

# Интернет-технологии и распределённая обработка данных

Лекция 14

## Массивы JavaScript

1. Массивы в JavaScript
2. Создание массивов
3. Доступ к элементам и индексы
4. Методы массивов
5. Объекты, подобные массивам

# Массивы – общие сведения

*Массив* – разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования такой коллекцией.

Каждый элемент характеризуется **числовой** позицией в массиве, которая называется *индексом*.

Во многих языках программирования массивы **однотипны**, имеют **фиксированный размер**, содержат **непрерывную последовательность** элементов.

# Особенности массивов в JavaScript

1. **Гетерогенные:** элементы могут иметь разные типы.
2. **Динамические:** меняют размер по мере необходимости.
3. **Разреженные:** могут отсутствовать элементы с некоторыми индексами.
4. **Это специальные объекты:** числовые индексы работают как имена свойств. Обычно движки делают оптимизации: доступ по индексу выполняется быстрее, чем доступ к свойствам объектов.

# Создание массива

Способ 1: литерал массива.

```
var a = [];           // пустой массив  
var b = [1, 2 + 3];  // два элемента  
var c = [1, "Robot", [3]]; // три элемента
```

Запятая перед закрывающей скобкой игнорируется!

```
var d = [1, 2, 3, ]; // три элемента!
```

Способ 2: функция `Array()` (вызывается с `new` или без):

1. Без аргументов – пустой массив длины 0
2. Одно число (целое неотрицательное, иначе `RangeError`) – массив указанной длины, без элементов
3. Один нечисловой аргумент – массив из этого элемента
4. Более одного аргумента – массив, который состоит из указанных элементов

```
var a = new Array();           // => []
var b = new Array(2);         // => [ , ]
var c = new Array("Pobot");   // => [" Pobot "]
var d = new Array(1, "Pobot"); // => [1, " Pobot "]
var e = Array(1, " Pobot ");  // => [1, " Pobot "]
```

# Доступ к элементам и индексы

Любой массив – это объект.

Любой объект – это коллекция свойств и значений.

Любое свойство имеет имя.

Универсальное обращение к свойству: `obj[name]`, где `name` автоматически приводится к `string` (а если имя свойства – это правильный идентификатор, то можно так: `obj.name`).

*Индекс* массива – это **свойство**, имя которого выглядит как целое число от 0 до  $2^{32} - 2$ .

Так как целые числа не являются правильными идентификаторами, обращение по индексу выполняется в форме `obj[name]` (можно указать число или строку: `data[1]` или `data["1"]`).



Примеры показывают состав индексов массива:

```
var a = [];           // индексов нет
a[0] = 10;           // один индекс: 0
var x = a[1];        // один индекс: 0
var b = [1, ,3];     // два индекса: 0 и 2
delete b[0];        // один индекс: 2
b[2] = undefined;   // один индекс: 2
var c = new Array(); // индексов нет
var d = new Array(2); // индексов нет!
var e = new Array(1, " Robot "); // два индекса: 0 и 1
```

# Многомерные массивы

```
var matrix = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];
```

```
alert( matrix[1][1] ); // центральный элемент
```

# Свойство length

Любой массив имеет свойство length.

Интуитивно: это «длина» массива, т.е. количество элементов в нём. Но это справедливо только для плотных массивов – у разреженных свои нюансы!

Всегда верно: **length будет больше максимального индекса массива** (и почти всегда – ровно на 1).

```
var a = [];           // length = 0
a[0] = 10;           // length = 1
a[10] = 100;         // length = 11
delete a[10];        // length = 11
```

```
var b = [1, ,3];     // length = 3
b[2] = undefined;   // length = 3
```

```
var c = new Array(2); // length = 2
c[0] = 10;           // length = 2
```

```
// получить последний элемент
```

```
var lastItem = goods[goods.length - 1];
```

```
//добавить новый элемент
```

```
goods[goods.length] = 'Компьютер';
```

Свойство `length` можно изменять. Уменьшение `length` приводит к «усечению» массива – исчезают элементы и индексы:

```
var a = [1, 2, 3];  
a.length = 2;    // у массива a индексы 0 и 1
```

