БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ

Кафедра «Автоматизированные системы управления»

ТЕХНОЛОГИЯ ИНТЕРНЕТ-ПРОГРАММИРОВАНИЯ

Методические указания к лабораторной работе № 6 для студентов специальности
09.03.01 Автоматизированные системы обработки информации и управления

Могилев 2020

Лабораторная работа №6
Обработка массивов

Цель работы: Изучение методов обработки массивов

Порядок выполнения работы.
Изучить теоретические сведения.
Выполнить задание к лабораторной работе в соответствии с вариантом.
Оформить отчет.

Требования к отчету.
Цель работы.
Постановка задачи.
Текст программы

Основные положения

1 Понятие массива и основы работы с массивами

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

Они обычно используются для хранения упорядоченных коллекций данных, например – списка товаров на странице, студентов в группе и т.п.

Объявление

Синтаксис для создания нового массива – квадратные скобки со списком элементов внутри.

Пустой массив:

```
var arr = [];
```

Массив `fruits` с тремя элементами:

```
var fruits = ["Яблоко", "Апельсин", "Слива"];
```

Элементы нумеруются, начиная с нуля.

Чтобы получить нужный элемент из массива – указывается его номер в квадратных скобках:

```
var fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

Элемент можно всегда заменить:

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

…Или добавить:

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

Общее число элементов, хранимых в массиве, содержится в его свойстве `length`:

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

alert( fruits.length ); // 3
```

Через `alert` можно вывести и массив целиком.

При этом его элементы будут перечислены через запятую:

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

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

В массиве может храниться любое число элементов любого типа.

В том числе, строки, числа, объекты, вот например:

```
// микс значений
var arr = [ 1, 'Имя', { name: 'Петя' }, true ];

// получить объект из массива и тут же -- его свойство
alert( arr[2].name ); // Петя
```

2 Конец массива

`pop`

Удаляет *последний* элемент из массива и возвращает его:

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

alert( fruits.pop() ); // удалили "Груша"

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

`push`

Добавляет элемент *в конец* массива:

```
var fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");
```

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

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

3 Начало массива

`shift`

Удаляет из массива *первый* элемент и возвращает его:

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

alert( fruits.shift() ); // удалили Яблоко

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

`unshift`

Добавляет элемент *в начало* массива:

```
var fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

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

Методы `push` и `unshift` могут добавлять сразу по несколько элементов:

```
var fruits = ["Яблоко"];

fruits.push("Апельсин", "Персик");
fruits.unshift("Ананас", "Лимон");

// результат: ["Ананас", "Лимон", "Яблоко", "Апельсин", "Персик"]
alert( fruits );
```

4 Ключи массива

Массив – это объект, где в качестве ключей выбраны цифры, с дополнительными методами и свойством `length`.

Так как это объект, то в функцию он передаётся по ссылке:

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

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

```
alert( arr.length ); // 5
eat(arr);
eat(arr);
alert( arr.length ); // 3, в функцию массив не скопирован, а передана ссылка
```

Ещё одно следствие – можно присваивать в массив любые свойства.

Например:

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

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

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

Но массивы для того и придуманы в JavaScript, чтобы удобно работать именно *с упорядоченными, нумерованными данными*. Для этого в них существуют специальные методы и свойство `length`.

Как правило, нет причин использовать массив как обычный объект, хотя технически это и возможно.

Для перебора элементов обычно используется цикл:

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

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

Не используйте `for..in` для массивов

5 Особенности работы length

Встроенные методы для работы с массивом автоматически обновляют его длину `length`.

Длина `length` – не количество элементов массива, а `последний индекс + 1`.

Так уж оно устроено.

Это легко увидеть на следующем примере:

```
var arr = [];
arr[1000] = true;

alert(arr.length); // 1001
```

Кстати, если у вас элементы массива нумеруются случайно или с большими пропусками, то стоит подумать о том, чтобы использовать обычный объект. Массивы предназначены именно для работы с непрерывной упорядоченной коллекцией элементов.

Используем length для укорачивания массива

Обычно нам не нужно самостоятельно менять `length`... Но есть один фокус, который можно провернуть.

При уменьшении `length` массив укорачивается.

Причем этот процесс необратимый, т.е. даже если потом вернуть `length` обратно – значения не восстановятся:

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

arr.length = 2; // укоротить до 2 элементов
alert( arr ); // [1, 2]

arr.length = 5; // вернуть length обратно, как было
alert( arr[3] ); // undefined: значения не вернулись
```

Самый простой способ очистить массив – это `arr.length=0`.

6 Создание вызовом new Array

new Array()

Существует еще один синтаксис для создания массива:

```
var arr = new Array("Яблоко", "Груша", "и т.п.");
```

Он редко используется, т.к. квадратные скобки `[]` короче.

Кроме того, у него есть одна особенность. Обычно `new Array(элементы, ...)` создаёт массив из данных элементов, но если у него один аргумент-число `new Array(число)`, то он создает массив *без элементов, но с заданной длиной*.

7 Многомерные массивы

Массивы в JavaScript могут содержать в качестве элементов другие массивы. Это можно использовать для создания многомерных массивов, например матриц:

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

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

8 Заполнение случайными числами

Math.random
Синтаксис
var randomNumber = Math.random();

Описание, примеры

Заметим, что числа в JS хранятся в формате с плавающей точкой IEEE 754 с округлением в сторону ближайшего четного. Поэтому иногда, в очень редких случаях, возможна генерация обычно исключенной верхней границы.

Пример: Случайное число от 0(включительно) до 1

```
function getRandom()
{
return Math.random();
}
```

Пример: Случайное число между min и max

```
function getRandomArbitary(min, max)
{
return Math.random() * (max - min) + min;
}
```

Пример: Случайное целое между min и max

```
// использование Math.round() даст неравномерное распределение
function getRandomInt(min, max)
{
return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

9 Функции и замыкания

Функции — это объекты. У них есть свойства (например `length`) и методы (например `toSource`, `apply` и `call`). Функции можно хранить в переменных, передавать и возвращать из других функций:

```
var a = function (..) { ... }; // создаем новую функцию и поменщаем ссылку в a
a(); // вызываем
console.log(a.toString()); // вызваем метод у функции (он вернет ее текст)
```

Функция **при создании** привязывается к набору переменных родительской функции и потому видит ее переменные:

```
function f1() {
```

```
    var a = 1;
    var b = 2;

    function f2() {
        var d = 3;
        var e = 4;

        ...(код 2)..

        return function () {
            var f = 5;
            ...(код 3)...
        };
    }

    ...(код 1)...
}
```

- Код 1 видит переменные a, b и функцию f2 (и f1 тоже)
- Код 2 видит свои переменные d, e, а также родительские a, b, и f2 (и f1 тоже)
- Код 3 видит переменную f, а также d, e, a, b, f2 (и f1 тоже)

То есть внутренняя функции видит переменные внешней функции, которые были в момент ее создания. Это называется замыкание.

Внешняя функция не видит переменных внутренней. Код, находящийся вне функций, в глобальном контексте, не видит внутренние (локальные) переменные.

10 Порядок создания переменных

Локальные переменные (объявленные через var) создаются при входе в функцию, до выполнения ее кода. При этом им изначально присваивается undefined:

```
function test() {
    console.log(a); // undefined

    var a = 2;
    console.log(a); // 2
}

test();
```

Этот код выполняется так:

- создать переменную a и присвоить ей undefined
- выполнить первый console.log
- присвоить a значение 2
- выполнить второй console.log

11 Копирование по значению и по ссылке

Примитивные значения дублируются при копировании, копирование объектов просто копирует ссылку на один и тот же объект. Примитивные значения — это не-объекты, то есть null, undefined, числа, true/false, строки. Если ты их присваиваешь переменной, передаешь или возвращаешь из функции, создается новая независимая копия значения:

```
var a = "Hello";
var b = a; // В b независимая копия строки. Меняя ее, мы не изменяем то, что в a
```

Объекты (а это в том числе массивы (Array), функции (Function), регулярки (RegExp), даты (Date)) копируются и передаются в/из функции по ссылке:

```
var a = { x: 1, y: 2 };
var b = a; // в b ссылка на тот же самый объект, что и в a. Проверим:
b.x = 10;
console.log(a.x); // 10

var c = [];
function changeArray(arr) { arr.push(1); }
changeArray(c); // в функцию передается не копия, а ссылка на тот же массив.
console.log(c); // [1]
```

Задания к лабораторной работе

Все данные вводятся при помощи формы

Вариант 1

Задача 1

Напишите код для вывода alert случайного значения из массива:
var arr = ["Яблоко", "Апельсин", "Груша", "Лимон"];
P.S. Код для генерации случайного целого от min to max включительно:
var rand = min + Math.floor(Math.random() * (max + 1 - min));

Задача 2

Создайте массив. Заполните его случайными значениями от A до B. Значения A, B и размер массива вводятся на форме.

Напишите функцию создания генератора sequence(start, step). Она при вызове возвращает другую функцию-генератор, которая при каждом вызове дает число на 1 больше, и так до бесконечности. Начальное число, с которого начинать отсчет, и шаг, задается при создании генератора. Шаг можно не указывать, тогда он будет равен одному. Начальное значение по умолчанию равно 0. Генераторов можно создать сколько угодно.

Например

```
var generator = sequence(10, 3);
var generator2 = sequence(7, 1);

console.log(generator()); // 10
console.log(generator()); // 13

console.log(generator2()); // 7

console.log(generator()); // 16

console.log(generator2()); // 8
```

Создайте функцию take(gen, x) которая вызывает функцию gen заданное число (x) раз и возвращает массив с результатами вызовов.

```
var gen2 = sequence(0, 2);
console.log(take(gen2, 5)); // [0, 2, 4, 6, 8 ]
```

Вариант 2

Задача 1

Напишите код, который:

- Запрашивает по очереди значения при помощи prompt и сохраняет их в массиве.
- Заканчивает ввод, как только посетитель введёт пустую строку, не число или нажмёт «Отмена».
- При этом ноль 0 не должен заканчивать ввод, это разрешённое число.
- Выводит сумму всех значений массива

Задача 2

Создайте массив. Заполните его случайными значениями от A до B. Значения A, B и размер массива вводятся на форме.

Напишите функцию filt, которая принимает функцию-предикат и массив. Возвращает она массив значений, для которых предикат вернет true.

Например

```
var input = [1, 2, 3, 4, 5, 6];
function isEven(x) { return x % 2 == 0; } // проверяет на четность
console.log(filt(input, isEven)); // [2, 4, 6]
```

Функция не должна изменять исходный массив:

```
console.log(input); // [1, 2, 3, 4, 5, 6]
```

Вариант 3

Задача 1

Создайте функцию find(arr, value), которая ищет в массиве arr значение value и возвращает его номер, если найдено, или -1, если не найдено.

Например:

```
arr = ["test", 2, 1.5, false];

find(arr, "test"); // 0
find(arr, 2); // 1
find(arr, 1.5); // 2

find(arr, 0); // -1
```

Задача 2

Создайте массив. Заполните его случайными значениями от A до B. Значения A, B и размер массива вводятся на форме.

Напишите функцию map(fn, array), которая принимает на вход функцию и массив, и обрабатывает каждый элемент массива этой функцией, возвращая новый массив. Пример:

```
function square(x) { return x * x; } // возведение в квадрат
console.log(map(square, [1, 2, 3, 4])); // [1, 4, 9, 16]
console.log(map(square, [])); // []
```

Функция не должна изменять переданный ей массив: