# Методы массивов

# concat – объединяет массивы и элементы

const toDoList = ['Посадить дерево', 'Построить дом'];

const toDoListUpdated = toDoList.concat('Вырастить сына');

console.log(toDoListUpdated);

const moscowAttractions = ['Кремль', 'Третьяковская галерея'];

const spbAttractions = ['Эрмитаж', 'Мариинский театр'];

const volgogradAttractions = ['Мамаев Курган', 'Родина-мать'];

const russiaAttractions = moscowAttractions.concat(spbAttractions, volgogradAttractions);

# join – преобразовывает массив в строку

Создаёт строку из элементов массива, разделённых запятой. Их порядок не меняется:

const bremenMusicians = ['Кот', 'Пёс', 'Трубадур', 'Осёл', 'Петух'];

console.log(bremenMusicians.join());

*// "Кот,Пёс,Трубадур,Осёл,Петух"*

console.log(`Представляем музыкантов: ${bremenMusicians.join(', ')}`);

*// "Представляем музыкантов: Кот, Пёс, Трубадур, Осёл, Петух"*

*// при этом с исходным массивом ничего не произошло:*

console.log(bremenMusicians);

*// ["Кот", "Пёс", "Трубадур", "Осёл", "Петух"]*

Мы можем задать любой нужный разделитель:

const bremenMusicians = ['Кот', 'Пёс', 'Трубадур', 'Осёл', 'Петух'];

console.log(bremenMusicians.join(' | '));

*// "Кот | Пёс | Трубадур | Осёл | Петух"*

# push – добавляет переданные ему аргументы в конец массива

const emeraldCityHeroes = ['Лев', 'Дровосек', 'Страшила'];

emeraldCityHeroes.push('Элли', 'Тотошка');

console.log(emeraldCityHeroes);

*// ["Лев", "Дровосек", "Страшила", "Элли", "Тотошка"]*

# pop – удаляет последний элемент массива

Он не принимает аргументы — только возвращает значение удалённого элемента

const insects = ['Бабочка', 'Мотылёк', 'Божья коровка', 'Комар'];

console.log(insects.pop()); *// "Комар"*

console.log(insects); *// ["Бабочка", "Мотылёк", "Божья коровка"]*

# shift – удаляет первый элемент массива (не рекомендуется в практике)

const italyCities = ['Помпеи', 'Рим', 'Неаполь'];

const volcanoEruption = italyCities.shift();

*// метод shift() возвращает удалённый элемент*

console.log(volcanoEruption); *// "Помпеи"*

*// массив остался без первого элемента*

console.log(italyCities); *// ["Рим", "Неаполь"]*

# unshift – добавляет элементы в начало массива (не рекомендуется в практике)

const queue = ['Рабочие', 'Школьники', 'Студенты'];

queue.unshift('Пенсионеры', 'Инвалиды');

console.log(queue);

*// ["Пенсионеры", "Инвалиды", "Рабочие", "Школьники", "Студенты"]*

# slice – копирует часть массива и делает из неё новый массив

принимает на вход два аргумента:

* индекс элемента, с которого нужно начать копирование (включительно);
* индекс элемента, на котором нужно остановиться (не включительно).
* const months = [ 'Январь','Февраль','Март','Апрель','Май','Июнь','Июль','Август', 'Сентябрь','Октябрь','Ноябрь','Декабрь'];
* *// начиная с индекса 2 ("Март") и до, но* ***не включая индекс 5*** *("Июнь")*
* const spring = months.slice(2, 5);
* console.log(spring); *// ["Март", "Апрель", "Май"]*
* console.log(months); */\* ["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"] \*/*
* *// Как видите, исходный массив остался нетронутым*

Второй аргумент можно не указывать. Тогда скопированы будут все элементы до конца.

