05 - Массивы

1. Массивы

Массив – это упорядоченная коллекция значений. Значения в массиве называются элементами, и каждый элемент характеризуется **числовой позицией** в массиве, которая называется **индексом**. Массивы в языке JavaScript являются **не типизированными**: элементы массива могут иметь любой тип, причем разные элементы одного и того же массива могут иметь разные типы. Элементы массива могут даже быть объектами или другими массивами, что позволяет создавать сложные структуры данных, такие как массивы объектов и массивы массивов.

Отсчет индексов массивов в языке JavaScript **начинается с нуля** и для них используются 32-битные целые числа: первый элемент массива имеет индекс 0, а наибольший возможный индекс имеет значение 4294967294 (232–2), т. е. максимально возможный размер массива составляет 4294967295 элементов. **Массивы** в JavaScript **являются динамическими**: они могут увеличиваться и уменьшаться в размерах по мере необходимости; нет необходимости объявлять фиксированные размеры массивов при их создании или повторно распределять память при изменении их размеров. Массивы в JavaScript могут быть разреженными: не требуется, чтобы массив содержал элементы с непрерывной последовательностью индексов – в массивах могут отсутствовать элементы с некоторыми индексами.

Все массивы в JavaScript имеют **свойство length**. Для неразреженных массивов это свойство определяет количество элементов в массиве. Для разреженных массивов значение length больше числа всех элементов в массиве.

Массивы – это специализированная форма объектов, а индексы массивов означают чуть больше, чем просто имена свойств, которые по совпадению являются целыми числами. Реализации обычно оптимизируют операции с массивами, благодаря чему доступ к элементам массивов по их числовым индексам выполняется значительно быстрее, чем доступ к обычным свойствам объектов.

Массивы наследуют свои свойства от прототипа Array.prototype, который определяет богатый набор методов манипулирования массивами. Большинство из этих методов являются универсальными, т. е. они могут применяться не только к истинным массивам, но и к любым объектам, «похожим на массивы».

Создание массивов. Легче всего создать массив **с помощью литерала**, который представляет собой простой список разделенных запятыми элементов массива в квадратных скобках. Например:

```
var empty = []; // Пустой массив
var primes = [2, 3, 5, 7, 11]; // Массив с пятью числовыми элементами
var misc = [ 1.1, true, "a", ]; // 3 элемента разных типов + завершающая запятая
```

Значения в литерале массива не обязательно должны быть константами – это могут быть любые выражения:

```
var base = 1024;
var table = [base, base+1, base+2, base+3];
```

Литералы массивов могут содержать литералы объектов или литералы других массивов:

```
var b = [[1,{x:1, y:2}], [2, {x:3, y:4}]];
```

Если литерал массива содержит несколько идущих подряд запятых без значений между ними, создается **разреженный массив.** Элементы, соответствующие таким пропущенным значениям, отсутствуют в массиве, но при обращении к ним возвращается значение **undefined:**

```
var count = [1,,3]; // Элементы с индексами 0 и 2. count[1] => undefined
var undefs = [,,]; // Массив без элементов, но с длиной, равной 2
```

Синтаксис литералов массивов позволяет вставлять необязательную завершающую запятую, т. е. литерал [,,] соответствует массиву с двумя элементами, а не с тремя.

Другой способ создания массива состоит в **вызове конструктора Array()**. Вызвать конструктор можно тремя разными способами:

• Вызвать конструктор без аргументов:

```
var a = new Array();
```

В этом случае будет создан пустой массив, эквивалентный литералу [].

• Вызвать конструктор с единственным числовым аргументом, определяющим длину массива:

```
var a = new Array(10);
```

В этом случае будет создан пустой массив указанной длины. Такая форма вызова конструктора Array() может использоваться для предварительного распределения памяти под массив, если заранее известно количество его элементов. Обратите внимание, что при этом в массиве не сохраняется никаких значений и даже свойства-индексы массива с именами «0», «1» и т.д. в массиве не определены.

• Явно указать в вызове конструктора значения первых двух или более элементов массива или один нечисловой элемент:

```
var a = new Array(5, 4, 3, 2, 1, "testing, testing");
```

В этом случае аргументы конструктора становятся значениями элементов нового массива. Использование литералов массивов практически всегда проще, чем подобное применение конструктора Array().

