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*

function scrollingText(word) {

  // write code here

  let massiv = [];

  let str = '';

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

  str =  word.slice(i, word.length)+word.slice(0,i);

  massiv.push(str.toUpperCase());

}

return massiv;

}

А тепер ти навчишся міняти порядок символів у рядку.

Напиши функцію scrollingText, яка:

* приймає рядок word;
* послідовно переставляє всі символи в рядку з нульового індексу на останній;
* повертає масив з отриманими комбінаціями рядка у верхньому регістрі.

Наприклад, для слова 'robot':

* перший крок: беремо першу літеру (r) слова 'robot' і переставляємо її в кінець слова — 'obotr';
* другий крок: беремо першу літеру (о) слова 'obotr' і переставляємо її в кінець – 'botro' і так далі.

У консолі ми побачимо:

[ 'ROBOT', // спочатку вхідне слово

'OBOTR', // потім із переставленим порядком символів

'BOTRO',

'OTROB',

'TROBO' ]

function isSpecialNumber(n) {

  // write code here

let str = String(n);

console.log(str);

  for (const ch of str) {

    if (ch > 5) {

      console.log('NOT!!');

       return ('NOT!!');

    }

    console.log('Special!!');

  }

  return ('Special!!');

  }

Створи функцію isSpecialNumber, яка приймає додатне число n і визначає, чи є воно **особливим**.

Число називається **особливим**, якщо **кожна** його цифра **не більша** ніж 5 (0, 1, 2, 3, 4 або 5).

Функція повинна повернути рядок 'Special!!', якщо число **особливе**, та 'NOT!!' — якщо ні.

Наприклад:

isSpecialNumber(2); *// 'Special!!'*

*// 2 — знаходиться в інтервалі від 0 до 5*

isSpecialNumber(9); *// 'NOT!!'*

*// 9 > 5*

isSpecialNumber(23); *// 'Special!!'*

*// всі цифри числа 23 знаходяться в інтервалі від 0 до 5*

isSpecialNumber(38); *// 'NOT!!'*

*// 8 > 5*

function isTidy(number) {

  // write code here

  let str = number.toString();

  let a = str[0];

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

    if (a > str[i]) {

       return false;

  } a = str[i];

  }

  return true;

}

А тепер перевіримо, чи всі наші числа **охайні**. Число вважається **охайним**, якщо кожна його цифра не менша за попередню.

Реалізуй функцію isTidy, яка приймає додатне число та повертає true, якщо воно охайне, інакше — false.

Наприклад:

isTidy (12); *// true*

*// цифри розташовані за зростанням*

isTidy (32); *// false*

*// цифри розташовані за спаданням*

isTidy (1024); *// false*

*// 1 > 0*

isTidy(3445); *// true*

*// однакові цифри можуть бути поруч*

isTidy (13579); *// true*

*// цифри розташовані за зростанням*

const s = [5, 7, 2];

function editInPlace() {

// Змініть код лише під цим рядком

s[0] = 2;

s[1] = 5;

s[2] = 7;

// Використання s = [2, 5, 7] може бути недійсним

// Змініть код лише над цим рядком

}

editInPlace();

**Видозмінюйте наданий масив за допомогою змінної**

Якщо ви ще не знаєте про const, див. [це завдання про ключове слово const](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/declare-a-read-only-variable-with-the-const-keyword).

Є багато значень змінної const у сучасній JavaScript.

Деякі розробники надають перевагу привласненню усіх змінних за допомогою const, доки вони не дізнаються про можливу необхідність перепризначення змінної. Тоді вони використовують let.

Проте варто розуміти, що об'єкти (включаючи масиви та функції), призначенні змінній за допомогою const, все ще можна змінювати. Правильне використання const запобігає зміненню ідентифікатора змінної.

const s = [5, 6, 7];

s = [1, 2, 3];

s[2] = 45;

console.log(s);

s = [1, 2, 3] призведе до помилки. After commenting out that line, the console.log will display the value [5, 6, 45].

Як ви помітили, можна змінювати об'єкт [5, 6, 7]. Змінна s все ще вказуватиме на змінений масив [5, 6, 45]. Як і усі масиви, елементи масиву в s є незмінними, бо було використано const. Ви не можете використовувати ідентифікатор змінної s, щоб вказувати на інший масив оператору призначення.

Масив оголошується як const s = [5, 7, 2]. Змініть масив на [2, 5, 7] використовуючи різні призначення елементів.

function freezeObj() {

const MATH\_CONSTANTS = {

PI: 3.14

};

// Змініть код лише під цим рядком

Object.freeze(MATH\_CONSTANTS);

// Змініть код лише над цим рядком

try {

MATH\_CONSTANTS.PI = 99;

} catch(ex) {

console.log(ex);

}

return MATH\_CONSTANTS.PI;

}

const PI = freezeObj();

**Запобігання мутаціям об'єкта**

З попереднього завдання бачимо, що const насправді не захищає ваші дані від змін. Щоб ваші дані не змінилися, JavaScript надає функцію Object.freeze для запобігання мутації даних.

Будь -яка спроба змінити об'єкт буде відхилена, з помилкою, якщо тег працює в строгому режимі.

let obj = {

name:"FreeCodeCamp",

review:"Awesome"

};

Object.freeze(obj);

obj.review = "bad";

obj.newProp = "Test";

console.log(obj);

The obj.review and obj.newProp призначення призведе до помилок, тому що наш редактор за замовчуванням працює в строгому режимі,і консоль покаже значення { name: "FreeCodeCamp", review: "Awesome" }.

У цьому завданні вам доведеться використовувати Object.freeze для запобігання зміни математичних констант. Вам необхідно заморозити об'єкт MATH\_CONSTANTS так, щоб ніхто не зміг змінити значення PI, додати або видалити властивості.

const magic = () => new Date();

**Використовуйте функцію Arrow для опису анонімної функції**

У JavaScript, нам часто не потрібно називати функцію, особливо при передачі функції в якості аргументу іншої функції. Натомість створюються вбудовані функції. Їх не потрібно називати, бо вони більш ніде не використовуються.

Щоб досягнути цього, використовується наступний синтаксис:

const myFunc = function() {

const myVar = "value";

return myVar;

}

ES6 надає нам синтаксичні можливості, щоб не використовувати анонімних функцій таким чином. Саме тому варто використовувати **синтаксичну функцію arrow**:

const myFunc = () => {

const myVar = "value";

return myVar;

}

Коли відсутнє тіло функції, а є лише повернене значення, то синтаксична функція arrow дозволяє опустити ключове слово return, а також дужки, що стосуються коду. Це допомагає спростити невелику функцію в однорядковий вираз:

const myFunc = () => "value";

Цей код все це поверне рядок value за змовчуванням.

Перепишіть функцію, призначеній змінній magic, що перетворюється наnew Date() використовуючи синтаксичну функцію arrow. Також переконайтесь, що ще нічого не визначено за допомогою ключового слова var.

const myConcat = (arr1, arr2) => arr1.concat(arr2);

console.log(myConcat([1, 2], [3, 4, 5]));

# Запис функції Arrow з параметрами

Так само, як і у звичайній функції, ви можете передавати аргументи в функцію Arrow.

const doubler = (item) => item \* 2;

doubler(4);

doubler(4) повинен перетворюватись на 8.

Якщо функція arrow має один параметр, то дужки можуть бути опущені.

const doubler = item => item \* 2;

Функція може вміщати в себе більше одного аргументу.

const multiplier = (item, multi) => item \* multi;

multiplier(4, 2);

multiplier(4, 2) повинен перетворюватись на 8.

Перепишіть функцію myConcat, яка додає вміст arr2 до arr1, щоб функція використовувала синтаксис функції arrow.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Вам слід замінити ключове слово var.
* В очікуванні:myConcat має бути постійною змінною (використовуйте const).
* В очікуванні:myConcat має бути функцією з 2 параметрами
* В очікуванні:myConcat() перетворюється на [1, 2, 3, 4, 5].
* В очікуванні:Не використовуйте ключове слово function.

// Змініть код лише під цим рядком

const increment = (number, value = 1) => number + value;

// Змініть код лише над цим рядком

# Виставлення параметрів за замовчуванням для ваших функцій

Для допомоги у створенні більш гнучких функцій, ES6 пропонує default parameters для функцій.

Погляньте на цей код:

const greeting = (name = "Anonymous") => "Hello " + name;

console.log(greeting("John"));

console.log(greeting());

Консоль буде відображати рядки Hello John та Hello Anonymous.

Параметр за замовчуванням спрацьовує, коли аргумент не вказано (не визначено). Як можна побачити у прикладі вище, параметр name отримає за замовчуванням значення Anonymous, якщо ви не вказали значення цього параметру. Ви можете додати значення за замовчуванням будь-якій кількості параметрів.

Модифікуйте функцію increment, додавши параметри за замовчуванням, щоб одиниця була додана до number, якщо value не вказано.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Результатом increment(5, 2) повинен бути 7.
* В очікуванні:Результатом increment(5) повинен бути 6.
* В очікуванні:Значення параметру за замовчуванням 1 слід використовувати для value.

const sum = (...args) => {

//const args = [x, y, z];

return args.reduce((a, b) => a + b, 0);

}

# Використання параметру скидання і параметру функцій

Щоб створити гнучкіші функції, ES6 пропонує параметр скидання для параметрів функцій. За допомогою параметра скидання можна створити функції зі змінною кількістю аргументів. Ці аргументи зберігаються в масиві, який буде доступний пізніше з внутрішньої функції.

Погляньте на цей код:

function howMany(...args) {

return "You have passed " + args.length + " arguments.";

}

console.log(howMany(0, 1, 2));

console.log(howMany("string", null, [1, 2, 3], { }));

Консоль відображатиме рядки You have passed 3 arguments. і You have passed 4 arguments..

Параметр скидання усуває потребу в перевірці args масиву і дає змогу застосувати map(), filter() і reduce() у масиві параметрів.

Змініть функції sum за допомогою параметру скидання у такий спосіб, щоб функція sum могла прийняти будь-яку кількість аргументів і повернути їх суму.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Результатом sum(0,1,2) має бути 3
* В очікуванні:Результатом sum(1,2,3,4) має бути 10
* В очікуванні:Результатом sum(5) має бути 5
* В очікуванні:Результатом sum() має бути 0
* В очікуванні:sum має бути функцією зі стрілкою, що використовує синтаксис параметру скидання (...) у параметрі args.

const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];

let arr2;

arr2 = [...arr1]; // Змініть цей рядок

console.log(arr2);

# Використовуйте спред оператор для оцінки аналізу масиву

ES6 представляє собою spread operator, який дозволяє розширювати масиви та інші елементи.

Код ES5 нижче використовує apply() для розрахунку максимального значення в масиві:

var arr = [6, 89, 3, 45];

var maximus = Math.max.apply(null, arr);

maximus повинен мати значення 89.

Ми використали Math.max.apply(null, arr) тому що Math.max(arr) повертає NaN. Math.max() виконується через спільний аргумент, а не масив. Спред оператор робить синтаксис більш читабельним та зручним в обслуговуванні.

const arr = [6, 89, 3, 45];

const maximus = Math.max(...arr);

maximus повинен мати значення 89.

...arr перетворюється в відкритий масив. Іншими словами, це поширює масив. Проте, спред оператор працює тільки всередині, як аргумент в функції або літери в масиві. Наступний код не буде працювати:

const spreaded = ...arr;

Скопіюйте весь вміст arr1 в інший масив arr2 використовуючи спред оператор.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Масив arr2 має бути правильною копією arr1.
* В очікуванні:... спред оператор використовується для дублювання arr1.
* В очікуванні:arr2 має залишатися без змін, в той час як arr1 було змінено.

# const HIGH\_TEMPERATURES = {

# yesterday: 75,

# today: 77,

# tomorrow: 80

# };

# // Змініть код лише під цим рядком

# //const {yesterday, today, tomorrow} = HIGH\_TEMPERATURES;

# const {today} = HIGH\_TEMPERATURES;

# const {tomorrow} = HIGH\_TEMPERATURES;

# // Змініть код лише над цим рядком

# Використовуйте деструктивне привласнення для визначення змінних із масивів

Призначення деструктуризації є спеціальним синтаксисом, запровадженим у ES6, для чіткого призначення параметрів, взятих безпосередньо з об'єкта.

Розглянемо наступний приклад коду ES5:

const user = { name: 'John Doe', age: 34 };

const name = user.name;

const age = user.age;

name матиме значення рядка John Doe, and age матиме значення числа 34.

Ось аналогічна інструкція з використання синтаксису деструкції ES6:

const { name, age } = user;

Отже, name стане рядком John Doe, and age матиме значення числа 34.

Тут змінні name та age будуть створені і призначені значення їх відповідних значень об'єкта user. Ви можете побачити наскільки стало простіше.

Ви можете отримати потрібну кількість значень з потрібного вам об'єкта.

Замініть два визначення еквівалентами з деструктивного привласнення. Ви все ще призначаєте змінним today and tomorrow значення today та tomorrow з об'єкту HIGH\_TEMPERATURES.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Видаліть призначення синтаксису ES5.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну today.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну tomorrow.
* В очікуванні:today буде рівним 77 та tomorrow дорівнюватиме 80.

const HIGH\_TEMPERATURES = {

yesterday: 75,

today: 77,

tomorrow: 80

};

// Змініть код лише під цим рядком

const {today: highToday} = HIGH\_TEMPERATURES;

const {tomorrow: highTomorrow} = HIGH\_TEMPERATURES;

// Змініть код лише над цим рядком

# Використовуйте деструктивне привласнення для визначення змінних із масивів

Деструктуризація дозволяє вам призначити нове ім'я змінної при вилученні. Щоб зробити це, поставте нове ім'я після двокрапки при призначенні.

Використовуйте то й же об'єкт що і в останньому прикладі:

const user = { name: 'John Doe', age: 34 };

Розглянемо, як ви можете дати нові імена змінним у призначенні:

const { name: userName, age: userAge } = user;

Ви можете прочитати це як "отримати значення user.name та та назначити нову змінну з назвою userName" і т.д. Значення userName буде John Doe, і значення userAge буде числом 34.

Замініть два визначення еквівалентами з деструктивного привласнення. Ви все ще призначаєте змінним highToday та highTomorrow значення today та and tomorrow з об'єкту HIGH\_TEMPERATURES.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Видаліть призначення синтаксису ES5.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну highToday.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну highTomorrow.
* В очікуванні:highToday повинна дорівнювати 77 та highTomorrow має бути рівний 80.

const LOCAL\_FORECAST = {

yesterday: { low: 61, high: 75 },

today: { low: 64, high: 77 },

tomorrow: { low: 68, high: 80 }

};

// Змініть код лише під цим рядком

const {today: {low: lowToday}} = LOCAL\_FORECAST;

const {today: {high: highToday}} = LOCAL\_FORECAST;

// Змініть код лише над цим рядком

# Використовуйте деструктивне привласнення для визначення змінних із масивів

Слідуйте тим самим принципам, що і у попередніх 2 уроках, які були присвячені визначенню змінних із масивів.

