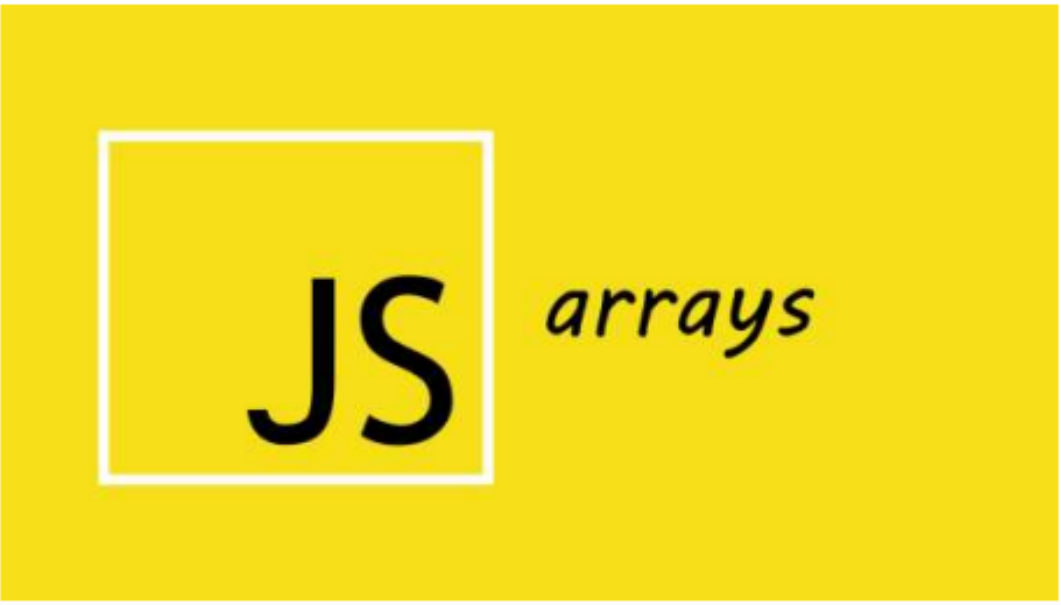
Лабораторная работа №7. Массивы в JavaScript



Содержание:

На этом занятии мы познакомимся с массивами, научимся их создавать, выполнять операции над их элементами, а также рассмотрим основные методы и свойства, доступные при работе с ними.

Что такое массив?

Массив – это специальная структура данных, которая предназначена для хранения упорядоченных коллекций значений.

Массивы очень популярная конструкция и довольно часто используются в коде, т.к. позволяют нам хранить несколько значений в одной переменной.

Например, вместо того чтобы использовать 5 переменных можно объявить одну со всеми этими значениями:

```
const ocean1 = 'Атлантический';
const ocean2 = 'Индийский';
const ocean3 = 'Тихий';
const ocean4 = 'Северный Ледовитый';
const ocean5 = 'Южный';
// вместо них массив
```

```
const oceans = ['Атлантический', 'Индийский', 'Тихий', 'Северный Ледовитый', 'Южный'];
```

Каждое значение в массиве имеет свой порядковый номер (индекс). Значения называются элементами. Первый элемент массива имеет индекс 0, второй – 1, третий – 2 и т.д.

На следующем рисунке показан массив, состоящий из 5 элементов: 123, 7, 50, -9, 24.

Массив, состоящий из 5 элементов

индексы элементов массива

При этом необязательно, чтобы все элементы массива имели один и тот же тип данных. Элементами могут быть любые значения – даже другие массивы. Это позволяет создавать сложные структуры данных, например, такие как массивы объектов или массивы массивов.

Создание массива

Создавать новые массивы в JavaScript можно двумя способами:

- с использованием литерала, т.е. посредством квадратных скобок `[]` и перечислением в них элементов через запятую;
- путем создания нового экземпляра класса `Array`.

Пример создания пустого массива:

```
// посредством литерала массива
const arr = [];
// с использованием конструктора Array()
const otherArr = new Array();
```

Метод создания массива с помощью литерала (квадратных скобок) более предпочтителен и в большинстве случаев лучше использовать именно его.

При объявлении массива в нём можно сразу создать элементы. Для этого внутрь скобок необходимо поместить элементы, отделив их друг от друга запятой:

```
const numArr = [3, -5, 9, 1, 21];
// с помощью Array()
// const numArr = new Array(3, -5, 9, 1, 21);
```

Внимание! Если конструктору `Array()` передать один аргумент, который является числом, то он создаст массив с указанным количеством элементов. Значения элементов при этом будут неопределёнными (пустыми):

```
const arr = new Array(3); // [ , , ]
```

При создании массивов в конец разрешается вставлять необязательную завершающую запятую:

```
const coffee = [
  Lavazza,
  Nescafe,
  Jardin,
];
```

Доступ к элементам

Получение доступа к элементу массива выполняется через квадратные скобки, внутрь которых нужно поместить индекс.

