**Рекурсия**

Задача 1 Реализуйте функцию sequenceSum, которая находит сумму последовательности целых чисел. Последовательность задается тремя значениями: begin - начало последовательности, end - конец последовательности, step - шаг. Например: begin = 2, end = 6, step = 2, дает нам такую последовательность 2, 4, 6. Сумма такой последовательности будет: 12.

import { sequenceSum } from './solution';

sequenceSum(1, 5, 1); // 1 + 2 + 3 + 4 + 5 = 15

sequenceSum(1, 5, 3); // 1 + 4 = 5

Решение

export const sequenceSum = (begin, end, step) => {

if (begin > end) {

return 0;

}

return begin + sequenceSum(begin + step, end, step);

}

Задача 2 Число в спепень

Решение

stepen = (x, n) => {

if (n===0) {

return 1;

}

return x \* stepen(x ,n-1);

};

stepen(3,3) //27

Итерративная функция (внутри рекурсивный процесс)

**Задача 3**

Найти наименьший целый делитель числа.

**Решение**

smallestDivisor = (num) => {

const iter = (acc) => {

if (num % acc === 0) {

return acc;

}

return iter(acc + 1);

};

return iter(2);

};

smallestDivisor(6); //2

**Задача 4**

Возведение числа в степень.

**Решение**

const pow = (base, exp) => {

if (exp === 0) {

return 1;

}

const iter = (count, acc) => {

if (count === 0) {

return acc;

}

else

return iter(count - 1, acc \* base);

};

else

return iter(exp, acc = 1);

};

pow(2,4);

**Задача 4**

куб числа

**Решение**

cube = (x) => {

const iter = (current, acc) => {

if (current === 0) {

return acc;

}

return iter(current - 1, acc\*x);

};

return iter(3, acc=1);

};

cube(2); //2

//куб числа

**Задача 5**

Сумма всех чисел от 0 до x, кратных 5.

**Решение**

const sumOfNumbers = (x) => {

const iter = (current, acc) => {

if (current === 0) {

return acc;

}

else if (current % 5 !== 0) {

return iter(current - 1, acc) ;

}

return iter(current - 1, acc+current);

};

return iter(x, acc=0);

};

sumOfNumbers(21); //50

Императивный подход + цикл WHILE

**Задача 6**

Реализуйте и экспортируйте функцию smallestDivisor используя итеративный процесс (императивно). Функция должна находить минимальный делитель переданного числа.

Для этого достаточно последовательно проверять делимость начиная с двойки. Если делитель не найден, значит число простое и делителем является само число.

Если переданное число меньше единицы, то функция должна вернуть NaN.

smallestDivisor(15); // 3

smallestDivisor(17); // 17

smallestDivisor(0); // NaN

smallestDivisor(-3); // NaN

**Решение**

export const smallestDivisor = (num) => {

let result = 2;

if (num < 1) {

return NaN;

}

if (num === 1) {

return num;

}

while (num % result !== 0) {

result = result + 1;

}

return result;

};

**Задача 7**

Число в степень

**Решение**

stepen = (base, exp) => {

let result = 1;

let i = 0;

if (base === 0) {

return 0;

}

if (base === 1) {

return 1;

}

while (i !== exp) {

result = base \* result;

i=i+1;

}

return result;

};

stepen(2 , 4);

**Задача 8**

Сумма чисел кратных 3, от 1 до Х включительно.

**Решение**

const sumOfNumbers = (x) => {

let base = 1;

let acc = 0;

while (base !== x + 1 ) {

if (base % 3 ===0) {

acc = acc + base;

}

base = base + 1;

console.log(acc);

}

return acc;

};

sumOfNumbers(15);

// сумма чисел кратных трем

**Задача 9**

Наибольшим общим делителем (НОД) для двух целых чисел m и n называется наибольший из их общих делителей. Пример: для чисел 70 и 105 наибольший общий делитель равен 35. Наибольший общий делитель существует и однозначно определён, если хотя бы одно из чисел m или n не равно нулю.

solution.js

Экспортируйте функцию по умолчанию, которая находит наибольший общий делитель двух положительных и целых чисел.

gcd(2, 3); // 1

gcd(6, 3); // 3

gcd(14, 21); // 7

**Решение**

export default gsd = (m , n) => {

let div = Math.min(m , n);

if (m <= 0 || n <= 0 ) {

return ('M or N <=0');

}

if (Math.floor(m) - m !== 0 || Math.floor(n) - n !== 0) {

return ('Введите целые числа');

