22 популярные задачи

на собеседовании по Javascript

> *Задачи*

- https://www.codewars.com/kata

  > *Варианты решения, Методы языка*

- https://developer.mozilla.org/ru/docs/Web/JavaScript

> *Автор: Кирилл Остапенко @khbfr*

Оглавление

Введение

Любая задача решается применением конструкции - шаблона решения:

> *Цикл перебора For loop*

```
for ([начало]; [условие]; [шаг]) выражения
```

> *И логическая часть - условия If __ Else*

```
if (условие)
   инструкция1
else
   инструкция2
```

Давайте взглянем на приведённую задачу и применим к ней этот шаблон

Задача: дублирующий кодер

Цель этого упражнения — преобразовать строку в новую строку, где каждый символ в новой строке — это «(», если этот символ встречается в исходной строке только один раз, или «)», если этот символ встречается в исходной строке более одного раза. Нужно игнорировать заглавные буквы при определении, является ли символ дубликатом.

Требуемый результат

```
"din"      =>  "((("
"recede"   =>  "()()()"
"Success"  =>  ")())())"
"(( @"     =>  "))(("
```

Решение циклом for

```
function duplicateEncode(word){
    let unique='';
    word = word.toLowerCase();
    for(let i=0; i<word.length; i++){
        if(word.lastIndexOf(word[i]) == word.indexOf(word[i])){
            unique += '(';
        }
        else{
            unique += ')';
        }
    }
    return unique;
}
```

Отлично, но для любой задачи есть много вариантов решения. Попробуем использовать map, метод создаёт новый массив с результатом вызова указанной функции для каждого элемента массива.

Решение через map

```
function duplicateEncode(word){
  return word
    .toLowerCase()
    .split('')
    .map( function (a, i, w) {
      return w.indexOf(a) == w.lastIndexOf(a) ? '(' : ')'
    })
    .join('');
}
```

А это решение, любимо начинающими разработчиками - так называемый one line

Решение one line

```
function duplicateEncode(word) {
  word = word.toLowerCase();
  return word.replace(/./g, m => word.indexOf(m) == word.lastIndexOf(m) ? '(' : ')');
}
```

Ок, начало положено - давайте попробуем решить и другие задачи

Задача: найдите недостающую букву

Напишите метод, который принимает на вход массив последовательных (возрастающих) букв и возвращает отсутствующую букву в массиве. Вы всегда получите допустимый массив. И всегда будет отсутствовать ровно одна буква. Длина массива всегда будет не менее 2. Массив всегда будет содержать буквы только в одном регистре.

Требуемый результат

```
["a","b","c","d","f"] -> "e"
["O","Q","R","S"] -> "P"
```

(Используйте английский алфавит из 26 букв!)

Решение

```
function findMissingLetter(array) {
  let first = array[0].charCodeAt(0)
  for (let i = 1; i < array.length; i++) {
    if (first + i !== array[i].charCodeAt(0)) {
      return String.fromCharCode(first + i)
    }
  }
  throw new Error("Invalid input")
}
```

- Метод charCodeAt() возвращает числовое значение Юникода для символа по указанному индексу

- Статический метод String.fromCharCode() возвращает строку, созданную из указанной последовательности значений единиц кода UTF-16

Решение через slice

```
const findMissingLetter = (array) => {
  const alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'.split('');
  const start = alphabet.indexOf(array[0]);
  return alphabet.slice(start, start + array.length).find(el => !array.includes(el));
};
```

- Метод slice() возвращает новый массив, содержащий копию части исходного массива
- Метод find() возвращает значение первого найденного в массиве элемента
- Метод includes() определяет, содержит ли массив определённый элемент, возвращая в зависимости от этого true или false

Решение с использованием filter

```
function findMissingLetter(str) {
    let alphabet = 'abcdefghijklmnopqrstuvwxyz';
    if(str[0] == str[0].toUpperCase()) {
      alphabet = alphabet.toUpperCase(); }
    alphabet = alphabet.split('');
    let index = alphabet.indexOf(str[0]);
    let strip = alphabet.slice(index, index + str.length)
    return strip.filter(x => !str.includes(x))[0];
}
```

- Метод filter() создаёт новый массив со всеми элементами, прошедшими проверку, задаваемую в передаваемой функции.

Задача: перемещение нулей в конец

Напишите алгоритм, который берет массив и перемещает все нули в конец, сохраняя порядок остальных элементов.

