Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал

«Минский радиотехнический колледж»

Учебная дисциплина «Программные средства создания Internet-приложений»

**Инструкция**

по выполнению лабораторной работы

«Работа с массивами в JavaScript. Применение методов работы с массивами.»

Минск

2017

**Лабораторная работа № 18**

**Тема работы: «Работа с массивами в JavaScript. Применение методов работы с массивами»**

**1. Цель работы**

Формирование практических умений работы с массивами и применения методов для работы с ними.

**2. Задание**

Создать HTML-документ используя средства JS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

**Массив** – это группа ячеек памяти, которые объединены общим именем и, как правило, имеют одинаковый тип данных (хотя это и необязательно).

Значения в массиве называются элементами, и каждый элемент характеризуется числовой позицией в массиве, которая называется индексом.

Массивы в языке JavaScript являются нетипизированными: 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.

*Важно понимать различие между терминами «седьмой элемент массива» и «элемент массива под номером семь». Поскольку индексы массива отсчитываются от 0, то «седьмой элемент массива» имеет индекс 6.*

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.

Первый способ создания массива - с помощью литерала, который представляет собой простой список разделенных запятыми элементов массива в квадратных скобках. Значения в литерале массива не обязательно должны быть константами - это могут быть любые выражения, в том числе и литералы объектов:

**var empty = []; // Пустой массив**

**var numbers = [2, 3, 5, 7, 11]; // Массив с пятью числовыми элементами**

**var misc = [ 1.1, true, "a", ]; // 3 элемента разных типов + завершающая запятая**

**var base = 1024;**

**var table = [base, base+1, base+2, base+3]; // Массив с переменными**

**var arrObj = [[1,{x:1, y:2}], [2, {x:3, y:4}]]; // 2 массива внутри, содержащие**

**объекты**

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

Для того чтобы включить в литерал массива неопределенный элемент, достаточно пропустить значение между запятыми. Следующий массив содержит пять элементов, в том числе три неопределенных:

var sparseArray = [1,,,,5];

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

1. вызвать конструктор без аргументов:

var **arr =** new **Array();**

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

1. вызвать конструктор с единственным числовым аргументом, определяющим длину массива:

**var arr = 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.*

1. явно указать в вызове конструктора значения первых двух или более элементов массива или один нечисловой элемент:

**var arr = new Array(5, 4, 3, 2, 1, "тест");**

*В этом случае аргументы конструктора становятся значениями элементов нового массива. Использование литералов массивов практически всегда проще, чем подобное применение конструктора Array(). В общем случае конструктор создаёт массив с элементами, переданными ему в аргументах. Но невозможно с его помощью создать массив с одним целочисленным элементом. Т.к. в этом случае будет создан пустой массив с length, равным единственному переданному аргументу (см п.2).*

Элементами массива могут быть переменные любого типа. Интересная особенность JavaScript - массив может одновременно содержать элементы различных типов, в том числе массивы:

*Пример: массив с элементами разных типов*

**var a = new Array(1.2, "JavaScript", true, { x:1, y:3 });**

Фактически массивы в JavaScript представляют собой экземпляры объекта Array. От обычных объектов массивы отличаются тем, что специальным образом оперируют с числовыми свойствами и свойством length. Чтобы узнать длину массива, используется свойство length.

var myArray = new Array();

...

// узнать количество элементов

alert(myArray.length)

Обратите внимание, что значение Array.length на единицу больше номера последнего элемента массива, т.к. нумерация в массиве начинается с нуля, а свойство length показывает общее количество элементов.

*Вместе с тем, существует распространённое ошибочное мнение, что length равно максимальному индексу в массиве плюс 1. Но это лишь частный случай. В общем случае length строго больше максимального индекса в массиве. Например, мы можем самостоятельно менять length.*

**var a = [1, 2, 3];**

**a.length = 5;**

**alert(a.length + '; ' + [0 in a, 1 in a, 2 in a, 3 in a, 4 in a]); // 5;**

**// true,true,true,false,false**

*Видим, что length равен 5, хотя элементов в массиве всего 3.*

**a.length = 1;**

**alert(a.length + '; ' + [0 in a, 1 in a, 2 in a, 3 in a, 4 in a]); // 1;**

// true,false,false,false,false

*length уменьшилась и все элементы, индексы которых больше или равны length, были удалены.*

В общем случае невозможно узнать, сколько элементов в массиве, не перебирая их. Но в подавляющем большинстве случаев создаваемые массивы не имеют пропусков, и свойство length у них вручную не изменяется, поэтому length равно количеству элементов в массиве.

При добавлении элементов массивы автоматически определяют значение свойства **length.** При удалении элементов length не изменяется.

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.

// Создать массив с одним элементом

var arr = ["world"];

// Прочитать элемент 0

var value = arr[0];

// Записать значение в элемент 1

arr[1] = 3.14;

// Записать значение в элемент 2

i = 2; arr[i] = 3;

// Записать значение в элемент 3

arr[i + 1] = 'привет';

// Прочитать элементы 0 и 2, записать значение в элемент 3

arr[arr[i]] = arr[0];

Массивы в JavaScript не обязательно должны быть "сплошными", т.е. содержать все элементы. При необходимости можно создавать так называемые "разреженные" массивы:

**var myArray = new Array();**

**myArray[0] = 'Питер'**

**// пропустим myArray[1]**

**myArray[2] = 'Гомель'**

**// пропустим myArray[3]**

**myArray[4] = 'Рязань'**

**myArray[5] = 'Псков'**

**for(i = 0; i < 6; i++)**

**alert(myArray[i])**

Добавление элементов производится простой инициализацией соответствующего элемента.

**var myArray = new Array();**

**myArray[0] = 'Питер';**

Добавлять элементы в массив, равно как и удалять их, можно также, как и обычные свойства любых других объектов. С тем лишь отличием, что при добавлении числовых свойств может изменятся свойство length, а при изменении свойства length могут удаляться числовые свойства. В общем случае алгоритм установки свойств у массивов примерно следующий:

1. при добавлении несуществующего числового свойства i, если length меньше или равен i, то length устанавливается равным i + 1;
2. при изменении свойства length:

- если присваиваемое значение меньше 0, то бросается RangeError;

- удаляются все числовые свойства, индексы которых больше и равны новому length.

**var a = [0, 1, 2];**

**alert(a.length); // 3**

**a[5] = 5;**

**alert(a.length); // 6 добавление элемента в крнец массива**

**delete a[5];**

**alert(a.length); // 6: при удалении элементов length не изменяется.**

**a.length = 1;**

**alert([0 in a, 1 in a, 2 in a]); // true,false,false: элементы с индексами 1 и 2 удалены**

**try {**

**a.length = -1;**

**} catch (e) {**

**alert(e.message); // RangeError**

Из того факта, что при установке length удаляются "лишние" элементы, следует способ очистки массива. Если вас по какой-то причине не устраивает присваивание переменной нового пустого массива, а нужно именно обнулить существующий, то достаточно присвоить его свойству length значение 0.

var **a = [0, 1, 2, 3, 4, 5], b = a;**

**// Имеем две переменных, хранящих ссылки на один и тот же массив**

**a = []; // Если сделаем так, то в a и b будут лежать ссылки на разные массивы**

**alert([a.length, b.length]); // 0,6**

var **a = [0, 1, 2, 3, 4, 5], b = a;**

**a.length = 0; // Очистили массив, на который ссылаются значения обеих переменных**

**alert([a.length, b.length]) // 0,0**

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

**var arr = [1,2,'three'];**

**delete arr[2];**

**2 in arr; // false, индекс 2 в массиве не определен**

**arr.length; // 3: оператор delete не изменяет свойство length массива**

*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.*

Кроме того имеется возможность удалять элементы в конце массива простым присваиванием нового значения свойству length.

*Для перебора элементов обычно используется цикл:*

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

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

**alert( arr[i] );**

**}**

Минимально возможный индекс в массиве — 0, максимально возможный — length - 1, поэтому для того, чтобы перебрать все элементы массива, нужно перебрать числа от 0 до length - 1включительно. А для этого существует цикл for.

var **a = [0, 1, 2, 3, 4, 5], sum = 0;**

for **(**var **i = 0; i < a.length; i++) {**

**sum += a[i];**

**}**

**alert(sum); // 15**

Если в массиве нет пропусков, т.е. в большинстве случаев, то такого цикла вполне достаточно. Однако если перебирается разреженный массив, то нужно проверять, существует ли элемент с текущим индексом.

var a = [1, 2, , , 3, , 5];

var props = '';

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

props += a[i] + ' ';

}

alert(props); // 1 2 undefined undefined 3 undefined 5

// Перебрали и несуществующие элементы

props = '';

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

if (i in a) {

props += a[i] + ' ';

}

}

alert(props); // 1 2 3 5

// Перебрали только существующие элементы

// Не забываем, что undefined обманчиво

a[2] = undefined;

props = '';

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

if (i in a) {

props += a[i] + ' ';

}

}

alert(props); // 1 2 undefined 3 5

Наиболее правильной конструкцией для перебора элементов массива является.

for (var i = 0, length = a.length; i < length; i++) {

if (i in a) {

// делаем что-то с элементом массива.

}

}

А чтобы не писать такие конструкции, лучше воспользоваться [методом forEach](http://alljs.ru/articles/array/iterators)

*Пример использования массива:*

**<html>**

**<head>**

**<script>**

**function start ()**

**{**

**var colors = new Array ("cyan", "magenta", "yellow", "black");**

**var integers1 = [2,4,6,8];**

**var integers2 = [2, , ,8];**

**outputArray ("Содержимое массива colors", colors);**

**outputArray ("Содержимое массива integers1", integers1);**

**outputArray ("Содержимое массива integers2", integers2);**

**}**

**function outputArray (header,theArray)**

**{**

**document.writeln ("<H2>"+header+"</H2>");**

**document.writeln ("<TABLE BORDER=’1’ WIDTH=’100%’>");**

**document.writeln ("<TR><TD WIDTH=’100’><B>Индекс</B>"+"<TD><B>Значение</B></TR>");**

**for (var i=0; i<theArray.lenght; i++)**

**document.writeln ("<TR><TD>"+i+"<TD>"+theArray[i]+"</TR>");**

**document.writeln ("</TABLE>");**

**}**

**</script>**

**</head>**

**<body onload= "start()"></body>**

**</html>**

Недостатки этого способа:

1. цикл for..in выведет все свойства объекта, а не только цифровые.

*В браузере, при работе с объектами страницы, встречаются коллекции элементов, которые по виду как массивы, но имеют дополнительные нецифровые свойства. При переборе таких «похожих на массив» коллекций через for..in эти свойства будут выведены, а они как раз не нужны.*

*Бывают и библиотеки, которые предоставляют такие коллекции. Классический for надёжно выведет только цифровые свойства, что обычно и требуется.*

1. цикл for (var i=0; i<arr.length; i++) в современных браузерах выполняется в 10-100 раз быстрее. Казалось бы, по виду он сложнее, но браузер особым образом оптимизирует такие циклы.

В большинстве языков программирования имеется два способа передачи аргументов функциям (или методам): передача по значению и передача по ссылке. В JavaScript числа булевы значения передаются функциям по значениям, а все объекты и массивы передаются функциям по ссылке.

Когда функции передается ссылочный тип данных, сценарий в теле функции, модифицирующий параметр, модифицирует и оригинальное значение в контексте вызова.

*Рассмотрим следующую модификацию функции fiddle:*

**<html>**

**<head>**

**<script type='text/javascript'>**

**function start ()**

**{**

**var x=["одно", "второе", "третье"];**

**function fiddle (argl)**

**{**

**argl[0] = "другое";**

**document.write("Внутри функции fiddle argl = "+argl+"<br />");**

**}**

**document.write("До вызова функции х = "+x+"<br />");**

**fiddle(x);**

**document.write("После вызова функции х = "+x+"<br />");**

**}**

**</script>**

**</head>**

**<body ONLOAD="start ()">**

**</body>**

**</html>**

*В этой ситуации функция fiddle может изменить значения массива, содержащегося в переменной х, как показано ниже:*

*В результате его выполнения получим:*

*До вызова функции x = одно, второе, третье*

*Внутри функции fiddle argl = другое,второе, третье*

*После вызова функции к = другое,второе,третье*

*В этой ситуации функция fiddle может изменить значения массива, содержащегося в переменной х, как показано выше. Это "передача по ссылке" в действии.*

Массивы могут содержать любой тип данных JavaScript, в том числе ссылки на другие массивы или на объекты или функции.

Например:

**document.images[1].width**

Этот код ссылается на свойство width объекта, хранящегося во втором элементе массива, в свою очередь хранящегося в свойстве images объекта document.

JavaScript не поддерживает «настоящие» многомерные массивы. Однако, в JavaScript массивы могут содержать значения любых типов, в том числе и другие массивы. Эту особенность можно использовать для создания так называемых многомерных массивов.

**Многомерный массив**– это массив, значениями которого являются другие массивы:

**var arr = [**

**[1,2,3],**

**[4,5,6],**

**[7,8,9],**

**];**

Для доступа к элементам, которые в качестве значения содержат массив, используются всё те же квадратные скобки. В первых скобках указывается индекс элемента для доступа к внутреннему массиву, так как в качестве значения элемента содержится ещё один массив, то во вторых скобках указывается индекс элемента внутреннего массива:

*Пример:*

**list = new Array()**

**list[0] = newArray("Меню 1", "Меню 2", "Меню 3")**

**alert(list [0][0]);**

*Теперь, вызывая list[0][0], мы получим «Меню 1», вызывая list[0][1] — «Меню 2», и т.д.*

**var arr = [  [1,2,3],  [4,5,6],  [7,8,9],];**

**alert(arr[1][2]);   // 6**

Массивы нескольких размерностей можно создавать, объявляя каждый элемент одномерного массива снова одномерным массивом. Полученные элементы уже двумерного массива можно снова объявить как одномерные массивы, создав, таким образом, трехмерный массив и т. д. Следующий код создает двумерный массив размерности (4 4):

**a = new Array()**

**for (i=0; i < 4; i++) {**

**a[i] = new Array()**

**for (j=0; j < 4; j++) {**

**a[i][j] = значение**

**}**

**}**

Обратите внимание, что при ссылке на элементы такого массива индекс каждой размерности элемента следует задавать в самостоятельных квадратных скобках: сначала в квадратных скобках указать первый индекс, затем в других квадратных скобках второй индекс и т. д.

Для перебора многомерного массива придётся использовать вложенный цикл. Первый цикл будет использоваться для доступа к элементам внешнего массива, а второй – для доступа к элементам вложенных массивов:

**var arr = [**

**[1,2,3],**

**[4,5,6],**

**[7,8,9],**

**];**

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

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

**if (n === arr.length - 1)**

**document.write(arr[j][n]);**

**else**

**document.write(arr[j][n] + ", ");**

**}**

**document.write("<br>");**

**}**

*Следующие элементы главного массива заготавливаем как массивы пунктов подменю:*

**list = new Array();**

**list[0] = newArray("Меню 1", "Меню 2", "Меню 3");**

**list[1] = newArray("Меню 1.1", "Меню 1.2", "Меню 1.3");**

**list[2] = newArray("Меню 2.1", "Меню 2.2");**

**list[3] = newArray("Меню 3.1", "Меню 3.2", "Меню 3.3", "Меню 3.4");**

**Вызываются аналогично: например, list[1][2] («Меню 1.3») или list[2][0] («Меню 2.1»).**

*Теперь формируем список:*

**/\*Первый уровень, первый пункт\*/**

**document.writeln("<ul><li>" + list[0][0] + "</li>");**

**/\*Второй уровень\*/**

**document.writeln("<ul><li>" + list[1][0] + "</li>");**

**document.writeln("<li>" + list[1][1] + "</li>");**

**document.writeln("<li>" + list[1][2] + "</li></ul>");**

**/\*Первый уровень, второй пункт\*/**

**document.writeln("<li>" + list[0][1] + "</li>");**

**/\*Второй уровень\*/**

**document.writeln("<ul><li>" + list[2][0] + "</li>");**

**document.writeln("<li>" + list[2][1] + "</li></ul>");**

**/\*Первый уровень, третий пункт\*/**

**document.writeln("<li>" + list[0][2] + "</li>");**

**/\*Второй уровень\*/**

**document.writeln("<ul><li>" + list[3][0] + "</li>");**

**document.writeln("<li>" + list[3][1] + "</li>");**

**document.writeln("<li>" + list[3][2] + "</li>");**

**document.writeln("<li>" + list[3][3] + "</li></ul></ul>");**

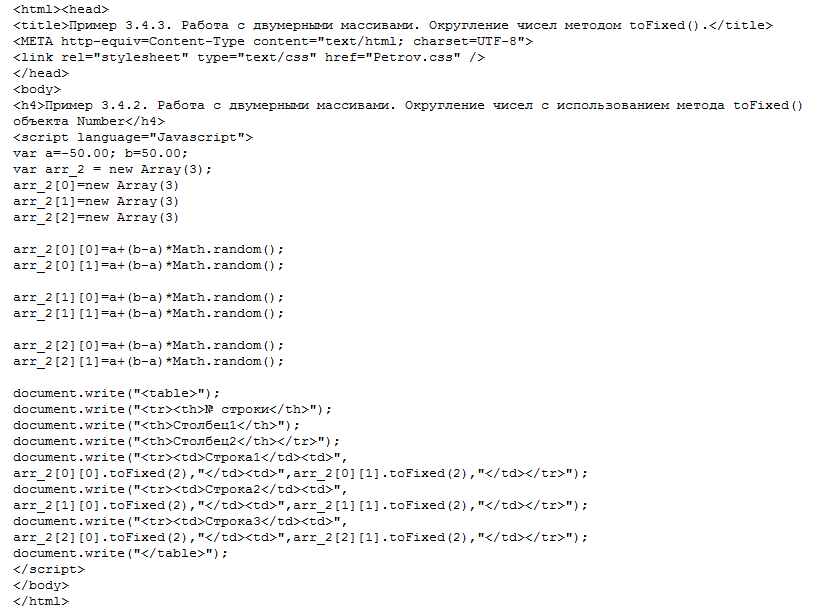
Результат:

* Меню 1
  + Меню 1.1
  + Меню 1.2
  + Меню 1.3
* Меню 2
  + Меню 2.1
  + Меню 2.2
* Меню 3
  + Меню 3.1
  + Меню 3.2
  + Меню 3.3
  + Меню 3.4

В листинге создаётся и выводится двумерный массив размерностью (4х3). Значения элементов массива создаются с помощью датчика случайных чисел Math,random(). Math,random() выдаёт случайное вещественное число, равномерно распределённое на интервале [0,1]. После выполнения выражения a+(b-a)\*Math.random() может быть получено вещественное случайное число на интервале [a,b]. Но поскольку нам нужно получить целые числа, было использовано выражение a+Math.round((b-a)\*Math.random()). В результате мы получаем таблицу случайных целых чисел, равномерно распределённых на интервале [-50,50].

Для округления вещественных чисел с точностью до нужного числа знаков после запятой можно использовать метод toFixed(). Синтаксис метода -  имя\_переменной.toFixed(точность), где точность – параметр, определяющий число знаков после десятичной точки с округлением последней цифры по правилам округления.

В листинге числа вычисляются с точностью до сотых. Точность указывается в качестве параметра метода.



*Объект Array рекомендуется использовать только для массивов с числовыми индексами.*

*Для ассоциативных массивов в javascript применяется [Object](http://javascript.ru/Object).*

*Ассоциативный массив - это массив, у которого в качестве ключей выступают строки. Представить структуру данных такого массива можно как совокупность пар "ключ-значение". Данная структура данных характеризуется тем, что значение связано с ключом, т.е. если вы предоставите массиву некоторый ключ, то он вернёт значение, которое с ним связано.*

*Создание ассоциативных массивов в JavaScript осуществляется на основе типа object (объект). В этом типе данных в качестве ключей можно использовать свойства, а в качестве значений - значения свойств.*

*Несмотря на то, что можно и вручную изменять элементы массива, рекомендуется использовать для этого встроенные методы, что будет гарантировать отсутствие пропусков в массиве и корректное значение свойства length. Корректное в том смысле, что оно будет равно количеству элементов массива.*

Стандарт ECMAScript 3 определяет в составе Array.prototype множество удобных функций для работы с массивами, которые доступны как методы любого массива.