Використовуйте змінну схожу до попередніх масивів:

const user = {

johnDoe: {

age: 34,

email: 'johnDoe@freeCodeCamp.com'

}

};

Розглянемо приклад, як отримати значення властивостей об'єкта та призначити їх змінним із таким самим ім'ям:

const { johnDoe: { age, email }} = user;

Приклад того, як ви можете призначити значення з різними іменами:

const { johnDoe: { age: userAge, email: userEmail }} = user;

Замініть два визначення еквівалентами з деструктивного привласнення. Ви все ще призначаєте змінним lowToday та highToday значення today.low та today.high з об'єкту LOCAL\_FORECAST.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Видаліть призначення синтаксису ES5.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну lowToday.
* В очікуванні:Використовуйте деструкцію, щоб створити змінну highToday.
* В очікуванні:lowToday повинна дорівнювати 64 та highToday</code має бути рівний <code>77.

let a = 8, b = 6;

// Змініть код лише під цим рядком

[a, b] = [b, a];

# Використовуйте деструктивне привласнення для визначення змінних із масивів

ES6 створює деструктивні масиви так само легко, як і деструктивні об'єкти.

Однією з ключових відмінностей між спред оператором та деструктивним масивом є те, що спред оператор розпаковує весь вміст масиву в список, розділений комою. Отже, ви не можете вибрати, які елементи ви бажаєте призначити змінним.

Деструкція масиву дозволяє нам зробити це:

const [a, b] = [1, 2, 3, 4, 5, 6];

console.log(a, b);

В консолі відображатимуться значення a та b як 1, 2.

Змінна a призначає перше значення масиву, а b присвоюється друге значення масиву. Також ми можемо отримати доступ до значення у масиві з деструкцією за допомогою коми для досягнення бажаного індексу:

const [a, b,,, c] = [1, 2, 3, 4, 5, 6];

console.log(a, b, c);

В консолі відображатимуться значення a, b, та c як 1, 2, 5.

Використайте привласнення для заміни значень a and b, так, що a отримує значення, збережене в b, і b отримує значення, збережене в a.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Значення a має бути 6, після перегортання.
* В очікуванні:Значення b має бути 8, після перегортання.
* В очікуванні:Ви повинні використовувати деструктивний масив, щоб замінити a та b.

function removeFirstTwo(list) {

// Змініть код лише під цим рядком

const [a, b, ...shorterList] = list; // Змініть цей рядок

// Змініть код лише над цим рядком

return shorterList;

}

const source = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const sourceWithoutFirstTwo = removeFirstTwo(source);

# Використовуйте решту параметрів з деструктивним призначенням для перепризначення масиву елементів

У деяких ситуаціях нам потрібно зберегти елементи в окремий масив.

Результат схожий на Array.prototype.slice(), як показано нижче:

const [a, b, ...arr] = [1, 2, 3, 4, 5, 7];

console.log(a, b);

console.log(arr);

У такому разі консоль відображатиме значення 1, 2 та [3, 4, 5, 7].

Змінні a та b приймають перші та другі значення з масиву. Після цього, завдяки параметру arr решта значень подаються у масиві. Решта елементів працює тільки-но у вигляді останньої змінної списку. Як і в даних, ви не можете використовувати інший параметр для охоплення підмасиву, який залишає останній елемент вихідного масиву.

Use a destructuring assignment with the rest parameter to emulate the behavior of Array.prototype.slice(). removeFirstTwo() should return a sub-array of the original array list with the first two elements omitted.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:removeFirstTwo([1, 2, 3, 4, 5]) should be [3, 4, 5]
* В очікуванні:removeFirstTwo() should not modify list
* В очікуванні:Array.slice() не варто використовувати.
* В очікуванні:Необхідно використовувати деструктуризацію list.

const stats = {

max: 56.78,

standard\_deviation: 4.34,

median: 34.54,

mode: 23.87,

min: -0.75,

average: 35.85

};

// Змініть код лише під цим рядком

const half = ({max, min}) => (max + min) / 2.0;

// Змініть код лише над цим рядком

# Використовуйте призначення за рахунок деструктуризації, щоб передати об'єкт як параметри функції

У деяких випадках ви можете деструктурувати об'єкт в самому аргументі функції.

Розглянемо наступний код:

const profileUpdate = (profileData) => {

const { name, age, nationality, location } = profileData;

}

Ефективний спосіб деструктуризації об'єкта в функцію. Це можна виконати шляхом аналізу:

const profileUpdate = ({ name, age, nationality, location }) => {

}

Коли profileData переходить у функцію, значення деструктуруються з параметра функції в використання функції.

Використовуйте деструктивне привласнення в межах аргументу half щоб відправити лише max та min всередині функції.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:stats має бути object.
* В очікуванні:half(stats) має бути 28.015
* В очікуванні:Деструкція є важливим елементом.
* В очікуванні:Необхідно використовувати деструктивний параметр.

const result = {

success: ["max-length", "no-amd", "prefer-arrow-functions"],

failure: ["no-var", "var-on-top", "linebreak"],

skipped: ["no-extra-semi", "no-dup-keys"]

};

function makeList(arr) {

"use strict";

// change code below this line

const failureItems = [];

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

failureItems.push(`<li class="text-warning">${arr[i]}</li>`);

}

// change code above this line

return failureItems;

}

const failuresList = makeList(result.failure);

# Створіть рядки за допомогою шаблонних літералів

Нова особливість ES6 полягає у template literal. Вона означає особливий тип рядка, що полегшує процес створення складних рядків.

Шаблонні літерали дозволяють нам створювати багатолінійні рядки та використовувати можливості інтерполяції для створення рядків.

Розгляньмо наступний код:

const person = {

name: "Zodiac Hasbro",

age: 56

};

const greeting = `Hello, my name is ${person.name}!

I am ${person.age} years old.`;

console.log(greeting);

Консоль показуватиме рядки Hello, my name is Zodiac Hasbro! та I am 56 years old..

Тут відбувається багато процесів. По-перше, у прикладі вживаються зворотні лапки (`), а не одинарні чи подвійні (' чи "), щоб обрамити рядок. По-друге, зверніть увагу, що рядок залишається багатолінійним як у коді, так і в кінцевому вигляді. Завдяки цьому, \n вставляється у самі рядки. Наведений вище синтаксис ${variable} є заповнювачем місця (placeholder). Це означає, що більше не потрібно використовувати конкатенацію з оператором +. Для додання змінних до рядків варто видалити змінну в шаблонному рядку та розмістити її у конструкцію ${ і }. Інші вирази також можуть додаватися до літералу у рядку схожим чином, наприклад ${a + b}. Такий новий спосіб створення рядків надає більше можливостей створювати функціональні рядки.

Використовуйте синтаксис шаблонних літералів зі зворотними лапками для створення масиву елемента списку - рядків (li). Текст кожного елемента списку має бути одним з елементів масиву зі значенням failure на об'єкті result та мати атрибут class зі значенням text-warning. Функція makeList повинна повернути масив елементів списку у рядках.

Завдяки методу, що використовує ітератор (будь-який цикл), ми можемо отримати бажаний результат (наведений нижче).

[

'<li class="text-warning">no-var</li>',

'<li class="text-warning">var-on-top</li>',

'<li class="text-warning">linebreak</li>'

]

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:failuresList має бути масивом, що містить повідомлення result failure.
* В очікуванні:failuresList має дорівнювати вказаному результату.
* В очікуванні:Варто також використовувати шаблонні рядки та інтерполяцію виразу.
* В очікуванні:Варто використовувати ітератор.

const createPerson = (name, age, gender) => {

// Змініть код лише під цим рядком

return ({

name,

age,

gender

});

// Змініть код лише над цим рядком

};

# Давайте напишемо точні декларації об'єкта за допомогою Object Property Shorthand

ES6 надає гарну підтримку для легкого визначення літералів об'єкта.

