



Лекция №4



Массив



Array

Массив (**Array**) в JavaScript является глобальным объектом, который используется для создания массивов; которые представляют собой высокоуровневые спископодобные объекты.



Синтаксис

```
[element0, element1, ..., elementN]
```

```
new Array(element0, element1[, ..., elementN])
```

```
new Array(arrayLength)
```



Описание

Массивы являются спископодобными объектами, чьи прототипы содержат методы для операций обхода и изменения массива. Ни размер JavaScript-массива, ни типы его элементов не являются фиксированными. Поскольку размер массива может увеличиваться и уменьшаться в любое время, то нет гарантии, что массив окажется плотным. То есть, при работе с массивом может возникнуть ситуация, что элемент массива, к которому вы обратитесь, будет пустым и вернёт `undefined`. В целом, это удобная характеристика; но если эта особенность массива не желательна в вашем специфическом случае, вы можете рассмотреть возможность использования типизированных массивов.



Доступ к элементам массива

Массивы в JavaScript индексируются с нуля: первый элемент массива имеет индекс, равный 0, а индекс последнего элемента равен значению свойства массива `length` минус 1.

```
var arr = ['первый элемент', 'второй элемент', 'последний элемент'];  
console.log(arr[0]);           // напечатает 'первый элемент'  
console.log(arr[1]);           // напечатает 'второй элемент'  
console.log(arr[arr.length - 1]); // напечатает 'последний элемент'
```



Взаимосвязь свойства `length` с числовыми свойствами

Свойство массивов `length` связано с числовыми свойствами. Некоторые встроенные методы массива (например, `join`, `slice`, `indexOf` и т.д.) учитывают значение свойства `length` при своём вызове. Другие методы (например, `push`, `splice` и т.д.) в результате своей работы также обновляют свойство `length` массива.

```
var fruits = [];  
fruits.push('банан', 'яблоко', 'персик');  
  
console.log(fruits.length); // 3
```




Объекты



Основы объектов

Объект — это совокупность связанных данных и/или функциональных возможностей. Обычно состоят из нескольких переменных и функций, которые называются свойствами и методами, если они находятся внутри объектов. Разберём пример, чтобы показать, как они выглядят.

Чтобы начать, скопируйте себе [oojs.html](#)  файл. В нём содержится очень мало: `<script>` элемент для написания в нём исходного кода. Мы будем использовать это как основу для изучения основ синтаксиса объектов. Во время работы с этим примером у вас должна быть открытая [консоль JavaScript инструментов разработчика](#), готовая к вводу некоторых команд.



Как и во многих случаях в JavaScript, создание объекта часто начинается с определения и инициализации переменной. Попробуйте ввести следующий код JavaScript в ваш файл, а затем сохраните файл и обновите страницу браузера:

```
const person = {};
```



Если вы введёте `person` в текстовое JS консоль и нажмёте клавишу Enter, должен получиться следующий результат:

```
Object { }
```



Точечная запись (Dot notation)

Выше вы получили доступ к свойствам и методам используя **точечную запись (dot notation)**. Имя объекта (person) действует как **пространство имён (namespace)** — оно должно быть введено первым, для того чтобы получить доступ ко всему что заключено (**encapsulated**) внутри объекта. Далее вы пишете точку, затем элемент, к которому хотите получить доступ — это может быть имя простого свойства, элемент массива, или вызов одного из методов объекта, например:

```
person.age  
person.interests[1]  
person.bio()
```



Скобочная запись (Bracket notation)

Существует другой способ получить свойства объекта — использовать скобочную запись (bracket notation). Вместо написания этого кода:

```
person.age  
person.name.first
```



Вы можете использовать следующий

```
person['age']  
person['name']['first']
```



Запись элементов в объект

До сих пор мы рассматривали только возврат (или получение) элементов объекта — вы так же можете установить (обновить) значение элемента объекта просто объявив элемент, который вы хотите установить (используя точечную или скобочную запись), например:

```
person.age = 45;  
person['name']['last'] = 'Cratchit';
```



Что такое "this"?

Возможно, вы заметили что-то странное в наших методах. Посмотрите на этот пример:

```
greeting: function() {  
    alert('Hi! I\'m ' + this.name.first + '.');  
}
```



Вы, вероятно, задаётесь вопросом, что такое "this"? Ключевое слово `this`, ссылается на текущий объект, внутри которого пишется код — поэтому в нашем случае `this` равен объекту `person`. Но почему просто не написать `person`? Как вы увидите в статье [Object-oriented JavaScript for beginners](#) (Объектно-ориентированный JavaScript для начинающих), когда мы начинаем создавать конструкторы и т.д., `this` очень полезен — он всегда будет гарантировать, что используется верное значение, когда контекст элемента изменяется (например, два разных экземпляра объекта `person` могут иметь разные имена, но захотят использовать своё собственное имя при приветствии).

ЦИКЛЫ



Зацикливание кода

Языки программирования очень полезны для быстрой реализации повторяющихся задач. От базовых числовых операций до любой другой ситуации, когда у вас есть много похожих операций, которые нужно выполнить. В этой статье мы рассмотрим структуры циклов, доступные в JavaScript, которые можно использовать для этих целей.



Цикл обычно составляет одну или несколько из следующих функций:

- **Счётчик**, который инициализируется с определённого значения — начальной точки цикла (На рисунке выше первый этап: "у меня нет еды (i have no food)")
- **Условие выхода** — критерий, при котором цикл останавливается, — обычно наступает, когда цикл достигает определённого значения. Это иллюстрируется выше словами "Достаточно ли у меня еды? (Do I have enough food?)". Предположим, фермеру нужно 10 порций еды, чтобы прокормить семью.
- **Итератор** постепенно увеличивает счётчик на некоторое значение на каждом шаге цикла, пока не достигнуто условия выхода. Мы явно не показали это в изображении, но если предположить что фермер собирает две порции еды в час, то после каждого часа, количество еды, которое у него имеется, увеличивается на две порции, и он проверяет достаточно ли у него еды сейчас. Если у него собралось 10 порций (условие выхода), он может остановить сбор и вернуться домой.



