**Задание 26**

**Дайте определение понятию «Массив». Опишите виды массивов в языке JavaScript, способы объявления массивов в языке JavaScript. Опишите порядок доступа к элементам массива в языке JavaScript.**

Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array.

Существует два варианта синтаксиса для создания пустого массива:

let arr = new Array();

let arr = [];

Практически всегда используется второй вариант синтаксиса. В скобках можно указать начальные значения элементов:

let fruits = ["Яблоко", "Апельсин", "Слива"];

Элементы массива нумеруются, начиная с нуля. Получить элемент можно указав его номер в квадратных скобках:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Заменить элемент:

fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]

Добавить новый к существующему массиву:

fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]

Общее число элементов массива содержится в его свойстве length:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits.length ); // 3

Вывести массив целиком можно при помощи alert.

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits ); // Яблоко, Апельсин, Слива

В массиве могут храниться элементы любого типа.

Например разные типы значений

let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];

Получить элемент с индексом 1 (объект) и затем показать его свойство

alert( arr[1].name ); // Джон

Получить элемент с индексом 3 (функция) и выполнить её

arr[3](); // привет

Висячая запятая: список элементов массива, как и список свойств объекта, может оканчиваться запятой:

let fruits = [

"Яблоко",

"Апельсин",

"Слива"*,*

];

«Висячая запятая» упрощает процесс добавления/удаления элементов, так как все строки становятся идентичными.

Допустим, нам нужен последний элемент массива, некоторые языки программирования позволяют использовать отрицательные индексы для той же цели, как-то так: fruits[-1].

Однако, в JavaScript такая запись не сработает. Её результатом будет undefined, поскольку индекс в квадратных скобках понимается буквально.

Можно явно вычислить индекс последнего элемента, а затем получить к нему доступ вот так: fruits[fruits.length - 1].

let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[fruits.length-1] ); // Plum

Есть более короткий синтаксис: fruits.at (-1):

let fruits = ["Apple", "Orange", "Plum"];

То же самое, что и fruits[fruits.length-1]

alert( fruits.at(-1) ); // Plum

Другими словами, arr.at(i):

* это ровно то же самое, что и arr[i], если i >= 0.
* для отрицательных значений i, он отступает от конца массива.

Очередь – один из самых распространённых вариантов применения массива. В области компьютерных наук так называется упорядоченная коллекция элементов, поддерживающая два вида операций:

* push добавляет элемент в конец.
* shift удаляет элемент в начале, сдвигая очередь, так что второй элемент становится первым.

Массивы поддерживают обе операции.

На практике необходимость в этом возникает очень часто. Например, очередь сообщений, которые надо показать на экране.

Существует и другой вариант применения для массивов – структура данных, называемая стек.

Она поддерживает два вида операций:

* push добавляет элемент в конец.
* pop удаляет последний элемент.

Таким образом, новые элементы всегда добавляются или удаляются из «конца».

Примером стека обычно служит колода карт: новые карты кладутся наверх и берутся тоже сверху:

Массивы в JavaScript могут работать и как очередь, и как стек. Мы можем добавлять/удалять элементы как в начало, так и в конец массива.

В компьютерных науках структура данных, делающая это возможным, называется двусторонняя очередь.

**Методы, работающие с концом массива: pop**

Удаляет последний элемент из массива и возвращает его:

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удаляем "Груша" и выводим его

alert( fruits ); // Яблоко, Апельсин

И fruits.pop() и fruits.at(-1) возвращают последний элемент массива, но fruits.pop() также изменяет массив, удаляя его.

push

Добавляет элемент в конец массива:

let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ....

**Методы, работающие с началом массива:**

**shift**

Удаляет из массива первый элемент и возвращает его:

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша

**unshift**

Добавляет элемент в начало массива:

let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу несколько элементов:

let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");

fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]

alert( fruits );

**Внутреннее устройство массива**

Массив – это особый подвид объектов. Квадратные скобки, используемые для того, чтобы получить доступ к свойству arr[0] – это по сути обычный синтаксис доступа по ключу, как obj[key], где в роли obj у нас arr, а в качестве ключа – числовой индекс.

Массивы расширяют объекты, так как предусматривают специальные методы для работы с упорядоченными коллекциями данных, а также свойство length. Но в основе всё равно лежит объект.

Следует помнить, что в JavaScript существует 8 основных типов данных. Массив является объектом и, следовательно, ведёт себя как объект.

Например, копируется по ссылке:

let fruits = ["Банан"]

let arr = fruits; // копируется по ссылке (две переменные ссылаются на один и тот же массив)

alert( arr === fruits ); // true

arr.push("Груша"); // массив меняется по ссылке

alert( fruits ); // Банан, Груша - теперь два элемента

Но то, что действительно делает массивы особенными – это их внутреннее представление. Движок JavaScript старается хранить элементы массива в непрерывной области памяти, один за другим, так, как это показано на иллюстрациях к этой главе. Существуют и другие способы оптимизации, благодаря которым массивы работают очень быстро.

Но все они утратят эффективность, если мы перестанем работать с массивом как с «упорядоченной коллекцией данных» и начнём использовать его как обычный объект.

Например, технически мы можем сделать следующее:

let fruits = []; // создаём массив

fruits[99999] = 5; // создаём свойство с индексом, намного превышающим длину массива

fruits.age = 25; // создаём свойство с произвольным именем

Это возможно, потому что в основе массива лежит объект. Мы можем присвоить ему любые свойства.

Но движок поймёт, что мы работаем с массивом, как с обычным объектом. Способы оптимизации, используемые для массивов, в этом случае не подходят, поэтому они будут отключены и никакой выгоды не принесут.

Варианты неправильного применения массива:

* Добавление нечислового свойства, например: arr.test = 5.
* Создание «дыр», например: добавление arr[0], затем arr[1000] (между ними ничего нет).
* Заполнение массива в обратном порядке, например: arr[1000], arr[999] и т.д.

Массив следует считать особой структурой, позволяющей работать с *упорядоченными данными*. Для этого массивы предоставляют специальные методы. Массивы тщательно настроены в движках JavaScript для работы с однотипными упорядоченными данными, поэтому, пожалуйста, используйте их именно в таких случаях. Если вам нужны произвольные ключи, вполне возможно, лучше подойдёт обычный объект {}.

**Перебор элементов**

Одним из самых старых способов перебора элементов массива является цикл for по цифровым индексам:

let arr = ["Яблоко", "Апельсин", "Груша"];

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

alert( arr[i] );

}

Но для массивов возможен и другой вариант цикла, for..of:

let fruits = ["Яблоко", "Апельсин", "Слива"];

// проходит по значениям

for (let fruit of fruits) {

alert( fruit );

}

Цикл for..of не предоставляет доступа к номеру текущего элемента, только к его значению, но в большинстве случаев этого достаточно. А также это короче.

Технически, так как массив является объектом, можно использовать и вариант for..in:

let arr = ["Яблоко", "Апельсин", "Груша"];

*for (let key in arr) {*

alert( arr[key] ); // Яблоко, Апельсин, Груша

}

Но на самом деле это – плохая идея. Существуют скрытые недостатки этого способа:

1. Цикл for..in выполняет перебор *всех свойств* объекта, а не только цифровых.

В браузере и других программных средах также существуют так называемые «псевдомассивы» – объекты, которые *выглядят, как массив*. То есть, у них есть свойство length и индексы, но они также могут иметь дополнительные нечисловые свойства и методы, которые нам обычно не нужны. Тем не менее, цикл for..in выведет и их. Поэтому, если нам приходится иметь дело с объектами, похожими на массив, такие «лишние» свойства могут стать проблемой.

1. Цикл for..in оптимизирован под произвольные объекты, не массивы, и поэтому в 10-100 раз медленнее. Увеличение скорости выполнения может иметь значение только при возникновении узких мест. Но мы всё же должны представлять разницу.

Свойство length автоматически обновляется при изменении массива. Если быть точными, это не количество элементов массива, а наибольший цифровой индекс плюс один.

