Урок 4.5 – 4.6. Методы массивов, Set, Map, рекурсия

Методы массивов (продолжение)

`filter` - фильтрация массива

Возвращает новый массив, удовлетворяющий условиям.

```
const filteredArray = arr2.filter((item, index, arrray) => {
    // Функция выполняется для каждого элемента
    // В filteredArray попадут те элементы, где функция вернула true.
    return item[2] === 'a';
 // третья буква а
});

// Короткая запись
const filteredArrayShort = arr2.filter(item => item[2] === 'a');
```

`map` - возвращает переформированный массив

```
const objectsList = [
    {
        name: 'Andrew',
        lastName: 'Gulin'
    },
    {
        name: 'Boris',
        lastName: 'Ivanov'
    },
    {
        name: 'Alexey',
        lastName: 'Petrov'
    }
];

const objectsRemapped = objectsList.map((item, index, array) => {
    // Функция выполняется для каждого элемента.
    // Возвращаемое значение будет использоваться
    // вместо текущего значения элемента
    return {
```

```
        ...item, // копируем изначальный объект с name и lastName
        id: index // добавляем новое свойство
    }
});
```

`reduce` - обработка каждого элемента с сохранением результата

```
const arrNumbers = [
    {
        x: 5,
        y: 10
    },
    {
        x: 15,
        y: 20
    },
    {
        x: 39,
        y: 27
    }
];

// Считаем (x+y)*(x+y)*(x+y)
const result = arr2.reduce((accumulator, item, index, array) => {
    accumulator *= (item.x + item.y);
    // Необходимо вернуть новое значение аккумулятора
    return accumulator;
}, 1); // Начальное значение аккумулятора после функции
```

Set

Set - множество значений. По сути, очень сильно перекликается с массивом.

Главное отличие - не может содержать повторяющихся значений.

```
const set = new Set();

set.add('яблоко');
set.add('яблоко');
set.add('банан');
//  в результате в set будет только два значения  - яблоко и банан

set.clear(); // очистить множество
set.delete('банан'); // удалить элемент по значению
```

```
set.size; // количество элементов в множестве


// Пробежаться по значениям множества
for (let entry of set) {
}
```

Map

Раньше для создания объекта с парами ключ-значение использовали объект, но у Map есть несколько преимуществ (именно для пар ключ-значение).Сравнение объектов и мап на MDN

```
const map = new Map();

map.set('Ключ', 'Значение');
map.set('Ключ2', 'Значение');
map.size; // Количество пар ключ-значение
map.get('Ключ'); // Возвращает знаачение с ключом "Ключ"
map.has('Ключ');  // true, если map содержит значение с ключом "Ключ"
map.delete('Ключ'); // Удаляет пару ключ-значение с ключом "Ключ"

// Пробежаться по элементам map
for (let [key, value] of map) {
}

// Другой вариант
for (let entry of map) {
    const key = entry[0];
    const value = entry[1];
}
```

Рекурсия

- Рекурсивная функкия в JavaScript - функция, которая в своём теле использует вызов самой себя
- С рекурсией нужно быть аккуратным: если не задать условие выхода из бесконечного вызова функцией самой себя - программа вылетит из-за превышения лимита стека

- Стек вызовов — это структура данных, которая, говоря упрощённо, записывает сведения о месте в программе, где мы находимся.
- Функции перед выполнением записываются в стек вызовов и удаляются оттуда, когда завершили выполнение
- Однако, вызывая в функции саму себя, текущая функция не завершает исполнение, поэтому с каждым новым вызовом новая функция добавляется в стек.

