Modify the Array

Pop -> Премахва елемент от края на array.

let nums = [10, 20, 30, 40, 50, 60, 70];

console.log(nums.length); *// 7*

console.log(nums.pop());  *// 70*

*// [ 10, 20, 30, 40, 50, 60 ]*

Push -> Добавя нов елемент в края на array.

let nums = [10, 20, 30, 40];

console.log(nums.length);  *// 4*

console.log(nums.push(50)); *// 8*

*// [ 10, 20, 30, 40, 50]*

Shift -> Премахва елемент от началото на array.

let nums = [10, 20, 30, 40, 50, 60, 70];

console.log(nums.length); // 7

console.log(nums.shift()); // 10 (removed element)

// [ 20, 30, 40, 50, 60, 70, 80 ]

Unshift -> Добавя елемент от началото на array.

let nums = [40, 50, 60];

console.log(nums.unshift(30));  *// 4 (nums.length)*

console.log(nums.unshift(10,20)); *// 6 (nums.lenghth)*

*// [ 10, 20, 30, 40, 50, 60, 70, 80 ]*

Splice -> Променя съдържанието на array, като

премахва, заменя или добавя нови елементи. (Multifunctional)

let nums = [1, 3, 4, 5, 6];

//splice(1/index/, 0/replaceCount/, 2/numberToReplace/;(3nums)

nums.splice(1, 0, 2); *// inserts at index 1*

console.log(nums);

*// [ 1, 2, 3, 4, 5, 6 ]*

nums.splice(4,1,19); *// replaces 1 element at index 4*

console.log(nums);

*// [ 1, 2, 3, 4, 19, 6 ] (2nums)*  
let el = nums.splice(2,1); *// removes 1 element at index 2*

console.log(nums); *// [ 1, 2, 4, 19, 6 ]*

console.log(el); *// [ 3 ]*

Fill -> Запълва всички елементи на масив от начален индекс до краен индекс със статична стойност.

let arr = [1, 2, 3, 4];  
*// fill with 0 from position 2 until position 4*

*/num/fromIndex/toIndex*

console.log(arr.fill(0, 2, 4)); *// [1, 2, 0, 0]*  
*// fill with 5 from position 1*

console.log(arr.fill(5, 1)); *// [1, 5, 5, 5]*  
console.log(arr.fill(6)); *// [6, 6, 6, 6]*

Reverse -> Първият елемент от масива става последният, а последният елемент от масива става първи.

let arr = [1, 2, 3, 4];

arr.reverse();

console.log(arr); *// [ 4, 3, 2, 1 ]*

Sort -> Сортира елементите на масива и връща сортирания масив. (Сортира по стринг!)

let array1 = [1, 30, 4, 21, 100000];

array1.sort();

console.log(array1); *// [1, 100000, 21, 30, 4]*

*// --------------------*

let array2 = [1, 30, 4, 21, 100000];

array2.sort(compareNumbers);

console.log(array2); *// [ 1, 4, 21, 30, 100000 ]*  
function compareNumbers(a, b) { return a - b; }

negative -> a first ; positive -> b first

* If compareFunction(a, b) returns less than 0, sort a to an index lower than b (i.e. a comes first).
* If compareFunction(a, b) returns 0, leave a and b unchanged with respect to each other, but sorted with respect to all different elements.
* If compareFunction(a, b) returns greater than 0, sort b to an index lower than a (i.e. b comes first).

Sorting Objects -> Обект може да бъде сортиран по value на неговото property.

let items = [

    { name: 'Edward', value: 21 },

    { name: 'Sharpe', value: 37 },

    { name: 'And', value: 45 }

];

*// sort by value*

items.sort(function (a, b) {*// sort by value*

    return a.value - b.value;

});

*// sort by name*

items.sort(function (a, b) {

    let nameA = a.name.toUpperCase(); *// ignore upper and lowercase*

    let nameB = b.name.toUpperCase(); *// ignore upper and lowercase*

    if (nameA < nameB) { return -1; }

    if (nameA > nameB) { return 1; }

    return 0;

});

Accessor Methods

Join -> Създава и връща нов низ чрез обединяване на всички елементи в масив (или обект, подобен на масив), разделен със запетаи или определен низ сепаратор.

let elements = ['Fire', 'Air', 'Water'];

console.log(elements.join()); *// "Fire,Air,Water"*

console.log(elements.join('')); *// "FireAirWater"*

console.log(['Fire'].join(".")); *// Fire*

IndexOf -> Методът indexOf () връща първия индекс, при който даден елемент може да бъде намерен в масива, или -1, ако не присъства.

const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison')); // 1  
// start from index 2

console.log(beasts.indexOf('bison', 2)); // 4  
console.log(beasts.indexOf('giraffe')); // -1

Concat -> Методът concat () се използва за обединяване на два или повече масива

Този метод не променя съществуващите масиви, а вместо това връща нов масив.

