**Массивы: методы**

## [Метод split](http://learn.javascript.ru/array-methods" \l "metod-split)

Ситуация из реальной жизни. Мы пишем сервис отсылки сообщений и посетитель вводит имена тех, кому его отправить: Маша, Петя, Марина, Василий.... Но нам-то гораздо удобнее работать с массивом имен, чем с одной строкой.

К счастью, есть метод split(s), который позволяет превратить строку в массив, разбив ее по разделителю s. В примере ниже таким разделителем является строка из запятой и пробела.

var names = 'Маша, Петя, Марина, Василий';

var arr = names.split(', ');

for (var i = 0; i < arr.length; i++) {

alert( 'Вам сообщение ' + arr[i] );

}

**Второй аргумент split**

У метода split есть необязательный второй аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано – остаток массива будет отброшен:

alert( "a,b,c,d".split(',', 2) ); // a,b

**Разбивка по буквам**

Вызов split с пустой строкой разобьёт по буквам:

var str = "тест";

alert( str.split('') ); // т,е,с,т

## [Метод join](http://learn.javascript.ru/array-methods" \l "metod-join)

Вызов arr.join(str) делает в точности противоположное split. Он берет массив и склеивает его в строку, используя str как разделитель.

Например:

var arr = ['Маша', 'Петя', 'Марина', 'Василий'];

var str = arr.join(';');

alert( str ); // Маша;Петя;Марина;Василий

**new Array + join = Повторение строки**

Код для повторения строки 3 раза:

alert( new Array(4).join("ля") ); // ляляля

Как видно, new Array(4) делает массив без элементов длины 4, который join объединяет в строку, вставляя между его элементами строку "ля".

В результате, так как элементы пусты, получается повторение строки. Такой вот небольшой трюк.

## [Удаление из массива](http://learn.javascript.ru/array-methods" \l "udalenie-iz-massiva)

Так как массивы являются объектами, то для удаления ключа можно воспользоваться обычным delete:

var arr = ["Я", "иду", "домой"];

delete arr[1]; // значение с индексом 1 удалено

// теперь arr = ["Я", undefined, "домой"];

alert( arr[1] ); // undefined

Да, элемент удален из массива, но не так, как нам этого хочется. Образовалась «дырка».

Это потому, что оператор delete удаляет пару «ключ-значение». Это – все, что он делает. Обычно же при удалении из массива мы хотим, чтобы оставшиеся элементы сдвинулись и заполнили образовавшийся промежуток.

Поэтому для удаления используются специальные методы: из начала – shift, с конца – pop, а из середины – splice, с которым мы сейчас познакомимся.

## [Метод splice](http://learn.javascript.ru/array-methods" \l "metod-splice)

Метод splice – это универсальный раскладной нож для работы с массивами. Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно.

Его синтаксис:

**arr.splice(index[, deleteCount, elem1, ..., elemN])**

Удалить deleteCount элементов, начиная с номера index, а затем вставить elem1, ..., elemN на их место. Возвращает массив из удалённых элементов.

Этот метод проще всего понять, рассмотрев примеры.

Начнём с удаления:

var arr = ["Я", "изучаю", "JavaScript"];

arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент

alert( arr ); // осталось ["Я", "JavaScript"]

В следующем примере мы удалим 3 элемента и вставим другие на их место:

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 3 первых элемента и добавить другие вместо них

arr.splice(0, 3, "Мы", "изучаем")

alert( arr ) // теперь ["Мы", "изучаем", "JavaScript"]

Здесь видно, что splice возвращает массив из удаленных элементов:

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 2 первых элемента

var removed = arr.splice(0, 2);

alert( removed ); // "Я", "сейчас" <-- array of removed elements

Метод splice также может вставлять элементы без удаления, для этого достаточно установить deleteCount в 0:

var arr = ["Я", "изучаю", "JavaScript"];

// с позиции 2

// удалить 0

// вставить "сложный", "язык"

arr.splice(2, 0, "сложный", "язык");

alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"

Допускается использование отрицательного номера позиции, которая в этом случае отсчитывается с конца:

var arr = [1, 2, 5]

// начиная с позиции индексом -1 (перед последним элементом)

// удалить 0 элементов,

// затем вставить числа 3 и 4

arr.splice(-1, 0, 3, 4);

alert( arr ); // результат: 1,2,3,4,5

## [Метод slice](http://learn.javascript.ru/array-methods" \l "metod-slice)

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

Например:

var arr = ["Почему", "надо", "учить", "JavaScript"];

var arr2 = arr.slice(1, 3); // элементы 1, 2 (не включая 3)

alert( arr2 ); // надо, учить

Аргументы ведут себя так же, как и в строковом slice:

* Если не указать end – копирование будет до конца массива:

var arr = ["Почему", "надо", "учить", "JavaScript"];

alert( arr.slice(1) ); // взять все элементы, начиная с номера 1

* Можно использовать отрицательные индексы, они отсчитываются с конца:

var arr2 = arr.slice(-2); // копировать от 2-го элемента с конца и дальше

* Если вообще не указать аргументов – скопируется весь массив:

var fullCopy = arr.slice();

**Совсем как в строках**

Синтаксис метода slice одинаков для строк и для массивов. Тем проще его запомнить.

## [Сортировка, метод sort(fn)](http://learn.javascript.ru/array-methods" \l "sortirovka-metod-sort-fn)

Метод sort() сортирует массив на месте. Например:

var arr = [ 1, 2, 15 ];

arr.sort();

alert( arr ); // 1, 15, 2

Не заметили ничего странного в этом примере?

Порядок стал 1, 15, 2, это точно не сортировка чисел. Почему?

**Это произошло потому, что по умолчанию sort сортирует, преобразуя элементы к строке.**

Поэтому и порядок у них строковый, ведь "2" > "15".

### [Свой порядок сортировки](http://learn.javascript.ru/array-methods" \l "svoy-poryadok-sortirovki)

Для указания своего порядка сортировки в метод arr.sort(fn) нужно передать функцию fn от двух элементов, которая умеет сравнивать их.

Внутренний алгоритм функции сортировки умеет сортировать любые массивы – апельсинов, яблок, пользователей, и тех и других и третьих – чего угодно. Но для этого ему нужно знать, как их сравнивать. Эту роль и выполняет fn.

Если эту функцию не указать, то элементы сортируются как строки.

Например, укажем эту функцию явно, отсортируем элементы массива как числа:

function compareNumeric(a, b) {

if (a > b) return 1;

if (a < b) return -1;

}

var arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);

alert(arr); // 1, 2, 15

Обратите внимание, мы передаём в sort() именно саму функцию compareNumeric, без вызова через скобки. Был бы ошибкой следующий код:

arr.sort( compareNumeric() ); // не сработает

Как видно из примера выше, функция, передаваемая sort, должна иметь два аргумента.

Алгоритм сортировки, встроенный в JavaScript, будет передавать ей для сравнения элементы массива. Она должна возвращать:

* Положительное значение, если a > b,
* Отрицательное значение, если a < b,
* Если равны – можно 0, но вообще – не важно, что возвращать, если их взаимный порядок не имеет значения.

**Алгоритм сортировки**

В методе sort, внутри самого интерпретатора JavaScript, реализован универсальный алгоритм сортировки. Как правило, это [««быстрая сортировка»»](http://algolist.manual.ru/sort/quick_sort.php), дополнительно оптимизированная для небольших массивов.

Он решает, какие пары элементов и когда сравнивать, чтобы отсортировать побыстрее. Мы даём ему функцию – способ сравнения, дальше он вызывает её сам.

Кстати, те значения, с которыми sort вызывает функцию сравнения, можно увидеть, если вставить в неё alert:

[1, -2, 15, 2, 0, 8].sort(function(a, b) {

alert( a + " <> " + b );

});

**Сравнение compareNumeric в одну строку**

Функцию compareNumeric для сравнения элементов-чисел можно упростить до одной строчки.

function compareNumeric(a, b) {

return a - b;

}

Эта функция вполне подходит для sort, так как возвращает положительное число, если a > b, отрицательное, если наоборот, и 0, если числа равны.

## [reverse](http://learn.javascript.ru/array-methods" \l "reverse)

Метод [arr.reverse()](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reverse) меняет порядок элементов в массиве на обратный.

var arr = [1, 2, 3];

arr.reverse();

alert( arr ); // 3,2,1

## [concat](http://learn.javascript.ru/array-methods" \l "concat)

Метод [arr.concat(value1, value2, … valueN)](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/concat) создаёт новый массив, в который копируются элементы из arr, а также value1, value2, ... valueN.

Например:

var arr = [1, 2];

var newArr = arr.concat(3, 4);

alert( newArr ); // 1,2,3,4

У concat есть одна забавная особенность.

Если аргумент concat – массив, то concat добавляет элементы из него.

Например:

var arr = [1, 2];

var newArr = arr.concat([3, 4], 5); // то же самое, что arr.concat(3,4,5)

alert( newArr ); // 1,2,3,4,5

## [indexOf/lastIndexOf](http://learn.javascript.ru/array-methods" \l "indexof-lastindexof)

Эти методы не поддерживаются в IE8-. Для их поддержки подключите библиотеку [ES5-shim](https://github.com/kriskowal/es5-shim).

Метод [«arr.indexOf(searchElement[, fromIndex])»](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/indexOf) возвращает номер элемента searchElement в массиве arr или -1, если его нет.

Поиск начинается с номера fromIndex, если он указан. Если нет – с начала массива.

**Для поиска используется строгое сравнение ===.**

Например:

var arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1

alert( arr.indexOf(false) ); // 2

alert( arr.indexOf(null) ); // -1

Как вы могли заметить, по синтаксису он полностью аналогичен методу [indexOf для строк](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/String/indexOf).

Метод [«arr.lastIndexOf(searchElement[, fromIndex])»](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/lastIndexOf) ищет справа-налево: с конца массива или с номера fromIndex, если он указан.

**Методы indexOf/lastIndexOf осуществляют поиск перебором**

Если нужно проверить, существует ли значение в массиве – его нужно перебрать. Только так. Внутренняя реализация indexOf/lastIndexOf осуществляет полный перебор, аналогичный циклу for по массиву. Чем длиннее массив, тем дольше он будет работать.

**Коллекция уникальных элементов**

Рассмотрим задачу – есть коллекция строк, и нужно быстро проверять: есть ли в ней какой-то элемент. Массив для этого не подходит из-за медленного indexOf. Но подходит объект! Доступ к свойству объекта осуществляется очень быстро, так что можно сделать все элементы ключами объекта и проверять, есть ли уже такой ключ.

Например, организуем такую проверку для коллекции строк "div", "a" и "form":

var store = {}; // объект для коллекции

var items = ["div", "a", "form"];

for (var i = 0; i < items.length; i++) {

var key = items[i]; // для каждого элемента создаём свойство

store[key] = true; // значение здесь не важно

}

Теперь для проверки, есть ли ключ key, достаточно выполнить if (store[key]). Если есть – можно использовать значение, если нет – добавить.

Такое решение работает только со строками, но применимо к любым элементам, для которых можно вычислить строковый «уникальный ключ».

## [Object.keys(obj)](http://learn.javascript.ru/array-methods" \l "object-keys-obj)

Ранее мы говорили о том, что свойства объекта можно перебрать в цикле for..in.

Если мы хотим работать с ними в виде массива, то к нашим услугам – замечательный метод [Object.keys(obj)](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Object/keys). Он поддерживается везде, кроме IE8-:

var user = {

name: "Петя",

age: 30

}

var keys = Object.keys(user);

alert( keys ); // name, age

## [Итого](http://learn.javascript.ru/array-methods" \l "itogo)

Методы массивов:

* push/pop, shift/unshift, splice – для добавления и удаления элементов.
* join/split – для преобразования строки в массив и обратно.
* slice – копирует участок массива.
* sort – для сортировки массива. Если не передать функцию сравнения – сортирует элементы как строки.
* reverse – меняет порядок элементов на обратный.
* concat – объединяет массивы.
* indexOf/lastIndexOf – возвращают позицию элемента в массиве (не поддерживается в IE8-).

Дополнительно:

* Object.keys(obj) возвращает массив свойств объекта.

Изученных нами методов достаточно в 95% случаях, но существуют и другие. Для знакомства с ними рекомендуется заглянуть в справочник [Array](http://javascript.ru/Array) и [Array в Mozilla Developer Network](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array).