Требуемый результат

```
moveZeros([false,1,0,1,2,0,1,3,"a"])
// returns[false,1,1,2,1,3,"a",0,0]
```

Решение по шаблону For .. if _ else

```
const moveZeros = function (arr) {
  let result = [];
  let zeros = [];
  for (let i in arr) {
    if (arr[i] === 0) {
      zeros.push(arr[i]);
    } else if (arr[i] !== 0) {
      result.push(arr[i]);
    }
```

```
  }
  return result.concat(zeros);
}
```

- Метод push() добавляет один или более элементов в конец массива и возвращает новую длину массива.
- Метод concat() возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в качестве аргументов.

Решение с аккумулятором

```
const moveZeros = function (arr) {
  return arr.reduceRight(function(prev, curr) {
    if (curr !== 0) {
      prev.unshift(curr);
    }
    else {
      prev.push(curr);
    }
    return prev;
  }, []);
}
```

- Метод reduceRight() применяет функцию к аккумулятору и каждому значению массива (справа-налево), сводя его к одному значению.
- Метод unshift() добавляет один или более элементов в начало массива и возвращает новую длину массива.

Задача: преобразование строки в верблюжий регистр

Завершите метод, чтобы он преобразовывал слова, разделенные тире/подчеркиванием, в верблюжий регистр. Первое слово в выводе должно быть написано с заглавной буквы только в том случае, если исходное слово было написано с заглавной буквы (известный как верхний верблюжий регистр, также часто называемый регистром Паскаля).

Требуемый результат

```
"the-stealth-warrior" gets converted to "theStealthWarrior"
"The_Stealth_Warrior" gets converted to "TheStealthWarrior"
```

Хоть и эду задачу можно решить шаблоном For .. if _ else, лучше углубимся в методы языка.

Решение через RegExp

```
function toCamelCase(str){
      let regExp=/[-_]\w/ig;
      return str.replace(regExp,function(match){
            return match.charAt(1).toUpperCase();
       });
}
```

- Конструктор RegExp создаёт объект регулярного выражения для сопоставления текста с шаблоном.
- Метод match() возвращает получившиеся совпадения при сопоставлении строки с регулярным выражением.
- Метод replace() возвращает новую строку с некоторыми или всеми сопоставлениями с шаблоном, заменёнными на заменитель. Шаблон может быть строкой или регулярным выражением, а заменитель может быть строкой или функцией, вызываемой при каждом сопоставлении.
- Метод toUpperCase() возвращает значение строки, на которой он был вызван, преобразованное в верхний регистр.

Решение For .. if _ else

Ок - ок, если нужно значит так надо

```
function toCamelCase(str){
  let arr = str.split('');
  for(i = 0; i < arr.length; i++){
    let letter = arr[i];
    if(letter == '_' || letter == '-') {
      arr[i + 1] = arr[i + 1].toUpperCase();
      arr[i] = '';
    }
  }
  return arr.join('');
}
```

- Метод split() разбивает объект String на массив строк путём разделения строки указанной подстрокой.

Задача: сумма цифр / цифровой корень

Учитывая n, возьмите сумму цифр n. Если это значение имеет более одной цифры, продолжайте уменьшать таким образом, пока не будет получено однозначное число. Ввод будет неотрицательным целым числом.

Требуемый результат

```
    16  -->  1 + 6 = 7
   942  -->  9 + 4 + 2 = 15  -->  1 + 5 = 6
132189  -->  1 + 3 + 2 + 1 + 8 + 9 = 24  -->  2 + 4 = 6
493193  -->  4 + 9 + 3 + 1 + 9 + 3 = 29  -->  2 + 9 = 11  -->  1 + 1 = 2
```

Решение For .. if _ else и рекурсия

```
function digital_root(n) {
  if (n < 10)
    return n;

  for (let sum = 0, i = 0, n = String(n); i < n.length; i++)
    sum += Number(n[i]);
```

```
  return digital_root(sum);
}
```

Среди функций отдельно можно выделить рекурсивные функции. Их суть состоит в том, что функция вызывает саму себя.

Решение с eval

Не используйте eval без необходимости!

```
function digital_root(n){
    n = eval(n.toString().split('').join('+'));
    if (n > 9) {
        return digital_root(n);
    }
    return n;
}
```

- Метод eval() выполняет JavaScript-код, представленный строкой.
- Метод join() объединяет все элементы массива (или массивоподобного объекта) в строку.

Решение с reduce

```
function digital_root(n) {
  while (n > 9) { n = (''+n).split('').reduce(function(s,d) {return +s + +d;}); }
  return n;
}
```

- Метод reduce() применяет функцию reducer к каждому элементу массива (слева-направо), возвращая одно результирующее значение.

Задача: построить башню

Постройте башню в форме пирамиды, учитывая положительное целое число этажей. Башенный блок представлен символом «*».

Требуемый результат

Например, башня в 3 этажа выглядит так:

```
  "  *  ",
  " *** ",
  "*****"
```

Решение с repeat

```
function towerBuilder(n) {
  return Array.from({length: n}, function(v, k) {
    const spaces = ' '.repeat(n - k - 1);
    return spaces + '*'.repeat(k + k + 1) + spaces;
  });
}
```

- Метод repeat() конструирует и возвращает новую строку, содержащую указанное количество соединённых вместе копий строки, на которой он был вызван.

Решение по шаблону

```
function towerBuilder(nFloors) {
  let arrTower = [],
      blocks;
  function buildFlour(blocks) {
    let maxSize = nFloors == 1 ? 1 : nFloors * 2 - 1,
        floor = '';
    for(let i = 0; i < maxSize; i++) {
      let space = (maxSize - blocks) / 2;
      if(maxSize - blocks !== 0) {
          if((i + 1) <= space || (i + 1) > blocks + space) {
            floor += ' ';
          } else {
            floor += '*';
          }
      } else {
        floor += '*';
      }
    }
    return floor;
  }
  for(var i = 0; i < nFloors; i++) {
    var blocks = i == 0 ? 1 : blocks + 2;
    arrTower.push(buildFlour(blocks));
  }
  return arrTower;
}
```

Выглядит как монстр, не правда ли ;)

Задача: глубокий подсчет массива

Array.prototype.length даст вам количество элементов верхнего уровня в массиве. Ваша задача — создать функцию deepCount, которая возвращает количество ВСЕХ элементов в массиве, включая любые внутри массивов внутреннего уровня.

Требуемый результат

```
deepCount([1, 2, 3]);
//>>>>> 3
deepCount(["x", "y", ["z"]]);
//>>>>> 4
deepCount([1, 2, [3, 4, [5]]]);
//>>>>> 7
```

Решение по шаблону

```
function deepCount(a){
  let count = a.length;
  for (let i=0; i<a.length; i++) if (Array.isArray(a[i])) count += deepCount(a[i]);
  return count;
}
```

- Свойство length объекта, который является экземпляром типа Array , устанавливает или возвращает число элементов этого массива.

Решение с while

```
function deepCount(arr) {
  const stack = [...arr]
  let size = 0
  while (stack.length) {
    const next = stack.pop()
    size += 1
    if (Array.isArray(next)) {
      stack.push(...next)
    }
  }
  return size
}
```

- Оператор while создаёт цикл, выполняющий заданную инструкцию, пока истинно проверяемое условие. Логическое значение условия вычисляется перед исполнением тела цикла.
- Метод pop() удаляет последний элемент из массива и возвращает его значение.
- Метод Array.isArray() возвращает true, если объект является массивом и false, если он массивом не является.

Решение с JSON

```
function deepCount(a) {
  return JSON.stringify(a).replace(/[^[,]|\[]/g, '').length;
}
```

- Метод JSON.stringify() преобразует значение JavaScript в строку JSON, возможно с заменой значений, если указана функция замены, или с включением только определённых свойств, если указан массив замены.

Задача: Перестановки

В этой задаче вы должны создать все перестановки непустой входной строки и удалить дубликаты, если они есть. Это означает, что вы должны перетасовать все буквы из ввода во всех возможных порядках.

Требуемый результат

```
* With input 'a'
* Your function should return: ['a']
* With input 'ab'
```

```
* Your function should return ['ab', 'ba']
* With input 'aabb'
* Your function should return ['aabb', 'abab', 'abba', 'baab', 'baba', 'bbaa']
```

Решение по шаблону

```
function permutations(string) {
  let arr = string.split(''), tmp = arr.slice(), heads = [], out = [];
  if(string.length == 1) return [string];
  arr.forEach(function(v, i, arr) {
    if(heads.indexOf(v) == -1) {
      heads.push(v);
      tmp.splice(tmp.indexOf(v), 1);
      permutations(tmp.join('')).forEach(function(w) {out.push(v + w);});
      tmp.push(v);
    }
  });
  return out;
}
```

- Метод slice() возвращает новый массив, содержащий копию части исходного массива.
- Метод forEach() выполняет указанную функцию один раз для каждого элемента в массиве.
- Метод splice() изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые.

Решение с substring

```
function permutations(str) {
 return (str.length <= 1) ? [str] :
         Array.from(new Set(
           str.split('')
              .map((char, i) => permutations(str.substr(0, i) + str.substr(i +
1)).map(p => char + p))
              .reduce((r, x) => r.concat(x), [])
         ));
}
```

- Метод Array.from() создаёт новый экземпляр Array из массивоподобного или итерируемого объекта.
- Метод substring() возвращает подстроку строки между двумя индексами, или от одного индекса и до конца строки.
- Метод concat() возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в качестве аргументов.