# splice – удаляет элементы из массива и добавляет на их место новые

Первый аргумент метода — индекс элемента, с которого надо начать удалять. Второй — сколько элементов нужно удалить

const week = ['Понедельник','Вторник','Среда','Четверг','Пятница','Суббота','Воскресенье'];

*// начиная с индекса 0 удалим пять элементов и вставим на их место другие пять*

const removedItems = week.splice(0, 5, 'Воскресенье', 'Суббота', 'Воскресенье', 'Суббота', 'Воскресенье');

console.log(removedItems); *// ["Понедельник", "Вторник", "Среда", "Четверг", "Пятница"]*

console.log(week); *// ["Воскресенье", "Суббота", "Воскресенье", "Суббота", "Воскресенье", "Суббота", "Воскресенье"]*

Можно ничего не удалять — только добавить новые элементы:

const planets = ['Меркурий','Венера','Марс','Юпитер','Сатурн','Уран','Нептун'];

planets.splice(2, 0, "Земля"); *// []*

console.log(planets); *// ["Меркурий", "Венера", "Земля", "Марс", "Юпитер", "Сатурн", "Уран", "Нептун"]*

# forEach – перебирает элементы и для каждого выполняет свой код

Метод  forEach удобно использовать для работы с массивоподобными объектами. Например, с NodeList.

const elements = document.querySelectorAll('.text');

elements.forEach((item) => {

item.classList.add('text\_is-active');

});

Метод forEach — более удобная для работы с массивом версия цикла for.

Но с forEach нельзя использовать директивы continue и break. Поэтому метод не используют, когда нужно прервать выполнение цикла каким-то условием.

# map – создаёт новый массив на основе существующего. В качестве аргумента принимает функцию. Эта функция должна вернуть значение, которое станет элементом нового массива

const firstArr = [0, 1, 2, 3, 4];

const secondArr = firstArr.map(function (item) { *// Берём каждый элемент массива*

return item \* item; *// Возводим каждый элемент в квадрат*

});

});

console.log(secondArr); *// [0, 1, 4, 9, 16]*

Метод map нельзя применить к массивоподобным объектам. Чтобы избежать этой ошибки — нужно превратить массивоподобный объект в полноценный массив c использованием Array.from. Метод map удобно использовать в тех случаях, когда нужно сохранить исходный массив.

const elements = document.querySelectorAll('.text');

const elementsArr = Array.from(elements);

const newArr = elementsArray.map((item) => {

item.classList.add('text\_is-active');

});

# filter – отсеивает элементы массива по заданному признаку

const a = [1, 9, 2, 2, 3, 4, 1, 7, 8, 0, 9, 0, 1, 5, 3];

*// отберём элементы больше 5*

const b = a.filter(function (item) {

return item > 5

});

console.log(b); *// [9, 7, 8, 9]*

# some – проверяет, есть ли в массиве хотя бы один элемент, который соответствует определённому правилу

const oceanResidents = ['Флаундер', 'Немо', 'Губка Боб', 'Аквамен'];

const nemo = oceanResidents.some(function (resident) {

return resident === 'Немо';

});

console.log(nemo); *// true*

*// Функция принимает массив полей*

const hasInvalidInput = (inputList) => {

*// проходим по этому массиву методом some*

return inputList.some((inputElement) => {

*// Если поле не валидно, колбэк вернёт true*

*// Обход массива прекратится и вся фунцкция*

*// hasInvalidInput вернёт true*

return !inputElement.validity.valid;

})

};

# find – единственное отличие: some возвращает булево значение, а find — значение элемента, на котором он завершил проверку правилу

const birds = ['Ворона','Чёрно-белая ворона','Белая ворона','Ворона обыкновенная'];

const includesCrow = birds.some(function (bird) {

return bird.includes('ворона');

});

const crow = birds.find(function (bird) {

return bird.includes('ворона');

});

console.log(includesCrow); *// true*

console.log(crow); *// "Чёрно-белая ворона"*

# every – похож на some, но проверяет все элементы массива и возвращает true, только если все элементы прошли проверку

const jokes = ['смешная шутка и доля правды','не очень смешной анекдот + доля правды', 'доля правды в дурацкой шутке, услышав которую, все улыбнулись из вежливости'];

const allJokesWithTruth = jokes.every(function (joke) {

return joke.indexOf('доля правды') > -1;

});

console.log(allJokesWithTruth); *// true*

# some – проверяет, есть ли в массиве хотя бы один элемент, который соответствует определённому правилу

const oceanResidents = ['Флаундер', 'Немо', 'Губка Боб', 'Аквамен'];

const nemo = oceanResidents.some(function (resident) {

return resident === 'Немо';

});

console.log(nemo); *// true*

# reduce – сводит массив к одному значению

*// Записываем функцию в переменную*

const multiply = function (a, b) {

return a \* b;

};

multiply(2, 3); *// 6*

*// передаём функцию как аргумент*

const result = [1, 2, 3].reduce(multiply);

console.log(result); *// 6 - произведение всех чисел в массиве*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*/\* Создадим массив, хранящий в себе историю выигрышей*