Так как индексы нумеруются с 0, то для получение первого, второго и третьего элемента нужно использовать индексы 0, 1 и 2.

```
const colors = ['black', 'white', 'grey'];
console.log( colors[0] ); // 'black'
console.log( colors[1] ); // 'white'
console.log( colors[2] ); // 'grey'
```

При попытке получить доступ к несуществующему элементу возвращается `undefined`:

```
console.log( colors[3] ); // undefined
```

Чтобы изменить элемент, ему нужно просто присвоить новое значение:

```
colors[1] = 'yellow'; // ['black', 'yellow', 'grey']
```

Сейчас в переменной `$colors` три элемента. Для добавления нового элемента в массив, можно просто присвоить нужное значение следующему индексу:

```
colors[3] = 'red'; // ['black', 'yellow', 'grey', 'red']
```

Если при добавлении элемента случайно пропустить индекс, то в массиве появится неопределенный элемент:

```
colors[5] = 'green'; // ['black', 'yellow', 'grey', 'red', , 'green']
console.log( colors[4] ); // undefined
```

Определить количество элементов в массиве можно с помощью свойства `length`:

```
console.log( colors.length ); // 6
```

Зная количество, получить последний элемент можно так:

```
const lastIndex = colors.length - 1;
console.log( colors[lastIndex] ); // "green"
```

Пример массива, элементы которого содержат различные типы данных:

```
const arr = [5, {name: 'Василий', age: 35}, [7, 54, 2], true, function() {
console.log('good'); }];

// получим элемент с индексом 1, который является объектом, а затем значение его свойства age
console.log( arr[1].age ); // 35
// получим элемент с индексом 2, который является массивом, а затем его элемент с индексом 1
console.log( arr[4][1] ); // 35
// получим элемент с индексом 4, т.е. функцию и выполним её
arr[2](); // "good"
```

В качестве элементов массива можно использовать выражения:

```
const ratio = 5;
const point = [3 * ration, 7 * ratio];
```

Массивы являются объектами

Массивы в JavaScript не являются каким-то определённым типом данных. Это объект, у которого прототипом является `Array`.

Например, получим с помощью оператора `typeof` тип данных:

```
const arr = ['One', 'Two', 'Three'];
console.log(typeof arr); // "object"
```

А так как массивы являются объектами, то при его копировании, передаётся не он сам, а ссылка на него:

```
const names = ['Иван', 'Вася'];
const copyNames = names;
copyNames[1] = 'Петр';
console.log(names); // ['Иван', 'Петр']
```

Создать копию массива можно следующими способами:

```
const names = ['Иван', 'Вася'];
// клонирование массива с помощью slice
const cloneNames1 = names.slice();
// с помощью оператора spread
const cloneNames2 = [...names];
// с помощью Array.from
const cloneNames3 = Array.from(names);
```

Конструктором массивов является `Array`:

```
const arr = ['One', 'Two', 'Three'];
console.log(arr.constructor === Array); // true
```

При этом прототипом массивов является «Array.prototype»:

```
console.log( Object.prototype.toString.call(arr) ); // "[object Array]"
console.log(  Object.prototype.toString.call(Array.prototype)  );  //  "[object
Array]"
```

Проверить содержит ли переменная массив можно следующим образом:

```
// 1 способ
console.log( Array.isArray(arr) ); // true
// 2 способ
console.log( arr instanceof Array ); // true
// 3 способ
console.log( arr.constructor === Array ); // true
```

Поиск элемента в массиве

Найти элемент в массиве можно с помощью метода `indexOf()`:

```
const disks = ['500Gb', '1Tb', '2Tb'];
const index = disks.indexOf('1Tb'); // 1
```

В качестве результата он возвращает индекс первого найденного элемента.

Если элемент не найден, то `indexOf()` возвращает `-1`. Это можно использовать, например, чтобы проверить существует ли элемент в массиве:

```
const disks = ['500Gb', '1Tb', '2Tb'];
if (disks.indexOf('1Tb') > -1) {
  console.log( 'Этот элемент в массиве есть!' );
}
```

Метод `indexOf()` позволяет искать элементы не только с начала, но и с определённого индекса. Для этого ему нужно его указать в качестве второго аргумента:

```
const nums = ['One', 'Two', 'One'];
nums.indexOf('One', 1); // 2
```

Метод `lastIndexOf()` выполняет то же самое что `indexOf()`, только осуществляет это с конца:

```
// arr – массив, element – искомый элемент, from – индекс (по умолчанию arr.length)
arr.lastIndexOf(element[, from])
```

Пример:

```
const nums = ['One', 'Two', 'Three', 'One'];
nums.lastIndexOf('One'); // 3
nums.lastIndexOf('Two'); // 1
```

Начиная с версии ECMAScript 7 появился новый метод `includes()`.