При увеличении `length` не появляется ни новых элементов, ни новых индексов:

```
var b = [1, 2];  
b.length = 3;    // у массива b индексы 0 и 1
```

# Обход массива

```
var a = [10, 20, 30];
```

Обычно обход выполняется при помощи **for**:

```
for(var i = 0; i < a.length; i++)  
    alert(a[i]);
```

(Микро)оптимизация:

```
for(var i = 0, len = a.length; i < len; i++)  
    alert(a[i]);
```

Пропускаем несуществующие и undefined-элементы:

```
for(var i = 0; i < a.length; i++) {  
    if (a[i] === undefined) continue;  
    // тело цикла  
}
```

Пропускаем только несуществующие элементы:

```
for(var i = 0; i < a.length; i++) {  
    if (!(i in a)) continue;  
    // тело цикла  
}
```

Для обхода «сильно» разреженного массива можно применить цикл `for/in`:

```
for(var index in sparseArray) {  
    var value = sparseArray[index];  
    // далее операции с индексами и значениями  
}
```

Желательно задуматься о проверках, чтобы делать перебор только собственных свойств, которые одновременно являются индексами. (`for..in` выведет все свойства объекта, а не только цифровые)



# Методы массивов

Каждый массив обладает набором методов, изначально определённых в объекте `Array.prototype`.

Многие из этих методов допускают косвенное применение к объектам, подобным массивам. Но некоторые методы меняют тот массив, у которого вызываются, а не все объекты, подобные массивам, допускают изменение (пример – строки).

# Преобразование массива в строку

`join(separator)` – вызывает у каждого элемента `toString()`, получившиеся результаты соединяет, используя указанную строку-разделитель (или запятую, если вызывается без аргумента);

`toString()` – аналог `join()`, вызванного без аргумента;

`toLocaleString()` – работает как `toString()`, но у каждого элемента вызывается `toLocaleString()`;

`valueOf()` – аналог `toString()`

Методы преобразования в строку перебирают индексы массива от 0 до length-1. Их не интересует фактическое наличие индекса!

```
var a = [1, , , , 5];  
alert(a.join("+")); // ВЫВОДИТ "1++++5"
```

```
var b = new Array(5);  
alert(b.join("+")); // ВЫВОДИТ "++++«"
```

```
var monthNames = ['Янв', 'Фев', 'Мар', 'Апр'];  
// присваивает 'Янв,Фев,Мар,Апр' переменной myVar  
var myVar = monthNames.toString();
```

```
var number = 1337;  
var date = new Date();  
var myArr = [number, date, 'foo'];  
var str = myArr.toLocaleString();  
console.log(str);  
// напечатает '1337,6.12.2013 19:37:35,foo',  
// если работает под германской локалью (de-DE) с  
//временной зоной Европа/Берлин
```

# Метод split

split(s) позволяет превратить строку в массив, разбив ее по разделителю s

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

```
var arr = names.split(', '); //запятая и пробел - разделитель
```

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

```
  alert( 'Вам сообщение ' + arr[i] );
```

```
}
```

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

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

Вызов `split` с пустой строкой разобьёт по буквам:

```
var str = "тест";
```

```
alert( str.split("") ); // т,е,с,т
```

# Массив как стек

`pop()`

Удаляет последний элемент из массива и возвращает удалённое значение.

`push(element1, ..., elementN)`

Добавляет элементы в конец массива и возвращает новую длину массива.



```
var stack = [2, 3];
var popped = stack.pop();
// stack = [2], popped = 3

stack.push(8);
var total = stack.push(10, 12);
// stack = [2, 8, 10, 12], total = 4

// КОНКАТЕНАЦИЯ МАССИВОВ
Array.prototype.push.apply(stack, [-1, -2]);
```