Метод **push** добавляет переданные элементы в конец массива.

var **a = [];**

**a.push(5, 6, 7);**

**alert(a.length + ';' + a); // 3;5,6,7**

**var arr = []; // Создать пустой массив**

**arr.push('zero'); // Добавить значение в конец**

**arr.push('one',2); // Добавить еще два значения**

Метод **pop** возвращает последний элемент массива и удаляет его.

var **a = [];**

**a.push(5, 6, 7);**

**alert(a.pop()); // 7**

**alert(a.length + ';' + a); // 2;5,6**

***Пример:***

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.push("Киви") + "<br />"); //5**

**document.write(fruits.push("Лемон","Ананас") + "<br />"); //7**

**document.write(fruits); //** **Банан,Апельсин, Яблоко,Манго,Киви,Лемон,Ананас**

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.pop() + "<br />"); Манго**

**document.write(fruits + "<br />"); Банан,Апельсин,Яблоко**

**document.write(fruits.pop() + "<br />"); Яблоко**

**document.write(fruits); Банан,Апельсин**

Метод **unshift** добавляет переданные элементы в начало массива, при этом существующие элементы в массиве смещаются в позиции с более высокими индексами. Элементы будут располагаться в том же порядке, в каком они были переданы.

var **a = [1];**

**a.unshift(9, 8, 7);**

**alert(a.length + ';' + a); // 4;9,8,7,1**

***Пример:***

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.unshift("Киви") + "<br />"); //5**

**document.write(fruits.unshift("Лемон","Ананас") + "<br />"); //7**

**document.write(fruits); //** **Лемон,Ананас,**

**Киви,Банан,Апельсин,Яблоко,Манго**

Метод **shift** возвращает первый элемент массива и удаляет его.

var **a = [1];**

**a.unshift(9, 8, 7);**

**alert(a.shift()); // 9**

**alert(a.length + ';' + a); // 3;8,7,1**

***Пример:***

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.shift() + "<br />"); //Банан**

**document.write(fruits + "<br />"); //Апельсин,Яблоко,Манго**

**document.write(fruits.shift() + "<br />"); //Апельсин**

**document.write(fruits); //Яблоко,Манго**

*В Internet Explorer младше 8-й версии unshift возвращает undefined, во всех остальных браузерах — новое значение length. Так что лучше не надеяться на возвращаемое от unshift значение.*

var **a = [];**

**alert(a.unshift(1, 2, 3)); // IE 6,7: undefined**

// IE8 и остальные: 3

Метод **Array.splice()** - это универсальный метод, выполняющий вставку или удаление элементов массива. В отличие от методов slice() и concat(), метод splice() изменяет исходный массив, относительно которого он был вызван.

Метод splice() может удалять элементы из массива, вставлять новые элементы или выполнять обе операции одновременно. Он изменяет значение свойства length и сдвигает элементы массива с более низкими или высокими индексами по мере необходимости, 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. Если первый аргумент отрицательный, то индекс, с которого начнётся удаление, будет равен **«length + первый аргумент»**. 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. За этими аргументами может следовать любое количество дополнительных аргументов, определяющих элементы, которые будут вставлены в массив, начиная с позиции, указанной в первом аргументе.

Метод splice() возвращает массив удаленных элементов или (если ни один из элементов не был удален) пустой массив.

**var arr = [1,2,3,4,5,6,7,8];**

**arr.splice(4); // Вернет [5,6,7,8], arr = [1,2,3,4]**

**arr.splice(1,2); // Вернет [2,3], arr = [1,4]**

**arr.splice(1,1); // Вернет [4]; arr = [1]**

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

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

**var a = [0, 1, 2, 3, 4, 5, 6];**

**// Удалим три средних элемента**

**a.splice(2, 3);**

**alert(a); // 0,1,5,6**

**// Добавим два элемена перед последним**

**a.splice(-1, 0, 7, 8);**

**alert(a); // 0,1,5,7,8,6**

**// Заменим второй и третий элементы на три строки**

**a.splice(1, 2, 'a', 'b', 'c');**

**alert(a); // 0,a,b,c,7,8,6**

*В простейшем случае, если нужно удалить элемент с индексом i, то нужно у массива вызвать метод splice с аргументами i и 1.*

*Вообще говоря, второй аргумент у метода splice необязательный, но поведение функции с одним аргументов отличается в разных браузерах.*

*Поэтому всегда следует передавать в этот метод как минимум два элемента.*

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write("Добавлено: " + fruits.splice(2,0,"Лемон") + "<br />");**

**document.write(fruits); //** **Добавлено: Банан,Апельсин,Лемон,Яблоко,Манго**

*Таким образом с помощью метода****splice****можно удалять элементы из середины массива или добавлять произвольное количество в произвольное место массива.*

Метод **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 arr = [1,2,3];**

**arr.join(); // '1,2,3'**

**arr.join("-"); // '1-2-3'**

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.join() + "<br />");**

**document.write(fruits.join("+") + "<br />");**

**document.write(fruits.join(" и "));**

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 arr = [1,2,3];**

**arr.reverse().join(); // "3,2,1"**

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.reverse());**

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). 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.

