JavaScript

Урок №16: Основы. Методы перебора 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

Перебирающие методы массивов.

Метод **forEach()** выполняет указанную функцию один раз для каждого элемента в массиве.

Метод **filter()** создаёт новый массив со всеми элементами, ***прошедшими проверку,*** задаваемую в передаваемой функции.

Метод **every()** проверяет, удовлетворяют ли ***все элементы массива условию***, заданному в передаваемой функции.

Метод **some()** проверяет, удовлетворяет ли ***хоть какой-нибудь элемент*** массива условию, заданному в передаваемой функции.

Метод **map()** создаёт новый массив с результатом вызова указанной функции для каждого элемента массива.

Метод **reduce()** применяет функцию к аккумулятору и каждому значению массива (слева-направо), сводя его к одному значению.

Полезные ссылки: Методы

Задачи часть 1:

1. На основе массива [1,2,3,5,8,9,10] сформировать новый массив, каждый элемент которого будет хранить информацию о числе и его четности: *[{digit: 1, odd: true}, {digit: 2, odd: false}, {digit: 3, odd: true}...]*
2. Проверить, содержит ли массив [12, 4, 50, 1, 0, 18, 40] элементы, равные нулю. Если да - вернуть true.
3. Проверить, все элементы массива имеют длину более 3х символов ['yes', 'hello', 'no', 'easycode', 'what']. Если да - вернуть true
4. Дан массив объектов, где каждый объект содержит информацию о букве и месте её положения в строке {буква: “a”, позиция_в_предложении: 1}:

   [{char:"a",index:12}, {char:"w",index:8}, {char:"Y",index:10}, {char:"p",index:3}, {char:"p",index:2},
   {char:"N",index:6}, {char:" ",index:5}, {char:"y",index:4}, {char:"r",index:13}, {char:"H",index:0},
   {char:"e",index:11}, {char:"a",index:1}, {char:" ",index:9}, {char:"!",index:14}, {char:"e",index:7}]

   Напишите функцию, которая из элементов массива соберет и вернёт строку, основываясь на index каждой буквы. Например:
   [{char:"H",index:0}, {char:"i",index: 1}, {char:"!",index:2}] → “Hi!”

Метод Sort.

По умолчанию сортирует элементы массива как строки:
['d', 'a', 'b', 'c'].**sort**() *// ["a", "b", "c", "d"]*
[10, 2, 15].**sort**() *// [10, 15, 2] → т.к '10' < '15' < '2'*

Для того, чтобы отсортировать массив "как надо", метод sort принимает функцию, внутри которой мы определяем как (по какому правилу) нужно отсортировать наша массив.

[...].**sort**(function (prev, next) { return ...})
Функция в методе sort будет выполнена для каждой пары элементов внутри массива (текущий-следующий или предыдущий-следующий). Функция, передаваемая в метод sort может вернуть три возможных значения:
- **отрицательное** - в этом случае prev будет сдвинут в начало массива, а next станет следующим после prev
- **положительное** - в этом случае prev будет сдвинут правее (prev и next поменяются местам)
- **ноль** - оставить элементы без изменений

Полезные ссылки: Sort Быстрая сортировка

Метод Sort.

Для сортировки по числовому значению следующий код можно упростить:

```
[10, 2, 15].sort(function (previous, next) {
    if (previous < next) return -1;
    if (previous > next) return 1;
}) // [2, 10, 15]
⇒
[10, 2, 15].sort(function (previous, next) {
    return previous - next;
}) // [2, 10, 15]
```

Задачи часть 2:

1. Отсортируйте массив массивов так, чтобы вначале располагались наименьшие массивы (размер массива определяется его длиной): *[[14, 45], [1], ['a', 'c', 'd']] → [[1], [14, 45], ['a', 'c', 'd']]*
2. Есть массив объектов:

```
[
    {cpu: 'intel', info: {cores:2, cache: 3}},
    {cpu: 'intel', info: {cores:4, cache: 4}},
    {cpu: 'amd', info: {cores:1, cache: 1}},
    {cpu: 'intel', info: {cores:3, cache: 2}},
    {cpu: 'amd', info: {cores:4, cache: 2}}
]
```

   Отсортировать их по возрастающему количеству ядер (**cores**).

Задачи часть 3:

3. Создать функцию, которая будет принимать массив продуктов и две цены. Функция должна вернуть все продукты, цена которых находится в указанном диапазоне, и сортировать от дешевых к дорогим:

```
let products = [
    {title: 'prod1', price: 5.2}, {title: 'prod2', price: 0.18},
    {title: 'prod3', price: 15}, {title: 'prod4', price: 25},
    {title: 'prod5', price: 18.9}, {title: 'prod6', price: 8},
    {title: 'prod7', price: 19}, {title: 'prod8', price: 63}
];
```

filterCollection(products, 15, 30) → [{...price: 15}, {...price: 18.9}, {...price: 19}, {...price: 25}]