const num1 = [1, 2, 3];

const num2 = [4, 5, 6];

const num3 = [7, 8, 9];

const numbers = num1.concat(num2, num3);

console.log(numbers); *//  [1, 2, 3, 4, 5, 6, 7, 8, 9]*

Includes -> Определя дали масивът съдържа определен елемент, връща true или false, както е подходящо.

let arr = ['a', 'b', 'c'];

arr.includes('a'); // true

arr.includes('a', -1); // false (-1 => fromIndex)

Slice -> Методът slice () връща shallow copy на част от масив в нов обект от масив, избран от начало до край.Оригиналният масив няма да бъде променен.

let fruits = ['Banana', 'Orange', 'Lemon', 'Apple', 'Mango'];

let citrus = fruits.slice(1, 3); // До 3 ВКЛЮЧИТЕЛНО! За да го включи.

*// citrus contains ['Orange','Lemon']*

let fruitsCopy = fruits.slice();

*// fruits contains ['Banana', 'Orange', 'Lemon', 'Apple',   
'Mango']*

**Iteration Methods**

ForEach -> Методът forEach () изпълнява предоставена функция веднъж за всеки елемент от масива.

const items = ['item1', 'item2', 'item3'];

const copy = [];

*// For loop*

for (let i = 0; i < items.length; i++) {

  copy.push(items[i]);

}

*// ForEach*

items.forEach(item => { copy.push(item); });

Filter -> Създава нов масив с всички елементи, които преминават теста, реализиран от предоставената функция.

function isBigEnough(value) {

return value >= 10;

};

let filtered = [12, 5, 8, 130, 44].filter(isBigEnough);

*// filtered is [12, 130, 44]*

*/-------------------------------*

let fruits = ['apple', 'banana', 'grapes', 'mango', 'orange'];  
 *// Filter array items based on search criteria (query)*

function filterItems(arr, query) {

  return arr.filter(function(el) {

      return el.toLowerCase().indexOf(query.toLowerCase()) !== -1;

// Ако не намери false -1, ако намери true 1 => 1 !== -1

  });

};  
console.log(filterItems(fruits, 'ap')); *// ['apple', 'grapes']*

Find -> Връща намерената стойност в масива.

let array1 = [5, 12, 8, 130, 44];  
let found = array1.find(function(element) {

  return element > 10;

});  
console.log(found); *// 12*

Some -> Методът some () тества дали поне един елемент в масива преминава теста, реализиран от предоставената функция.Връща (bool) булева стойност.

let array = [1, 2, 3, 4, 5];

let even = function(element) {

  // checks whether an element is even

  return element % 2 === 0;

};

console.log(array.some(even)); //true

Map -> Създава нов масив с резултатите от извикване на предоставена функция на всеки елемент в извикващия масив.

let numbers = [1, 4, 9];

let roots = numbers.map(function(num) {

return Math.sqrt(num)

});

*// roots is now [1, 2, 3]*

*// numbers is still [1, 4, 9]*

*// ----------------------------------*

const myUsers = [

    { name: 'chuloo', likes: 'grilled chicken' },

    { name: 'chris', likes: 'cold beer' },

    { name: 'sam', likes: 'fish biscuits' }

];

const usersByFood = myUsers.map(item => {

    const container = {};  
    container[item.name] = item.likes;

    container.age = item.name.length \* 10;  
    return container;

});

console.log(usersByFood);

*//{ chuloo: 'grilled chicken', age: 60 },*

*//{ chris: 'cold beer', age: 50 },*

*//{ sam: 'fish biscuits', age: 30 }*

Reduce -> Методът reduce () изпълнява функция на reducer върху всеки елемент от масива, което води до една изходна стойност.

Методът приема 2 параметъра :

-Reducer function

-Initial value

const array1 = [1, 2, 3, 4];

const reducer =

(accumulator, currentValue) => accumulator + currentValue;  
console.log(array1.reduce(reducer)); *// 10*

console.log(array1.reduce(reducer, 5)); *// 15*

Reducer Function: (Взема 4 аргумента)

-Accumulator

-Current Value

-Current Index (Optional)

-Source Array (Optional)

**Array of Arrays**

Nested Arrays

Looping Through a Nested Array:

**arr.forEach(printRow);**

**function printRow(row){**

**console.log(row);**

**row.forEach(printNumber);**

**}**

**function printNumber(num){**

**console.log(num);**

**};**

*//-----------------------------------*

*// 3 5 17*

*//-1 7 14*

*// 1-8 89*

**function diagonalSums(input) {**

**let firstDiagonal = 0;**

**let secondDiagonal = 0;**

**let firstIndex = 0;**

**let secondIndex = input[0].length - 1; // start last element in the row**

**input.forEach(array => {**

**firstDiagonal += array[firstIndex++];**

**secondDiagonal += array[secondIndex--];**

**});**

**console.log(firstDiagonal + ' ' + secondDiagonal);**

**}**