Object.assign() 메소드를 사용하여 새로운 객체 newPerson을 만듭니다. Object.assign() 메소드는 기존 객체를 복사하여 새로운 객체를 만듭니다.

3-5. 객체 비교

```
let person1 = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

let person2 = {
  name: "홍길동",
  age: 30,
  gender: "남자"
};

console.log(person1 === person2);    // false
console.log(JSON.stringify(person1) === JSON.stringify(person2));    // true
```

위의 예제에서는 두 개의 객체를 생성하고, 객체 비교를 합니다. 객체를 비교할 때는 일반적으로 === 연산자를 사용할 수 없습니다. 대신 JSON.stringify() 함수를 사용하여 객체를 문자열로 변환한 후, 문자열 비교를 합니다.

3-6. 객체 병합

```
let person1 = {
  name: "홍길동",
  age: 30
};

let person2 = {
  gender: "남자"
};

let mergedPerson = {...person1, ...person2};

console.log(mergedPerson);    // { name: "홍길동", age: 30, gender: "남자" }
```

위의 예제에서는 객체 병합을 합니다. 객체 병합을 할 때는 전개 연산자(...)를 사용합니다.

▼ 배열과 배열 메소드

자바스크립트에서는 배열(Array)을 사용하여 여러 개의 값을 저장하고 관리할 수 있습니다. 이번에는 자바스크립트의 배열과 배열 메소드에 대해서 알아보겠습니다.

1. 배열 생성

1-1. 기본적인 배열 생성

```
let fruits = ["사과", "바나나", "오렌지"];
```

위의 예제에서는 배열 fruits를 생성합니다. 배열을 만들 때는 대괄호([])를 사용하며, 각 요소는 쉼표(,)로 구분합니다.

1-2. 배열의 크기 지정

```
let numbers = new Array(5);
```

위의 예제에서는 크기가 5인 배열 numbers를 생성합니다. new Array()를 사용하여 배열의 크기를 지정할 수 있습니다.

2. 배열 요소 접근

```
let fruits = ["사과", "바나나", "오렌지"];

console.log(fruits[0]); // "사과"
console.log(fruits[1]); // "바나나"
console.log(fruits[2]); // "오렌지"
```

위의 예제에서는 배열 fruits의 첫 번째 요소부터 세 번째 요소까지 출력합니다. 배열의 요소에 접근할 때는 대괄호([]) 안에 인덱스 값을 넣습니다.

3. 배열 메소드

3-1. push() 메소드

```
let fruits = ["사과", "바나나"];

fruits.push("오렌지");

console.log(fruits); // ["사과", "바나나", "오렌지"]
```

위의 예제에서는 push() 메소드를 사용하여 배열 fruits의 끝에 "오렌지"를 추가합니다. push() 메소드는 배열의 끝에 요소를 추가합니다.

3-2. pop() 메소드

```
let fruits = ["사과", "바나나", "오렌지"];

fruits.pop();

console.log(fruits);    // ["사과", "바나나"]
```

위의 예제에서는 pop() 메소드를 사용하여 배열 fruits의 마지막 요소를 삭제합니다. pop() 메소드는 배열의 마지막 요소를 삭제합니다.

3-3. shift() 메소드

```
javascriptCopy code
let fruits = ["사과", "바나나", "오렌지"];

fruits.shift();

console.log(fruits);    // ["바나나", "오렌지"]
```

위의 예제에서는 shift() 메소드를 사용하여 배열 fruits의 첫 번째 요소를 삭제합니다. shift() 메소드는 배열의 첫 번째 요소를 삭제합니다.

3-4. unshift() 메소드

```
let fruits = ["바나나", "오렌지"];

fruits.unshift("사과");

console.log(fruits);    // ["사과", "바나나", "오렌지"]
```

위의 예제에서는 unshift() 메소드를 사용하여 배열 fruits의 맨 앞에 "사과"를 추가합니다. unshift() 메소드는 배열의 맨 앞에 요소를 추가합니다.

3-5. splice() 메소드

```
let fruits = ["사과", "바나나", "오렌지"];

fruits.splice(1, 1, "포도");

console.log(fruits);    // ["사과", "포도", "오렌지"]
```


위의 예제에서는 splice() 메소드를 사용하여 배열 fruits의 두 번째 요소를 삭제하고, 그 자리에 "포도"를 추가합니다. splice() 메소드는 배열의 요소를 삭제하거나, 새로운 요소를 추가할 수 있습니다.

3-6. slice() 메소드

```
let fruits = ["사과", "바나나", "오렌지"];

let slicedFruits = fruits.slice(1, 2);

console.log(slicedFruits); // ["바나나"]
```

위의 예제에서는 slice() 메소드를 사용하여 배열 fruits의 두 번째 요소부터 세 번째 요소까지 새로운 배열로 만듭니다. slice() 메소드는 배열의 일부분을 새로운 배열로 만듭니다.

3-7. forEach() 메소드

```
let numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number) {
  console.log(number);
});
```

위의 예제에서는 forEach() 메소드를 사용하여 배열 numbers의 모든 요소를 출력합니다. forEach() 메소드는 배열의 각 요소에 대해 콜백 함수를 실행합니다.

3-8. map() 메소드

```
let numbers = [1, 2, 3, 4, 5];

let squaredNumbers = numbers.map(function(number) {
  return number * number;
});

console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

위의 예제에서는 map() 메소드를 사용하여 배열 numbers의 모든 요소를 제공한 새로운 배열을 만듭니다. map() 메소드는 배열의 각 요소에 대해 콜백 함수를 실행하고, 그 결과를 새로운 배열로 반환합니다.

3-9. filter() 메소드

```
let numbers = [1, 2, 3, 4, 5];

let evenNumbers = numbers.filter(function(number) {
  return number % 2 === 0;
});

console.log(evenNumbers); // [2, 4]
```

위의 예제에서는 `filter()` 메소드를 사용하여 배열 `numbers`에서 짝수만 추출한 새로운 배열을 만듭니다. `filter()` 메소드는 배열의 각 요소에 대해 콜백 함수를 실행하고, 그 결과가 `true`인 요소만 새로운 배열로 반환합니다.

3-10. `reduce()` 메소드

```
let numbers = [1, 2, 3, 4, 5];

let sum = numbers.reduce(function(accumulator, currentValue) {
  return accumulator + currentValue;
}, 0);

console.log(sum);
```

3-11. `find()` 메소드

```
let numbers = [1, 2, 3, 4, 5];

let foundNumber = numbers.find(function(number) {
  return number > 3;
});

console.log(foundNumber); // 4
```

위의 예제에서는 `find()` 메소드를 사용하여 배열 `numbers`에서 3보다 큰 첫 번째 요소를 찾습니다. `find()` 메소드는 배열의 각 요소에 대해 콜백 함수를 실행하고, 그 결과가 `true`인 첫 번째 요소를 반환합니다.

3-12. `some()` 메소드

```
let numbers = [1, 2, 3, 4, 5];

let hasEvenNumber = numbers.some(function(number) {
  return number % 2 === 0;
});
```

```
console.log(hasEvenNumber);    // true
```

위의 예제에서는 `some()` 메소드를 사용하여 배열 `numbers`에서 짝수가 있는지 확인합니다. `some()` 메소드는 배열의 각 요소에 대해 콜백 함수를 실행하고, 그 결과가 `true`인 요소가 하나라도 있는지 확인합니다.

3-13. every() 메소드

```
let numbers = [2, 4, 6, 8, 10];

let isAllEvenNumbers = numbers.every(function(number) {
  return number % 2 === 0;
});

console.log(isAllEvenNumbers);    // true
```

위의 예제에서는 `every()` 메소드를 사용하여 배열 `numbers`의 모든 요소가 짝수인지 확인합니다. `every()` 메소드는 배열의 각 요소에 대해 콜백 함수를 실행하고, 그 결과가 `true`인 요소가 모든 요소인지 확인합니다.

3-14. sort() 메소드

```
let numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];

numbers.sort(function(a, b) {
  return a - b;
});

console.log(numbers);    // [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

위의 예제에서는 `sort()` 메소드를 사용하여 배열 `numbers`를 오름차순으로 정렬합니다. `sort()` 메소드는 배열의 요소를 정렬합니다.

3-15. reverse() 메소드

```
let numbers = [1, 2, 3, 4, 5];

numbers.reverse();

console.log(numbers);    // [5, 4, 3, 2, 1]
```

위의 예제에서는 reverse() 메소드를 사용하여 배열 numbers의 순서를 뒤집습니다. reverse() 메소드는 배열의 요소를 역순으로 정렬합니다.

5. 숙제

▼ 01 - 문자열 연습하기

1. 문자열 연습하기

<https://school.programmers.co.kr/learn/courses/30/lessons/12916>

문제 설명



코딩테스트의 요구사항입니다. 처음에는 눈에 잘 안들어올 수 있고, 실제 문제보다 설명이 난이도가 높게 느껴질 수 있습니다. 이런 경우 입출력 예를 함께 보면서 능동적으로 읽어가면 크게 도움이 됩니다.

대문자와 소문자가 섞여있는 문자열 s가 주어집니다. s에 'p'의 개수와 'y'의 개수를 비교해 같으면 True, 다르면 False를 return 하는 solution을 완성하세요. 'p', 'y' 모두 하나도 없는 경우는 항상 True를 리턴합니다. 단, 개수를 비교할 때 대문자와 소문자는 구별하지 않습니다.

예를 들어 s가 "pPoooyY"면 true를 return하고 "Pyy"라면 false를 return합니다.

제한사항



요구사항에 추가적으로 작용하는 제한사항입니다. 제한사항에 맞게 구현을 해야해서 꼭 읽어야 하지만, 문제를 풀어가는데 있어 힌트로 작용하는 내용도 많습니다.

- 문자열 s의 길이 : 50 이하의 자연수
- 문자열 s는 알파벳으로만 이루어져 있습니다.

입출력 예



우리가 작성할 solution 함수의 실제 input과 올바른 output의 명세입니다.

s	answer
"pPoooyY"	true
"Pyy"	false

▼ 02 - 반복문, 조건문 연습하기

2. 반복문, 조건문 연습하기

<https://school.programmers.co.kr/learn/courses/30/lessons/76501>

문제 설명



코딩테스트의 요구사항입니다. 처음에는 눈에 잘 안들어올 수 있고, 실제 문제보다 설명이 난이도가 높게 느껴질 수 있습니다. 이런 경우 입출력 예를 함께 보면서 능동적으로 읽어가면 크게 도움이 됩니다.

어떤 정수들이 있습니다. 이 정수들의 절댓값을 차례대로 담은 정수 배열 `absolutes`와 이 정수들의 부호를 차례대로 담은 불리언 배열 `signs`가 매개변수로 주어집니다. 실제 정수들의 합을 구하여 `return` 하도록 `solution` 함수를 완성해주세요.

제한사항



요구사항에 추가적으로 작용하는 제한사항입니다. 제한사항에 맞게 구현을 해야해서 꼭 읽어야 하지만, 문제를 풀어가는데 있어 힌트로 작용하는 내용도 많습니다.

- `absolutes`의 길이는 1 이상 1,000 이하입니다.

- absolutes의 모든 수는 각각 1 이상 1,000 이하입니다.
- signs의 길이는 absolutes의 길이와 같습니다.
 - signs[i] 가 참이면 absolutes[i] 의 실제 정수가 양수임을, 그렇지 않으면 음수임을 의미합니다.

입출력 예



우리가 작성할 solution 함수의 실제 input과 올바른 output의 명세 입니다.

absolutes	signs	result
[4, 7, 12]	[true, false, true]	9
[1, 2, 3]	[false, false, true]	0

▼ 정답 코드 / 해설 영상

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d0687350-8e80-4b3c-bc40-210bac4681d3/1%E1%84%8C%E1%85%AE%E1%84%8E%E1%85%A1\(1\).mp4](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d0687350-8e80-4b3c-bc40-210bac4681d3/1%E1%84%8C%E1%85%AE%E1%84%8E%E1%85%A1(1).mp4)

```
function solution(s){
  var result = true;
  s = s.toUpperCase();
  var num = 0;
  for(var i = 0; i < s.length; i++){
    if(s[i] === 'P') num++; // p이면 갯수 더하기
    if(s[i] === 'Y') num--; // y이면 갯수 빼기
  }
  result = (num === 0); // p, y 갯수가 같으면 0 (p가 많으면 양수, 반대 음수)

  return result;
}
```

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4abfd4a0-1bfe-4901-b013-ec5088faaec4/1%E1%84%8C%E1%85%AE%E1%84%8E%E1%85%A1\(2\).mp4](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4abfd4a0-1bfe-4901-b013-ec5088faaec4/1%E1%84%8C%E1%85%AE%E1%84%8E%E1%85%A1(2).mp4)

```
function solution(absolutes, signs) {
  let answer = 0;
  // 두 배열 길이 같음
  for (let i = 0; i < absolutes.length; i++) {
    // 부호에 따라 +-
    signs[i] ? answer += absolutes[i] : answer -= absolutes[i]
  }
  return answer;
}
```