}

while (m % div !==0 || n % div !== 0) {

div = div - 1;

}

return div;

};

**Решение 2**

export default gsd = (x , y) => {

if (x <=0 || y <=0 ) {

return ('M or N <=0');

}

if (Math.floor(x) - x !== 0 || Math.floor(y) - y !== 0) {

return ('Введите целые числа');

}

while (x !== 0 && y !== 0) {

if (x > y) {

x = x % y

} else {

y = y % x;

}

}

return x + y;

};

gsd(30 , 18);

**Решение учителя**

const gcd = (a, b) => {

if (b === 0) {

return a;

}

return gcd(b, a % b);

};

export default gcd;

**Задача 10**

Простое число — натуральное (целое положительное) число (больше единицы), имеющее ровно два различных натуральных делителя — единицу и самого себя.

solution.js

Реализуйте и экспортируйте функцию isPrime определяющую, является ли число простым.

Пример:

isPrime(1); // false

isPrime(7); // true

isPrime(10); // false

**Решение**

export const isPrime = (x) => {

let div = 2;

if (x <= 0) {

return false;

}

if (Math.floor(x) - x !== 0) {

return ('Введите целые числа');

}

if (x >= 1) {

while(x % div !== 0 && div <= x) {

div++;

}

if (x === div) {

return true;

} else {

return false;

}

}

};

isPrime(7);

ЦиклFOR

**Задача 11**

Реализуйте и экспортируйте функцию по умолчанию, которая считает сумму всех натуральных чисел меньших чем n (первый аргумент), которые делятся на числа a или b (второй и третий аргументы) без остатка. n - может быть только натуральным числом.

// 1 % 2 || 1 % 4

// 2 % 2 || 2 % 4

// ...

// 9 % 2 || 9 % 4

solution(10, 2, 4); // 20

**Решение**

export default (n, a, b) => {

let acc = 0;

if ((Math.floor(a) - a) !== 0 || (Math.floor(b) - b) !== 0 || (Math.floor(n) - n) !== 0 || n <= 1) {

return 0;

}

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

if (i % a === 0 || i % b === 0){

acc += i;

}

}

return acc;

} ;

(101, 7, 3);

**Строки**

**Задача 12**

Реализуйте и экспортируйте функцию reverse, которая переворачивает строку.

Пример:

reverse('hello, world!'); // !dlrow ,olleh

Подсказки

Для подсчета длины строки используйте функцию length из модуля strings. Пример: strings.length('cat'); // 3

**Решение**

import \* as strings from './strings';

// BEGIN (write your solution here)

export const reverse = (str) => {

let acc = "";

for (let i = strings.length(str)-1; i >= 0 ; i--) {

acc = acc + str[i];

console.log(acc);

}

return acc;

}

reverse("hello, world");

**SWITCH**

**Задача 13**

Функция calc представляет из себя простейший калькулятор, которому передается бинарная операция, а так же два числа, а на выходе получается результат применения операции к этим числам.

Реализуйте и экспортируйте функцию calc, аргументами которой являются:

Операция в виде строки. Поддерживаются +, -, /,\*.

Первый операнд.

Второй операнд.

Если передается операция которая не поддерживается, то функция должна вернуть NaN.

Примеры:

calc('+', 3, 10); // 13

calc('-', -8, 6); // -14

calc('$', 0, 9); // NaN

**Решение**

export const calc = (operation, num1, num2) => {

switch (operation) {

case '+':

return num1 + num2;

case '-':

return num1-num2;

case '\*':

return num1\*num2;

case '/':

return num1/num2;

default:

return NaN;

}

}

calc('%', 8, 2);

**Задача 14**

Функция compare, принимающая две строки first и second, работает по следующему алгоритму:

Если первая строка больше второй то возвращается 1.

Если вторая строка больше первой то возвращается -1.

Во всех остальных случаях возвращается 0.

Подсказки

Вычисление длины строки: strings.length(str).

Перевод строки в верхний регистр strings.toUpperCase(str).

Проверка на то, что символ в верхнем регистре: strings.toUpperCase(char) === char;

**Решение**

import \* as strings from './strings';

const bigLettersCount = (str) => {

// BEGIN (write your solution here)

let result = 0;

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

if (strings.toUpperCase(str[i]) === str[i]) {

result++;

}

}

return result;

// END

};

const compare = (first, second) => {

const firstCount = bigLettersCount(first);

const secondCount = bigLettersCount(second);

// BEGIN (write your solution here)

if (firstCount > secondCount) {

return 1;

}

else if (secondCount > firstCount) {

return -1;

}

return 0;

// END

};

export const greaterThan = (first, second) => {

return compare(first, second) === 1;

};

export const lessThan = (first, second) => {

return compare(first, second) === -1;

};

// END

**Задача** **15**

Реализуйте и экспортируйте по умолчанию функцию, которая делает заглавной первую букву каждого слова в предложении.

solution('hello, world!'); // Hello, World!

Подсказки

Вычисление длины строки: strings.length(str).

Перевод строки/буквы в верхний регистр strings.toUpperCase(str).

**Решение**

import \* as strings from './strings';

// BEGIN (write your solution here)

export default (str) => {

let res = "";

let i = 0;

for (i = 0; i < strings.length(str); i++ ) {

if (i === 0 || str[i-1] === ' ') {

res += strings.toUpperCase(str[i]);

console.log(res);

} else {

res += str[i];

console.log(res);

}

}

return res;

};

('hello, world');

// END

**Задача 16**

Реализуйте и экспортируйте функцию addDigits, которая работает следующим образом:

Дано неотрицательное целое число num. Сложите все входящие в него цифры до тех пор, пока не останется одна цифра.

Для числа 38 процесс будет выглядеть так:

3 + 8 = 11

1 + 1 = 2

Результат: 2

Пример:

addDigits(10); // 1

addDigits(19); // 1

addDigits(38); // 2

addDigits(1259); // 8

Подсказка

Выделите процесс суммирования цифр в числе в отдельную функцию

**Решение**

import \* as strings from './strings';

// BEGIN (write your solution here)

const sum = (str) => {

str = String(str);

let acc = 0;

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

acc = acc + Number(str[i]);

console.log(acc);

}

return acc; // в виде цифры

};

export const addDigits = (num) => {

let result = num;

while (result > 9) {

result = sum(result);//здесь num передается в виде строки

console.log(result);

}

return result;

};

// END

addDigits(7777);

**Решение 2**

export const addDigits = (num) => {

let acc = 0;

while (num > 0) {

acc += num % 10;

num = Math.floor(num/10);

}

if(String(acc).length > 1) {

return addDigits(acc);

}

return acc;

};

// END

addDigits(7777);

**ПРАКТИКА**

**Задача**

**Реализуйте и экспортируйте функцию invertCase, которая меняет в строке регистр каждой буквы на противоположный.**

**invertCase('Hello, World!') // hELLO, wORLD!**

**invertCase('I loVe JS') // i LOvE js**

**РЕШЕНИЕ**

import \* as strings from './strings';

// BEGIN (write your solution here)

export const invertCase = (str) => {

let acc = "";

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

if (strings.toUpperCase(str[i]) === str[i]) {

acc = acc + strings.toLowerCase(str[i]);

} else if (strings.toUpperCase(str[i]) !== str[i]) {

acc = acc + strings.toUpperCase(str[i]);

console.log(acc);

}

}

return acc;

};

// END

invertCase("Hello, world!");

**Задача**

Счастливым билетом называют такой билет с шестизначным номером, где сумма первых трех цифр равна сумме последних трех.

Например, билет с номером 385916 - счастливый, так как 3 + 8 + 5 = 9 + 1 + 6

solution.js

Напишите и экспортируйте по умолчанию функцию, проверяющую является ли номер счастливым. Функция должна возвращать true, если билет счастливый, или false, если нет.

Примеры использования:

import isHappyTicket from './solution';

isHappyTicket(385916); // true

isHappyTicket(231002); // false

isHappyTicket(128722); // true

Подсказки

Преобразовать число в строку можно с помощью функции String:

String(1234); // "1234"

Преобразовать строку в число можно с помощью функции Number:

Number("456"); // 456

Используйте функцию length из модуля strings чтобы узнать длину строки:

strings.length('hello'); // 5

Используйте функцию substr из модуля strings чтобы получить подстроку из строки:

strings.substr('hello', 0, 4); // "hell"

**Решение**

import \* as strings from './strings';

// BEGIN (write your solution here) (write your solution here)

export default (str) => {

let tikket = String(str);

let sum1 = strings.substr(tikket, 0, 3);//строка из первых трех

console.log(sum1);

let sum2 = strings.substr(tikket, 3, 3);//строка из вторых трех

let result1 = 0

let result2 = 0;

let i = 0;

while(i<3){

result1 += Number(sum1[i]);

result2 += Number(sum2[i]);

console.log(result1);

i++;

}

if (result1 === result2) {

return true;

} else if (result1 !== result2){

return false;

}

};

// END

(312321);

**Задача,**

Палиндром — число, слово или текст, одинаково читающееся в обоих направлениях. Например: радар, топот.

solution.js

Реализуйте и экспортируйте по умолчанию функцию isPalindrome с использованием рекурсии.

Примеры использования:

import isPalindrome from './solution';

isPalindrome('radar'); // true

isPalindrome('a'); // true

isPalindrome('abs'); // false

Подсказки

Используйте функцию length из модуля strings чтобы узнать длину строки:

strings.length('hello'); // 5

Используйте функцию substr из модуля strings чтобы получить подстроку из строки:

strings.substr('hello', 0, 4); // "hell"

**Решение**

import \* as strings from './strings';

// BEGIN (write your solution here) (write your solution here)

const isPalindrome = (str) => {

let acc = '';

while (strings.length(acc) !== strings.length(str)) {

acc += str[strings.length(str) - 1 - strings.length(acc)];

}

if (acc === str) {

return true;

}

return false;

};

export default isPalindrome;

// END

**Функция Reverse**

const reverse = (value) => {

let string = String(value);

let acc = '';

while (acc.length !== string.length) {

acc += string[string.length - 1 - acc.length];

}

if (acc !== string) {

}

return acc;

}

reverse(2 , 3);

**Уровень 2 Составные данные**

**Задача 17**

points.js

Реализуйте и экспортируйте следующие функции для работы с точками:

quadrant - функция, которая вычисляет квадрант, в котором находится точка. Ниже приведена схема, показывающая номера квадрантов на плоскости.

+

2 | 1

|

+----------------+

|

3 | 4

+

const point = makePoint(1, 5);

quadrant(point); // 1

quadrant(makePoint(3, -3)); // 4

Если точка не принадлежит ни одному квадранту, то функция должна возвращать null.

symmetricalPoint - функция, возвращающая новую точку, симметричную относительно начала координат. Такая симметричность означает, что меняются знаки у x и y.

symmetricalPoint(makePoint(1, 5)); // makePoint(-1, -5)

distance - функция, вычисляющая расстояние между точками по формуле: d = sqrt((x2−x1)^2+(y2−y1)^2)

distance(makePoint(-2, -3), makePoint(-4, 4)); // ≈ 7.28

**Решение**

import { makePoint, getX, getY } from 'hexlet-points';

export const quadrant = (point) => {

const x = getX(point);

const y = getY(point);

if (x > 0 && y > 0) {

return 1;

} else if (x < 0 && y > 0) {

return 2;

} else if (x < 0 && y < 0) {

return 3;

} else if (x > 0 && y < 0) {

return 4;

}

return null;

};

export const symmetricalPoint = point => makePoint(-getX(point), -getY(point));

export const distance = (point1, point2) => {

const dX = getX(point2) - getX(point1);

const dY = getY(point2) - getY(point1);

return Math.sqrt(Math.pow(dX, 2) + Math.pow(dY, 2));

};

**// END**

**Задача 18**

Рассмотрим задачу представления отрезков на прямой плоскости. Каждый отрезок представляется как пара точек: начало и конец. Он может быть быть описан, например, так: [(1, 3), (5, 4)]. Это означает, что наш отрезок начинается в точке (1, 3) и заканчивается в точке (5, 4) координатной плоскости.

В этом задании необходимо разработать абстракцию для работы с отрезками (англ. сегмент), которая позволяет строить их из точек, извлекать из отрезков составные части (начало или конец сегмента), а так же получать текстовое представление сегмента.

Абстракция заключается в том, что конкретное представление сегмента определяется внутри функций, описывающих работу с отрезками и зависит от создателя библиотеки. С точки зрения клиента библиотеки (кода который ее вызывает), не важно как конкретно устроен сегмент, важно только то, что есть некоторый набор функций (абстракция), позволяющий работать с ним.

segments.js

Определите и экспортируйте следующие функции:

Конструктор makeSegment, который принимает на вход две точки и возвращает сегмент. Первая точка это начало сегмента, вторая это конец.

Селекторы startSegment и endSegment, которые извлекают из сегмента его начальную и конечную точку соответственно.

Вспомогательную функцию toStr, которая возвращает текстовое представление сегмента: [(1, 2), (-4, -2)].

Функцию midpointSegment, которая находит точку на середине отрезка по формулaм: x = (x1 + x2) / 2 и y = (y1 + y2) / 2.

import \* as points from 'hexlet-points';

// не важно, чем является segment с точки зрения реализации, главное, что с ним можно

// работать используя функции для работы с сегментами

const segment = makeSegment(points.makePoint(1, 2), points.makePoint(-4, -2));

console.log(toStr(segment)); // [(1, 2), (-4, -2)]

const point1 = startSegment(segment); // (1, 2)

console.log(points.toString(point1);

const point2 = endSegment(segment);

console.log(points.toString(point2)); // (-4, -2)

points.toString(startSegment(segment)) == points.toString(makePoint(1, 2)); // true

points.toString(midpointSegment(segment)); // (-1.5, 0)

Поскольку на текущий момент мы знакомы только с парами для работы с составными данными, то и реализация сегментов должна быть основана на них.

**Решение**

**Черновик**

import { makePoint, getX, getY, toString } from 'hexlet-points';

import { cons, car, cdr } from 'hexlet-pairs';

// BEGIN (write your solution here)

const point1 = makePoint(3 , 3);

const point2 = makePoint(2 , 2);

export const makeSegment = (point1 , point2) => cons(point1 , point2);

const segment = (point1, point2) => makeSegment(point1, point2);

console.log(segment);

export const startSegment = (segment) => car(segment);

export const endSegment = (segment) => cdr(segment);

export const toStr = (segment) => {

return "[" + toString(car(segment)) + ", " + toString(cdr(segment)) + "]";

};

console.log(toStr);

export const midpointSegment = (segment) => {

return makePoint(((getX(car(segment)) + getX(cdr(segment)))/2), ((getY(car(segment)) + getY(cdr(segment)))/2));

};

**Новое решение**

import { makePoint, getX, getY, toString } from 'hexlet-points';

import { cons, car, cdr } from 'hexlet-pairs';

// BEGIN (write your solution here)

//const makepoint = (x , y) => cons(x, y); //задаем функцию makepoint, аргументы x , y. Делаем ПАРУ

//const point1 = (x1, y1) => makePoint(x1, y1); // точка1 равна вызванной функции с аргументами (x1 , y1)

//const point2 = (x2, y2) => makePoint(x2, y2);

export const makeSegment = (point1, point2) => cons(point1, point2); //возвращает конструктор, который обьединяет 2 пары.

export const startSegment = segment => car(segment);//начинается работа с переменной сегмент( начало сегмента - точка с двумя координатами))

export const endSegment = segment => cdr(segment);

export const toStr = segment => '[' + toString(car(segment)) + ', ' + toString(cdr(segment)) + ']';

export const midpointSegment = (segment) => {

const middleX = segment => (getX(car(segment)) + getX(cdr(segment)))/2;//

const middleY = segment => (getY(car(segment)) + getY(cdr(segment)))/2;

return makePoint(middleX(segment), middleY(segment));//вызываем внутренние функции а аргументом segment, без него - не работает

};

const segment = makeSegment(makePoint(1, 2), makePoint(-4, -2)); // !! в примере немного другое выражение и оно не работает.

console.log(toStr(segment)); // [(1, 2), (-4, -2)]

console.log(toStr(endSegment(segment)));

//const point2 = endSegment(segment);

//console.log(toString(point2)); // (-4, -2)

// END

**Барьеры абстракции**

**Задача**

rectangles.js

Реализуйте абстракцию (набор функций) для работы с прямоугольниками, стороны которого всегда параллельны осям. При такой постановке, достаточно знать только три параметра для однозначного задания прямоугольника на плоскости: координаты левой-верхней точки, ширину и высоту. Зная их, мы всегда можем построить прямоугольник одним единственным способом.

|

4 | точка ширина

| \*-------------

3 | | |

| | | высота

2 | | |

| --------------

1 |

|

|---------------------------

0 1 2 3 4 5 6 7

Основной интерфейс:

makeRectangle - создает прямоугольник. Принимает параметры: левую-верхнюю точку, ширину и высоту.

Вспомогательные функции для выполнения расчетов:

square - возвращает площадь прямоугольника (a \* b).

perimeter - возвращает периметр прямоугольника (2 \* (a + b)).

containsTheOrigin - проверяет, принадлежит ли центр координат прямоугольнику (не лежит на границе прямоугольника, а находится внутри). Чтобы в этом убедиться, достаточно проверить, что все точки прямоугольника лежат в разных квадрантах.

Так как это интерфейсные функции, то они должны быть экспортированы. Если этого не сделать, система модулей js не даст ими воспользоваться.

