Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 16**

**Инструкционно-технологическая карта**

Тема: Массивы. Объявление, синтаксис, инициализация.

Цель: Научится писать скрипты с применением массивов.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

Объекты позволяют хранить данные со строковыми ключами. Это замечательно.

Необходимо упорядоченная коллекция данных, в которой присутствуют 1-й, 2-й, 3-й элементы и т.д. Например, она понадобится нам для хранения списка чего-либо: пользователей, товаров, элементов HTML и т.д.

В этом случае использовать объект неудобно, так как он не предоставляет методов управления порядком элементов. Не возможно вставить новое свойство «между» уже существующими. Объекты просто не предназначены для этих целей.

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

## Объявление

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.

Можно получить элемент, указав его номер в квадратных скобках:

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

alert( fruits[0] ); // Яблоко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Можно заменить элемент:

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

…Или 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 строка:

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.

## Методы pop/push, shift/unshift

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

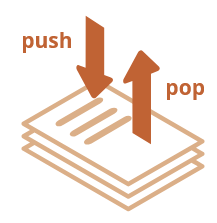
**push** добавляет элемент в конец.

**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. Можно 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

**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. 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 можно 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. Можно 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.

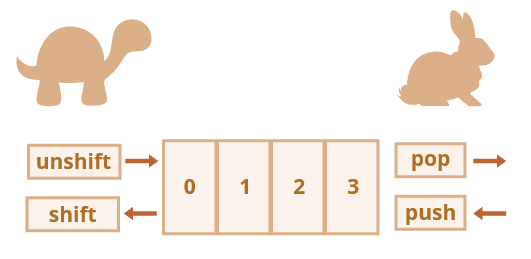
Варианты неправильного применения массива:

* 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 нужны произвольные ключи, вполне возможно, лучше подойдёт обычный объект {}.

## Эффективность

Методы **push/pop** выполняются быстро, а методы **shift/unshift** – медленно.



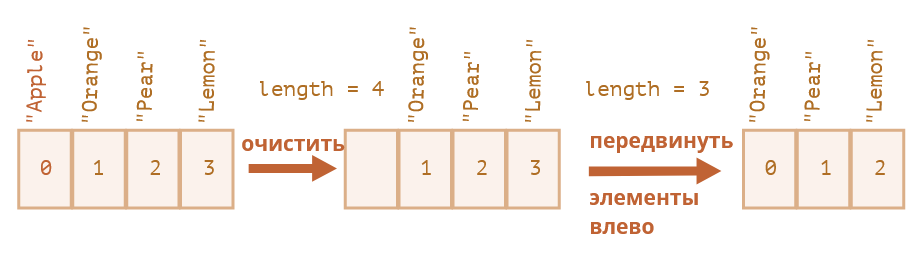
Почему работать с концом массива быстрее, чем с его началом? Посмотрите, что происходит во время выполнения:

fruits.shift(); // удаляем первый элемент с начала

Просто взять и удалить элемент с номером 0 недостаточно. Нужно также заново пронумеровать остальные элементы.

Операция **shift** должна выполнить 3 действия:

1. Удалить элемент с индексом 0.
2. Сдвинуть все элементы влево, заново пронумеровать их, заменив 1 на 0, 2 на 1 и т.д.
3. Обновить свойство **length**.



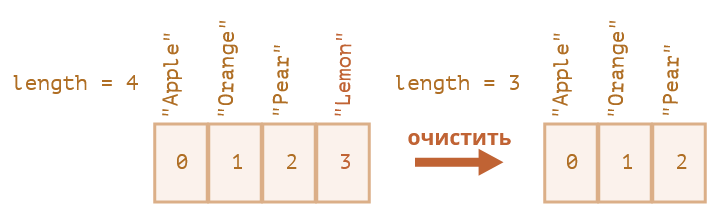
**Чем больше элементов содержит массив, тем больше времени потребуется для того, чтобы их переместить, больше операций с памятью.**

То же самое происходит с **unshift**: чтобы добавить элемент в начало массива, нам нужно сначала сдвинуть существующие элементы вправо, увеличивая их индексы.

Чтобы удалить элемент в конце массива, метод **pop** очищает индекс и уменьшает значение **length**.

Действия при операции **pop**:

fruits.pop(); // удаляем один элемент с конца



**Метод pop не требует перемещения, потому что остальные элементы остаются с теми же индексами. Именно поэтому он выполняется очень быстро.**

Аналогично работает метод **push**.

## 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.
2. 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.
3. 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.

В общем, не следует использовать цикл **for..in** для массивов.

## 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

# Контрольные вопросы

1. Приведите пример объявления массива
2. Какие операции применяются к массиву, позволяя использовать его как двустороннюю очередь?
3. Какой цикл не рекомендуется использовать для перебора элементов массива?
4. Какие бывают виды массивов по размерности?

# Задания

Задание 1.

1. Напишите функцию JavaScript, чтобы проверить, является ли `input` массивом или нет.