Чтение и запись элементов массива. Доступ к элементам массива осуществляется с помощью оператора []. Слева от скобок должна присутствовать ссылка на массив. Внутри скобок должно находиться произвольное выражение, возвращающее неотрицательное целое значение. Этот синтаксис пригоден как для чтения, так и для записи значения элемента массива. Следовательно, допустимы все приведенные далее JavaScript-инструкции:

```
var a = ["world"]; // Создать массив с одним элементом
```

```
var value = a[0]; // Прочитать элемент 0
a[1] = 3.14; // Записать значение в элемент 1
i = 2;
a[i] = 3; // Записать значение в элемент 2
a[i + 1] = "hello"; // Записать значение в элемент 3
a[a[i]] = a[0]; // Прочитать элементы 0 и 2, записать значение в элемент 3
```

Следует четко отличать индексы в массиве от имен свойств объектов. Все индексы являются именами свойств, но только свойства с именами, представленными целыми числами в диапазоне от 0 до 232–2 являются индексами. Все массивы являются объектами, и вы можете добавлять к ним свойства с любыми именами. Однако если вы затрагиваете свойства, которые являются индексами массива, массивы реагируют на это, обновляя значение свойства length при необходимости.

Разреженные массивы. Разреженным называется массив, индексы элементов которого не образуют непрерывную последовательность чисел, начиная с 0. Обычно свойство length массива определяет количество элементов в массиве. В разреженном массиве значение свойства length больше количества элементов. Разреженный массив можно создать с помощью конструктора Array() или путем присваивания значения элементу с индексом, большим, чем текущая длина массива.

```
a = new Array(5); // Нет элементов, но a.length имеет значение 5.
a = []; // Создаст пустой массив со значением length = 0.
a[1000] = 0; // Добавит один элемент, но установит длину равной 1001.
```

Существенно разреженные массивы обычно более медленны и потребляют больше памяти, чем плотные массивы, а поиск элементов в таких массивах занимает примерно столько же времени, что и поиск обычных свойств объектов. Обратите внимание, что литералы с пропущенными значениями (когда в определении подряд следуют запятые, например [1,,3]) создают разреженные массивы, в которых пропущенные элементы просто не существуют:

```
var a1 = [,]; // Массив без элементов с длиной, равной 1
var a2 = [undefined]; // Массив с одним неопределенным элементом
0 in a1 // => false: a1 не имеет элемента с индексом 0
0 in a2 // => true: a2 имеет элемент с индексом 0 и со значением undefined
```

Длина массива. Любой массив имеет свойство length, и это свойство отличает массивы от обычных объектов JavaScript. Для плотных (т. е. неразреженных) массивов свойство length определяет количество элементов в массиве. Его значение на единицу больше самого большого индекса в массиве:

```
[].length // => 0: массив не имеет элементов
['a','b','c'].length // => 3: наибольший индекс равен 2, длина равна 3
```

Для разреженных массивов значение свойства length больше числа элементов, и все, что можно сказать в этом случае, – это то, что значение свойства length гарантированно будет превышать индекс любого элемента в массиве. Или, говоря иначе, массивы (разреженные или нет) никогда не будут содержать элемент, индекс которого будет больше или равен значению свойства length массива. Особенность в поведении, обеспечивающем работу свойства length, заключается в том, что при присваивании свойству length неотрицательного целого числа n, меньшего, чем его текущее значение, все элементы массива с индексами, большими или равными значению n, удаляются из массива:

```
a = [1,2,3,4,5]; // Создать массив с пятью элементами.
a.length = 3; // теперь массив a содержит элементы [1,2,3].
a.length = 0; // Удалит все элементы. a - пустой массив [].
a.length = 5; // Длина равна 5, но элементы отсутствуют, подобно Array(5)
```

В свойство length массива можно также записать значение больше, чем его текущее значение. В этом случае в массив не добавляются новые элементы, а просто создается разреженная область в конце массива.

