**Длина length – не количество элементов массива, а последний индекс + 1**.

var arr = ["Я", "изучаю", "JavaScript"];

arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент

alert( arr ); // осталось ["Я", "JavaScript"]

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 3 первых элемента и добавить другие вместо них

arr.splice(0, 3, "Мы", "изучаем")

alert( arr ) // теперь ["Мы", "изучаем", "JavaScript"]

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 2 первых элемента

var removed = arr.splice(0, 2);

alert( removed );

var arr = ["Я", "изучаю", "JavaScript"];

// с позиции 2

// удалить 0

// вставить "сложный", "язык"

arr.splice(2, 0, "сложный", "язык");

alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"

var arr = [1, 2, 5]

// начиная с позиции индексом -1 (перед последним элементом)

// удалить 0 элементов,

// затем вставить числа 3 и 4

arr.splice(-1, 0, 3, 4);

alert( arr ); // результат: 1,2,3,4,5

SLICE(он лишь вырезает, но не изменяет!!!)

var arr = ["Почему", "надо", "учить", "JavaScript"];

var arr2 = arr.slice(1, 3); // элементы 1, 2 (не включая 3)

alert( arr2 );

var arr = ["Почему", "надо", "учить", "JavaScript"];

alert( arr.slice(1) ); // взять все элементы, начиная с номера 1

var arr2 = arr.slice(-2); // копировать от 2-го элемента с конца и дальше

var fullCopy = arr.slice();

**EVERY/SOME**

Эти методы используются для проверки массива.

* Метод [«arr.every(callback[, thisArg])»](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/every) возвращает true, если вызов callback вернёт true для каждогоэлемента arr.
* Метод [«arr.some(callback[, thisArg])»](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/some) возвращает true, если вызов callback вернёт true для какого-нибудь элемента arr.

var arr = [1, -1, 2, -2, 3];

function isPositive(number) {

return number > 0;

}

alert( arr.every(isPositive) ); // false, не все положительные

alert( arr.some(isPositive) ); // true, есть хоть одно положительное

**LET/CONST**

Переменные let:

* Видны только после объявления и только в текущем блоке.
* Нельзя переобъявлять (в том же блоке).
* При объявлении переменной в цикле for(let …) – она видна только в этом цикле. Причём каждой итерации соответствует своя переменная let.

Переменная const – это константа, в остальном – как let.

**Классы тоже не хоистятся!!!!!**

**Array.isArray(param) возвращает true,если объект-массив**

// Все следующие вызовы вернут true

Array.isArray([]);

Array.isArray([1]);

Array.isArray(new Array());

// Небольшой факт: Array.prototype сам является массивом:

Array.isArray(Array.prototype);

// Все следующие вызовы вернут false

Array.isArray();

Array.isArray({});

Array.isArray(null);

Array.isArray(undefined);

Array.isArray(17);

Array.isArray("Array");

Array.isArray(true);

Array.isArray(false);

Array.isArray({ \_\_proto\_\_: Array.prototype });

**-**Операции удаления в массиве будут возвращать эти самые элементы,а вот добавления - уже длину нового массива.

-В reduce& reduceRight в callback будут передаваться 4 аргумента.т.к там есть еще и аккумулятор, а в map-3.

-Array.fill(чем заполнять, откуда, до какого)

-Sort возвращает массив!!!

-Метод **Object.assign()** используется для копирования значений всех собственных перечисляемых свойств из одного или более исходных объектов в целевой объект. После копирования он возвращает целевой объект.

-**Object.entries()** возвращает массив, элементами которого являются массивы, соответсвующие перечисляемому свойству пары [key, value], найденной прямо в object. Порядок свойств тот же, что и при прохождении циклом по свойствам объекта вручную.(двумерный массив)

-Метод **Object.freeze()** замораживает объект: это значит, что он предотвращает добавление новых свойств к объекту, удаление старых свойств из объекта и изменение существующих свойств или значения их атрибутов перечисляемости, настраиваемости и записываемости. В сущности, объект становится эффективно неизменным. Метод возвращает замороженный объект.

-bind в отличие от call и apply не возращает выполнение функции,а возвращает новую функцию с привязкой к установленному this.

- Операторы равенства (== и ===) не могут использоваться для проверки значения на равенство NaN. Вместо них используйте функции [Number.isNaN()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Number/isNaN" \o "Метод Number.isNaN() определяет, является ли переданное значение NaN. Это более надёжная версия оригинальной глобальной функции isNaN().) или [isNaN()](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/isNaN" \o "Функция isNaN() определяет является ли литерал или переменная нечисловым значением (NaN) или нет. При работе с функцией необходимо проявлять осторожность так как она имеет свои особенности. В качестве альтернативы можно использовать метод Number.isNaN() из ECMAScript 6, или дополнительно проверять литерал или переменную на нечисловое значение при помощи typeof.).

NaN === NaN; // false

Number.NaN === NaN; // false

isNaN(NaN); // true

isNaN(Number.NaN); // true

**-**два массива выведут true через == если у тебя arr2=arr1,т.е. ты передал ссылку.

**You don’t know JS**

- Поднимаются только до ближайшего скоупа!!!

- Сперва поднимаются функции, а потом уже переменные

- bind слабее new.

-Метод **Object.create()** создаёт новый объект с указанными объектом прототипа и свойствами.

Примитивы

Number

String

Boolean

Null

Undefined

Object- объект

Typeof null – object

Typeof function – function

This

This – это привязка во время вызова функции к какому-либо объекту(динамически) и говорит нам, с каким объектом в качество вот этого this мы должны работать.

Всплытие

<https://learn.javascript.ru/event-bubbling>

Делегирование

<https://learn.javascript.ru/event-delegation>