**[Добавление/удаление элементов](https://learn.javascript.ru/array-methods" \l "dobavlenie-udalenie-elementov)**

Мы уже знаем методы, которые добавляют и удаляют элементы из начала или конца:

* arr.push(...items) – добавляет элементы в конец,
* arr.pop() – извлекает элемент из конца,
* arr.shift() – извлекает элемент из начала,
* arr.unshift(...items) – добавляет элементы в начало.

Есть и другие.

**Delete в массивах**

Элемент удаляется, но в массиве остается неизменное количество элементов. Это происходит, потому что delete obj.key удаляет значение по ключу key, и это все что он делает. Это хорошо для объектов. Но для массивов мы обычно хотим, чтобы оставшиеся элементы сдвинулись и заняли освободившееся место. Мы ждём, что массив станет короче.

let arr = ["I", "go", "home"];

delete arr[1]; // удалить "go"

alert( arr[1] ); // undefined

// теперь arr = ["I", , "home"];

alert( arr.length ); // 3

**arr.splice**

Умеет всё: добавлять, удалять и заменять элементы.

Синтаксис:

arr.splice(start[, deleteCount, elem1, ..., elemN])

Он изменяет arr начиная с индекса start: удаляет deleteCount элементов и затем вставляет elem1, ..., elemN на их место. Возвращает массив из удалённых элементов.

Удаление 1 элемента

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

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

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

Начиная с индекса 1, он убрал 1 элемент.

Удалить 3 элемента и заменить их двумя другими.

let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 3 первых элемента и заменить их другими

arr.splice(0, 3, "Давай", "танцевать");

alert( arr ) // теперь ["Давай", "танцевать", "прямо", "сейчас"]

Splice возвращает массив из удалённых элементов

let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас"];

// удалить 2 первых элемента

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

alert( removed ); // "Я", "изучаю" <-- массив из удалённых элементов

Вставить элементы без удаления

Для этого достаточно установить deleteCount в 0:

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

// с индекса 2

// удалить 0 элементов

// вставить "сложный", "язык"

arr.splice(2, 0, "сложный", "язык");

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

Отрицательные индексы разрешены

Они определяют позицию с конца массива:

let arr = [1, 2, 5];

// начиная с индекса -1 (перед последним элементом)

// удалить 0 элементов,

// затем вставить числа 3 и 4

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

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

**arr.slice**

намного проще, чем похожий на него arr.splice.

Синтаксис:

arr.slice([start], [end])

Он возвращает новый массив, в который копирует все элементы с индекса start до end (не включая end). start и end могут быть отрицательными, в этом случае отсчёт позиции будет вестись с конца массива.

Это похоже на строковый метод str.slice, но вместо подстрок возвращает подмассивы.

let arr = ["t", "e", "s", "t"];

alert( arr.slice(1, 3) ); // e,s (копирует с 1 до 3)

alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)

Можно вызвать slice без аргументов: arr.slice() создаёт копию arr. Это часто используют, чтобы создать копию массива для дальнейших преобразований, которые не должны менять исходный массив.

**Arr.concat**

Метод [arr.concat](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/concat) создаёт новый массив, в который копирует данные из других массивов и дополнительные значения.

arr.concat(arg1, arg2...)

Он принимает любое количество аргументов, которые могут быть как массивами, так и простыми значениями.

В результате – новый массив, включающий в себя элементы из arr, затем arg1, arg2 и так далее.

Если аргумент argN – массив, то копируются все его элементы. Иначе копируется сам аргумент.

let arr = [1, 2];

// создать массив из: arr и [3,4]

alert( arr.concat([3, 4]) ); // 1,2,3,4

// создать массив из: arr и [3,4] и [5,6]

alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// создать массив из: arr и [3,4], потом добавить значения 5 и 6

alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6

Обычно он копирует только элементы из массивов. Другие объекты, даже если они выглядят как массивы, добавляются как есть:

let arr = [1, 2];

let arrayLike = {

0: "что-то",

length: 1

};

alert( arr.concat(arrayLike) ); // 1,2,[object Object]

…Но если массивоподобный объект имеет специальное свойство Symbol.isConcatSpreadable, то он обрабатывается как массив, с помощью concat: вместо него добавляются его элементы:

let arr = [1, 2];

let arrayLike = {

0: "что-то",

1: "ещё",

[Symbol.isConcatSpreadable]: true,

length: 2

};

alert( arr.concat(arrayLike) ); // 1,2,что-то,ещё

## [Перебор: forEach](https://learn.javascript.ru/array-methods" \l "perebor-foreach)

Метод [arr.forEach](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach) позволяет запускать функцию для каждого элемента массива.

Синтаксис:

arr.forEach(function(item, index, array) {

// ... делать что-то с item

});

Вывести на экран каждый элемент массива:

// Вызов alert для каждого элемента

["Бильбо", "Гэндальф", "Назгул"].forEach(alert);

Вывести на экран каждый элемент массива и его позицию

["Бильбо", "Гэндальф", "Назгул"].forEach((item, index, array) => {

alert(`У ${item} индекс ${index} в ${array}`);

});

Результат функции (если она что-то возвращает) отбрасывается и игнорируется.

## [Поиск в массиве](https://learn.javascript.ru/array-methods" \l "poisk-v-massive)

**arr.indexOf**

Метод indexOf() возвращает первый индекс, по которому данный элемент может быть найден в массиве или -1, если такого индекса нет.

Синтаксис:

arr.indexOf(searchElement[, fromIndex = 0])

searchElement - Искомый элемент в массиве.

fromIndex - Индекс, с которого начинать поиск. Поддерживаются отрицательные индексы. Если индекс является отрицательным числом, он трактуется как смещение с конца массива. Обратите внимание: если индекс отрицателен, массив всё равно просматривается от начала к концу. Метод использует строгое сравнение.

let arr = [1, 0, false];

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

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

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

**arr.indexOf**

похож на indexOf, но ищет справа налево.

let fruits = ['Яблоко', 'Апельсин', 'Яблоко']

alert( fruits.indexOf('Яблоко') ); // 0 (первый 'Яблоко')

alert( fruits.lastIndexOf('Яблоко') ); // 2 (последний 'Яблоко')

**arr.includes**

Метод includes() определяет, содержит ли массив определённый элемент, возвращая в зависимости от этого true или false. Метод использует строгое сравнение.

let arr = [1, 0, false];

alert( arr.includes(1) ); // true

Если мы хотим проверить наличие элемента в массиве и нет необходимости знать его индекс, предпочтительно использовать

arr.includes. includes  правильно обрабатывает NaN, в отличие от indexOf:

const arr = [NaN];

alert( arr.indexOf(NaN) ); // -1 (неверно, должен быть 0)

alert( arr.includes(NaN) );// true (верно)

Это связано с тем, что includes был добавлен в JavaScript гораздо позже и использует более современный алгоритм сравнения.

[**arr.find**](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/find)

Позволяет найти объект с определённым условием

Синтаксис:

let result = arr.find(function(item, index, array) {

// если true - возвращается текущий элемент и перебор прерывается

// если все итерации оказались ложными, возвращается undefined

});

Функция вызывается по очереди для каждого элемента массива:

* item – очередной элемент.
* index – его индекс.
* array – сам массив.

Если функция возвращает true, поиск прерывается и возвращается item. Если ничего не найдено, возвращается undefined.

Найти пользователя с id == 1:

let users = [

{id: 1, name: "Вася"},

{id: 2, name: "Петя"},

{id: 3, name: "Маша"}

];

let user = users.find(item => item.id == 1);

alert(user.name); // Вася

Обратите внимание, что в данном примере мы передаём find функцию item => item.id == 1 с одним аргументом. Это типично, другие аргументы этой функции используются редко.

**arr.findIndex**

Синтаксис

let result = arr.findIndex(function(item, index, array) {});

Возвращает индекс, на котором был найден элемент, а не сам элемент. Значение -1 возвращается, если ничего не найдено. Ищет слева направо.

let users = [

{id: 1, name: "Вася"},

{id: 2, name: "Петя"},

{id: 3, name: "Маша"},

{id: 4, name: "Вася"}

];

// Найти индекс первого Васи

alert(users.findIndex(user => user.name == 'Вася')); // 0

**arr.findLastIndex**

Синтаксис

let result = arr.findIndex(function(item, index, array) {});

Возвращает индекс, на котором был найден элемент, а не сам элемент. Значение -1 возвращается, если ничего не найдено. Ищет справа налево.

let users = [

{id: 1, name: "Вася"},

{id: 2, name: "Петя"},

{id: 3, name: "Маша"},

{id: 4, name: "Вася"}

];

// Найти индекс последнего Васи

alert(users.findLastIndex(user => user.name == 'Вася')); // 3

**arr.filter**

Возвращает массив из всех подходящих под условие элементов:

let results = arr.filter(function(item, index, array) {

// если `true` -- элемент добавляется к results и перебор продолжается

// возвращается пустой массив в случае, если ничего не найдено

});

let users = [

{id: 1, name: "Вася"},

{id: 2, name: "Петя"},

{id: 3, name: "Маша"}

];

// возвращает массив, состоящий из двух первых пользователей

let someUsers = users.filter(item => item.id < 3);

alert(someUsers.length); // 2

## [Преобразование массива](https://learn.javascript.ru/array-methods" \l "preobrazovanie-massiva)

**map**

Вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции.

let result = arr.map(function(item, index, array) {

// возвращается новое значение вместо элемента

});

Преобразовать каждый элемент в его длину:

let lengths = ["Бильбо", "Гэндальф", "Назгул"].map(item => item.length);

alert(lengths); // 6,8,6

**sort(fn)**

Сортирует массив на месте, меняя в нём порядок элементов.

Он также возвращает отсортированный массив, но обычно возвращаемое значение игнорируется, так как изменяется сам arr.

**По умолчанию элементы сортируются как строки.**

Элементы преобразуются в строки при сравнении. Для строк применяется лексикографический порядок. Чтобы использовать наш собственный порядок сортировки, нам нужно предоставить функцию в качестве аргумента arr.sort().

**Reverse**

На месте меняет порядок элементов в arr на обратный.

let arr = [1, 2, 3, 4, 5];

arr.reverse();

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

Он также возвращает массив arr с изменённым порядком элементов.

**str.split(delim)**

Он разбивает строку на массив по заданному разделителю delim.

Например разделителем является строка из запятой и пробела.

let names = 'Вася, Петя, Маша';

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

У метода split есть необязательный второй числовой аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано, то остаток массива будет отброшен. На практике это редко используется:

let arr = 'Вася, Петя, Маша, Саша'.split(', ', 2);

alert(arr); // Вася, Петя

Разбивка по буквам

Вызов split(s) с пустым аргументом s разбивает строку на массив букв:

let str = "тест";

alert( str.split('') ); // т,е,с,т

**arr.join(glue)**

Он создаёт строку из элементов arr, вставляя glue между ними.

let arr = ['Вася', 'Петя', 'Маша'];

let str = arr.join(';'); // объединить массив в строку через ;

alert( str ); // Вася;Петя;Маша

**reduce/reduceRight**

используются для вычисления единого значения на основе всего массива.

let value = arr.reduce(function(accumulator, item, index, array) {

// ...

}, [initial]);

Функция применяется по очереди ко всем элементам массива и «переносит» свой результат на следующий вызов.

Аргументы:

* accumulator – результат предыдущего вызова этой функции, равен initial при первом вызове (если передан initial),
* item – очередной элемент массива,
* index – его позиция,
* array – сам массив.

При вызове функции результат её предыдущего вызова передаётся на следующий вызов в качестве первого аргумента.

Так, первый аргумент является по сути аккумулятором, который хранит объединённый результат всех предыдущих вызовов функции. По окончании он становится результатом reduce. при отсутствии initial в качестве первого значения берётся первый элемент массива, а перебор стартует со второго. Но такое использование требует крайней осторожности. Если массив пуст, то вызов reduce без начального значения выдаст ошибку. Поэтому рекомендуется всегда указывать начальное значение.

Метод [arr.reduceRight](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/reduceRight) работает аналогично, но проходит по массиву справа налево.

Получить сумму всех элементов массива одной строкой:

let arr = [1, 2, 3, 4, 5];

let result = arr.reduce((sum, current) => sum + current, 0);

alert(result); // 15

## [Array.isArray](https://learn.javascript.ru/array-methods" \l "array-isarray)

Массивы не образуют отдельный тип данных. Они основаны на объектах.

Поэтому typeof не может отличить простой объект от массива:

alert(typeof {}); // object

alert(typeof []); // тоже object

…Но массивы используются настолько часто, что для этого придумали специальный метод: [Array.isArray(value)](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/isArray). Он возвращает true, если value массив, и false, если нет.

alert(Array.isArray({})); // false

alert(Array.isArray([])); // true

## [Большинство методов поддерживают «thisArg»](https://learn.javascript.ru/array-methods" \l "bolshinstvo-metodov-podderzhivayut-thisarg)

Почти все методы массива, которые вызывают функции – такие как find, filter, map, за исключением метода sort, принимают необязательный параметр thisArg. Используется редко.

Вот полный синтаксис этих методов:

arr.find(func, thisArg);

arr.filter(func, thisArg);

arr.map(func, thisArg);

Значение параметра thisArg становится this для func.