**Array.sort(функция\_сортировки);**

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:

**var arr = [33, 4, 1111, 222];**

**arr.sort(); // Алфавитный порядок: 1111, 222, 33, 4**

**arr.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**

**});**

*Сортируем в обратном направлении, от большего к меньшему*

**arr.sort(function(a,b) {return b-a});**

*Обратите внимание, насколько удобно использовать в этом фрагменте неименованную функцию. Функция сравнения используется только здесь, поэтому нет необходимости давать ей имя.*

*По алфавиту:*

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.sort()); // по возрастанию**

**//по возрастанию**

**function sortNumber(a, b)**

**{**

**return a - b;**

**var n = ["10", "5", "40", "25", "100", "1"];**

**document.write(n.sort(sortNumber));**

**// по убыванию**

**function sortNumber(a, b)**

**{**

**return b - a;**

**}**

**var n = ["10", "5", "40", "25", "100", "1"];**

**document.write(n.sort(sortNumber));**

*Рассмотрим небольшой пример:*

**// функция сортировки массива**

**function sortByLength(a, b)**

**{**

**if(a.length > b.length) return 1;**

**else**

**if(a.length = b.length) return 0;**

**else return -1;**

**}**

**// инициализация массива**

**var myArray = new Array()**

**myArray[0] = 'Ярославль'**

**myArray[1] = 'Ямал'**

**myArray[2] = 'Ставрополь'**

**myArray[3] = 'Омск'**

**myArray[4] = 'Анапа'**

**alert('исходный массив:\n\n' + myArray.join('\n'))**

**// сортируем по алфавиту**

**myArray.sort();**

**alert('сортированный по алфавиту:\n\n' + myArray.join('\n'))**

**// сортируем в обратном порядке**

**myArray.reverse();**

**alert('перевёрнутый:\n\n' + myArray.join('\n'))**

**// сортируем по длине с помощью собственной функции**

**myArray.sort(sortByLength);**

**alert('сортированный по длине:\n\n' + myArray.join('\n'))**

*Функция сортировки должна всегда иметь два параметра и возвращать число. Порядок сортировки определяется знаком возвращаемого числа. Если число отрицательное - первый аргумент должен в массиве располагаться раньше второго, если положительное - позже. Если элементы равны, то обычно возвращается ноль, а порядок элементов не изменяется.*

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 arr = [1,2,3];**

**arr.concat(4, 5); // Вернет [1,2,3,4,5]**

**arr.concat([4,5]); // Вернет [1,2,3,4,5]**

**arr.concat([4,5],[6,7]) // Вернет [1,2,3,4,5,6,7]**

**arr.concat(4, [5,[6,7]]) // Вернет [1,2,3,4,5,[6,7]]**

**var parents = ["Руслан", "Людмила"];**

**var children = ["Иван", "Мария"];**

**var family = parents.concat(children);**

**document.write(family);**

**var parents = ["Руслан", "Людмила"];**

**var brothers = ["Антон", "Харитон", "Иван"];**

**var children = ["Демьян", "Мария"];**

**var family = parents.concat(brothers, children);**

**document.write(family);**

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

**arr.slice(0,3); // Вернет [1,2,3]**

**arr.slice(3); // Вернет [4,5]**

**arr.slice(1,-1); // Вернет [2,3,4]**

**arr.slice(-3,-2); // Вернет [3]**

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.slice(0,1) + "<br />");**

**document.write(fruits.slice(1) + "<br />");**

**document.write(fruits.slice(-2) + "<br />");**

**document.write(fruits);**

Метод [**toString()**](http://uroki-javascript.ru/tryit.php?filename=tryjsref_tostring_array) - конвертировать массив в строку.

**var fruits = ["Банан", "Апельсин", "Яблоко", "Манго"];**

**document.write(fruits.toString());**

Заметим, что метод toString() массива (наследуется от встроенного объекта Object и переопределяется в конструкторе String()) , вызываемый по умолчанию тогда, когда надо представить объект-массив в виде строки, по существу вызывает его метод join() без параметра. Например, если необходимо отобразить в документе содержимое массива, то можно использовать следующий оператор:

**document.write("<p>", array1, "</p>");**

Результатом будет новый абзац со значениями массива array1, разделенными запятыми:

**Первый,Второй,Третий**

Кроме того, во всех современных браузерах, кроме Internet Explorer, у массивов есть методы, предназначенные для перебора элементов и выполнения последующих различных действий над ними. Это методы forEach, map, filter, every, some, reduce и reduceRight.