*и проигрышей одного игрока в казино. \*/*

const winsAndLoses = [190, 117, -381, -394, -36, 137, -473, 372, -383];

*/\* Посчитаем, какая сумма денег останется у этого игрока*

*к концу вечера, если вначале у него было с собой 1000. \*/*

const total = winsAndLoses.reduce(function (previousValue, item) {

return previousValue + item;

}, 1000); *// Начальное значение передаём методу reduce как второй аргумент.*

console.log(total); *// 149. Чего ещё ожидать от азартных игр?*

Первый аргумент метода reduce — колбэк. А вот вторым нужно задать промежуточное значение previousValue при первой итерации.

Если как результат нужен объект, тогда методу reduce вторым аргументом передают пустой объект.

const order = ['яблоко', 'банан', 'апельсин', 'банан', 'яблоко', 'банан'];

const result = order.reduce(function (prevVal, item) {

if (!prevVal[item]) {

*// если ключа ещё нет в объекте, значит это первое повторение*

prevVal[item] = 1;

} else {

*// иначе увеличим количество повторений на 1*

prevVal[item] += 1;

}

*// и вернём изменённый объект*

return prevVal;

}, {}); *// Начальное значение — пустой объект.*

console.log(result); *// { яблоко: 2, банан: 3, апельсин: 1 }*

# sort – сортирует массив по возрастанию, убыванию и вообще как угодно

Сортировка по возрастанию .

const myNumbers = [0, 3.14, 2.718, 13];

myNumbers.sort(function (a, b) {

return a - b;

*/\* Если a меньше b, вернётся отрицательное число.*

*Это значит, что a должно расположиться в итоговом массиве*

*раньше, b — позже. Например:*

*a = 0, b = 3.14*

*a - b = -3.14*

*Получили отрицательное число, значит a стоит раньше b \*/*

});

console.log(myNumbers); *// [0, 2.718, 3.14, 13] — так и есть*

# Методы работы с данными

# indexOf - ищет символ в строке и возвращает его индекс

'эспрессо'.indexOf('c'); *// 1*

# includes - проверяет, есть ли в строке набор символов

'Гарри Поттер и узник Азкабана'.includes('Гарри Поттер'); *// true*

# **startsWith** **и endsWith** – сравнивает начало строки с переданной ему подстрокой

'Вендетта'.startsWith('В'); *// true*

'Родина'.startsWith('Картинка в букваре'); *// false*

# toLowerCase и toUpperCase - возвращают новую строку, все символы которой переведены в нижний и верхний регистр соответственно

'Включите Caps Lock'.toLowerCase(); *// "включите caps lock"*

'Выключите Caps Lock'.toUpperCase(); *// "ВЫКЛЮЧИТЕ CAPS LOCK"*

# split – превращает строку в массив

Разделителем может быть любой символ или набор символов:

'Пришёл. Увидел. Победил.'.split(' '); *// ["Пришёл.", "Увидел.", "Победил."]*

'Пришёл. Увидел. Победил.'.split('. '); *// ["Пришёл", "Увидел", "Победил."]*

# slice – возвращает часть строки

На вход он принимает два аргумента — с какого индекса начинать отсчёт и на каком закончить, но второй необязательный – тогда до конца строки

'Не прислоняться'.slice(6, 10); *// "слон"*

'Яндекс.Практикум'.slice(7); *// "Практикум"*

# Math – работает с числами

Разделителем может быть любой символ или набор символов:

*// округляет переданное число «вниз»*

Math.floor(9.99); *// 9*

*// округляет «вверх»*

Math.ceil(9.01); *// 10*

*// округляет до ближайшего целого*

Math.round(9.51); *// 10*

*// возвращает наибольшее из переданных чисел*

Math.max(1, 2, 3, 4, 5); *// 5*

*// возвращает наименьшее из переданных чисел*

Math.min(1, 2, 3, 4, 5); *// 1*

*// возвращает случайное число от 0 включительно до 1 не включительно*

Math.random(); *// 0.31764219954126016*

# parseIntфункция приводит переданный аргумент к целому числу

parseInt('38 попугаев'); *// 38*

Если первый символ передаваемой строки не цифра, parseInt вернёт NaN.