Правила записи цикла

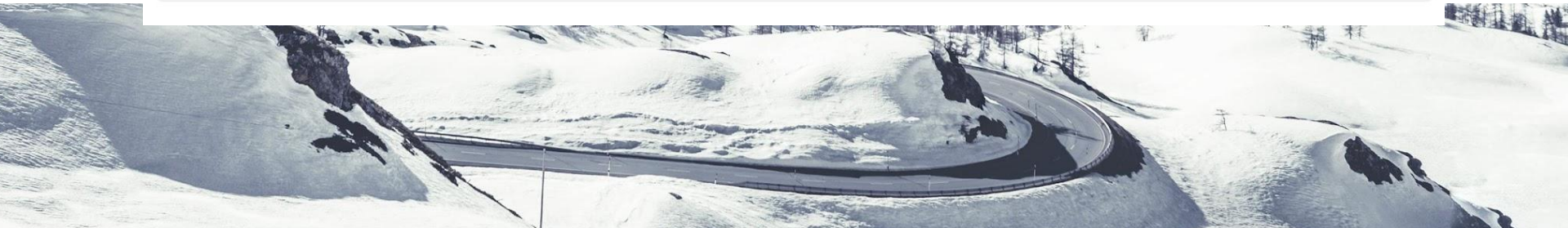
Рассмотрим некоторые конкретные конструкции циклов. Первый вариант, который вы будете использовать чаще всего, это цикл [for](#). Он имеет следующий синтаксис:

```
for (initializer; exit-condition; final-expression) {  
    // код для выполнения  
}
```



Посмотрим на пример, чтобы разобраться в этом более детально.

```
var cats = [ 'Билл', 'Макс', 'Пикси', 'Алиса', 'Жасмин' ];  
var info = 'Моих кошек зовут '  
var para = document.querySelector('p');  
  
for (var i = 0; i < cats.length; i++) {  
    info += cats[i] + ', '  
}  
  
para.textContent = info;
```



Выход из цикла с помощью break

Если вы хотите выйти из цикла до завершения всех итераций, вы можете использовать оператор [break](#) . Мы уже встречались с ним в предыдущей статье, когда рассматривали оператор [switch](#): когда происходит событие, соответствующее условию, прописанному ключевым словом `case` внутри оператора `switch` , условие `break` моментально выходит из конструкции `switch` и запускает следующий после неё код.

Тоже самое и с циклами — условие `break` моментально закончит цикл и заставит браузер запустить следующий после цикла код.



Пропуск итераций с продолжением

Оператор [continue](#) работает аналогичным образом, как и `break`, но вместо полного выхода из цикла он переходит к следующей итерации цикла.

Рассмотрим другой пример, в котором в качестве вводных данных принимается число, а возвращаются только числа, которые являются квадратами целых чисел.



Код HTML в основном такой же, как и в предыдущем примере — простой ввод текста и абзац для вывода. JavaScript в основном такой же, хотя сам цикл немного другой:

```
var num = input.value;

for (var i = 1; i <= num; i++) {
    var sqRoot = Math.sqrt(i);
    if (Math.floor(sqRoot) !== sqRoot) {
        continue;
    }

    para.textContent += i + ' ';
}
```



Циклы `while` и `do...while`

for — не единственный тип цикла, доступный в JavaScript. На самом деле существует множество других типов циклов. И хотя сейчас не обязательно знать их все, стоит взглянуть на структуру нескольких других, чтобы вы могли распознать те же функции, но в работе немного по-другому.

Рассмотрим цикл [while](#). Синтаксис этого цикла выглядит так:

```
initializer
while (exit-condition) {
    // code to run

    final-expression
}
```

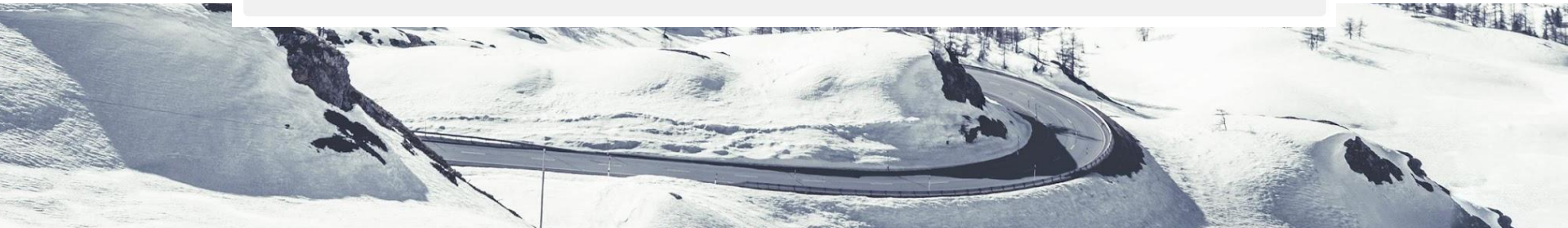
for...of

Сводка

Оператор `for...of` выполняет цикл обхода итерируемых объектов (включая `Array`, `Map`, `Set`, объект `аргументов` и подобных), вызывая на каждом шаге итерации операторы для каждого значения из различных свойств объекта.

Синтаксис

```
for (variable of iterable) {  
    statement  
}
```



Полезные ссылки

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array

<https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/Basics>

https://developer.mozilla.org/ru/docs/Learn/JavaScript/Building_blocks/Looping_code





Спасибо!