Решение с yield

```
function permutations(chs) {
    return [...new Set(
        Array.from( heapsPerms((chs+'').split('')),
```

```
        str => str.join('') )
    )];
}
function *heapsPerms(chs, n = chs.length) {
    if (n <= 1) yield chs.slice();
    else for (let i = 0; i < n; i++) {
        yield *heapsPerms(chs, n-1);
        swap(chs, (n % 2 !== 0) ? 0 : i, n-1);
    }
}
function swap(iterable, i, j) {
    [iterable[i], iterable[j]] = [iterable[j], iterable[i]];
}
```

- Ключевое слово yield используется для остановки и возобновления функций-генераторов (function* или legacy generator function).

```
- [rv] = yield [[выражение]];
- выражение
Возвращаемое выражение. Если не указано, то возвращается значение undefined.
- rv
Возвращает необязательное значение, которое передаётся в next() генератора, чтобы возобновить его выполнение.
```

Задача: сведение вложенной карты

Напишите функцию, которая берет иерархическую карту свойств и преобразует ее в единую плоскую карту с разными уровнями, разделенными косой чертой ('/').

Требуемый результат

Например, при таком вводе:

```
{
  'a': {
    'b': {
      'c': 12,
      'd': 'Hello World'
    },
    'e': [1,2,3]
  }
}
```

вернуть новую карту:

```
  'a/b/c': 12,
  'a/b/d': 'Hello World',
  'a/e': [1,2,3]
```

Решение с Object

```
function flattenMap(map) {
  let di = {}
  function recur(d,prev=''){
      if (d.constructor!=Object) return
      for (let [i,j] of Object.entries(d)){
          if (j!=null && j.constructor==Object) recur(j,prev+(prev?'/':'')+i)
          else{
            if (j==null || j.constructor != Function)
di[(prev+'/'+i).replace(/^\/+/,'')] = j
          }
      }
  }
  recur(map)
  return di
}
```

- constructor - это специальный метод, служащий для создания и инициализации объектов, созданных с использованием class.
- Конструктор Object создаёт объект-обёртку.

```
// Инициализатор объекта или литерал
{ [ nameValuePair1[, nameValuePair2[, ...nameValuePairN] ] ] }

// Вызов в качестве конструктора
new Object([value])
```

Ещё одно решение с Object

```
function flattenMap(map) {
    let result = {};
    function recurse (cur, prop) {
        if (Object(cur) !== cur || Array.isArray(cur)) {
            return result[prop] = cur;
        }
        for (let p in cur) {
            recurse(cur[p], prop ? prop+"/"+p : p);
        }
    }
    recurse(map, "");
    return result;
}
```

К сожалению эту задачу с нашим шаблоном решить невозможно.

Задача: сумма интервалов

Напишите функцию которая принимает массив интервалов и возвращает сумму всех длин интервалов. Перекрывающиеся интервалы следует учитывать только один раз. Интервалы представлены парой целых чисел в виде массива. Первое значение интервала всегда будет меньше второго значения.

- Пример интервала: [1, 5] — это интервал от 1 до 5. Длина этого интервала равна 4.

Требуемый результат

```
sumIntervals( [
   [1,2],
   [6, 10],
   [11, 15]
] ); // => 9

sumIntervals( [
   [1,4],
   [7, 10],
   [3, 5]
] ); // => 7

sumIntervals( [
   [1,5],
   [10, 20],
   [1, 6],
   [16, 19],
   [5, 11]
] ); // => 19
```

Решение по шаблону

```
function sumIntervals(intervals) {
  let sum = 0, max = -Infinity;
  const sortedIntervals = intervals.sort(([a, b], [c, d]) => a == c ? b - d : a - c);
  for (const [start, stop] of sortedIntervals) {
    if (max < start)
      max = start;
    if (max < stop)
      [max, sum] = [stop, sum + stop - max]
  }
  return sum;
}
```

- Метод sort() на месте сортирует элементы массива и возвращает отсортированный массив.
- Глобальное свойство Infinity является числовым значением, представляющим бесконечность.