Например, единственный элемент, имеющий большой индекс, даёт большую длину:

let fruits = [];

fruits[123] = "Яблоко";

alert( fruits.length ); // 124

Ещё один интересный факт о свойстве length – его можно перезаписать.

Если мы вручную увеличим его, ничего интересного не произойдёт. Зато, если мы уменьшим его, массив станет короче. Этот процесс необратим, как мы можем понять из примера:

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

arr.length = 2; // укорачиваем до двух элементов

alert( arr ); // [1, 2]

arr.length = 5; // возвращаем length как было

alert( arr[3] ); // undefined: значения не восстановились

Таким образом, самый простой способ очистить массив – это arr.length = 0;.

**new Array()**

Существует ещё один вариант синтаксиса для создания массива:

let arr = *new Array*("Яблоко", "Груша", "и тд");

Он редко применяется, так как квадратные скобки [] короче. Кроме того, у него есть хитрая особенность.

Если new Array вызывается с одним аргументом, который представляет собой число, он создаёт массив *без элементов, но с заданной длиной*.

Давайте посмотрим, как можно оказать себе медвежью услугу:

let arr = new Array(2); // создастся ли массив [2]?

alert( arr[0] ); // undefined! нет элементов.

alert( arr.length ); // length 2

Как мы видим, в коде, представленном выше, в new Array(number) все элементы равны undefined.

Чтобы избежать появления таких неожиданных ситуаций, мы обычно используем квадратные скобки, если, конечно, не знаем точно, что по какой-то причине нужен именно Array.

**Многомерные массивы**

Массивы могут содержать элементы, которые тоже являются массивами. Это можно использовать для создания многомерных массивов, например, для хранения матриц:

let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // 5, центральный элемент

[**toString**](https://learn.javascript.ru/array#tostring)

Массивы по-своему реализуют метод toString, который возвращает список элементов, разделённых запятыми.

Например:

let arr = [1, 2, 3];

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

alert( String(arr) === '1,2,3' ); // true

Давайте теперь попробуем следующее:

alert( [] + 1 ); // "1"

alert( [1] + 1 ); // "11"

alert( [1,2] + 1 ); // "1,21"

Массивы не имеют ни Symbol.toPrimitive, ни функционирующего valueOf, они реализуют только преобразование toString, таким образом, здесь [] становится пустой строкой, [1] становится "1", а [1,2] становится "1,2".

Когда бинарный оператор плюс "+" добавляет что-либо к строке, он тоже преобразует это в строку, таким образом:

alert( "" + 1 ); // "1"

alert( "1" + 1 ); // "11"

alert( "1,2" + 1 ); // "1,21"

**Итого**

Массив – это особый тип объекта, предназначенный для работы с упорядоченным набором элементов.

Объявление:

let arr = [item1, item2...];

let arr = new Array(item1, item2...);

Вызов new Array(number) создаёт массив с заданной длиной, но без элементов.

* Свойство length отражает длину массива или, если точнее, его последний цифровой индекс плюс один. Длина корректируется автоматически методами массива.
* Если мы уменьшаем length вручную, массив укорачивается.

Получение элементов:

* Мы можем получить элемент по его индексу, например arr[0].
* Также мы можем использовать метод at(i) для получения элементов с отрицательным индексом, для отрицательных значений i, он отступает от конца массива. В остальном он работает так же, как arr[i], если i >= 0.

Мы можем использовать массив как двустороннюю очередь, используя следующие операции:

* push(...items)добавляет items в конец массива.
* pop() удаляет элемент в конце массива и возвращает его.
* shift() удаляет элемент в начале массива и возвращает его.
* unshift(...items) добавляет items в начало массива.

Чтобы пройтись по элементам массива:

* for (let i=0; i<arr.length; i++) – работает быстрее всего, совместим со старыми браузерами.
* for (let item of arr) – современный синтаксис только для значений элементов (к индексам нет доступа).
* for (let i in arr) – никогда не используйте для массивов!