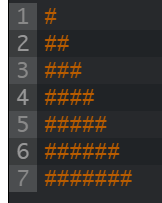
JavaScript задания

1. **Треугольник.** Напишите цикл,  выводит такой треугольник:



1. **FizzBuzz**. Напишите программу, которая выводит через console.log все числа от 1 до 100, с двумя исключениями. Для чисел, нацело делящихся на 3, она должна выводить ‘Fizz’, а для чисел, делящихся на 5 (но не на 3) – ‘Buzz’.Когда сумеете – исправьте её так, чтобы она выводила «FizzBuzz» для всех чисел, которые делятся и на 3 и на 5.
2. **Шахматная доска.** Напишите программу, создающую строку, содержащую решётку 8х8, в которой линии разделяются символами новой строки. На каждой позиции либо пробел, либо #. В результате должна получиться шахматная доска

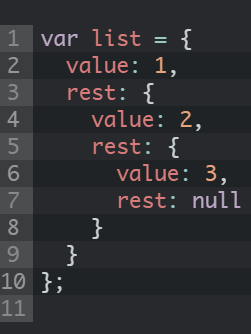
****

1. **Минимум**. Напишите функцию min, принимающую два аргумента, и возвращающую минимальный из них.
2. **Считаем бобы.**Символ номер N строки можно получить, добавив к ней .charAt(N) ( “строчка”.charAt(5) ) – схожим образом с получением длины строки при помощи .length. Возвращаемое значение будет строковым, состоящим из одного символа (к примеру, “к”). У первого символа строки позиция 0, что означает, что у последнего символа позиция будет string.length – 1. Другими словами, у строки из двух символов длина 2, а позиции её символов будут 0 и 1.Напишите функцию countBs, которая принимает строку в качестве аргумента, и возвращает количество символов “B”, содержащихся в строке.Затем напишите функцию countChar, которая работает примерно как countBs, только принимает второй параметр — символ, который мы будем искать в строке (вместо того, чтобы просто считать количество символов “B”). Для этого переделайте функцию countBs.
3. **Сумма диапазона.**  Напишите функцию range, принимающую два аргумента, начало и конец диапазона, и возвращающую массив, который содержит все числа из него, включая начальное и конечное.Затем напишите функцию sum, принимающую массив чисел и возвращающую их сумму. Запустите указанную выше инструкцию и убедитесь, что она возвращает 55.В качестве бонуса дополните функцию range, чтобы она могла принимать необязательный третий аргумент – шаг для построения массива. Если он не задан, шаг равен единице. Вызов функции range(1, 10, 2) должен будет вернуть [1, 3, 5, 7, 9]. Убедитесь, что она работает с отрицательным шагом так, что вызов range(5, 2, -1) возвращает [5, 4, 3, 2].
4. **Обращаем массив вспять.** Напишите две функции, reverseArray и reverseArrayInPlace. Первая получает массив как аргумент и выдаёт новый массив, с обратным порядком элементов. Вторая работает как оригинальный метод reverse – она меняет порядок элементов на обратный в том массиве, который был ей передан в качестве аргумента. Не используйте стандартный метод reverse.
5. **Список.**Объекты могут быть использованы для построения различных структур данных. Часто встречающаяся структура – список (не путайте с массивом). Список – связанный набор объектов, где первый объект содержит ссылку на второй, второй – на третий, и т.п.

Списки удобны тем, что они могут делиться частью своей структуры. Например, можно сделать два списка, {value: 0, rest: list} и {value: -1, rest: list}, где list – это ссылка на ранее объявленную переменную. Это два независимых списка, при этом у них есть общая структура list, которая включает три последних элемента каждого из них. Кроме того, оригинальный список также сохраняет свои свойства как отдельный список из трёх элементов.

Напишите функцию arrayToList, которая строит такую структуру, получая в качестве аргумента [1, 2, 3], а также функцию listToArray, которая создаёт массив из списка. Также напишите вспомогательную функцию prepend, которая получает элемент и создаёт новый список, где этот элемент добавлен спереди к первоначальному списку, и функцию nth, которая в качестве аргументов принимает список и число, а возвращает элемент на заданной позиции в списке, или же undefined в случае отсутствия такого элемента.

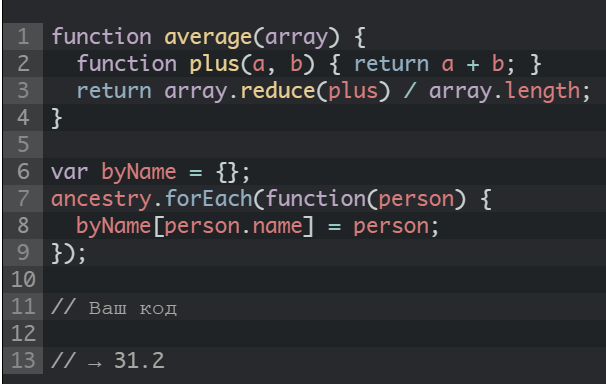
Если ваша версия nth не рекурсивна, тогда напишите её рекурсивную версию.



1. **Глубокое сравнение.** Оператор == сравнивает переменные объектов, проверяя, ссылаются ли они на один объект. Но иногда полезно было бы сравнить объекты по содержимому.Напишите функцию deepEqual, которая принимает два значения и возвращает true, только если это два одинаковых значения или это объекты, свойства которых имеют одинаковые значения, если их сравнивать рекурсивным вызовом deepEqual.Чтобы узнать, когда сравнивать величины через ===, а когда – объекты по содержимому, используйте оператор typeof. Если он выдаёт “object” для обеих величин, значит нужно делать глубокое сравнение. Не забудьте об одном дурацком исключении, случившемся из-за исторических причин: “typeof null” тоже возвращает “object”
2. **Свертка**. Используйте метод reduce в комбинации с concat для свёртки массива массивов в один массив, у которого есть все элементы входных массивов.
3. **Разница в возрасте матерей и их детей**

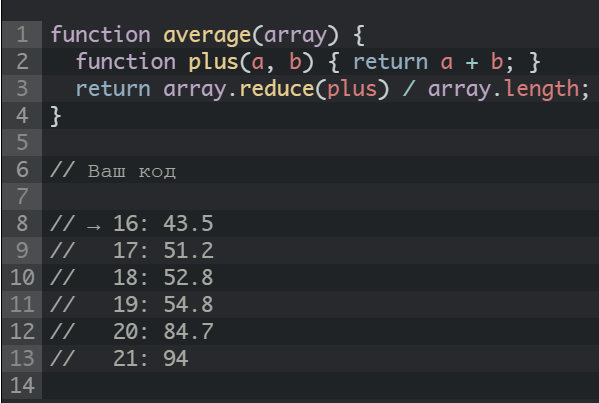
Используя набор данных из примера, подсчитайте среднюю разницу в возрасте между матерями и их детьми (это возраст матери во время появления ребёнка). Можно использовать функцию average, приведённую в главе.

Обратите внимание – не все матери, упомянутые в наборе, присутствуют в нём. Здесь может пригодиться объект byName, который упрощает процедуру поиска объекта человека по имени.



1. Историческая ожидаемая продолжительность жизни.

Мы считали, что только последнее поколение людей дожило до 90 лет. Давайте рассмотрим этот феномен поподробнее. Подсчитайте средний возраст людей для каждого из столетий. Назначаем столетию людей, беря их год смерти, деля его на 100 и округляя: Math.ceil(person.died / 100).



1. Every и some

У массивов есть ещё стандартные методы every и some. Они принимают как аргумент некую функцию, которая, будучи вызванной с элементом массива в качестве аргумента, возвращает true или false. Так же, как && возвращает true, только если выражения с обеих сторон оператора возвращают true, метод every возвращает true, когда функция возвращает true для всех элементов массива. Соответственно, some возвращает true, когда заданная функция возвращает true при работе хотя бы с одним из элементов массива. Они не обрабатывают больше элементов, чем необходимо – например, если some получает true для первого элемента, он не обрабатывает оставшиеся.

Напишите функции every и some, которые работают так же, как эти методы, только принимают массив в качестве аргумента.