*В IE эти методы отсутствуют, но их можно* [*реализовать*](http://alljs.ru/s/files/Array.js)*, расширив прототип Array (*[*сжатая версия файла*](http://alljs.ru/s/files/Array.pack.js)*).*

Метод [**forEach**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/forEach) используется для перебора массива. Он для каждого элемента массива вызывает функцию callback.

[**arr.forEach(callback[, thisArg])**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/forEach)

Этой функции он передаёт три параметра callback(item, i, arr):

**item**– очередной элемент массива;

**i** – его номер;

**arr** – массив, который перебирается.

*Например:*

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

**arr.forEach(function(item, i, arr) {**

**alert( i + ": " + item + " (массив:" + arr + ")" );**

**});**

Второй, необязательный аргумент forEach позволяет указать контекст this для callback. *Мы обсудим его в деталях чуть позже, сейчас он нам не важен.*

Метод forEach ничего не возвращает, его используют только для перебора, как более «элегантный» вариант, чем обычный цикл for.

Метод [**map**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/map) используется для *трансформации* массива. Он создаёт новый массив, который будет состоять из результатов вызова callback(item, i, arr) для каждого элемента arr.

**arr.map(callback[, thisArg])**

*Например:*

**var names = ['HTML', 'CSS', 'JavaScript'];**

**var nameLengths = names.map(function(name) {**

**return name.length;**

**});**

**// получили массив с длинами**

**alert( nameLengths ); // 4,3,10**

Метод [**filter**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/filter) используется для фильтрации массива через функцию. Он создаёт новый массив, в который войдут только те элементы arr, для которых вызов callback(item, i, arr) возвратит true.

**arr.filter(callback[, thisArg])**

*Например:*

**var arr = [1, -1, 2, -2, 3];**

**var positiveArr = arr.filter(function(number) {**

**return number > 0;**

**});**

**alert( positiveArr ); // 1,2,3**

Методы **every/ some** используются для проверки массива.

Метод [**every**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/every) возвращает true, если вызов callback вернёт true для каждого элемента arr.

**[arr.every(callback[, thisArg])](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/every)**

Метод [**some**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/some) возвращает true, если вызов callback вернёт true для какого-нибудь элемента arr.

[**arr.some(callback[, thisArg])**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/some)

**var arr = [1, -1, 2, -2, 3];**

**function isPositive(number) {**

**return number > 0;**

**}**

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

**alert( arr.some(isPositive) ); // true, есть хоть одно положительное**

Метод [**reduce**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reduce) используется для последовательной обработки каждого элемента массива с сохранением промежуточного результата.

[**arr.reduce(callback[, initialValue])**](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reduce)

*Это один из самых сложных методов для работы с массивами. Но его стоит освоить, потому что временами с его помощью можно в несколько строк решить задачу, которая иначе потребовала бы в разы больше места и времени.*

Метод reduce используется для вычисления на основе массива какого-либо единого значения, иначе говорят «для свёртки массива». Чуть далее мы разберём пример для вычисления суммы.

Он применяет функцию callback по очереди к каждому элементу массива слева направо, сохраняя при этом промежуточный результат.

Аргументы функции:

**callback(previousValue, currentItem, index, arr)**

**previousValue** – последний результат вызова функции, он же «промежуточный результат»;

**currentItem** – текущий элемент массива, элементы перебираются по очереди слева-направо;

**index** – номер текущего элемента;

**arr** – обрабатываемый массив.

Кроме callback, методу можно передать «начальное значение» – аргумент initialValue. Если он есть, то на первом вызове значение previousValue будет равно initialValue, а если у reduce нет второго аргумента, то оно равно первому элементу массива, а перебор начинается со второго.

*Проще всего понять работу метода reduce на примере.*

*Например, в качестве «свёртки» мы хотим получить сумму всех элементов массива.*

*Вот решение в одну строку:*

**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);**

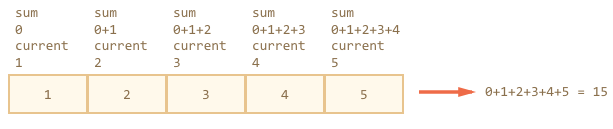
**alert( result ); // 15**

*Разберём, что в нём происходит.*

*При первом запуске sum – исходное значение, с которого начинаются вычисления, равно нулю (второй аргумент reduce).*

*Сначала анонимная функция вызывается с этим начальным значением и первым элементом массива, результат запоминается и передаётся в следующий вызов, уже со вторым аргументом массива, затем новое значение участвует в вычислениях с третьим аргументом и так далее.*

*Поток вычислений получается такой:*

**

*В виде таблицы, где каждая строка – вызов функции на очередном элементе массива:*

|  | **Sum** | **Current** | **результат** |
| --- | --- | --- | --- |
| **первый вызов** | 0 | 1 | 1 |
| **второй вызов** | 1 | 2 | 3 |
| **третий вызов** | 3 | 3 | 6 |
| **четвёртый вызов** | 6 | 4 | 10 |
| **пятый вызов** | 10 | 5 | 15 |

*Как видно, результат предыдущего вызова передаётся в первый аргумент следующего.*

*Кстати, полный набор аргументов функции для reduce включает в себя function(sum, current, i, array), то есть номер текущего вызова i и весь массив arr, но здесь в них нет нужды.*

*Посмотрим, что будет, если не указать initialValue в вызове arr.reduce:*

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

**// убрали 0 в конце**

**var result = arr.reduce(function(sum, current) {**

**return sum + current**

**});**

**alert( result ); // 15**

*Результат – точно такой же! Это потому, что при отсутствии initialValue в качестве первого значения берётся первый элемент массива, а перебор стартует со второго.*

*Таблица в ычислений будет такая же, за вычетом первой строки.*

Метод [reduceRight](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/reduceRight) работает аналогично, но идёт по массиву справа-налево.

