JS TASKS FOR STUDENTS:

Всем привет, тут буду писать описание задач. Задачи будут поделены на уровни сложности, но они всего лишь условные. Удачи, самураи ^-^

Ответы на эти вопросы предполагают реализацию функций на JavaScript. За каждым вопросом следуют тесты, которые должны успешно проходить решение (когда напишите функцию, надо будет ее вызвать и передать значения, что описаны ниже, чтобы проверить корректность работы и проверить сошелся ли у вас ожидаемый результат с тестами)

**Простые задания**

1. Реализуйте функцию isPrime(), которая возвращает true или false, указывая, является ли переданное ей число простым.

isPrime(0) // false

isPrime(1) // false

isPrime(17) // true

isPrime(10000000000000) // false

1. Реализуйте функцию factorial(), которая возвращает факториал переданного ей числа.

factorial(0) // 1

factorial(1) // 1

factorial(6) // 720

1. Реализуйте функцию fib(), возвращающую n-ное число Фибоначчи.

fib(0) // 0

fib(1) // 1

fib(10) // 55

fib(20) // 6765

1. Реализуйте функцию isSorted(), которая возвращает true или false в зависимости о того, отсортирован ли переданный ей числовой массив.

isSorted([]) // true

isSorted([-Infinity, -5, 0, 3, 9]) // true

isSorted([3, 9, -3, 10]) // false

1. Реализуйте функцию reverse(), которая обращает порядок следования символов переданной ей строки. Не пользуйтесь встроенной функцией reverse().

reverse('') // ''

reverse('abcdef') // 'fedcba'

1. Реализуйте функцию isPalindrome(), которая возвращает true или false в зависимости от того, является ли переданная ей строка палиндромом (функция нечувствительна к регистру и к наличию в строке пробелов).

isPalindrome('') // true

isPalindrome('abcdcba') // true

isPalindrome('abcd') // false

isPalindrome('A man a plan a canal Panama') // true

1. Реализуйте функцию isBalanced() которая принимает строку и возвращает true или false, указывая на то, сбалансированы ли фигурные скобки, находящиеся в строке.

isBalanced('}{') // false

isBalanced('{{}') // false

isBalanced('{}{}') // true

isBalanced('foo { bar { baz } boo }') // true

isBalanced('foo { bar { baz }') // false

isBalanced('foo { bar } }') // false

**Средней сложности задания**

1. Реализуйте функцию fib2(). Она похожа на функцию fib() из предыдущей группы заданий, но поддерживает числа вплоть до 50. Подсказка: используйте мемоизацию.

fib2(0) // 0

fib2(1) // 1

fib2(10) // 55

fib2(50) // 12586269025

1. Реализуйте функцию isBalanced2(). Она похожа на функцию isBalanced() из предыдущей группы заданий, но поддерживает три типа скобок: фигурные {}, квадратные [], и круглые (). При передаче функции строки, в которой имеются неправильные скобочные последовательности, функция должна возвращать false.

isBalanced2('(foo { bar (baz) [boo] })') // true

isBalanced2('foo { bar { baz }') // false

isBalanced2('foo { (bar [baz] } )') // false

1. Реализуйте функцию includes(), которая возвращает true или false в зависимости от того, встречается ли переданное ей число в переданном ей отсортированном массиве.

includes([1, 3, 8, 10], 8) // true

includes([1, 3, 8, 8, 15], 15) // true

includes([1, 3, 8, 10, 15], 9) // false

**Сложные задачки**

Некоторые задания из этой группы связаны с созданием структур данных. Не нужно запоминать все тонкости их функционирования, достаточно понимания их устройство, при этом сведения о предоставляемом ими интерфейсе можно найти в интернете. Далее, нужно знать, для чего эти структуры данных используются, каковы их ограничения в сравнении с другими структурами данных.

1. Реализуйте функцию permute(), которая возвращает массив строк, содержащий все пермутации заданной строки.

permute('') // []

permute('abc') // ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']

1. Реализуйте класс BinarySearchTree. Он должен поддерживать 4 метода: add(), has(), remove(), и size().

let tree = new BinarySearchTree

tree.add(1, 2, 3, 4)

tree.add(5)

tree.has(2) // true

tree.has(5) // true

tree.remove(3) // undefined

tree.size() // 4

Дополнительный ресурс для самураев, ссылка на некоторые инкубаторские материалы, где можете найти много полезного для изучения:  
  
1) [Incubator Students Public](https://docs.google.com/spreadsheets/d/1giJS8F2y1JSqY3tRLmnilzl9gl-LufGinowDO1O58y4/edit#gid=1853606021)

2) [Interview Questions](https://docs.google.com/spreadsheets/d/1M90pf_IKHxCHbC8UIuf1kqIMcmUkx2SZAYPDJ52qsn8/edit#gid=0)

Также ниже просили осветить тему перебираемых (или их еще называют итерируемых типов данных).

**Перебираемые (или итерируемые) типы данных в JavaScript** - это как коробки с игрушками, в которых много разных вещей, и ты можешь поочередно доставать каждую игрушку и играть с ней. Термины "итерируемые" и "перебираемые" часто используются взаимозаменяемо в контексте JavaScript. Оба термина относятся к типам данных, которые можно перебирать с использованием циклов, таких как цикл for или for...of.

Давайте сравним массив (перебираемый тип данных) с обычным списком вещей:

1. Массив (перебираемый):

Допустим, у вас есть коробка с разными игрушками: машинка, кукла, мяч и т.д.

Вы можете взять каждую игрушку по очереди и поиграть с ней.

В программировании, когда мы говорим о массиве, мы можем перебирать (или проходить через) каждую вещь в этой коробке, используя цикл.

let toys = ['машинка', 'кукла', 'мяч'];

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

console.log(toys[i]);

}

1. Обычный список (не перебираемый):

Теперь представь, что у тебя есть простой список вещей: книга, ручка, бумага. Но, к сожалению, у тебя нет специальной коробки, и ты не можешь поочередно доставать каждую вещь.

let items = 'книга, ручка, бумага';

// К сожалению, мы не можем так просто перебрать вещи в этом списке

// и взять каждую по очереди.

**НЮАНС:**

Здесь items.split(', ') превращает строку в массив, разбив её по запятой и пробелу. Теперь вы можете использовать цикл for...of, чтобы перебрать каждый элемент в массиве и, следовательно, каждую "вещь" в строке.

Таким образом, строка items тоже может быть перебрана, но для этого нужно предварительно преобразовать её в массив с помощью метода split.

let items = 'книга, ручка, бумага';

for (let items of items.split(', ')) {

console.log(items);

}

Вот несколько примеров итерируемых типов данных в JS:

* Array - Массив представляет собой упорядоченный набор элементов, к которым можно обращаться по индексу. Они итерируемы, потому что вы можете использовать цикл for...of для перебора каждого элемента массива.

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

for (let el of arr) {

console.log(el);

}

* String - Строка представляет собой последовательность символов. Она итерируема, потому что вы можете использовать цикл for...of для перебора каждого символа строки(в предыдущем примере строка была не итерируемой, так как мы итерировались не посимвольно, а конкретно по словам). В данном случае мы можем итерироваться посимвольно

let str = 'Привет, мир!';

for (let sym of str) {

console.log(sym);

}

Объекты в JavaScript не считаются итерируемыми напрямую, что означает, что вы не можете использовать цикл for...of для непосредственного перебора их свойств. Однако вы можете перебирать свойства объекта с использованием цикла for...in. В этом случае цикл будет перебирать имена (ключи) свойств объекта, и вы сможете получить доступ к значениям, используя эти ключи.

const object = { a: 1, b: 2, c: 3 };

for (const property in object) {

console.log(`${property}: ${object[property]}`);

}