Он позволяет проверить содержит ли массив указанный элемент:

```
// arr – массив, element – искомый элемент, from – индекс (по умолчанию 0)
arr.includes(element[, from])
```

Он похож на `indexOf()`, но в отличие от него возвращает `true` или `false`.

```
[1, 2, 3].includes(2); // true
[1, 2, 3].includes(4); // false
```

С использованием второго аргумента:

```
[1, 2, 3].includes(2, 1); // true
[1, 2, 3].includes(2, -1); // false
```

При отрицательных значениях `from` поиск выполняется начиная с `array.length + from`.

В отличие от `indexOf()`, в котором используется строгое равенство (Strict Equality Comparison), в `includes()` используется алгоритм равенства SameValueZero. Это значит, что вы можете, например, определить, содержит ли массив NaN:

```
[1, 2, NaN].includes(NaN); // true
[1, 2, NaN].indexOf(NaN); // -1
```

Также в отличие от `indexOf()`, `includes()` не пропускает отсутствующие элементы:

```
[1, , 3].includes(undefined); // true
[1, , 3].indexOf(undefined); // -1
```

Удаление элементов массива

Удаление элемента массива с помощью `delete` делает его неопределённым (пустым). Длина массива при этом не изменяется.

```
const nums = ['One', 'Two', 'Three', 'Four', 'Five'];
delete nums[2]; // ['One', 'Two', , 'Four', 'Five']
```

Если нужно конкретно удалить элемент из массива, то можно воспользоваться методом `splice`.

Пример удаления элемента массива по индексу:

```
const nums = ['One', 'Two', 'Three', 'Four', 'Five'];
nums.splice(2, 1); // ['One', 'Two', 'Four', 'Five']
По значению это можно сделать так:
const nums = ['One', 'Two', 'Three', 'Four', 'Five'];
for (let i = 0; i < nums.length; i++) {
  if (nums[i] === 'Three') {
    nums.splice(i--, 1);
  }
}
```

С помощью `filter()`:

```
const nums = ['One', 'Two', 'Three', 'Four', 'Five'];
const result = nums.filter(value => value !== 'Three');
console.log(result); // ['One', 'Two', 'Four', 'Five']
```

Добавление и удаление элементов

В JavaScript для добавления и удаления элементов имеются следующие методы:

- `push()` – для добавления одного или нескольких элементов в конец массива;
- `unshift()` – для добавления одного или нескольких элементов в начало массива;
- `pop()` – для удаления последнего элемента;
- `shift()` – для удаления первого элемента.

Пример использования методов:

```
const planets = ['Меркурий'];
// добавим в конец 2 элемента
planets.push('Земля', 'Марс'); // ["Меркурий", "Земля", "Марс"]
// добавим в начало 1 элемент
planets.unshift('Венера'); // ["Венера", "Меркурий", "Земля", "Марс"]
// удалим последний элемент
planets.pop(); // ["Венера", "Меркурий", "Земля"]
// удалим первый элемент
planets.shift(); // ["Меркурий", "Земля"]
```

Методы `push()` и `unshift()` в качестве результата возвращают количество элементов в массиве, а `pop` и `unshift` - удаленный элемент:

```
const list = ['One'];
console.log(list.push('Two', 'Three')); // 3
console.log(list.unshift('Four')); // 4
console.log(list.pop()); // "Four"
console.log(list.shift()); // "One"
```

Функции для работы с массивами (методы объекта Array)

Объект Array содержит следующие методы (функции) для работы с массивами:

- slice
- splice
- join
- split
- reverse
- sort

slice - копирование участка массива

Метод **slice** предназначен для копирования участка массива. При этом он не изменяет исходный массив, а возвращает в качестве результата новый массив, состоящий из выбранных элементов.

Метод **slice** имеет 2 параметра:

- 1 параметр (обязательный) - предназначен для указания индекса элемента, с которого необходимо начать копировать элементы;
- 2 параметр (необязательный) - предназначен для указания индекса элемента, до которого необходимо копировать (при этом он не включается в новый массив). Если его не указать, то будут скопированы элементы до конца указанного массива.

```
let namePlanets = ["Венера", "Меркурий", "Земля", "Марс", "Юпитер"];
let newNamePlanets = namePlanets.slice(2, 4); // ["Земля", "Марс"]
```

splice - изменение содержимого массива

Метод **splice** предназначен для изменения содержимого массива. Он может использоваться как для добавления элементов в массив, так и для их удаления.

Синтаксис метода **splice**:

```
array.splice(startIndex, deleteCount [, element1[, element2[, ...]]]);
/*
  startIndex (обязательный) - стартовый индекс элемента, с которого нужно начать изменение массива.
      Если в качестве startIndex указать число, большее длины массива, то стартовый индекс будет установлен на конец массива.
      Если в качестве startIndex указать отрицательное число, то отсчет стартового элемента будет вестись с конца.
  deleteCount (обязательный) - число, показывающее какое количество элементов необходимо удалить из массива.
      Если элементы не нужно удалять из массива, то deleteCount необходимо установить 0. После этого нужно указать как минимум один новый элемент, который нужно добавить в массив.
      Если в качестве deleteCount указать число, которое будет превышать количество оставшихся элементов в массиве, начиная с startIndex, то в этом случае они всё равно будут удалены (т.е. все элементы до конца массива, начиная со стартового индекса)
  element1, element2, ... (необязательные) - элементы которые нужно добавить в массив.
*/
```

Примеры использования метода splice.

Применения метода splice для **удаления** части элементов из массива.

```
let namePlanets = ["Венера", "Меркурий", "Земля", "Марс"];
namePlanets.splice(2, 2); //["Земля", "Марс"]
console.log(namePlanets); // ["Венера", "Меркурий"]
```

Применение метода splice для **удаления** элемента из массива и **добавления** в него новых.

```
let namePlanets = ["Венера", "Меркурий", "Земля", "Марс"];
namePlanets.splice(1, 1, "Уран", "Нептун", "Сатурн"); // ["Меркурий"]
console.log(namePlanets); // ["Венера", "Уран", "Нептун", "Сатурн", "Земля", "Марс"]
```

Применение метода splice только для **добавления** новых элементов в массив.

```
let namePlanets = ["Юпитер", "Сатурн", "Уран"];
namePlanets.splice(0, 0, "Венера", "Меркурий", "Земля", "Марс"); // []
console.log(namePlanets); // ["Венера", "Меркурий", "Земля", "Марс", "Юпитер", "Сатурн", "Уран"]
```

join - преобразование массива в строку

Метод **join** предназначен для соединения всех элементов массива в строку.

Синтаксис метода **join**:

```
array.join([separator]);
/*
  separator (необязательный) - разделитель, который используется в качестве соединительной строки между каждым элементом массива.
            Если данный параметр не указать, то в качестве соединительной строки будет использоваться ",".
            Если в качестве параметра указать пустую строку, то элементы массивы в возвращаемой строке между собой ничем разделены не будут
*/
```

Пример.

```
let berries = ["Виноград", "Виноград", "Смородина", "Шиповник"];
let berriesStr1 = berries.join(); // "Виноград,Виноград,Смородина,Шиповник"
let berriesStr2 = berries.join(""); // "ВиноградВиноградСмородинаШиповник"
let berriesStr3 = berries.join(", "); // "Виноград, Виноград, Смородина, Шиповник"
let berriesStr4 = berries.join(" + "); // "Виноград + Виноград + Смородина + Шиповник"
```

Если в качестве **separator** использовать не строку, то он будет преобразован к строке.

```
let berries = ["Виноград", "Виноград", "Смородина", "Шиповник"];
let berriesStr1 = berries.join(false); // "ВиноградfalseВиноградfalseСмородинаfalseШиповник"
let berriesStr2 = berries.join(4/2); // "Виноград2Виноград2Смородина2Шиповник"
```

Элементы массива, которые имеют в качестве значения null или undefined, будут приведены к пустой строке.

```
let arr = [0, undefined, 5, null, -4];
let arrStr = arr.join(", "); // "0, , 5, , -4"
```

Преобразование строки в массив

Создание массива из строки посредством её разбивания с помощью разделителя в JavaScript осуществляется с помощью метода `split()`. Разделитель указывается в качестве аргумента.

```
const str = 'Процессоры||Материнские платы||Видеокарты';
const items = str.split('||'); // ['Процессоры', 'Материнские платы', 'Видеокарты']
```

Переворот массива

Перестановка элементов массива в обратном порядке осуществляется в JavaScript с помощью `reverse()`:

```
const nums = [1, 2, 3, 4, 5];
// переворачиваем массив
nums.reverse(); // [5, 4, 3, 2, 1]
```

Сортировка элементов массива

Сортировка массива выполняется с помощью метода `sort()`. По умолчанию он сортирует массив в порядке следования символов в кодировке Unicode.

```
const auto = ['Mazda', 'Audi', 'Toyota', 'Nissan', 'Tesla'];
// сортируем массив
auto.sort(); // ['Audi', 'Mazda', 'Nissan', 'Tesla', 'Toyota']
```

В обратном порядке:

```
auto.sort().reverse(); // ['Toyota', 'Tesla', 'Nissan', 'Mazda', 'Audi']
```