Розглянемо наступний код:

const getMousePosition = (x, y) => ({

x: x,

y: y

});

getMousePosition це проста функція, яка повертає об'єкт з двома властивостями. ES6 втілює синтаксичний цукор (синтаксичний прийом, який полегшує сприйняття тексту програми), щоб не доводилось багато писати x: x. Можна просто написати x один раз, і він конвертується у x: x (або еквівалент) автоматично. Ось та ж сама функція, переписана під новий синтаксис:

const getMousePosition = (x, y) => ({ x, y });

Скористайтеся скороченням властивостей об'єкта з його літералами, щоб створити і звернутися до об'єкта з такими властивостями: name, age і gender.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:createPerson("Zodiac Hasbro", 56, "male") має стати {name: "Zodiac Hasbro", age: 56, gender: "male"}.
* В очікуванні:Ваш код не має містити key:value.

// Змініть код лише під цим рядком

const bicycle = {

gear: 2,

setGear(newGear) {

this.gear = newGear;

}

};

// Змініть код лише над цим рядком

bicycle.setGear(3);

console.log(bicycle.gear);

# Напишіть стислі декларативні функції за допомогою ES6

При визначенні функцій у об’єктах у ES5 ми маємо використовувати ключове слово function наступним чином:

const person = {

name: "Taylor",

sayHello: function() {

return `Hello! My name is ${this.name}.`;

}

};

За допомогою ES6 ви можете взагалі видалити ключове слово function і двокрапку під час визначення функцій в об’єктах. Ось приклад такого синтаксису:

const person = {

name: "Taylor",

sayHello() {

return `Hello! My name is ${this.name}.`;

}

};

Реорганізуйте функцію setGear всередині об’єкта bicycle, щоб використати скорочений синтаксис, описаний вище.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Традиційні вираження функцій не повинні використовуватися.
* В очікуванні:setGear повинна бути декларативною функцією.
* В очікуванні:bicycle.setGear(48) повинен змінити значення gear для 48.

# Використовуйте клас синтаксису для визначення функції Constructor

ES6 забезпечує новий синтаксис для створення об'єктів, використовуючи ключове слово class.

Варто зазначити, що синтаксис class є просто синтаксисом, а не повноцінною реалізацією об'єктно-орієнтованої парадигми, на відміну від таких мов як Java, Python, Ruby, і т.д.

In ES5, an object can be created by defining a constructor function and using the new keyword to instantiate the object.

In ES6, a class declaration has a constructor method that is invoked with the new keyword. If the constructor method is not explicitly defined, then it is implicitly defined with no arguments.

// Explicit constructor

class SpaceShuttle {

constructor(targetPlanet) {

this.targetPlanet = targetPlanet;

}

takeOff() {

console.log("To " + this.targetPlanet + "!");

}

}

// Implicit constructor

class Rocket {

launch() {

console.log("To the moon!");

}

}

const zeus = new SpaceShuttle('Jupiter');

// prints To Jupiter! in console

zeus.takeOff();

const atlas = new Rocket();

// prints To the moon! in console

atlas.launch();

Варто зазначити, що ключове слово class оголошує нову функцію, до якої додається конструктор. Конструктор викликається, коли ключове слово new використовується для створення нового об'єкту.

**Примітка:** UpperCamelCase слід використовувати в конвенції для назв класів ES6, як і в SpaceShuttle використаного вище.

Метод constructor це спеціальний метод для створення та ініціалізації об'єкта, створений за допомогою класу. Ви можете дізнатись про це більше у розділі "Об"єктно-орієнтоване програмування" JavaScript алгоритми та структура даних.

Використайте ключове слово class та напишіть constructor для створення класу Vegetable.

Клас Vegetable дозволяє створити об'єкт vegetable з параметром name, що опускає constructor.

Запустити тест (Ctrl + Enter)Скинути весь код

Отримати допомогу

## Tests

* В очікуванні:Vegetable має бути class визначений методом constructor.
* В очікуванні:Використовуйте ключове слово class.
* В очікуванні:Vegetable має бути інстальовано.
* В очікуванні:carrot.name повинен повертатись як carrot.