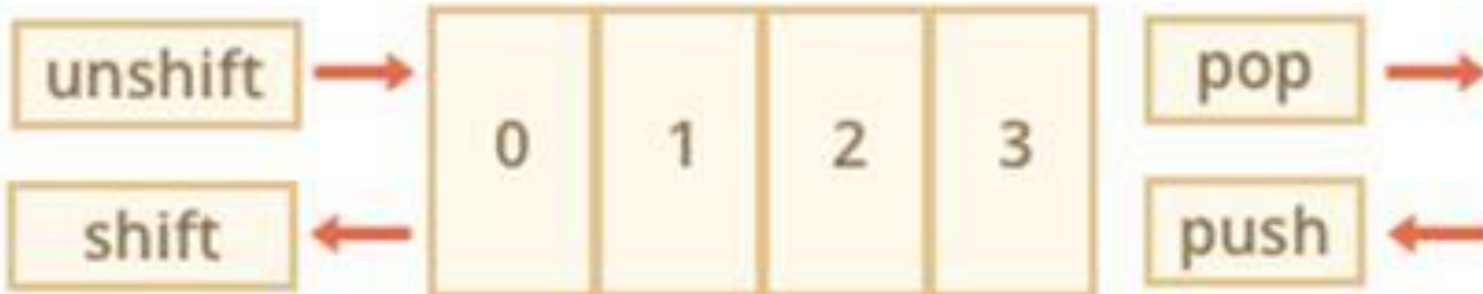
`shift()`

Удаляет и возвращает первый элемент массива, смещая последующие элементы к началу.

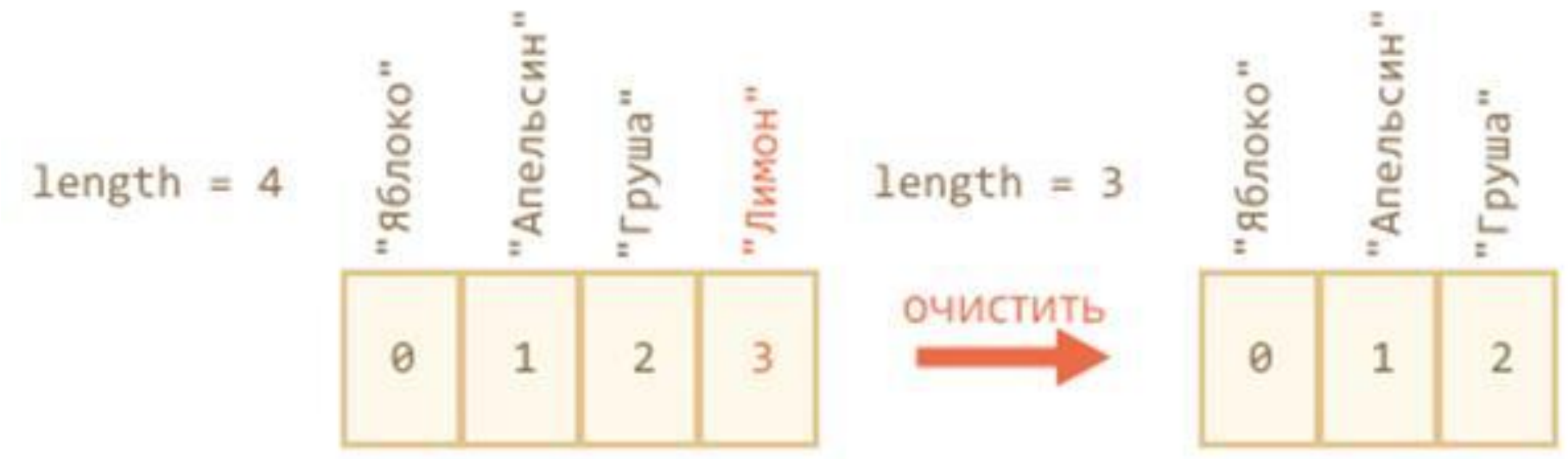
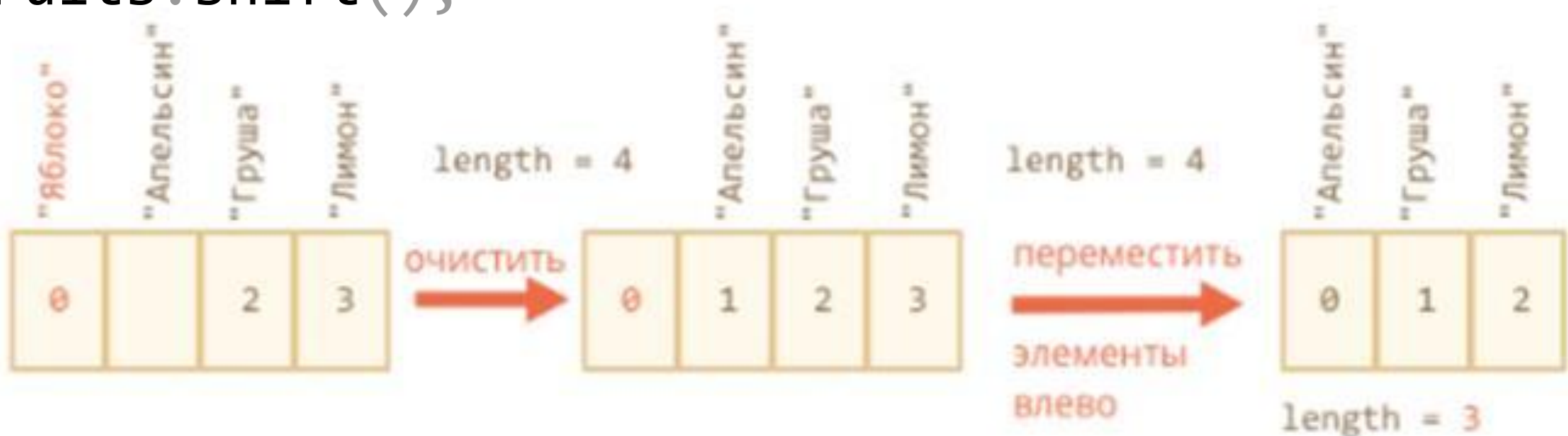
`unshift(element1, ..., elementN)`

Добавляет элементы в начало массива (блоком) и возвращает новую длину массива.

```
var queue = [2, 3];  
var first = queue.shift();  
// queue = [3], first = 2  
  
var total = queue.unshift(-1);  
// queue = [-1, 3], total = 2  
  
queue.unshift(10, 20);  
// queue = [10, 20, -1, 3]
```



```
fruits.shift();
```



```
fruits.pop();
```

```
*var styles = ["Джаз", "Блюз"];
styles.push("Рок-н-Ролл");
styles[styles.length - 2] = "Классика";
alert( styles.shift() );
styles.unshift("Рэп", "Регги");
```

Рэп,Регги,Классика,Рок-н-  
Ролл

//вывод случайного значения из массива:

```
var arr = ["Яблоко", "Апельсин", "Груша", "Лимон"];
var rand = Math.floor(Math.random() * arr.length);
alert( arr[rand] );
```

# Изменение порядка элементов

`reverse()`

На месте обращает порядок следования элементов массива.

`sort(compareFunction)`

На месте сортирует массив по возрастанию. Может принимать функцию сравнения (без неё – сравнивается строковое представление элементов).

```
var a = [1, 11, 2, -3];  
a.sort(); // [-3, 1, 11, 2]
```

```
var arr = [ 1, 2, 15 ];  
arr.sort();  
alert( arr ); // 1, 15, 2
```

```
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  
  
// -1, если a меньше b  
// 1, если a больше b  
// 0, если a и b равны  
a.sort(function(a, b) { return a - b; });  
  
a.reverse(); // [11, 2, 1, -3]
```



увидеть значения, с которыми sort вызывает функцию сравнения

```
[1, -2, 15, 2, 0, 8].sort(function(a, b) {  
  alert( a + " <> " + b );  
}  
);
```

```
concat(elementOrArray1, ..., elementOrArrayN)
```

Возвращает новый массив, состоящий из массива, на котором он был вызван, соединённого с другими массивами и/или значениями, переданными в качестве аргументов.

Метод различает простые аргументы и аргументы-массивы (но без рекурсивной вложенности).

```
var a = [1];
```

```
var b = a.concat(2, 3); // [1, 2, 3]
```

```
var c = a.concat([2, 3]); // [1, 2, 3]
```

```
var d = a.concat([2, 3], 4, [5, 6]);  
// [1, 2, 3, 4, 5, 6]
```

```
var e = a.concat([2, [3, 4]]); // [1, 2, 3, 4]
```

```
var arr = [1, 2];
```

```
var newArr = arr.concat(arr, [3, 4], 5);
```

```
alert( newArr );
```

```
// 1,2,1,2,3,4,5
```

Если в качестве аргумента передать методу `concat()` разреженный массив, он будет присоединён со всеми своими «дырами»:

```
var a = [1, 2, 3];  
var b = a.concat([4, , , 7]);  
// b = [1, 2, 3, 4, , , 7]  
// индексы b = "0", "1", "2", "3", "6"
```

```
slice(begin [,end])
```

Возвращает копию части массива в виде нового массива.

Возвращаются элементы с индексами от `begin` до `end`, не включая `end`.

Если индексы отрицательные – отнимаем от длины.

Копировать весь массив: вызов `slice()` без аргументов.

```
var a = [10, 20, 30, 40, 50];  
var b = a.slice(1, 3);    // [20, 30]  
var c = a.slice(1);      // [20, 30, 40, 50]  
var d = a.slice(3, 1);   // []  
var e = a.slice(-3, -1); // [30, 40]  
var f = a.slice();       // [10, 20, 30, 40, 50]
```

Метод `slice()` в разреженном массиве воспринимает последовательность индексов непрерывной:

```
var a = [1, , , , 6]; // здесь индексы "0" и "5"
```

```
var b = a.slice(2, 4);
```

```
// в b должны оказаться элементы с индексами "2" и "3"
```

```
// но их в a нет, и в b они не попадут
```

```
// т.е. в b нет ни индексов, ни элементов
```

```
// однако b.length = 2 (!!!)
```



```
splice(start, deleteCount, [item1], ..., [itemN])
```

Удалить deleteCount элементов, начиная с номера start, а затем вставить item1, ..., itemN на их место.

```
// начиная с позиции 1, удалить 1 элемент
```

```
arr.splice(1, 1);
```

Изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые.

Если start отрицателен – отнимаем от длины.

Возвращаемое значение: массив, содержащий удалённые элементы.

```
var a = [10, 20, 30, 40, 50];  
var b = a.splice(1, 3);  
// a=[10, 50], b=[20, 30, 40]
```

```
var c = a.splice(1, 0, 20, 30, 40);  
// a=[10, 20, 30, 40, 50], c=[]
```

```
var d = a.splice(1, 1, -1);  
// a=[10, -1, 30, 40, 50], d=[20]
```

```
var e = a.splice(-1, 1, -1);  
// a=[10, -1, 30, 40, -1], e=[50]
```

Метод `splice()` в разреженном массиве ведёт себя подобно методу `slice()` (считает последовательность индексов непрерывной):

```
// a: индексы "0" и "4", a[0]=1, a[4]=5, длина = 5  
var a = [1, , , , 5];
```

```
var b = a.splice(1, 2);  
// b: длина = 2, индексов нет  
alert(b.length);  
for(x in b)  
    alert(x);  
// a: индексы "0" и "2", a[0]=1, a[2]=5, длина = 3
```

```
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
```

# Поиск элементов

`indexOf(element [, startIndex])`

`lastIndexOf(element [, startIndex])`

Методы ищут указанный элемент в массиве и возвращают первую позицию элемента или -1.

Первый ищет от начала массива (или от указанной позиции) к концу, второй – от конца (или от указанной позиции) к началу.