Решение с несколькими функциями

```
function sumIntervals(intervals) {
  intervals = intervals.sort((a, b) => a[0] - b[0]);
  let merged = merge(intervals);
  return merged.reduce((sum, int) => sum + int[1] - int[0], 0);
}
function merge(ints) {
```

```
  let m = [ints.shift()];
  while (ints.length) {
    m = m.concat(sum(m.pop(), ints.shift()));
  }
  return m;
}
function sum(a, b) {
  if ((Math.max(...a) < Math.min(...b)) === (Math.max(...b) > Math.min(...a))) {
    return a[0] < b[0] ? [a, b] : [b, a];
  }
  return [[Math.min(a[0], b[0]), Math.max(a[1], b[1])]];
}
```

- Метод shift() удаляет первый элемент из массива и возвращает его значение. Этот метод изменяет длину массива.
- Метод Math.max() возвращает наибольшее из нуля или более чисел.
- Метод Math.min() возвращает наименьшее из нуля или более чисел.

Задача: разница в анаграмме

Если даны два слова, сколько букв нужно убрать из них, чтобы получились анаграммы? Слово является анаграммой другого слова, если они имеют одинаковые буквы (обычно в другом порядке).

Требуемый результат

```
First word : c od e w ar s (4 letters removed)
Second word : ha c k er r a nk (6 letters removed)
Result : 10
```

Решение по шаблону

```
function anagramDifference(w1,w2){
  let ww1=w1.split('');
  let ww2=w2.split('');
  for (let i=0; i<ww1.length; ++i)
  {
    if (ww2.indexOf(ww1[i])!=-1)
    {
      ww2.splice(ww2.indexOf(ww1[i]),1)
      ww1.splice(i,1);
      i--;
    }
  }
  return ww1.length+ww2.length
}
```

- Метод splice() изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые.

Решение с Object

```
function anagramDifference(w1,w2){
  return Object.values([w1,"",w2].reduce((a,c,j) => {
    c.split("").forEach(e => {
      (a[e] || a[e]===0) ? a[e] += j-1 : a[e] = j-1;
    });
    return a
  }, {})).reduce((a,c) => a + Math.abs(c), 0)
}
```

- Метод reduce() применяет функцию reducer к каждому элементу массива (слева-направо), возвращая одно результирующее значение.
- Метод Math.abs() возвращает абсолютное значение числа.

Задача: сумма пар

Учитывая список целых чисел и одно значение суммы, верните первые два значения (анализируйте слева) в порядке появления, которые в сумме образуют сумму.

Требуемый результат

```
sum_pairs([11, 3, 7, 5],         10)
#              ^--^      3 + 7 = 10
== [3, 7]

sum_pairs([4, 3, 2, 3, 4],         6)
#          ^-----^         4 + 2 = 6, indices: 0, 2 *
#             ^-----^      3 + 3 = 6, indices: 1, 3
#                ^-----^   2 + 4 = 6, indices: 2, 4
#  * вся пара раньше, и, следовательно, это правильный ответ
== [4, 2]
```

Решение по шаблону

```
const sum_pairs=function(ints, s){
    let cache = new Set(), c, v;
    for( v of ints ){
      if( cache.has(c=s-v) ) return [c,v]
      cache.add(v)
    }
    return undefined
}
```

- Объекты Set позволяют вам сохранять уникальные значения любого типа, как примитивы, так и другие типы объектов.

Решение с WeakMap

```
const sum_pairs = (arr, sum) => {
  const map = new WeakMap();
  for (let i = 0; i < arr.length; ++i) {
    if (map[arr[i]]) {
      return [sum - arr[i], arr[i]];
```

```
    }
    map[sum - arr[i]] = true;
  }
  return [][0];
};
```

- Объект WeakMap — коллекция пар ключ-значение. В качестве ключей могут быть использованы только объекты, а значения могут быть произвольных типов.

Задача: допустимые скобки

Напишите функцию, которая принимает строку скобок и определяет, допустим ли порядок скобок. Функция должна возвращать true, если строка допустима, и false, если она недействительна.

Требуемый результат

```
"()"              =>  true
")(()))"          =>  false
"("               =>  false
"((()))(()(()))"  =>  true
```

Решение по шаблону

```
function validParentheses(parens){
  var n = 0;
  for (var i = 0; i < parens.length; i++) {
    if (parens[i] == '(') n++;
    if (parens[i] == ')') n--;
    if (n < 0) return false;
  }
  return n == 0;
}
```

Решение one line

```
function validParentheses(parens){
    return [...parens].reduce((a,c) => (a+c).replace("()",'') ) == ""?true:false;
}
```

- Метод replace() возвращает новую строку с некоторыми или всеми сопоставлениями с шаблоном, заменёнными на заменитель. Шаблон может быть строкой или регулярным выражением, а заменитель может быть строкой или функцией, вызываемой при каждом сопоставлении.

Задача: произведение последовательных чисел Фибоначчи

Числа Фибоначчи — это числа в следующей целочисленной последовательности (Fn): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... такие как F(n) = F(n-1) + F(n-2), где F(0) = 0 и F(1) = 1. Имея число, скажем, prod (для произведения), мы ищем два числа

Фибоначчи F(n) и F(n+1), проверяя F(n) * F(n+1) = произведение. Ваша функция productFib принимает целое число (prod) и возвращает массив: [F(n), F(n+1), True] или {F(n), F(n+1), 1} или (F(n), F(n+1), True) в зависимости от языка, если F(n) * F(n+1) = prod. Если вы не найдете два последовательных F(n), подтверждающих F(n) * F(n+1) = prod, вы вернетесь [F(n), F(n+1), False] или {F(n), F(n+1), 0} или (F(n), F(n+1), False) F(n) — наименьший, такой как F(n) * F(n+1) > prod.

Требуемый результат

```
productFib(714) # should return (21, 34, true),
                # since F(8) = 21, F(9) = 34 and 714 = 21 * 34

productFib(800) # should return (34, 55, false),
                # since F(8) = 21, F(9) = 34, F(10) = 55 and 21 * 34 < 800 < 34 * 55
-----
productFib(714) # should return [21, 34, true],
productFib(800) # should return [34, 55, false],
```

Решение по шаблону

```
function productFib(prod){
  let n = 0;
  let nPlus = 1;
  while(n*nPlus < prod) {
    nPlus = n + nPlus;
    n = nPlus - n;
  }
  return [n, nPlus, n*nPlus===prod];
}
```

- Оператор while создаёт цикл, выполняющий заданную инструкцию, пока истинно проверяемое условие. Логическое значение условия вычисляется перед исполнением тела цикла.

Решение математическое

```
function productFib( prod ) // mathy way
{
  const r = ( 1 + Math.sqrt(5) ) / 2;
  let fib1 = Math.round( Math.sqrt( prod / r ) ),
  fib2 = Math.round( fib1 * r );
  if ( fib1 * fib2 === prod ) return [ fib1, fib2, true ];
  for ( [fib1, fib2] = [0, 1]; fib1 * fib2 < prod; [fib1, fib2] = [fib2, fib1 + fib2]
);
  return [ fib1, fib2, false ];
}
```

- Метод Math.round() возвращает число, округлённое к ближайшему целому.
- Метод Math.sqrt() возвращает квадратный корень числа

Задача: где мои анаграммы?

Что такое анаграмма? Ну, два слова являются анаграммами друг друга, если они оба содержат одни и те же буквы. Например: 'абба' и 'бааб' == правда 'абба' и 'ббаа' == правда 'абба' и 'абба' == ложь 'абба' и 'абка' == ложь Напишите функцию, которая найдет все анаграммы слова из списка. Вам будет дано два входа слово и массив со словами. Вы должны вернуть массив всех анаграмм или пустой массив, если их нет.

Требуемый результат

```
anagrams('abba', ['aabb', 'abcd', 'bbaa', 'dada']) => ['aabb', 'bbaa']
anagrams('racer', ['crazer', 'carer', 'racar', 'caers', 'racer']) => ['carer', 'racer']
anagrams('laser', ['lazing', 'lazy',  'lacer']) => []
```

Решение по шаблону

```
function anagrams(word, words) {
  let result = [];
  let test = word.split("").sort().join("");

  for (let i=0;i<words.length;i++){
    if(words[i].split("").sort().join("") == test) {
      result.push(words[i]);
    }
  }
  return result;
}
```

Объяснение `.split("").sort().join("")`

Метод split() используется для разделения объекта String на массив строк путем разделения строки на подстроки. `console.log('32243'.split(""));` Вывод: ["3", "2", "2", "4", "3"]

Метод sort() используется для сортировки элементов массива на месте и возвращает массив. `console.log(["3", "2", "2", "4", "3"].sort());` Вывод: ["2", "2", "3", "3", "4"]

Метод join() используется для объединения всех элементов массива в строку. `console.log(["2", "2", "3", "3", "4"].join(""));` Выход: "22334"

Решение two liner ;)

```
function anagrams(word, words) {
  const sword = [...word].sort().join('');
  return words.filter(w => [...w].sort().join('') === sword);
}
```

- Метод filter() создаёт новый массив со всеми элементами, прошедшими проверку, задаваемую в передаваемой функции.

Задача: улитка

Учитывая массив n x n, вернуть элементы массива, расположенные от самых внешних элементов до среднего элемента, перемещаясь по часовой стрелке.

