Задачи по JavaScript. FED Training 2017.

1. Треугольник

Напишите цикл, выводящий такой треугольник:

```
#
##
###
####
#####
######
#######
```

2. 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.

3. Шахматная доска

Напишите программу, создающую строку, содержащую решётку 8x8, в которой линии разделяются символами новой строки. На каждой позиции либо пробел, либо #. В результате должна получиться шахматная доска. Когда справитесь, сделайте размер доски переменным, чтобы можно было создавать доски любого размера.

Пример доски 8x8:

```
# # # #
 # # # #
# # # #
 # # # #
```

4. Минимум

Напишите функцию min, принимающую два аргумента, и возвращающую минимальный из них.

5. Рекурсия

Ноль чётный.
Единица нечётная.
У любого числа N чётность такая же, как у N-2.

Напишите рекурсивную функцию isEven согласно этим правилам (любое число необходимо рекурсивно привести к 0 или 1). Она должна принимать число и возвращать булевское значение. Потестируйте её на 50 и 75. Попробуйте задать ей -1. Почему она ведёт себя таким образом? Можно ли её как-то исправить?

```
console.log(isEven(50));
// → true
console.log(isEven(75));
// → false
console.log(isEven(-1));
// → ??
```

6. Считаем бобы

Символ номер N строки можно получить, добавив к ней .charAt(N) `( "строчка".charAt(5) )`. 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.

7. 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].

```
console.log(sum(range(1, 10)));
// → 55
console.log(range(5, 2, -1));
// → [5, 4, 3, 2]
```

8. 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.

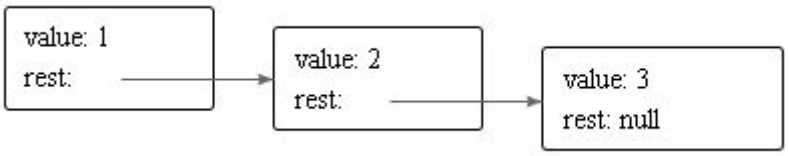
```
console.log(reverseArray(["A", "B", "C"]));
// → ["C", "B", "A"];
var arrayValue = [1, 2, 3, 4, 5];
reverseArrayInPlace(arrayValue);
console.log(arrayValue);
// → [5, 4, 3, 2, 1]
```

9. 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.

```
var list = {
  value: 1,
  rest: {
    value: 2,
    rest: {
      value: 3,
      rest: null
    }
  }
};
```

В результате объекты формируют цепочку:



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.

```
console.log(arrayToList([10, 20]));
// → {value: 10, rest: {value: 20, rest: null}}
console.log(listToArray(arrayToList([10, 20, 30])));
// → [10, 20, 30]
console.log(prepend(10, prepend(20, null)));
// → {value: 10, rest: {value: 20, rest: null}}
console.log(nth(arrayToList([10, 20, 30]), 1));
// → 20
```

10. 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”.

```
var obj = {here: {is: "an"}, object: 2};
console.log(deepEqual(obj, obj));
// → true
console.log(deepEqual(obj, {here: 1, object: 2}));
// → false
console.log(deepEqual(obj, {here: {is: "an"}, object: 2}));
// → true
```

11. Свертка

Используйте метод reduce в комбинации с concat для свёртки массива массивов в один массив, у которого есть все элементы входных массивов.

```
var arrays = [[1, 2, 3], [4, 5], [6]];
// Ваш код тут
// → [1, 2, 3, 4, 5, 6]
```

12. Разница в возрасте матерей и их детей

Используя набор данных из db.json, подсчитайте среднюю разницу в возрасте между матерями и их детьми (это возраст матери во время появления ребёнка).
Можно использовать функцию average, приведённую ниже. Обратите внимание – не все матери, упомянутые в наборе, присутствуют в нём. Здесь может пригодиться объект byName, который упрощает процедуру поиска объекта человека по имени.

```
function average(array) {
  function plus(a, b) { return a + b; }
  return array.reduce(plus) / array.length;
}

var byName = {};
ancestry.forEach(function(person) {
  byName[person.name] = person;
});

// Ваш код тут

// → 31.2
```

13. Историческая ожидаемая продолжительность жизни

Мы считали, что только последнее поколение людей дожило до 90 лет. Давайте рассмотрим этот феномен подробнее. Используя набор данных из db.json, 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).

```
function average(array) {
  function plus(a, b) { return a + b; }
  return array.reduce(plus) / array.length;
}

// Тут ваш код

// → 16: 43.5
//   17: 51.2
//   18: 52.8
//   19: 54.8
//   20: 84.7
//   21: 94
```

В качестве призовой игры напишите функцию groupBy, абстрагирующую операцию группировки. Она должна принимать массив и функцию, которая подсчитывает группу для элементов массива, и возвращать объект, который сопоставляет названия групп массивам членов этих групп.