Второй аргумент parseInt — система счисления, в которой число передаётся функции.

parseInt('100', 10); *// 100*

parseInt('100', 2); *// 4 (100 в двоичной системе)*

# parseFloat – функция аналогично parseInt, только выделяет число с плавающей точкой:

parseFloat('36.6'); *// 36.6*

parseFloat('36.6 нормальная температура человека '); *// 36.6*

# Number.isInteger – принимает число как аргумент и проверяет, целое оно или дробное

const eightAndAHalf = 8.5;

Number.isInteger(eightAndAHalf); *// false*

Number.isInteger(Math.floor(eightAndAHalf)); *// true*

# Объекты

# deleteоператор удаления ключа из объекта

В этом случае удалится и сам ключ one.

const obj = { one: 1 };

console.log(obj.one); *// 1*

delete obj.one;

console.log(obj.one); *// undefined*

# inоператор проверки наличия ключа в объекте

Это полезно для условных конструкций:

Функция должна возвращать число — количество языков из переданного массива, которые есть в нашем словаре:

const welcomeMessages = {

russian: 'Добро пожаловать',

english: 'Welcome',

french: 'Bienvenue',

italian: 'Benvenuto',

spanish: 'bienvenido',

chinese: '歡迎',

finnish: 'Tervetuloa'

};

const obj = { one: 1 };

function countLanguages(obj, propsArr) {

return propsArr.reduce(function (res, current) {

if (current in obj) {

res += 1;

}

return res;

}, 0);

}

console.log(countLanguages(welcomeMessages, ['english', 'french', 'mandarin'])); *// 2*

console.log(countLanguages(welcomeMessages, ['russian', 'czech'])); *// 1*

# Цикл for...inперебор свойств

В этом случае удалится и сам ключ one.

const emojis = {smile: '😃', kiss: '😚', smirk: '😏', disappointment: '😞',astonishment: '😲',dizziness: '😵'};

for (let emotion in emojis) {

console.log(`${emotion} - ${emojis[emotion]}`);

}

*/\* в консоли появится: smile — 😃 kiss — 😚 smirk — 😏 disappointment — 😞 astonishment — 😲 dizziness — 😵 \*/*

# Object.keys получить полный список ключей в массив и дальше перебирать уже изученными способами, например forEach.

const expenses = {'руп': 'на суп','трёшку': 'на картошку','пятёрку': 'на тетёрку', 'десятку': 'на куропатку','сотку': 'на \*одку'};

Object.keys(expenses).forEach(function (a) {

console.log(a);

});

*/\* в консоли выводятся ключи: руп трёшку пятёрку десятку сотку\*/*

# Object.entries и Object.values Метод Object.values возвращает значения всех свойств и методов объекта, а Object.entries — все пары «ключ-значение»

Будьте аккуратны с этими методами: их не поддерживает даже последняя версия Internet Explorer

const paulMcCartney = {

yesterday: 'all my troubles seemed so far away'

};

console.log( Object.values(paulMcCartney) ); *// ["all my troubles seemed so far away"]*

console.log( Object.entries(paulMcCartney) ); *// [["yesterday", "all my troubles seemed so far away"]]*

# Object.assign создает поверхностную копию объекта

const firstObj = {

one: 1,

two: 2

};

const secondObj = Object.assign({}, firstObj);

console.log(secondObj === firstObj); *// false*

# Глубокая копия объекта

const original = {

one: 1,

two: 2,

three: { message: 'I love JS' }

};

const copy = Object.assign({}, original);

copy.three = Object.assign({}, original.three);

*// копия*

console.log(copy);

*// { one: 1, two: 2, three: { message: "I love JS" } }*

*// копия имеет общие корни с оригиналом?*

console.log(copy === original); *// false*

*// свойство three копии имеет общие корни*

*// со свойством three оригинала?*

console.log(copy.three === original.three); *// false*

# Array.isArray проверка объекта на массив

const arr = [1, 2, 3];

const obj = {};

console.log(Array.isArray(arr)); *// true*

console.log(Array.isArray(obj)); *// false*

# Object.assign копирование массива

const boringTale = ['У попа была собака, он её любил,','Она съела кусок мяса, поп её'];

const boringTale2 = Object.assign([], boringTale);

console.log(boringTale === boringTale2); *// false.*

*// Это два разных массива, но с одинаковым набором элементов.*

Более простой способ скопировать массив — вызвать метод slice, не передавая тому никаких аргументов:

const boringTale = ['У попа была собака, он её любил,','Она съела кусок мяса, поп её'];

const boringTale2 = boringTale.slice();

boringTale === boringTale2; *// false*