4) Массивы:

ES5:

Создание

/\* обычный синтаксис \*/

var array = [ elem0, elem1, elem2, ... ]

/\* Синтаксис с new Array() \*/

var array = new Array( elem0, elem1, elem2, ...)

/\* Редкий синтаксис: аргумент new Array - одно число.При этом создается массив заданной длины,все значения в котором undefined \*/

var array = new Array(10)

Аргументы

elem0, elem1, ...

Если ни один из параметров elem0, elem1, ... не указан, то инициализируется пустой массив.

Если указан лишь один числовой параметр elem0 и массив объявлен через new Array, то будет инициализирован пустой массив длины, равной данному числу. При этом все элементы являются undefined.

В ином случае создается массив из перечисленных элементов.

Массив пронумерованных элементов, также может служить стеком или очередью. Для ассоциативных массивов используйте Object.

Методы

* Push- добавляет элементы, начиная с текущей длины [length](http://javascript.ru/Array/length) и возвращает новую, увеличенную длину массива.
* Slice-Данный метод не изменяет исходный массив, а просто возвращает его часть.

var arr = [ 1, 2, 3, 4, 5 ]

arr.slice(2) // => [3, 4, 5]

* Sort-Данный метод изменяет исходный массив. Получившийся массив также возвращается в качестве результата.
* Splice- объединяет в себе две разные функциональности: удаляет часть массива и добавляет новые элементы на место удаленных.
* и другие

Свойства

* index
* input
* length

ES6:

Массив приобрёл собственные методы

Array.from(arrayLike, mapFunc, thisArg)

Array.from()базовая функции для преобразования двух типов объектов, массивов:

* Array-like objects-имеют собственную длину и индексированные элементы
* Iterable objects- возможность получить 1 элемент за раз.

Следующий код является примером преобразования массива-как объект в массив:

**let** lis = document.querySelectorAll('ul.fancy li');

Array.from(lis).forEach(**function** (li) {

console.log(node);

});

Результатом querySelectorAll () не является массив и не имеет FOREACH (метод), и именно поэтому мы должны преобразовать его в массив прежде чем мы сможем использовать этот метод.

Array.from позволяет контролировать конструктор используя его результат.

**class** MyArray extends Array {

...

}

**let** instanceOfMyArray = MyArray.from(anIterable);

Параметр mapFunc делает Array.from() вариантом Array.prototype.map().

класс MyArray расширяет Массив{

...

}

console.log(MyArray.of(3, 11, 8) instanceof MyArray); // верно

console.log(MyArray.of(3).length === 1); // верно

**Методы перебора массивов:**

Array.prototype.entries()

Array.prototype.keys()

Array.prototype.values()

Результат каждого из перечисленных методов является последовательность значений, но они не будут возвращены в виде массива; они раскрываются поочередно, через итератор.

> Array.from([ 'a', 'b' ].keys())

[ 0, 1 ]

> Array.from([ 'a', 'b' ].values())

[ 'a', 'b' ]

> Array.from([ 'a', 'b' ].entries())

[ [ 0, 'a' ],

[ 1, 'b' ] ]

Array.prototype.find(predicate, thisArg)

возвращает первый элемент массива, для которых предикат обратного вызова возвращает true. Если нет ни одного такого элемента, он возвращает undefined.

Пример:

> [6, -5, 8].find(x => x < 0)

-5

> [6, 5, 8].find(x => x < 0)

undefined

Array.prototype.fill(value, start, end)

Заполняет массив с заданным значение:

> ['a', 'b', 'c'].fill(7)

[ 7, 7, 7 ]

> new Array(3).fill(7)

[ 7, 7, 7 ]

При необходимости можно ограничить, где заполнение начинается и заканчивается:

> ['a', 'b', 'c'].fill(7, 1, 2)

[ 'a', 7 'c' ]

Литература:

<http://javascript.ru/Array/index>

<http://www.2ality.com/2014/05/es6-array-methods.html>