полный перебор, аналогичный циклу `for`

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

```
var arr = [1, 0, false];  
alert( arr.indexOf(0) ); // 1  
alert( arr.indexOf(false) ); // 2  
alert( arr.indexOf(null) ); // -1
```

# Поиск с проверкой поддержки браузером метода indexOf

```
// создаем пустой массив и проверяем поддерживается ли
//indexOf
if ([] .indexOf) {
var find = function(array, value) {
return array.indexOf(value); }
} else {
var find = function(array, value) {
for (var i = 0; i < array.length; i++) {
if (array[i] === value) return i; }
return -1; } } }
```

# Object.keys(obj)

Позволяет работать со свойствами в виде массива

```
var user = {  
  name: "Петя",  
  age: 30  
}  
  
var keys = Object.keys(user);  
alert( keys ); // name, age
```



Массив – это объект, в функцию он передаётся по ссылке:

```
function eat(arr) {  
    arr.pop();  
}
```

```
var arr = ["нам", "не", "страшен", "серый", "волк"];
```

```
alert( arr.length ); // 5
```

```
eat(arr);
```

```
eat(arr);
```

```
alert( arr.length );
```

```
// 3, в функцию массив не скопирован, а передана ссылка
```

МОЖНО присваивать в массив любые свойства:

```
var fruits = []; // создать массив
```

```
fruits[99999] = 5; // присвоить свойство с любым номером
```

```
fruits.age = 25; // назначить свойство со строковым именем
```

```
//не рекомендуется
```

```
var arr = [1, 2, 3];
```

```
var arr2 = arr;
```

```
arr2[0] = 5;
```

```
//результат?
```

```
alert( arr[0] );
```

```
alert( arr2[0] );
```

5

5

```
arr2 == arr
```

```
true
```

```
var arr2 = [];
```

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

```
arr2[i] = arr[i];
```

1,2,3

```
alert( new Array(4).join("ля") );
```

ляляля

# Методы перебора элементов

Эти пять методов появились в ECMAScript 5. Методы принимают функцию  $f()$ , выполняемую для каждого элемента массива:

1. `every()` – возвращает `true`, если  $f()$  вернула `true` для каждого элемента;
2. `some()` – возвращает `true`, если  $f()$  вернула `true` хотя бы для одного элемента;
3. `filter()` – возвращает массив элементов, для которых  $f()$  вернула `true`;
4. `map()` – возвращает массив с результатами вызова  $f()$ ;
5. `forEach()` – просто выполняет  $f()$  для каждого элемента;

```
var arr = [1, -1, 2, -2, 3];  
function isPositive(number) {  
return number > 0;  
}
```

```
alert( arr.every(isPositive) ); // false, не все положительные
```

```
// true, есть хоть одно положительное
```

```
alert( arr.some(isPositive) );
```

```
var arr = ["Яблоко", "Апельсин", "Груша"];

arr.forEach(function(item, i, arr) {
alert( i + ": " + item + " (массив:" + arr + ")" );
});
```

- `item` – очередной элемент массива.
- `i` – его номер.
- `arr` – массив, который перебирается.

0: Яблоко (массив:Яблоко,Апельсин,Груша)

1: Апельсин (массив:Яблоко,Апельсин,Груша)

2: Груша (массив:Яблоко,Апельсин,Груша)



```
var names = ['HTML', 'CSS', 'JavaScript'];  
var nameLengths = names.map(function(name) {  
    return name.length;  
});  
// получили массив с длинами  
alert( nameLengths ); // 4,3,10
```

```
var a = [1, 2, -3, 4, 5];
```

```
var all = a.every(function(x){ return x > 0;}); //false
```

```
var any = a.some(function(x){ return x < 0;}); // true
```

```
var where = a.filter(function(x){ return x > 0;});
```

```
// where = [1, 2, 4, 5]
```

```
where.map(function(x){ return x * 10;}).forEach(alert);
```

```
// ВЫВОДИТ "10", "20", "40", "50"
```

# Редукция элементов

```
reduce(callback [,initialValue])
```

```
reduceRight(callback [,initialValue])
```

Методы (ECMAScript 5) формируют из массива скалярное значение при помощи функции-аккумулятора (сохр. промежуточный рез-т).

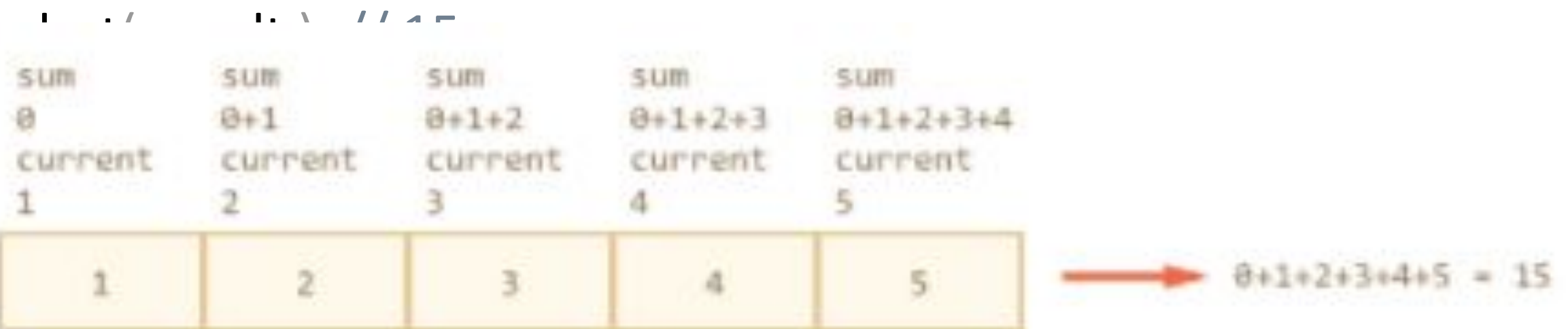
Эта функция с четырьмя аргументами: последний результат вызова функции, он же «промежуточный результат», текущий элемент, текущий индекс, исходный массив. Функция возвращает новое значение аккумулятора.

Если есть начальное значение – `initialValue`, то на первом вызове значение аккумулятора будет равно `initialValue`, а если у `reduce` нет второго аргумента, то оно равно первому элементу массива, а перебор начинается со второго.

```
var arr = [1, 2, 3, 4, 5]
```

```
// для каждого элемента массива запустить функцию,  
// промежуточный результат передавать первым аргументом  
далее
```

```
var result = arr.reduce(function(sum, current) {  
  return sum + current;  
}, 0);
```



```
var sum = function(prev, cur, index, arr){
return prev + cur;};
var res = [1, 2, -3, 4, 5].reduce(sum);
// res = 9
```

```
var flat = [[0, 1], [2, 3], [4, 5]].reduce(function(a, b) {
return a.concat(b);
});
// flat = [0, 1, 2, 3, 4, 5]
```

```
flat = [[0, 1], [2, 3], [4, 5]].reduceRight(function(a, b) {
return a.concat(b);
}, [6]);
// flat = [6, 4, 5, 2, 3, 0, 1]
```

# Проверка на массив

`Array.isArray(obj)`

Этот статический метод проверяет, является ли `obj` массивом, и возвращает `true` или `false`.

Метод появился в ECMAScript 5.

# Сводная таблица методов и свойств

length

join()	pop()	reverse()	indexOf()	every()
toString()	push()	sort()	lastIndexOf()	some()
toLocaleString()	shift()	concat()		filter()
valueOf()	unshift()	slice()		map()
		splice()		forEach()
				reduce()
				reduceRight()
				Array.isArray()



# Объекты, подобные массивам

Это объекты, обладающие индексами и свойством `length`. «Похожесть» на массивы позволяет применять к таким объектам многие методы массивов (если методу надо только `length` и индексы – всё будет ОК).

Примеры объектов, подобных массивам:

a) объект `Arguments`

b) строка (с поправкой на неизменяемость!)

```
var o = { "0": "A", "1": "B", length: 2};  
var s = Array.prototype.join.call(o, "+"); // s = "A+B"
```

```
var str = "JavaScript";
```

```
var res = Array.prototype.filter.call(str,  
    function(x) { return x >= "a"; });
```

```
// res = ["a", "v", "a", "c", "r", "i", "p", "t"]
```

```
res = Array.prototype.slice.call(str).sort().join("");
```

```
// res = "JSaaciprtv"
```