JavaScript

Урок №12: Основы. Массивы. Методы массивов.

Автор Бахшиллоев Бехзод
Ментор по Frontend в PRO UNITY

ОБЪЕКТЫ

```
let obj = new Object()
let obj = {}

Свойства объектов:
let obj = {
        name: 'John'
}
obj.name = 'John'

Методы объектов(действия, функции)

let obj = {
        sayName: function(){
                        alert('John')
                    }
}
```

МАССИВЫ

Разновидность объектов с элементами по порядку

```
let arr = ['1', {}, [], 25]
arr[0] == '1'
arr[2] == []
```

Методы массивов:
arr.push('a') - добавляет элемент в конец массива
arr.pop() - удаляет последний элемент из массива и возвращает его
arr.shift() - удаляет из массива первый элемент и возвращает его
arr.unshift('a') - добавляет элемент в начало массива
arr.split(s) - превращает строку в массив, s - разделитель
arr.join(s) - превращает массив в строку, s - разделитель
delete arr[1] - удаляет второй элемент
arr.splice(index, count, elem1...) - удалить count элементов, начиная с index и заменить на элементы elem1...
arr.slice(begin, end) - копирует часть массив с begin до end не включая
arr.sort(fn) - сортировка массива. Если не передать функцию сравнения – сортирует элементы как строки.
arr.reverse() - меняет порядок элементов на обратный.
arr.concat(item1...) - создаёт новый массив, в который копируются элементы из arr, а также item1...

Методы перебора

arr.forEach
arr.map
arr.every/some
arr.filter
arr.reduce

Массивы

Объекты позволяют хранить данные со строковыми ключами. Это замечательно.

Но довольно часто мы понимаем, что нам необходима *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. Мы не можем 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 мы можем указать начальные значения элементов:

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

Элементы массива нумеруются, начиная с нуля.

Мы можем 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
```

Другие способности

Мы можем 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
```

Хранение

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.

Получение последних элементов при помощи «at»

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.

Мы можем 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
```

Немного громоздко, не так ли? Нам нужно дважды написать имя переменной.

К счастью, 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
```

Методы массивов

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.

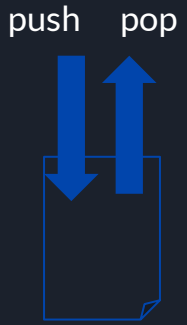
Ещё немного про push, pop

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:



Pop

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] = ...

Shift, unshift

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
```

push, unshift

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 );
```

Задачи:

1. Создать функцию, которая принимает число **n** и возвращает массив, заполненный числами от 1 до **n**: *getArray(10); // [1,2,3,4,5,6,7,8,9,10]*
Данное задание выполните после того как познакомитесь с методами массивов.

2. Создать функцию, которая принимает массив, а возвращает новый массив с дублированными элементами входного массива. **Данное задание выполните после того как познакомитесь с методами массивов**:
doubleArray([1,2,3]) // [1,2,3,1,2,3]

3. Создать функцию, которая принимает произвольное (любое) число массивов и удаляет из каждого массива первый элемент, а возвращает массив из оставшихся значений. **Данное задание выполните после того как познакомитесь с методами массивов**:
changeCollection([1,2,3], [‘a’, ’b’, ‘c’]) → [[2,3], [‘b’, ‘c’]], changeCollection([1,2,3]) → [[2,3]] и т.д.

4. Создать функцию которая принимает массив пользователей, поле на которое хочу проверить и значение на которое хочу проверять. Проверять что все аргументы переданы. Возвращать новый массив с пользователями соответсвующие указанным параметрам.
Данное задание выполните после того как познакомитесь с методами массивов

funcGetUsers(users, “gender”, “male”); // [{name: “Denis”, age: “29”, gender: “male”} , {name: “Ivan”, age: “20”, gender: “male”}]