Добавление и удаление элементов массива. Мы уже видели, что самый простой способ добавить элементы в массив заключается в том, чтобы присвоить значения новым индексам:

```
a = [] // Создать пустой массив.
a[0] = "zero"; // И добавить элементы.
a[1] = "one";
```

Для добавления одного или более элементов в конец массива можно также использовать метод push():

```
a = []; // Создать пустой массив
a.push("zero") // Добавить значение в конец. a = ["zero"]
a.push("one", "two") // Добавить еще два значения. a = ["zero", "one", "two"]
```

Добавить элемент в конец массива можно также, присвоив значение элементу a[a.length]. Для вставки элемента в начало массива можно использовать метод unshift() при этом существующие элементы в массиве смещаются в позиции с более высокими индексами.

Методы push() и pop(). Методы push() и pop() позволяют работать с массивами как со стеками. Метод push() добавляет один или несколько новых элементов в конец

массива и возвращает его новую длину. Метод pop() выполняет обратную операцию – удаляет последний элемент массива, уменьшает длину массива и возвращает удаленное им значение. Обратите внимание, что оба эти метода изменяют исходный массив, а не создают его модифицированную копию. Комбинация push() и pop() позволяет на основе массива реализовать стек с дисциплиной обслуживания «первым вошел – последним вышел». Например:

```
var stack = []; // стек: []
stack.push(1,2); // стек: [1,2] Вернет 2
stack.pop(); // стек: [1] Вернет 2
stack.push(3); // стек: [1,3] Вернет 2
stack.pop(); // стек: [1] Вернет 3
stack.push([4,5]); // стек: [1,[4,5]] Вернет 2
stack.pop() // стек: [1] Вернет [4,5]
stack.pop(); // стек: [] Вернет 1
```

Методы unshift() и shift(). Методы unshift() и shift() ведут себя почти так же, как push() и pop(), за исключением того, что они вставляют и удаляют элементы в начале массива, а не в конце. Метод unshift() смещает существующие элементы в сторону больших индексов для освобождения места, добавляет элемент или элементы в начало массива и возвращает новую длину массива. Метод shift() удаляет и возвращает первый элемент массива, смещая все последующие элементы на одну позицию вниз, чтобы занять место, освободившееся в начале массива. Например:

```
var a = []; // a:[]
a.unshift(1); // a:[1] Вернет: 1
a.unshift(22); // a:[22,1] Вернет: 2
a.shift(); // a:[1] Вернет: 22
a.unshift(3,[4,5]); // a:[3,[4,5],1] Вернет: 3
a.shift(); // a:[[4,5],1] Вернет: 3
a.shift(); // a:[1] Вернет: [4,5]
a.shift(); // a:[] Вернет: 1
```

Обратите внимание на поведение метода unshift() при вызове с несколькими аргументами. Аргументы вставляются не по одному, а все сразу. Это значит, что в результирующем массиве они будут следовать в том же порядке, в котором были указаны в списке аргументов. Будучи вставленными по одному, они бы расположились в обратном порядке.

Также удалять элементы массива можно с помощью оператора delete, как обычные свойства объектов:

```
a = [1,2,3];
delete a[1]; // теперь в массиве а отсутствует элемент с индексом 1
1 in a // => false: индекс 1 в массиве не определен
a.length // => 3: оператор delete не изменяет свойство length массива
```

Удаление элемента напоминает (но несколько отличается) присваивание значения undefined этому элементу. Обратите внимание, что применение оператора

delete к элементу массива не изменяет значение свойства length и не сдвигает вниз элементы с более высокими индексами, чтобы заполнить пустоту, оставшуюся после удаления элемента. После удаления элемента массив превращается в разреженный массив.

Наконец существует многоцелевой метод splice(), позволяющий вставлять, удалять и замещать элементы массивов. Он изменяет значение свойства length и сдвигает элементы массива с более низкими или высокими индексами по мере необходимости.

Метод slice(). Метод Array.slice() возвращает фрагмент, или подмассив, указанного массива. Два аргумента метода определяют начало и конец возвращаемого фрагмента.
Возвращаемый массив содержит элемент, номер которого указан в первом аргументе, плюс все последующие элементы, вплоть до (но не включая) элемента, номер которого указан во втором аргументе. Если указан только один аргумент, возвращаемый массив содержит все элементы от начальной позиции до конца массива. Если какой-либо из аргументов имеет отрицательное значение, он определяет номер элемента относительно конца массива. Так, аргументу –1 соответствует последний элемент массива, а аргументу –3 – третий элемент массива с конца. Вот несколько примеров:

```
var a = [1,2,3,4,5];
a.slice(0,3); // Вернет [1,2,3]
a.slice(3); // Вернет [4,5]
a.slice(1,-1); // Вернет [2,3,4]
a.slice(-3,-2); // Вернет [3]
```

Метод splice(). Метод Array.splice() – это универсальный метод, выполняющий вставку или удаление элементов массива. В отличие от методов slice(), метод splice() изменяет исходный массив, относительно которого он был вызван. Обратите внимание, что методы splice() и slice() имеют очень похожие имена, но выполняют совершенно разные операции. Метод splice() может удалять элементы из массива, вставлять новые элементы или выполнять обе операции одновременно. Элементы массива при необходимости смещаются, чтобы после вставки или удаления образовывалась непрерывная последовательность. Первый аргумент метода splice() определяет позицию в массиве, начиная с которой будет выполняться вставка и/или удаление. Второй аргумент определяет количество элементов, которые должны быть удалены (вырезаны) из массива. Если второй аргумент опущен, удаляются все элементы массива от указанного до конца массива. Метод splice() возвращает массив удаленных элементов или (если ни один из элементов не был удален) пустой массив. Например:

```
var a = [1,2,3,4,5,6,7,8];
a.splice(4); // Вернет [5,6,7,8]; a = [1,2,3,4]
a.splice(1,2); // Вернет [2,3]; a = [1,4]
a.splice(1,1); // Вернет [4]; a = [1]
```

Первые два аргумента метода splice() определяют элементы массива, подлежащие удалению. За этими аргументами может следовать любое количество дополнительных аргументов, определяющих элементы, которые будут вставлены в массив, начиная с позиции, указанной в первом аргументе. Например:

```
var a = [1,2,3,4,5];
a.splice(2,0,'a','b'); // Вернет []; a = [1,2,'a','b',3,4,5]
```

Методы indexOf() и lastIndexOf(). Методы indexOf() и lastIndexOf() отыскивают в массиве элемент с указанным значением и возвращают индекс первого найденного элемента или –1, если элемент с таким значением отсутствует. Метод indexOf() выполняет поиск от начала массива к концу, а метод lastIndexOf() – от конца к началу.

```
a = [0,1,2,1,0];
a.indexOf(1) // => 1: a[1] = 1
a.lastIndexOf(1) // => 3: a[3] = 1
a.indexOf(3) // => -1: нет элемента со значением 3
```

В отличие от других методов, описанных в этом разделе, методы indexOf() и lastIndexOf() не принимают функцию в виде аргумента. В первом аргументе им передается искомое значение. Второй аргумент является необязательным: он определяет индекс массива, с которого следует начинать поиск. Если опустить этот аргумент, метод indexOf() начнет поиск с начала массива, а метод lastIndexOf() – с конца. Во втором аргументе допускается передавать отрицательные значения, которые интерпретируются как смещение относительно конца массива, как в методе splice(): значение –1, например, соответствует последнему элементу массива.

Следующая функция отыскивает заданное значение в массиве и возвращает массив всех индексов, где было найдено совпадение. Здесь демонстрируется, как можно использовать второй аргумент метода indexOf() для поиска совпадений после первого.

```
// Отыскивает все вхождения значения x в массив и возвращает
// массив индексов найденных совпадений
function findall(a, x) {
    var results = [], // Возвращаемый массив индексов
        len = a.length, // Длина массива, где выполняется поиск
        pos = 0; // Начальная позиция поиска
    while(pos < len) { // Пока остались непроверенные элементы...
        pos = a.indexOf(x, pos); // Искать
        if (pos === -1) break; // Если ничего не найдено, поиск завершен.
        results.push(pos); // Иначе - сохранить индекс в массиве
        pos = pos + 1; // И продолжить поиск со следующего элемента
    }
    return results; // Вернуть массив индексов
}
```

Обратите внимание, что строки также имеют методы indexOf() и lastIndexOf(), которые действуют подобно методам массивов.

Метод join(). Метод Array.join() преобразует все элементы массива в строки, объединяет их и возвращает получившуюся строку. В необязательном аргументе методу можно передать строку, которая будет использоваться для отделения элементов в строке результата. Если строка-разделитель не указана, используется запятая. Например, следующий фрагмент дает в результате строку «1,2,3»:

```
var a = [1, 2, 3]; // Создать новый массив с указанными тремя элементами
a.join(); // => "1,2,3"
a.join(" "); // => "1 2 3"
a.join(""); // => "123"
var b = new Array(10); // Массив с длиной, равной 10, и без элементов
b.join('-') // => '---------': строка из 9 дефисов
```

Метод Array.join() является обратным по отношению к методу String.split(), создающему массив путем разбиения строки на фрагменты.

Метод reverse(). Метод Array.reverse() меняет порядок следования элементов в массиве на обратный и возвращает переупорядоченный массив. Перестановка выполняется непосредственно в исходном массиве, т. е. этот метод не создает новый массив с переупорядоченными элементами, а переупорядочивает их в уже существующем массиве. Например, следующий фрагмент, где используются методы reverse() и join(), дает в результате строку "3,2,1":

```
var a = [1,2,3];
a.reverse().join(); // => "3,2,1": теперь a = [3,2,1]
```

Метод sort(). Метод Array.sort() сортирует элементы в исходном массиве и возвращает отсортированный массив. Если метод sort() вызывается без аргументов, сортировка выполняется в алфавитном порядке (для сравнения элементы временно преобразуются в строки, если это необходимо):

```
var a = new Array("banana", "cherry", "apple");
a.sort();
var s = a.join(", "); // s == "apple, banana, cherry"
```

Неопределенные элементы переносятся в конец массива. Для сортировки в каком-либо ином порядке, отличном от алфавитного, методу sort() можно передать функцию сравнения в качестве аргумента. Эта функция устанавливает, какой из двух ее аргументов должен следовать раньше в отсортированном списке. Если первый аргумент должен предшествовать второму, функция сравнения должна возвращать отрицательное число. Если первый аргумент должен следовать за вторым в отсортированном массиве, то функция должна возвращать число больше нуля. А если два значения эквивалентны (т.е. порядок их следования не важен), функция сравнения должна возвращать 0. Поэтому, например, для сортировки элементов массива в числовом порядке можно сделать следующее:

```
var a = [33, 4, 1111, 222];
a.sort(); // Алфавитный порядок: 1111, 222, 33, 4
a.sort(function(a,b) { // Числовой порядок: 4, 33, 222, 1111
        return a-b; // Возвращает значение < 0, 0 или > 0
```

```
}); // в зависимости от порядка сортировки a и b
a.sort(function(a,b) {return b-a}); // Обратный числовой порядок
```

Обратите внимание, насколько удобно использовать в этом фрагменте неименованную функцию. Функция сравнения используется только здесь, поэтому нет необходимости давать ей имя. В качестве еще одного примера сортировки элементов массива можно реализовать сортировку массива строк без учета регистра символов, передав функцию сравнения, преобразующую свои аргументы в нижний регистр (с помощью метода toLowerCase()) перед сравнением.

```
a = ['ant', 'Bug', 'cat', 'Dog']
a.sort();  // сортировка с учетом регистра символов: ['Bug','Dog','ant',cat']
a.sort(function(s,t) { // Сортировка без учета регистра символов
      var a = s.toLowerCase();
      var b = t.toLowerCase();
      if (a < b) return -1;
      if (a > b) return 1;
      return 0;
}); // => ['ant','Bug','cat','Dog']
```

Метод concat(). Метод Array.concat() создает и возвращает новый массив, содержащий элементы исходного массива, для которого был вызван метод concat(), и значения всех аргументов, переданных методу concat(). Если какой-либо из этих аргументов сам является массивом, его элементы добавляются в возвращаемый массив. Следует, однако, отметить, что рекурсивного превращения массива из массивов в одномерный массив не происходит. Метод concat() не изменяет исходный массив. Ниже приводится несколько примеров:

```
var a = [1,2,3];
a.concat(4, 5) // Вернет [1,2,3,4,5]
a.concat([4,5]); // Вернет [1,2,3,4,5]
a.concat([4,5],[6,7]) // Вернет [1,2,3,4,5,6,7]
a.concat(4, [5,[6,7]]) // Вернет [1,2,3,4,5,[6,7]]
```

Метод forEach(). Метод forEach() выполняет обход элементов массива и для каждого из них вызывает указанную функцию. Как уже говорилось выше, функция передается методу forEach() в первом аргументе. При вызове этой функции метод forEach() будет передавать ей три аргумента: значение элемента массива, индекс элемента и сам массив. Если вас интересует только значение элемента, можно написать функцию с одним параметром – дополнительные аргументы будут игнорироваться:

```
var data = [1,2,3,4,5]; // Массив, элементы которого будут суммироваться

// Найти сумму элементов массива
var sum = 0; // Начальное значение суммы 0
data.forEach(function(value) { sum += value; }); // Прибавить значение к sum
sum // => 15
```

```
// Увеличить все элементы массива на 1
data.forEach(function(v, i, a) { a[i] = v + 1; });
data // => [2,3,4,5,6]
```

Обратите внимание, что метод forEach() не позволяет прервать итерации, пока все элементы не будут переданы функции. То есть отсутствует эквивалент инструкции break, которую можно использовать с обычным циклом for. catch(e) {

Метод map(). Метод map() передает указанной функции каждый элемент массива, относительно которого он вызван, и возвращает массив значений, возвращаемых этой функцией. Например:

```
a = [1, 2, 3];
b = a.map(function(x) { return x*x; }); // b = [1, 4, 9]
```

Метод map() вызывает функцию точно так же, как и метод forEach(). Однако функция, передаваемая методу map(), должна возвращать значение. Обратите внимание, что map() возвращает новый массив: он не изменяет исходный массив. Если исходный массив является разреженным, возвращаемый массив также будет разреженным: он будет иметь ту же самую длину и те же самые отсутствующие элементы.

Метод filter(). Метод filter() возвращает массив, содержащий подмножество элементов исходного массива. Передаваемая ему функция должна быть функцией-предикатом, т. е. должна возвращать значение true или false. Метод filter() вызывает функцию точно так же, как методы forEach() и map(). Если возвращается true или значение, которое может быть преобразовано в true, переданный функции элемент считается членом подмножества и добавляется в массив, возвращаемый методом. Например:

```
a = [5, 4, 3, 2, 1];
smallvalues = a.filter(function(x) { return x < 3 }); // [2, 1]
everyother = a.filter(function(x,i) { return i%2==0 }); // [5, 3, 1]
```

Обратите внимание, что метод filter() пропускает отсутствующие элементы в разреженных массивах и всегда возвращает плотные массивы. Чтобы уплотнить разреженный массив, можно выполнить следующие действия:

```
var dense = sparse.filter(function() { return true; });
```

А чтобы уплотнить массив и удалить из него все элементы со значениями undefined и null, можно использовать метод filter(), как показано ниже:

```
a = a.filter(function(x) { return x !== undefined && x != null; });
```

Методы every() и some(). Методы every() и some() применяют указанную функцию к элементам массива и возвращают true или false. Метод every() напоминает математический квантор всеобщности ∀: он возвращает true, только если переданная вами функция-предикат вернула true для всех элементов массива:

```
a = [1,2,3,4,5];
a.every(function(x) { return x < 10; }) // => true: все значения < 10.
```

```
a.every(function(x) { return x % 2 === 0; }) // => false: не все четные.
```

Метод some() возвращает true, если в массиве имеется хотя бы один элемент, для которого функция вернет true, а значение false возвращается методом, только если функция вернет false для всех элементов массива:

```
a = [1,2,3,4,5];
a.some(function(x) { return x%2===0; }) // => true: имеются четные числа.
a.some(isNaN) // => false: нет нечисловых элементов.
```

Обратите внимание, что оба метода, every() и some(), прекращают обход элементов массива, как только результат становится известен. Метод some() возвращает true, как только функция вернет true, и выполняет обход всех элементов массива, только если функция всегда возвращает false. Метод every() является полной противоположностью: он возвращает false, как только функция вернет false, и выполняет обход всех элементов массива, только если функция всегда возвращает true. Кроме того, отметьте, что в соответствии с правилами математики для пустого массива метод every() возвращает true, а метод some() возвращает false.

Многомерные массивы. JavaScript не поддерживает «настоящие» многомерные массивы, но позволяет неплохо имитировать их при помощи массивов из массивов. Для доступа к элементу данных в массиве массивов достаточно дважды использовать оператор []. Например, предположим, что переменная matrix – это массив массивов чисел. Каждый элемент matrix[x] – это массив чисел. Для доступа к определенному числу в массиве можно использовать выражение matrix[x][y]. Ниже приводится конкретный пример, где двумерный массив используется в качестве таблицы умножения:

```
// Создать многомерный массив
var table = new Array(10); // В таблице 10 строк
for(var i = 0; i < table.length; i++)
table[i] = new Array(10); // В каждой строке 10 столбцов

// Инициализировать массив
for(var row = 0; row < table.length; row++) {
      for(col = 0; col < table[row].length; col++) {
            table[row][col] = row*col;
      }
}

// Расчет произведения 5*7 с помощью многомерного массива
var product = table[5][7]; // 35
```

ДЗ. Пользователь вводит ФИО, привести ФИО к единому виду . Например: ввели иВаноВ Иван иванЫЧ, будет - Иванов Иван Иваныч

В домашних заданиях по массивам, длину массива задает пользователь, массив заполняется случайными целыми числами. необходимо предусмотреть некорректный

ввод данных. Каждая программа сначала выводит исходный массив, потом результат работы.

Операции с массивом. В одномерном массиве, состоящем из N вещественных чисел вычислить:

1. Сумму отрицательных элементов.

2. Произведение, находящихся между min и max элементами.

3. Произведение элементов с четными номерами.

4. Сумму элементов, находящихся между первым и последним

отрицательными элементами.

5. Максимальный элемент массива.

Сжать массив. Сжать массив, удалив из него все 0 и заполнить освободившиеся справа элементы значениями -1. Например дан массив [1, 4, 0, 5, 0 ,6, 8] - будет [1, 4, 5, 6, 8, -1, -1]. Массив может быть любой длинны.

Преобразование массива. Преобразовать массив так, чтобы сначала шли все отрицательные элементы, а потом положительные(0 считать положительным). Например дан массив [1, -4, 0, 6, 17, -5, 0] - будет [-4, -5, 1, 0, 6, 17, 0]

Модуль элементов. Заменить все отрицательные значения элементов массива их модулями. Например был массив [-1, 3, -5, 2] - будет [1, 3, 5, 2]

Среднее арифметическое массива. Написать программу, определяющую среднее арифметическое положительных/отрицательных/ненулевых элементов массива.

Сумма кол-во. Написать программу, определяющую сумму/количество положительных/отрицательных элементов массива. Например дан массив [1, -4, 6, 7, -3], программа выведет:

Сумма положительных элементов = 14

Кол-во положительных элементов = 3

Сумма отрицательных элементов = -7

Кол-во отрицательных элементов = 2

Исходный массив - [1, -4, 6, 7, -3]

Число в массиве. Написать программу, которая предлагает пользователю ввести число и считает сколько раз это число встречается в массиве

Копирование массивов. Написать программу, копирующую последовательно элементы 2-х массивов в один массив.

Копирование массивов 2. Написать программу, копирующую элементы 2-х массивов в один массив следующим образом: сначала копируются последовательно все

элементы, большие 0, затем последовательно все элементы, равные 0, а затем последовательно все элементы, меньшие 0.

Копирование массивов 3. Написать программу, копирующую последовательно элементы одного массива размером 10 элементов в 2 массива размером 5 элементов каждый.

Сумма элементов после 0. Написать программу, определяющую сумму элементов массива, находящихся в массиве после первого элемента со значением 0.

Например дан массив [1, 3, 0, -4, 7, 2] - ответ 5.

Матрица - это двухмерный массив. N или M задает пользователь.

Заполнение 2-х мерного массива случ. числами. Написать программу, которая позволяет двухмерный заполнить массив случайным образом значениями в диапазоне от -100 до 100 и выводит минимальное и максимальное значения. Пользователь вводит длину массивов.

Сумма элементов матрицы. Дана матрица N x M. Определите сумму всех ее элементов.