**1 Треугольник в цикле**

Напишите цикл, который за 7 вызовов console.log выводит такой треугольник:

#

##

###

####

#####

######

#######

Будет полезно знать, что длину строки можно узнать, приписав к переменной .length.

let abc = "abc";

console.log(abc.length);

// → 3

**2 FizzBuzz**

Напишите программу, которая выводит через console.log все цифры 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.  
  
(На самом деле, этот вопрос подходит для собеседований, и говорят, он позволяет отсеивать довольно большое число кандидатов. Поэтому, когда вы решите эту задачу, можете себя похвалить)

**3 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.

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

# # # #

Когда справитесь, сделайте размер доски переменным, чтобы можно было создавать доски любого размера.

Функции

**4 Минимум**

В предыдущей главе была представлена стандартная функция Math.min, которая возвращает наименьший из ее аргументов. Теперь мы можем сами создать нечто подобное. Напишите функцию min, которая принимает два аргумента и возвращает их минимум.

**5 Рекурсия**

Как мы видели, оператор % (оператор остатка) можно применять для проверки, является число четным или нечетным. Для этого нужно использовать %2, чтобы узнать, делится ли оно на два. Вот еще один способ определить, является ли положительное целое число четным или нечетным:

* ноль четный;
* единица нечетная;
* четность любого другого числа N совпадает с четностью N - 2.

Определите рекурсивную функцию isEven, соответствующую этому описанию. Функция должна принимать один параметр (положительное целое число) и возвращать логическое значение. Проверьте эту функцию на числах 50 и 75. Посмотрите, как она ведет себя

для -1. Почему? Можете ли вы придумать способ, как это исправить?

**6 Подсчет букв**

Чтобы получить N-й символ или букву из строки, нужно написать "string"[N]. Возвращаемым значением будет строка, содержащая только один символ (например, "Ь"). Первый символ имеет позицию О, в результате чего последний находится в позиции string. length - 1. Другими словами, длина строки из двух символов равна 2, а ее символы находятся в позициях О и 1.

Напишите функцию countBs, которая принимает строку в качестве единственного аргумента и возвращает число, показывающее, сколько больших букв «В» содержится в этой строке.

Затем напишите функцию countChar, которая ведет себя как countBs, за исключением того, что принимает второй аргумент, указывающий, какие именно символы нужно посчитать (вместо того чтобы считать только большие буквы ~в~). Перепишите countBs, чтобы использовать эту новую функцию.

[Структуры данных: объекты и массивы](http://habrahabr.ru/post/240813/)

**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].

|  |  |
| --- | --- |
| 1  2  3  4 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var list = {    value: 1,    rest: {      value: 2,      rest: {        value: 3,        rest: null      }    }  }; |

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  2  3  4  5  6  7  8 | 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”.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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 |