Требуемый результат

```
array = [[1,2,3],
         [8,9,4],
         [7,6,5]]
snail(array) #=> [1,2,3,4,5,6,7,8,9]
```

Решение по шаблону

```
const snail = function(array) {
  let maxx = array[0].length,
    maxy = maxx,
    minx = -1, miny = 0,
    x = 0, y = 0,
    result = [], dir = "r";

  for(let i = maxx*maxx;i>0;i--){
    result.push(array[y][x]);
    switch (dir){
      case "u": y--; break;
      case "l": x--; break;
      case "d": y++; break;
      case "r": x++; break;
    }
    if(x==maxx-1 && y==miny){ dir="d"; minx++; }
    else if(x==maxx-1 && y==maxy-1){ dir="l"; miny++;  }
    else if(x==minx && y==maxy-1){ dir="u"; maxx--; }
    else if(x==minx && y==miny){ dir="r"; maxy--; }
  }
  return result;
}
```

- Инструкция switch сравнивает выражение со случаями, перечисленными внутри неё, а затем выполняет соответствующие инструкции.

```
switch (expression) {
case value1:
  //Здесь выполняются инструкции, если результат выражения равен value1
  [break;]
case value2:
  //Инструкции, соответствующие value2
  [break;]
...
case valueN:
  //Инструкции, соответствующие значению valueN
  //statementsN
  [break;]
default:
  //Здесь находятся инструкции, которые выполняются при отсутствии
```

```
соответствующего значения
  //statements_def
  [break;]
}
```

Решение с array

```
function snail(array) {
  let vector = [];
  while (array.length) {
    vector.push(...array.shift());
    array.map(row => vector.push(row.pop()));
    array.reverse().map(row => row.reverse());
  }
  return vector;
}
```

- Метод reverse() на месте обращает порядок следования элементов массива. Первый элемент массива становится последним, а последний — первым.

Задача: добавление больших чисел

Напишите функцию, которая возвращает сумму двух чисел. Входные числа являются строками, и функция должна возвращать строку.

Требуемый результат

```
add("123", "321"); -> "444"
add("11", "99");   -> "110"
```

Решение по шаблону

```
function add(a, b) {
 let zrx = /^0+/;
  a = a.replace(zrx, '').split('').reverse();
  b = b.replace(zrx, '').split('').reverse();
  let result = [], max = Math.max(a.length, b.length);
  for (let memo = 0, i = 0; i < max; i++) {
    let res = parseInt(a[i] || 0) + parseInt(b[i] || 0) + memo;
    result[i] = res % 10;
    memo = (res - result[i]) / 10;
  }
  if (memo) {
    result.push(memo);
  }
  return result.reverse().join('');
}
```

- Функция parseInt() принимает строку в качестве аргумента и возвращает целое число в соответствии с указанным основанием системы счисления

Решение с arguments

```
function add(a, b) {
  const sum = +a.slice(-1) + +b.slice(-1) + +(arguments[2] || '').slice(0, 1);
  if (a.length <= 1 && b.length <= 1) return sum + (arguments[2] || '').slice(1);
  let acc = '' + ~~(sum / 10) + sum % 10 + (arguments[2] || '').slice(1);
  return add(a.slice(0, -1), b.slice(0, -1), acc)
}
```

- Объект arguments — это подобный массиву объект, который содержит аргументы, переданные в функцию.

Задача: простая свинья на латыни

Переместите первую букву каждого слова в конец, а затем добавьте «ay» в конец слова. Оставьте знаки препинания нетронутыми.

Требуемый результат

```
pigIt('Pig latin is cool'); // igPay atinlay siay oolcay
pigIt('Hello world !');     // elloHay orldway !
```

Решение с substring

```
function pigIt(str){
  //Code here
  return str.split(' ').map(word => {
    return word.substring(1) + word[0] + 'ay';
  }).join(' ');
}
```

- Метод substring() возвращает подстроку строки между двумя индексами, или от одного индекса и до конца строки.

Решение one liner и RegExp

```
function pigIt(str){
  return str.replace(/\w+/g, match => match.slice(1) + match.charAt(0) + "ay");
}
```

- Метод charAt() возвращает указанный символ из строки.

Задача: объединить два массива

Напишите функцию, которая объединяет два массива, поочередно беря элементы из каждого массива по очереди.

Требуемый результат

```
[a, b, c, d, e], [1, 2, 3, 4, 5] becomes  [a, 1, b, 2, c, 3, d, 4, e, 5]
[1, 2, 3], [a, b, c, d, e, f] becomes [1, a, 2, b, 3, c, d, e, f
```

Решение по шаблону

```
function mergeArrays(a, b) {
  var res = [];
  let i=0;
  while ((i<a.length) || (i<b.length) ) {
    if (i<a.length) res.push(a[i]);
    if (i<b.length) res.push(b[i]);
    i++;
  }
  return res;
}
```

Решение с lodash

```
const _ = require('lodash')

function mergeArrays(a, b) {
  return _.compact(_.flatten(_.zip(a, b)))
}
```

- lodash - Современная служебная библиотека JavaScript, обеспечивающая модульность, производительность и дополнительные возможности. https://lodash.com/

Задача: деревья - максимальная сумма

Вам дано бинарное дерево. Реализуйте метод maxSum, который возвращает максимальную сумму маршрута от корня к листу. Например, для следующего дерева: 17 /
3 -10 / /
2 16 1 / 13 Метод должен вернуть 23, так как [17,-10,16] — это маршрут от корня к листу с максимальной суммой.

Требуемый результат

Вам доступен класс TreeNode:

```
let TreeNode = function(value, left, right) {
  this.value = value;
  this.left = left;
  this.right = right;
};
```

Решение по шаблону

```
function maxSum(root) {
  if(root == null || root == undefined) return 0;
  let left = maxSum(root.left);
```

```
  let righ = maxSum(root.right);
  let sum = Math.max(left,righ) + root.value;
  return sum}
```

- Метод Math.max() возвращает наибольшее из нуля или более чисел.

Решение one line

```
maxSum=root=>!root?0:root.value+Math.max(maxSum(root.left),maxSum(root.right))
```

Задача: связанные списки — отсортированная вставка

Напишите функцию SortedInsert(), которая вставляет узел в правильное место предварительно отсортированного связанного списка, отсортированного в порядке возрастания. SortedInsert принимает заголовок связанного списка и данные, используемые для создания узла, в качестве аргументов. SortedInsert() также должен возвращать заголовок списка.

Требуемый результат

```
sortedInsert(1 -> 2 -> 3 -> null, 4) === 1 -> 2 -> 3 -> 4 -> null)
sortedInsert(1 -> 7 -> 8 -> null, 5) === 1 -> 5 -> 7 -> 8 -> null)
sortedInsert(3 -> 5 -> 9 -> null, 7) === 3 -> 5 -> 7 -> 9 -> null)
```

Решение с рекурсией

```
function Node(data) {
  this.data = data;
  this.next = null;
}
function sortedInsert(head, data) {
  return (head == null || head.data > data)
    ? push(head, data)
    : push(sortedInsert(head.next, data), head.data);
}
```

- Метод push() добавляет один или более элементов в конец массива и возвращает новую длину массива.

Решение с new Node

```
function Node(data) {
  this.data = data;
  this.next = null;
}

function sortedInsert(head, data) {
  if (head == null) {
    return new Node(data);
  }
  let previous, current = head;
```

```
  while(data > current.data) {
    previous = current;
    current = current.next;
    if (!current) break;
  }
  let node = new Node(data);
  node.next = current;
  if (previous) {
    previous.next = node;
  } else {
    return node
  }
  return head;
}
```

- Связный список — это структура данных, в которой один объект ссылается на следующий в последовательности, сохраняя его адрес. Каждый объект называется узлом. В дополнение к ссылке на следующий объект в последовательности каждый объект несет некоторые данные (например, целочисленное значение, ссылку на строку и т. д.)

Задача: крестики-нолики

Если бы нам нужно было настроить игру в крестики-нолики, мы бы хотели знать, решено ли текущее состояние доски. Наша цель — создать функцию, которая проверит это за нас!

Требуемый результат

Предположим, что доска имеет форму массива 3x3, где значение равно 0, если ячейка пуста, 1, если это «X», или 2, если это «O», например:

```
[[0, 0, 1],
 [0, 1, 2],
 [2, 1, 0]]
```

Мы хотим, чтобы наша функция возвращала:

> *-1, если доска еще не закончена И еще никто не выиграл (есть пустые места), 1, если "X" выиграл, 2, если "O" выиграл, 0, если это кошачья игра (т.е. ничья). Вы можете предположить, что переданная доска действительна в контексте игры в крестики-нолики.*

Решение по шаблону

```
function isSolved(board) {
  let row = board;
  let col = [0,1,2].map((i) => [0,1,2].map((h)=>board[h][i]));
  let di1 = [[0,1,2].map((i) => board[i][i])];
  let di2 = [[0,1,2].map((i) => board.reverse()[i][i])];
  let all = row.concat(col,di1,di2);
  if (all.some(x=>""+x=="1,1,1")){return 1;}
  else if (all.some(x=>""+x=="2,2,2")){return 2;}
  else if (all.some(x => x.includes(0))){return -1;}
```

```
  else{return 0;};
}
```

- Метод some() проверяет, удовлетворяет ли какой-либо элемент массива условию, заданному в передаваемой функции.
- Метод concat() возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в качестве аргументов.

Условия публичной лицензии на электронные версии