// Создание прямоугольника:

// p - левая верхняя точка

// 4 - ширина

// 5 - высота

//

// p 4

// -----------

// | |

// | | 5

// | |

// -----------

const p = makePoint(0, 1);

const rectangle = makeRectangle(p, 4, 5);

// Вычисление площади прямоугольника

square(rectangle); // 20;

perimeter(rectangle); // 18

containsTheOrigin(rectangle); // false

containsTheOrigin(makeRectangle(makePoint(-4, 3), 5, 4)); // true

containsTheOrigin(makeRectangle(makePoint(-4, 4), 5, 2)); // false

containsTheOrigin(makeRectangle(makePoint(-4, 3), 2, 8)); // false

**Решение**

import { makePoint, getX, getY, quadrant } from 'hexlet-points';

import { cons, car, cdr, toString } from 'hexlet-pairs';

// BEGIN (write your solution here)

export const makeRectangle = (p, width, heigth) => cons(p, cons(width, heigth));

export const square = rectangle => getX(cdr(rectangle)) \* getY(cdr(rectangle));

export const perimeter = rectangle => (getX(cdr(rectangle)) + getY(cdr(rectangle))) \* 2;

export const containsTheOrigin = (rectangle) => {

const point1 = car(rectangle);

const point2 = makePoint(getX(car(rectangle)), (getY(car(rectangle)) - getY(cdr(rectangle))));

const point3 = makePoint((getX(car(rectangle)) + getX(cdr(rectangle))), (getY(car(rectangle)) - getY(cdr(rectangle))));

const point4 = makePoint((getX(car(rectangle)) + getX(cdr(rectangle))), getY(car(rectangle)));

if(quadrant(point1) === 2 && quadrant(point2) === 3 && quadrant(point3) === 4 && quadrant(point4) === 1) {

return true;

}

return false;

};

// END

Решение учителя

// BEGIN

export const makeRectangle = (point, width, height) => cons(point, cons(width, height));

export const startPoint = rectangle => car(rectangle);

export const height = rectangle => cdr(cdr(rectangle));

export const width = rectangle => car(cdr(rectangle));

export const square = rectangle => height(rectangle) \* width(rectangle);

export const perimeter = rectangle => 2 \* (height(rectangle) + width(rectangle));

export const containsTheOrigin = (rectangle) => {

const point1 = startPoint(rectangle);

const point2 = makePoint(getX(point1) + width(rectangle), getY(point1) - height(rectangle));

return quadrant(point1) === 2 && quadrant(point2) === 4;

};

// END

**Обьекты первого класса**

**Задача 20**

Во многих языках программирования существует интересная функция flip. Ее необычность заключается в том, что единственная цель этой функции это преобразовать другую функцию так, чтобы порядок ее аргументов был обратным. Например, у нас может быть функция pow, которая возводит первый аргумент в степень второго. flip вернет нам новую функцию, у которой первый аргумент это степень, а второй это число, которое нужно возводить в эту степень.

Эта функция бывает очень полезна при активной работе с функциями высшего порядка, она часто упрощает возможность комбинировать функции между собой.

solution.js

Реализуйте и экспортируйте функцию flip. Данная версия flip работает только с функциями от двух аргументов.

// subtraction

const sub = (a, b) => a - b;

const reverseSub = flip(sub);

sub(5, 3); // 2

reverseSub(5, 3); // -2

// exponentiation

const wop = flip(Math.pow);

Math.pow(1, 2); // 1

wop(1, 2); // 2

Math.pow(3, 2); // 9

wop(3, 2); // 8

**Решение 1**

export const flip = (func) => (a , b) => func(b , a);

**Решение 2**

export const flip = (func) => {

return (a , b) => {

return func(b , a);

};

};

**Черновик**

**const flip = (f) => (value) => f(reverse(value));**

**//функция, которая получает аргумент и возвращает его задом на перед**

**const reverse = (value) => {**

**let string = String(value);**

**let acc = '';**

**while (acc.length !== string.length) {**

**acc += string[string.length - 1 - acc.length];**

**}**

**if (acc !== string) {**

**}**

**return acc;**

**}**

**reverse(2 , 3);**

**// любая функция, к примеру вычитания с двумя аргументами**

**const delta = (x , y) => {**

**return x - y};**

**//функция которая возвращает значение, с учетом того, что аргументы поменялись местами**

**const unDelta = flip(delta);**

**delta(5 , 3); //должна возвращать 2**

**unDelta(5 , 3);//должна возвращать -2**