Пример работы стека вызовов

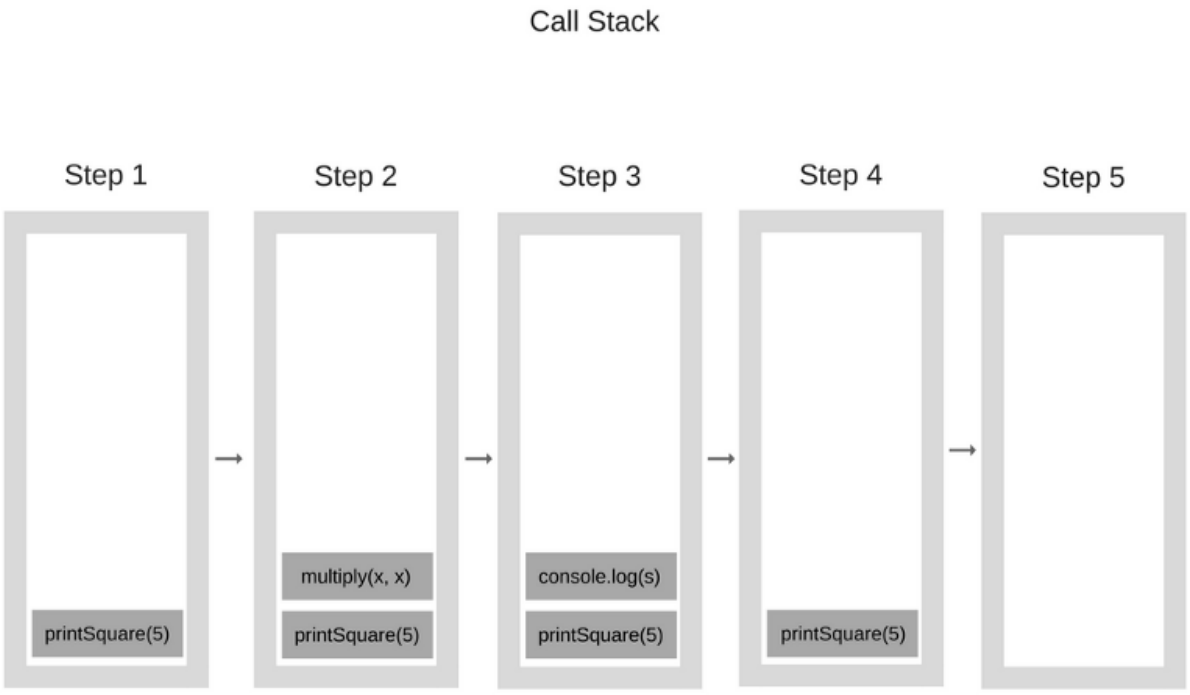
Предположим, есть такой код:

```
function multiply(x, y) {
    return x * y;
}

function printSquare(x) {
    const s = multiply(x, x);
    console.log(s);
}

printSquare(5);
```

Картинка ниже описывает пошагово состояние стека вызовов:

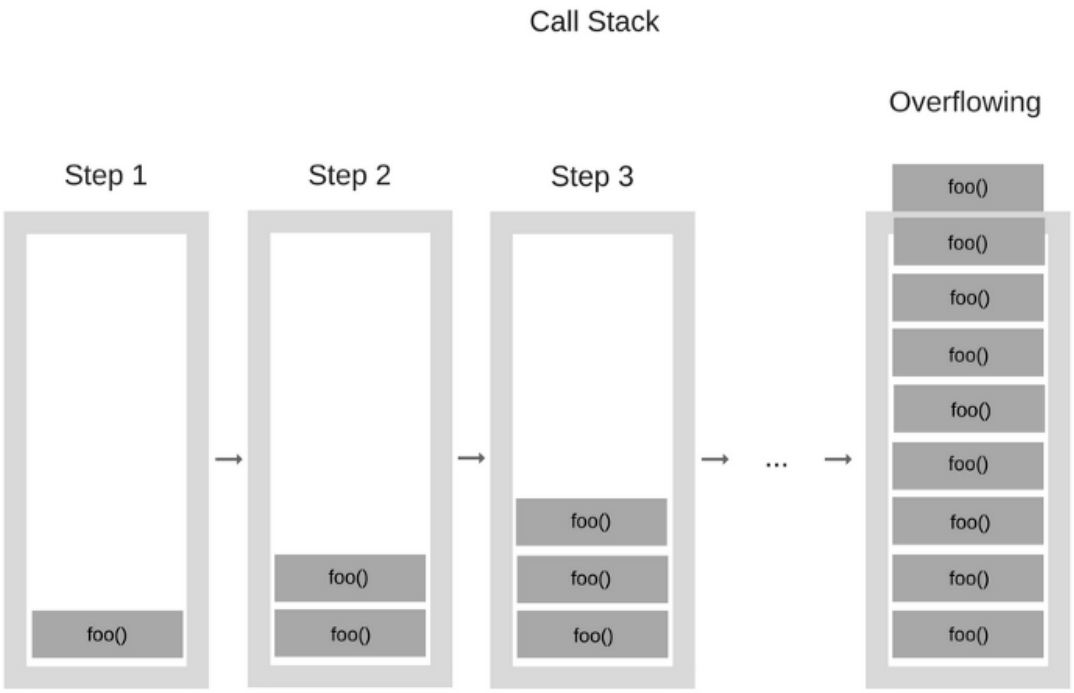


Если будет достигнут максимальный размер стека, произойдёт переполнение, например, в случае рекурсии без условия выхода:

```
function foo() {
    foo();
}

foo();
```

Стек в этом случае выглядит вот так:



Пример рекурсивной функции

Примером рекурсивной функции может выступить функция подсчёта факториала числа.

```
function factorial(n) {
    // обязательно иметь условие выхода.
    // И обязательно сделать так, чтобы вв нужный момент это условие выполнилось.
    // Иначе рекурсия будет бесконечной.
    if (n === 1) {
        return 1;
    }

    return n * factorial(n - 1);
}
```