14. Every и some

У массивов есть 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.

```
// Ваш код тут

console.log(every([NaN, NaN, NaN], isNaN));
// → true
console.log(every([NaN, NaN, 4], isNaN));
// → false
console.log(some([NaN, 3, 4], isNaN));
// → true
console.log(some([2, 3, 4], isNaN));
// → false
```

15. Повтор

Допустим, у вас есть функция primitiveMultiply, которая в 50% случаев перемножает 2 числа, а в остальных случаях выбрасывает исключение типа MultiplicatorUnitFailure. Напишите функцию, обёртывающую эту, и просто вызывающую её до тех пор, пока не будет получен успешный результат. Убедитесь, что вы обрабатываете только нужные вам исключения.

```
function MultiplicatorUnitFailure() {}

function primitiveMultiply(a, b) {
  if (Math.random() < 0.5)
    return a * b;
  else
    throw new MultiplicatorUnitFailure();
}

function reliableMultiply(a, b) {
  // Ваш код
}

console.log(reliableMultiply(8, 8));
// → 64
```

16. Регулярный гольф

«Гольфом» в коде называют игру, где нужно выразить заданную программу минимальным количеством символов. Регулярный гольф – практическое упражнение по написанию наименьших возможных регулярных выражений для поиска заданного шаблона, и только его.
Для каждой из подстрок напишите регулярное выражение для проверки их нахождения в строке. Регулярное выражение должно находить только эти указанные подстроки. Не волнуйтесь насчёт границ слов, если это не упомянуто особо. Когда у вас получится работающее регулярное выражение, попробуйте его уменьшить.
— car и cat
— pop и prop
— ferret, ferry, и ferrari
— Любое слово, заканчивающееся на ious
— Пробел, за которым идёт точка, запятая, двоеточие или точка с запятой.
— Слово длинее шести букв

— Слово без букв e

```
// Впишите свои регулярки

verify(/.../,
       ["my car", "bad cats"],
       ["camper", "high art"]);

verify(/.../,
       ["pop culture", "mad props"],
       ["plop"]);

verify(/.../,
       ["ferret", "ferry", "ferrari"],
       ["ferrum", "transfer A"]);

verify(/.../,
       ["how delicious", "spacious room"],
       ["ruinous", "consciousness"]);

verify(/.../,
       ["bad punctuation ."],
       ["escape the dot"]);

verify(/.../,
       ["hottentottententen"],
       ["no", "hotten totten tenten"]);

verify(/.../,
       ["red platypus", "wobbling nest"],
       ["earth bed", "learning ape"]);

function verify(regexp, yes, no) {
  // Ignore unfinished exercises
  if (regexp.source == "...") return;
  yes.forEach(function(s) {
    if (!regexp.test(s))
      console.log("Не нашлось '" + s + "'");
  });
  no.forEach(function(s) {
    if (regexp.test(s))
      console.log("Неожиданное вхождение '" + s + "'");
```

```
  });
}
```

17. Кавычки в тексте

Допустим, вы написали рассказ, и везде для обозначения диалогов использовали одинарные кавычки. Теперь вы хотите заменить кавычки диалогов на двойные, и оставить одинарные в сокращениях слов типа aren't.

Придумайте шаблон, различающий два этих использования кавычек, и напишите вызов метода replace, который производит замену.

18. Снова числа

Последовательности цифр можно найти простым регулярным выражением /\d+/.

Напишите выражение, находящее только числа, записанные в стиле JavaScript. Оно должно поддерживать возможный минус или плюс перед числом, десятичную точку, и экспоненциальную запись 5e-3 или 1E10 – опять-таки с возможными плюсом или минусом. Также заметьте, что до или после точки не обязательно могут стоять цифры, но при этом число не может состоять из одной точки. То есть, .5 или 5. – допустимые числа, а одна точка сама по себе – нет.

```
// Впишите сюда регулярное выражение.
var number = /^...$/;

// Tests:
["1", "-1", "+15", "1.55", ".5", "5.", "1.3e2", "1E-4",
 "1e+12"].forEach(function(s) {
  if (!number.test(s))
    console.log("Не нашла '" + s + "'");
});
["1a", "+-1", "1.2.3", "1+1", "1e4.5", ".5.", "1f5",
 "."].forEach(function(s) {
  if (number.test(s))
    console.log("Неправильно принято '" + s + "'");
});
```