Тестовые данные:

console.log (is\_array ('resource'));

console.log (is\_array ([1, 2, 4, 0]));

ложный

правда

1. Напишите функцию JavaScript для клонирования массива.

Тестовые данные:

console.log (array\_Clone ([1, 2, 4, 0]));

console.log (array\_Clone ([1, 2, [4, 0]]));

[1, 2, 4, 0]

[1, 2, [4, 0]]

1. Напишите функцию JavaScript, чтобы получить первый элемент массива. Передача параметра 'n' вернет первые 'n' элементы массива.

Тестовые данные:

console.log (первый ([7, 9, 0, -2]));

console.log (первый ([], 3));

console.log (первый ([7, 9, 0, -2], 3));

console.log (первый ([7, 9, 0, -2], 6));

console.log (первый ([7, 9, 0, -2], - 3));

Ожидаемый результат:

7

[]

[7, 9, 0]

[7, 9, 0, -2]

[]

1. Напишите функцию JavaScript, чтобы получить последний элемент массива. Передача параметра 'n' вернет последние 'n' элементы массива.

Тестовые данные:

console.log (последний ([7, 9, 0, -2]));

console.log (последний ([7, 9, 0, -2], 3));

console.log (последний ([7, 9, 0, -2], 6));

Ожидаемый результат:

-2

[9, 0, -2]

[7, 9, 0, -2]

1. Напишите простую программу на JavaScript, чтобы объединить все элементы следующего массива в строку.

Пример массива: myColor = ["Red", "Green", "White", "Black"];

Ожидаемый результат:

«Красный, зеленый, белый, черный»

«Красный, зеленый, белый, черный»

«Красный + зеленый + белый + черный»

1. Напишите программу на JavaScript для сортировки элементов массива.

Пример массива: var arr1 = [3, 8, 7, 6, 5, -4, 3, 2, 1];

Пример вывода: -4, -3,1,2,3,5,6,7,8

1. Напишите программу на JavaScript, чтобы найти наиболее часто повторяющийся элемент массива.

Пример массива: var arr1 = [3, 'a', 'a', 'a', 2, 3, 'a', 3, 'a', 2, 4, 9, 3];

Пример вывода: а (5 раз)

1. Напишите программу на JavaScript для вычисления суммы и произведения массива целых чисел.
2. Напишите программу на JavaScript для удаления дублирующихся элементов из массива (игнорируйте регистр символов).
3. Напишите программу на JavaScript для выполнения бинарного поиска в отсортированном массиве.

**Задание 2.**

| **№ вар** | **Задача** |
| --- | --- |
|  | Перевести заданное в десятичной системе число a в двоичную систему, используя функцию перевода. |
|  | Даны два натуральных числа. Найти наименьшее общее кратное этих чисел, используя функцию, реализующую алгоритм Евклида. |
|  | Получить все четырехзначные счастливые номера. Счастливым называется номер, у которого сумма первых двух цифр номера равна сумме последних двух цифр. Использовать функцию для расчета суммы цифр двухзначного числа. |
|  | Написать программу определения максимального предельного заряда и потенциала для трех шаров с диаметром r1, r2, r3, используя функции расчета максимального предельного заряда и потенциала. Напряженность поля, при которой начинается ударная ионизация, равна 3 МВ/м. |
|  | Для заданного массива экспериментальных данных рассчитать дисперсию, используя функции определения среднего и дисперсии. |
|  | Получить все шестизначные счастливые номера. Счастливым называется номер, у которого сумма первых трех цифр номера равна сумме последних трех цифр. Использовать функцию для расчета суммы цифр трехзначного числа. |
|  | Даны два натуральных числа. Определить, является хотя бы одно из них палиндромом (число, которое читается одинаково слева направо и справа налево), используя функцию распознавания таких чисел. |
|  | Рассчитать рабочий диапазон длин волн приемника, если емкость конденсатора в его колебательном контуре плавно изменяется от 50 до 500 пФ, а индуктивность катушки постоянна и равна 20 мкГн. Использовать  функцию для определения длины волны |
|  | Отсортировать по убыванию заданный массив чисел, используя функцию сортировки. |
|  | Перевести заданное в двоичной системе число a в десятичную систему. |
|  | Отсортировать по возрастанию заданный массив чисел, исключив отрицательные числа. Использовать функцию сортировки. |
|  | Даны два натуральных числа a и b, обозначающие соответственно числитель и знаменатель дроби. Сократить дробь, используя функцию определения наибольшего общего делителя. |
|  | В заданном массиве целых чисел найти максимальное и минимальное число, использую функции определения максимального и минимального числа. |
|  | Задан одномерный массив из 16 элементов. Сформировать двухмерный массив 4×4, используя функцию преобразования любого одномерного массива в двухмерный массив размерностью n×n с добавлением нулевых элементов. |
|  | Задан одномерный массив из 16 элементов. Сформировать двухмерный массив 4×4, в котором элементы увеличиваются слева направо, используя функции сортировки и преобразования одномерного массива в двухмерный массив. |

# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич