function makeAbbr(words) {

  // write code here

  let abbreviation = words.charAt().toUpperCase();

  for (let i = 0; i < words.length; i++) {

    if (words[i] === ' ') {

      abbreviation = abbreviation + words[i+1].toUpperCase();

    }

  }

  return abbreviation;

}

У цьому завданні створи функцію makeAbbr, яка приймає рядок зі слів words та повертає абревіатуру з них у верхньому регістрі.

Рядок words містить одне або декілька слів, розділених одним пробілом.

function decryptMessage(message) {

  // write code here

  let reserved = '';

  for (const ch of message) {

    reserved = ch + reserved;

  }

  return reserved;

}

А тепер навчимося перебирати рядок із кінця.

Створи функцію decryptMessage, яка приймає рядок message та повертає новий рядок, де символи з message розташовані у зворотному порядку.

function makeStickers(detailsCount, robotPart) {

  // write code here

  if (detailsCount === 0) {

    return [];

  }

  let empty = [];

  for (let i = 1; i <= detailsCount; i++) {

    empty.push(`${robotPart} ${"detail"} ${"#"}${i}`);

  }

  return empty;

}

Настав час запускати масове виробництво роботів!

Щоб роботи на лінії збиралися правильно, потрібно маркувати деталі. Різні частини робота будуть складатися з різної кількості деталей. Тож зробимо наліпки для них!

Напиши функцію makeStickers, яка приймає число detailsCount і рядок robotPart. Функція повинна повертати масив рядків у наступному форматі: {{robotPart}} detail #{{n}} (наприклад, Hand detail #1).

**Зверни увагу:** якщо detailsCount = 0, поверни порожній масив.

function doublePower(currentPowers) {

  // write code here

  if (currentPowers === 0) {

    return [];

  }

  let a = [];

  for (let i = 0; i < currentPowers.length; i++) {

    a.push(currentPowers[i]\*2);

  }

  return a;

}

Напиши функцію doublePower, яка приймає масив потужностей currentPowers та повертає новий масив із подвоєними значеннями.

Наприклад:

doublePower([100, 150, 200, 220]); *// [200, 300, 400, 440]*

doublePower([45, 34, 56, 67]); *// [90, 68, 112, 134]*

doublePower([]); *// []*

function isSorted(boxNumbers) {

  // write code here

  if (boxNumbers === 0) {

    return true;

  }

  for (let i = 1; i <= boxNumbers.length; i++) {

    if ((boxNumbers[i-1]) > boxNumbers[i]) {

      return false;

    } else if ((boxNumbers[i-1]) <= boxNumbers[i]) {

    }

  }

  return true;

}

А тепер навчимо наших роботів сортувати коробки на складі. Кожна коробка має свій унікальний номер, а роботи вчаться сортувати в порядку зростання.

Але сортування — справа нелегка, іноді трапляються помилки. Тому нам поки що доведеться перевіряти, чи правильно робот відсортував коробки.

Для цього напиши функцію isSorted, яка отримує масив чисел boxNumbers і повертає true, якщо всі числа розташовані в порядку зростання, або false — якщо ні.

**Зверни увагу:** числа в масиві можуть повторюватися.

Наприклад:

isSorted([1, 2, 3, 4, 5]); *// true*

isSorted([0, 1, 1, 1, 2]); *// true*

isSorted([1, 2, 11]); *// true*

isSorted([5]); *// true*

isSorted([]); *// true*

isSorted([0, 3, 1, 2, 2, 2]); *// false*

isSorted([1, 11, 2]); *// false*

function getLocation(coordinates, commands) {

  // write code here

  let x = coordinates[0];

  let y = coordinates[1];

  for (let i = 0; i < commands.length; i++) {

    if (commands[i] === 'forward') {

      y = y + 1;

    } else if (commands[i] === 'back') {

       y = y - 1;

    } else if (commands[i] === 'right') {

      x = x + 1;

    } else if (commands[i] === 'left') {

      x = x - 1;

    }

  }

  return [x, y];

}

Ускладнюємо роботу нашого робота! Тепер він вміє перетворювати команди руху на правильний сигнал і рухатися відповідно до нього:

* 'forward' означає y + 1 (крок уперед);
* 'back' означає y - 1 (крок назад);
* 'right' означає x + 1 (крок праворуч);
* 'left' означає x - 1 (крок ліворуч).

Але було б чудово, щоб робот знав, де він знаходиться навіть без GPS.

Для цього реалізуй функцію getLocation, яка приймає 2 параметри:

* масив початкових координат coordinates у вигляді [x, y];
* масив із командами commands у вигляді ['command1', 'command2', 'command3' ...].

Функція повинна повертати масив кінцевих координат [x, y] після рухів згідно команд із масиву commands.

Наприклад, ми маємо масив із координатами coordinates = [2, 1] та масив із командами commands = ['left', 'back', 'back']:

* координати після першої команди — [1, 1] (1 крок ліворуч);
* координати після другої команди — [1, 0] (1 крок назад);
* координати після третьої команди — [1, -1] (1 крок назад);
* результатом буде масив [1, -1].

Інші приклади:

getLocation([0, 0], ['forward', 'right']); *// [1, 1]*

getLocation([2, 3], ['back', 'back', 'back', 'right']); *// [3, 0]*

getLocation([0, 5], ['back', 'back', 'back',

function getPlan(startProduction, numberOfMonths, percent) {

  // write code here

  let goals = [];

  let currentProduction = startProduction;

  for (let i = 0; i < numberOfMonths; i++) {

   goals.push(currentProduction + Math.floor(currentProduction\*percent/100));

    currentProduction = Math.floor(goals[i]);

  }

  return goals;

}

А тепер настав час збільшити обсяги виробництва роботів!

Напиши функцію getPlan, яка приймає 3 аргументи:

* startProduction — поточна кількість роботів, яку ми виробляємо за місяць;
* numberOfMonths — кількість місяців, протягом якої виробництво має зростати;
* percent — відсоток, на який має зростати виробництво щомісяця.

Функція має повертати масив із цілями на кожен місяць (скільки роботів треба виробити щоб дотримуватись запланованого зростання).

Щоб краще зрозуміти, як це працює, розглянемо приклад. Припустимо, нам дано startProduction = 200, numberOfMonths = 3 та percent = 50:

* план на перший місяць — 200 + 50% = 300 роботів;
* на другий місяць це вже 300 + 50% = 450 роботів;
* і нарешті на третій місяць це 450 + 50% = 675 роботів.

В результаті маємо отримати масив [300, 450, 675].

**Зверни увагу:** ціль на наступний місяць потрібно рахувати на основі попереднього місяця.

Якщо число роботів виявиться дробовим, округли його за допомогою Math.floor.

Ще приклади:

getPlan(10, 4, 30); *// [13, 16, 20, 26]*

getPlan(1000, 6, 20); *// [1200, 1440, 1728, 20*

function getSpeedStatistic(testResults) {

  // write code here

  if (testResults.length === 0) {

    return [0, 0, 0];

  }

  let max = testResults[0];

  let min = testResults[0];

  let sum = 0;

  let average = 0;

  for (const number of testResults) {

    if (number > max) {

      max = number;

    } else if (number < min) {

      min = number;

    }

    sum = sum + number;

  }

  average = Math.floor(sum/testResults.length);

 // console.log([min, max, average]);

  return [min, max, average];

}

Перша партія роботів готова, тепер їх треба перевірити. Усі роботи унікальні, і швидкість руху в кожного своя. У цьому завданні тобі потрібно знайти найменшу, найбільшу та середню швидкості роботів.

Напиши функцію getSpeedStatistic, яка приймає масив швидкостей роботів testResults і повертає статистику у вигляді масиву, у якому:

* перший елемент — найменша швидкість;
* другий елемент — найбільша швидкість;
* третій елемент — середнє значення, округлене вниз (використай Math.floor).

**Зверни увагу:** якщо вхідний масив швидкостей порожній — поверни масив [0, 0, 0].

Наприклад:

getSpeedStatistic([]); *// [0, 0, 0]*

getSpeedStatistic([10]); *// [10, 10, 10]*

getSpeedStatistic([8, 9, 3, 12]); *// [3, 12, 8]*

getSpeedStatistic([10, 10, 11, 9, 12, 8]);

function compareRobots(firstRobotResults, secondRobotResults) {

  // write code here

  let firstSum = 0;

  let secondSum = 0;

   for (const ch of firstRobotResults) {

    firstSum = firstSum + ch;

  }

  for (const sec of secondRobotResults) {

    secondSum = secondSum + sec;

  }

  if (firstSum > secondSum) {

    return ("First robot for sale!");

  } else if (firstSum < secondSum) {

    return ("Second robot for sale!");

  } else if (firstSum = secondSum){

    return ("Both robots for sale!");

  } else if (firstSum === 0 && secondSum === 0) {

    return ("Both robots for sale!");

  }

}

Роботи протестовані. Повільних ми відправили на апгрейд. Служба доставки **Mate Post** хоче купити в нас десяток роботів для перевезення вантажу містом. А для цього їм потрібні роботи, які перевозитимуть за день найбільше вантажу.

Напишемо функцію compareRobots, яка отримує два масиви. Перший масив firstRobotResults — це ваги вантажів, перевезених за день першим роботом, другий secondRobotResults — відповідно, другим.

Перевір, хто з роботів може перевезти більше вантажу за день, і поверни рядок із рекомендацією, кого з роботів варто купити:

* 'First robot for sale!' — якщо **перший** робот перевозить більше вантажу;
* 'Second robot for sale!' — якщо **другий** робот перевозить більше вантажу;
* 'Both robots for sale!' — якщо обидва роботи перевозять **однакову** кількість вантажу.

Наприклад:

compareRobots([12, 4, 13], [1, 1, 4, 5, 12]); *// 'First robot for sale!' (29 > 23)*

compareRobots([9, 7, 9], [1, 3, 4, 5, 12]); *// 'Both robots for sale!' (25 = 25)*

compareRobots([1, 3, 4], [1, 1, 4, 5]); *//*

function checkNumber(number) {

    return [(number > 0),(number%2 === 0),(number%10 === 0)];

}

У цьому завданні згадаємо як працювати з числами.

Створи функцію checkNumber, яка приймає ціле число та перевіряє його за 3 критеріями:

* це число додатне?
* це число парне?
* це число кратне 10?

Функція має повернути масив із результатами перевірок у вигляді булевих значень: true або false.

Наприклад:

checkNumber(3); *// [true, false, false]*

checkNumber(10); *// [true, true, true]*

checkNumber(0); *// [false, true, true]*

checkNumber(-1); *// [false, false, false]*

function getArraysSum(arr1, arr2) {

  // write code here

  let a = 0;

  let b = 0;

  for (const ch of arr1) {

    a = a + ch;

  }

  for (const sec of arr2) {

    b = b + sec;

  }

  return a + b;

}

А тепер знайдемо суму елементів масивів.

У цьому завданні реалізуй функцію getArraysSum, яка приймає два масиви чисел **однакової довжини** та повертає суму всіх елементів цих масивів.

Наприклад:

getArraysSum([1, 2], [3, 4]); *// 10 (1 + 2 + 3 + 4)*

getArraysSum([1, 2, 3, 4], [5, 6, 7, 8]); *// 36*

getArraysSum([], []); *// 0*

function combineArrays(first, second) {

  // write code here

  let c = [];

  for (let i = 0; i < first.length; i++) {

    c.push(first[i] + second[i]);

  }

  return c;

}

Реалізуй функцію combineArrays, яка приймає 2 масиви чисел **однакового розміру** (first та second) та повертає масив чисел, де result[i] — це сума чисел first[i] та second[i].

Наприклад:

combineArrays([1, 2, 5], [3, 6, 1]); *// [4, 8, 6]*

combineArrays([1], [6]); *// [7]*

combineArrays([], []); *// []*

function findSmallestElement(numbers) {

  // write code here

  let min = numbers[0];

  for (const ch of numbers) {

    if (ch < min) {

      min = ch;

    }

  }

  return min;

}

Реалізуй функцію findSmallestElement, яка приймає масив numbers та повертає найменше з цього масиву.

Приклади:

findSmallestElement([5, 2, 8, 12, 6]) === 2

findSmallestElement([4, 4, 4, 4]) === 4

findSmallestElement([0, -1, -2, -3]) === -3

function getLastCharacter(input) {

  // write code here

  return input.slice(-1);

}

Implement function that returns last character of the string.

getLastCharacter('abc'); *// 'c'*

getLastCharacter('number 13'); *// '3'*

getLastCharacter('#$@!'); *// '!'*

function checkSubstring(text, part) {

  // write code here

  let a = text.toLowerCase();

  let b = part.toLowerCase();

  return a.includes(b);

}

Напиши функцію checkSubstring, яка приймає рядки text та part і перевіряє, чи text містить part. Регістр не має значення.

Приклад:

checkSubstring('mate academy', 'school') === false

checkSubstring('mate academy', 'mate') === true

checkSubstring('Mate academy', 'matE') === true

checkSubstring('Mate academy', '') === true

function doubleChars(message) {

  // write code here

  let a = [];

  for (const ch of message) {

    a.push(ch + ch);

  }

  return a.join("");

}

Напиши функцію doubleChars, яка приймає рядок message та повертає новий рядок де всі букви з message повторюються двічі.

Приклад:

doubleChars ('Mate academy') === 'MMaatte

function getAverageAge(years) {

  // write code here

  if (years.length === 0) {

    return 0;

  }

  let average = 0;

  let a = [];

  for (let i = 0; i < years.length; i++) {

    a = years[i].split("-");

    average = average + (a[1] - a[0]);

  }

  return Math.round(average/years.length);

}

Дано масив years, який містить роки життя різних людей у вигляді рядків в форматі 1714-1748 (рік народження - рік смерті).

Допиши функцию getAverageAge так, щоб вона повертала среднюю тривалість життя всіх людей, округлену до найближчого цілого (Math.round)

Наприклад:

getAverageAge(['1832-1905', '1876-1956', '1683-1724', '1714-1748']) === 57

getAverageAge([

'1907-1997',

'1761-1833',

'1535-1582',

'1918-2012',

'1877-1968',

'1696-1724',

'1602-1642',

'1692-1743',

'1695-1762',

'1570-1636',

'1762-1807',

'1668-1731',

]) === 63

function getSumOfRange(start, end) {

  // write code here

  let sum = 0;

  const a =[];

  for (let i = start; i <= end; i++) {

    a.push(i);

  }

  for (let i = 0; i < a.length; i++) {

    sum = sum + a[i];

  }

  console.log(a);

  return sum;

}

Допиши функцію getSumOfRange, яка приймає числа start, end, та повертає суму усіх чисел на відрізку від start до end включно.

Приклад:

getSumOfRange(1, 10) === 55

getSumOfRange(-5, 5) === 0

getSumOfRange(1, 500) === 125250

function splitString(str) {

let word = str.split("");

let result = [];

  if (word.length%2 === 0) {

    } else {

      word.push("\_");

  }

  for (let i = 0; i < word.length; i = i + 2) {

      result.push(word[i] + word [i+1]);

  }

  return result;

}

А тепер «розріжемо» рядок на частини 😎

Реалізуй функцію splitString, яка приймає рядок str, ділить його на частини по 2 символи, а потім повертає масив з отриманих частин.

**Зверни увагу:** якщо рядок містить непарну кількість символів, додай символ \_ після останнього символу.

Наприклад:

splitString('123456'); *// ['12', '34', '56']*

splitString('ab cd ef'); *// ['ab', ' c', 'd ', 'ef']*

splitString('abc'); *// ['ab', 'c\_']*

splitString(' '); *// [' \_']*

splitString(''); *// []*

function getLargestExpressionResult (a, b){

  let result = a + b;

    if (result < (a - b)){

    result = a - b;

  }

    if (result < (a \* b)){

    result = a \* b;

  }

  if (result < (a / b)){

    result = a / b;

  }

  return result;

}

У цьому завданні створи функцію getLargestExpressionResult, яка приймає 2 числа: a та b. Ця функція має порівняти результати наступних обчислень та повернути найбільший із них:

* a + b
* a - b
* a \* b
* a / b

**Зверни увагу**:

* числа a та b можуть бути від'ємними;
* не використовуй тернарний оператор чи ключове слово else.

Наприклад, якщо a = 10 та b = 5, то:

* a + b = 15
* a - b = 5
* a \* b = 50 — найбільший результат
* a / b = 2

Але якщо a = 10 та b = -5, то:

* a + b = 5
* a - b = 15 — найбільший результат
* a \* b = -50
* a / b = -2

getLargestExpressionResult(10, 5); *// 50*

getLargestExpressionResult(10, -5); *// 15*