**5. Порядок выполнения работы**

1. Создайте массив musicStyles c 3мя произвольными элементами.
2. Добавьте в конец массива новое значение
3. Замените предпоследнее значение на «Блюз». Учтите, что код не должен быть привязан к конкретному количеству элементов.
4. Удалите первое значение и выведите его в alert.
5. Добавьте в начало 2 новых значения.
6. Отсортировать элементы массива по убыванию, без учета регистра.
7. Создавать массив с размерностью (№ по журналу+4). Циклом While вывести элементы с 10 по 1. Для вывода использовать document.write().
8. Написать примеры кода, используя методы .join(Array) и .split(string)
9. Транспанирование матрицы.
10. Найти min и max элементы массива.
11. Выполнить индивидуальное задание.

Варианты индивидуальных заданий

Вариант 1. Найти сумму элементов массива кратных данному k.

Вариант 2. Дан одномерный массив, в котором есть нулевые элементы. Создать массив из номеров этих элементов.

Вариант 3. Дан массив состоящий из строк и чисел. Выяснить, какое число встречается раньше –положительное или отрицательное.

Вариант 4. Дана последовательность действительных чисел. Выяснить, будет ли она возрастающей.

Вариант 5. Дана последовательность натуральных чисел. Создать массив из четных чисел этой последовательности. Если таких чисел нет, то вывести сообщение об этом.

Вариант 6. Дана последовательность целых чисел. Указать наименьшую длину числовой оси, содержащую все эти числа.

Вариант 7. Определить, сколько процентов от всего количества элементов массива составляют четные элементы.

Вариант 8. Дан одномерный массив. Посчитать, сколько элементов массива до первого нулевого элемента.

Вариант 9. В массиве поменять местами наибольший и наименьший элементы.

Вариант 10. Подсчитать в массиве количество отрицательных положительных и нулевых элементов.

Вариант 11. Вывести на экран элементы массива, удовлетворяющие следующему условию M[i]>i.

Вариант 12. Определить, сколько процентов от всего количества элементов массива составляют нулевые элементы.

Вариант 13. В заданном одномерном массиве поменять местами соседние элементы, стоящие на четных местах, с элементами, стоящими на нечетных местах.

Вариант 14. В массиве A записаны оценки экзаменующихся, полученные на первом экзамене. Посчитать, сколько человек не допущено ко второму экзамену.

Вариант 15. Определить, сколько процентов от всего количества элементов массива составляют положительные элементы.

Вариант 16. Определить, сколько элементов в массиве больше k, меньше k и равных k.

Вариант 17. Вычислить среднее арифметическое элементов массива.

Вариант 18. Определить сколько процентов от всего количества элементов массива составляют нечетные элементы.

Вариант 19. Определить, сколько процентов от всего количества элементов массива составляют элементы равные заданному k.

Вариант 20. Вычислить сумму элементов массива, порядковые номера которых совпадают со значениями этого элемента.

Вариант 21. Заменить отрицательные элементы массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество проведенных замен.

Вариант 22. Найти произведение элементов массива, больших заданного M.

Вариант 23. Найти сумму элементов массива, стоящих на четных позициях.

Вариант 24. Найти сумму элементов массива, стоящих на нечетных позициях.

Вариант 25. Определить, сколько процентов от всего количества элементов массива составляют отрицательные элементы.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы и задания**

1. Укажите способы использования кода JavaScript в составе HTML-страницы.

2. На HTML-странице определена следующая форма:

<FORM name="f1">

<INPUT type="text" name="txt"> <br>

<INPUT type="button" name="but"> <br>

</FORM>

Каким образом можно поместить в поле ввода txt некоторый текст, изменить размер данного поля, изменить надпись на кнопке?

3. Как в JavaScript определяются комментарии?

4. Перечислите методы создания стандартных окон JavaScript.

5. Укажите параметры окна, которое открывается при выполнении следующего кода:

msgWindow=window.open("1.html", "displayWindow", "width=300,height=300,location=no, status=yes,toolbar=yes,menubar=no, scrollbars=no,resizable=no")

Как можно загрузить в данное окно документ new.html, используя атрибут TARGET тега <A>? Как можно закрыть данное окно?

**8. Рекомендуемая литература**

**Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. — СПб.: Питер, 2016. — 720с.

**Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. — СПб.: Питер, 2016.

**Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010.

**Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. — СПб.: Питер Пресс, 2017. — 272с.