Javascript

Javascript – динамический скриптовый язык программирования высокого уровня. Он отличается мультипарадигменностью. Речь идет о поддержке функционального, императивного, событийно-ориентированного стилей. Чаще всего язык используется для создания интерактивных веб-страниц и приложений. Востребованность, сферы применения. Неизменно высокий интерес к Javascript подтверждают специальные международные рейтинги. JavaScript — [мультипарадигменный](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Мультипарадигменное программирование) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Поддерживает [объектно-ориентированный](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивный](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [функциональный](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) стили. Является реализацией спецификации [ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript" \o "ECMAScript) (стандарт ECMA-262[[9]](https://ru.wikipedia.org/wiki/JavaScript#cite_note-ECMA-262-10)).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам [приложений](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Наиболее широкое применение находит в [браузерах](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) как язык сценариев для придания [интерактивности](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [веб-страницам](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0)[[10]](https://ru.wikipedia.org/wiki/JavaScript#cite_note-11).

Основные архитектурные черты: [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [слабая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [прототипное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Прототипное программирование), функции как [объекты первого класса](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0).

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Название «JavaScript» является зарегистрированным [товарным знаком](https://ru.wikipedia.org/wiki/%D0%A2%D0%BE%D0%B2%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D0%B7%D0%BD%D0%B0%D0%BA) корпорации [Oracle](https://ru.wikipedia.org/wiki/Oracle) в США.

В [1992 году](https://ru.wikipedia.org/wiki/1992_%D0%B3%D0%BE%D0%B4) компания Nombas (впоследствии приобретённая [Openwave](https://ru.wikipedia.org/w/index.php?title=Openwave&action=edit&redlink=1" \o "Openwave (страница отсутствует))) начала разработку встраиваемого скриптового языка Cmm (Си-минус-минус), который, по замыслу разработчиков, должен был стать достаточно мощным, чтобы заменить [макросы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D1%80%D0%BE%D1%81), сохраняя при этом схожесть с Си, чтобы разработчикам не составляло труда изучить его[[13]](https://ru.wikipedia.org/wiki/JavaScript#cite_note-nombasHistory-15). Главным отличием от Си была работа с памятью. В новом языке всё управление памятью осуществлялось автоматически: не было необходимости создавать [буфера](https://ru.wikipedia.org/wiki/%D0%91%D1%83%D1%84%D0%B5%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), объявлять переменные, осуществлять преобразование типов. В остальном языки сильно походили друг на друга: в частности, Cmm поддерживал стандартные функции и операторы Си. Cmm был переименован в ScriptEase, поскольку исходное название звучало слишком негативно, а упоминание в нём Си «отпугивало» людей. На основе этого языка был создан [проприетарный продукт](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%80%D0%B8%D0%B5%D1%82%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) CEnvi. В конце ноября [1995 года](https://ru.wikipedia.org/wiki/1995_%D0%B3%D0%BE%D0%B4) Nombas разработала версию CEnvi, внедряемую в веб-страницы. Страницы, которые можно было изменять с помощью скриптового языка, получили название Espresso Pages — они демонстрировали использование скриптового языка для создания игры, проверки пользовательского ввода в формы и создания анимации. Espresso Pages позиционировались как [демоверсия](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BC%D0%BE%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F" \o "Демоверсия), призванная помочь представить, что случится, если в браузер будет внедрён язык Cmm. Работали они только в 16-битовом [Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator) под управлением [Windows](https://ru.wikipedia.org/wiki/Windows).

JavaScript, наверное, самый известный мультипарадигменный язык, в котором очень много неочевидных особенностей. Но тем не менее любим ли мы его или ругаем, факт остается фактом — это основной язык, на котором работает современный web.  
  
В ушедшем году, вышел стандарт ECMAScript 2015 (неформально ES6), который сильно изменил, то к чему мы привыкли. Появилась масса новых возможностей, которые по сути представляют собой современное надмножество языка, пытающегося решить существующие проблемы. Class, let, const, стрелочные функции… разработчик, который ранее не видел код, написанный на ES6, не сразу догадается, что перед ним, по сути, старый добрый JS.  
  
Есть масса прекрасных статей, посвященных современному стандарту. В этом же посте я хочу показать, что нам может предложить современный JS, когда необходимо решить насущную задачу. Например, поздравить всех c Новым Годом.

JavaScript в элементе script

Самый простой способ внедрения JavaScript в HTML-документ – использование тега [<script>](https://wm-school.ru/tags/tag_script.html). Теги <script> часто помещают в элемент <head>, и ранее этот способ считался чуть ли не обязательным. Однако в наши дни теги <script> используются как в элементе <head>, так и в теле веб-страниц.

Таким образом, на одной веб-странице могут располагаться сразу несколько сценариев. В какой последовательности браузер будет выполнять эти сценарии? Как правило, выполнение сценариев браузерами происходит по мере их загрузки. Браузер читает HTML-документ сверху вниз и, когда он встречает тег <script>, рассматривает текст программы как сценарий и выполняет его. Остальной контент страницы не загружается и не отображается, пока не будет выполнен весь код в элементе <script>.

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
</head>  
<body>  
<p>Это обычный HTML документ</p>  
  
<script language="JavaScript">  
    alert("Привет, мир!");  
</script>  
  
<p>Выходим обратно в HTML</p>  
</body>  
</html>

Внешний JavaScript

Если JavaScript-кода много – его выносят в отдельный файл, который, как правило, имеет расширение .js.

Чтобы включить в HTML-документ JavaScript-кoд из внешнего файла, нужно использовать атрибут src (source) тега <script>. Его значением должен быть URL-aдpec файла, в котором содержится JS-код:

<script src="/scripts/script.js"></script>

В этом примере указан абсолютный путь к файлу с именем script.js, содержащему скрипт (из корня сайта). Сам файл должен содержать только JavaScript-кoд, который иначе располагался бы между тегами <script> и </script>.

Комментарии в javascript

В JavaScript есть два типа комментариев. Однострочные комментарии в JavaScript записываются с помощью двух слешей //.// This is a comment. Интерпретатор будет игнорировать все символы после // до конца строки. Многострочные комментарии заключаются в теги /\* и \*/. Если вы знакомы с CSS, вы уже знаете, как работают такие комментарии./ \* This is a comment\*/

Где писать код?

JavaScript код пишется прямо на HTML странице внутри тега script. Этот тег можно размещать в любом месте страницы.

JavaScript — вывод информации

[Первый способ: функция writeln](https://fokit.ru/javascript-vyvod-informatsii/?ysclid=lf7vn6szl2297775169#yak_1)

[Второй способ: функция alert](https://fokit.ru/javascript-vyvod-informatsii/?ysclid=lf7vn6szl2297775169#yak_2)

[Третий способ: функция getElementById](https://fokit.ru/javascript-vyvod-informatsii/?ysclid=lf7vn6szl2297775169#yak_3)

[Заключение](https://fokit.ru/javascript-vyvod-informatsii/?ysclid=lf7vn6szl2297775169#yak_4)

Здравствуйте, в прошлой статье мы рассматривали синтаксис языка JavaScript. Мы узнали что такое циклы, переменные, массивы, логические операторы, функции и так далее.

Все эти операции проходят внутри страницы и никак не отображаются. Сегодня же мы научимся выводить необходимые данные на экран. Выводить будем обычную числовую переменную с помощью ActionScript, самыми разными способами.

Для того чтобы использовать все способы, которые будут указаны ниже, нам необходимо обращаться к объекту document. Это отдельный элемент языка JavaScript, который предназначен для работы с веб-сайтами.

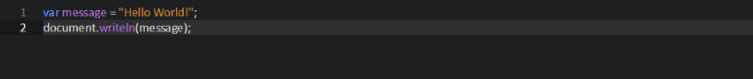
Первый способ: функция writeln

Для вывода переменной таким образом нам необходимо её создать. Напишем такую строку: var message = «Hello World!».

Теперь поговорим о функции writeln. Её лучше всего использовать только при загрузке исходного варианта страницы, так внутренний контент будет изменен.

С помощью таких функций как writeln можно создать страницу, полностью написанную на JavaScript. Кроме этого, данная функция может быть полезна в том случае, когда необходимо уменьшить количество запросов с сервера.

Для того чтобы воспользоваться данной функцией, записываем команду document.writeln, в скобках записываем название нашей переменной, которое затем будет показано на экране.



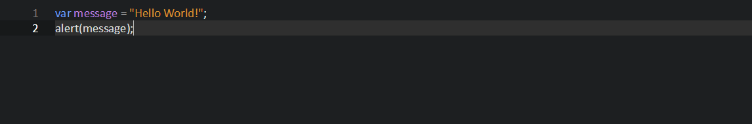
Второй способ: функция alert

Она не входит в пространство имен document, как writeln, поэтому его вписывать не нужно. Alert может отлично подойти для отладки кода веб-страницы. Его можно использовать и для взаимодействия с пользователем, но делать это не рекомендуется.

Alert имеет стандартный синтаксис любой JavaScript функции. Сначала записываем команду alert, в скобках записываем значение или переменной.

После этого в верхней части страницы всплывет окно с сообщением для пользователя, которое содержит кнопку «OK».

Данная функция может иметь множество применений, но не стоит злоупотреблять ей при работе с веб-сайтом, так как всплывающие окна могут вывести пользователя из себя. Кроме того, некоторые плагины блокируют функции типа alert.

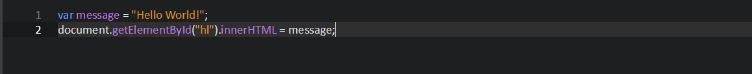


Третий способ: функция getElementById

Это самый сложный и наиболее популярный способ вывода информации на экран. С его помощью вы сможете изменять текстовый контент вашей HTML страницы.

Как мы знаем — все HTML теги могут иметь идентификаторы. Именно к ним и обращается наша функция. GetElementById находит элемент по его ID, изменяет содержимое тега по желанию, при этом остальной контент остается без изменений.

Данная функция входит в пространство имен document, что требует указывать его перед самой функцией.



Как вы могли заметить, кроме getElementById присутствует непонятная команда innerHTML, после него идет знак равенства и текстовая строка. Так вот, эта команда, вместе со знаком равенства, находит определенный тег и «приравнивает» его содержимое строке в кавычках.

Данный метод лучше всего подходит для вставки текста через JavaScript, поскольку он имеет целый ряд преимуществ:

Отсутствие всплывающих окон при работе функции.

Функция работает только с HTML-элементами.

Текст в кавычках может быть обернут в теги. Таким образом можно написать страницу на JavaScript с нуля.

Заключение

В этой статье мы поговорили с вами о том, как выводить информацию на экран. Это можно сделать с помощью сообщений, а также меняя внутренний контент веб-страницы. Мы лишь немного затронули пространство имен document, которое имеет целый ряд функция для работы с веб-страницей. В следующих уроках мы ознакомимся с ними детально.

Массивы

Объекты позволяют хранить данные со строковыми ключами. Это замечательно.

Но довольно часто мы понимаем, что нам необходима упорядоченная коллекция данных, в которой присутствуют 1-й, 2-й, 3-й элементы и т.д. Например, она понадобится нам для хранения списка чего-либо: пользователей, товаров, элементов HTML и т.д.

В этом случае использовать объект неудобно, так как он не предоставляет методов управления порядком элементов. Мы не можем вставить новое свойство «между» уже существующими. Объекты просто не предназначены для этих целей.

Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array.

[Объявление](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "obyavlenie)

Существует два варианта синтаксиса для создания пустого массива:

let arr = new Array();

let arr = [];

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

let fruits = ["Яблоко", "Апельсин", "Слива"];

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Мы можем заменить элемент:

fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]

…Или добавить новый к существующему массиву:

fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]

Общее число элементов массива содержится в его свойстве length:

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits.length ); // 3

Вывести массив целиком можно при помощи alert.

let fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits ); // Яблоко, Апельсин, Слива

В массиве могут храниться элементы любого типа.

Например:

// разные типы значений

let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];

// получить элемент с индексом 1 (объект) и затем показать его свойство

alert( arr[1].name ); // Джон

// получить элемент с индексом 3 (функция) и выполнить её

arr[3](); // привет

Висячая запятая

Список элементов массива, как и список свойств объекта, может оканчиваться запятой:

let fruits = [

"Яблоко",

"Апельсин",

"Слива",

];

«Висячая запятая» упрощает процесс добавления/удаления элементов, так как все строки становятся идентичными.

[Получение последних элементов при помощи «at»](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "poluchenie-poslednih-elementov-pri-pomoschi-at)

Новая возможность

Эта возможность была добавлена в язык недавно. В старых браузерах может понадобиться [полифил](https://learn.javascript.ru/polyfills).

Допустим, нам нужен последний элемент массива.

Некоторые языки программирования позволяют использовать отрицательные индексы для той же цели, как-то так: fruits[-1].

Однако, в JavaScript такая запись не сработает. Её результатом будет undefined, поскольку индекс в квадратных скобках понимается буквально.

Мы можем явно вычислить индекс последнего элемента, а затем получить к нему доступ вот так: fruits[fruits.length - 1].

let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[fruits.length-1] ); // Plum

Немного громоздко, не так ли? Нам нужно дважды написать имя переменной.

К счастью, есть более короткий синтаксис: fruits.at (-1):

let fruits = ["Apple", "Orange", "Plum"];

// то же самое, что и fruits[fruits.length-1]

alert( fruits.at(-1) ); // Plum

Другими словами, arr.at(i):

это ровно то же самое, что и arr[i], если i >= 0.

для отрицательных значений i, он отступает от конца массива.

[Методы pop/push, shift/unshift](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "metody-pop-push-shift-unshift)

[Очередь](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) – один из самых распространённых вариантов применения массива. В области компьютерных наук так называется упорядоченная коллекция элементов, поддерживающая два вида операций:

push добавляет элемент в конец.

shift удаляет элемент в начале, сдвигая очередь, так что второй элемент становится первым.

Массивы поддерживают обе операции.

На практике необходимость в этом возникает очень часто. Например, очередь сообщений, которые надо показать на экране.

Существует и другой вариант применения для массивов – структура данных, называемая [стек](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA).

Она поддерживает два вида операций:

push добавляет элемент в конец.

pop удаляет последний элемент.

Таким образом, новые элементы всегда добавляются или удаляются из «конца».

Примером стека обычно служит колода карт: новые карты кладутся наверх и берутся тоже сверху:

Массивы в JavaScript могут работать и как очередь, и как стек. Мы можем добавлять/удалять элементы как в начало, так и в конец массива.

В компьютерных науках структура данных, делающая это возможным, называется [двусторонняя очередь](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D1%83%D1%85%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D1%8F%D1%8F_%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C).

Методы, работающие с концом массива:

pop

Удаляет последний элемент из массива и возвращает его:

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удаляем "Груша" и выводим его

alert( fruits ); // Яблоко, Апельсин

И fruits.pop() и fruits.at(-1) возвращают последний элемент массива, но fruits.pop() также изменяет массив, удаляя его.

push

Добавляет элемент в конец массива:

let fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ....

Методы, работающие с началом массива:

shift

Удаляет из массива первый элемент и возвращает его:

let fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удаляем Яблоко и выводим его

alert( fruits ); // Апельсин, Груша

unshift

Добавляет элемент в начало массива:

let fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу несколько элементов:

let fruits = ["Яблоко"];

fruits.push("Апельсин", "Груша");

fruits.unshift("Ананас", "Лимон");

// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]

alert( fruits );

[Внутреннее устройство массива](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "vnutrennee-ustroystvo-massiva)

Массив – это особый подвид объектов. Квадратные скобки, используемые для того, чтобы получить доступ к свойству arr[0] – это по сути обычный синтаксис доступа по ключу, как obj[key], где в роли obj у нас arr, а в качестве ключа – числовой индекс.

Массивы расширяют объекты, так как предусматривают специальные методы для работы с упорядоченными коллекциями данных, а также свойство length. Но в основе всё равно лежит объект.

Следует помнить, что в JavaScript существует 8 основных типов данных. Массив является объектом и, следовательно, ведёт себя как объект.

Например, копируется по ссылке:

let fruits = ["Банан"]

let arr = fruits; // копируется по ссылке (две переменные ссылаются на один и тот же массив)

alert( arr === fruits ); // true

arr.push("Груша"); // массив меняется по ссылке

alert( fruits ); // Банан, Груша - теперь два элемента

…Но то, что действительно делает массивы особенными – это их внутреннее представление. Движок JavaScript старается хранить элементы массива в непрерывной области памяти, один за другим, так, как это показано на иллюстрациях к этой главе. Существуют и другие способы оптимизации, благодаря которым массивы работают очень быстро.

Но все они утратят эффективность, если мы перестанем работать с массивом как с «упорядоченной коллекцией данных» и начнём использовать его как обычный объект.

Например, технически мы можем сделать следующее:

let fruits = []; // создаём массив

fruits[99999] = 5; // создаём свойство с индексом, намного превышающим длину массива

fruits.age = 25; // создаём свойство с произвольным именем

Это возможно, потому что в основе массива лежит объект. Мы можем присвоить ему любые свойства.

Но движок поймёт, что мы работаем с массивом, как с обычным объектом. Способы оптимизации, используемые для массивов, в этом случае не подходят, поэтому они будут отключены и никакой выгоды не принесут.

Варианты неправильного применения массива:

Добавление нечислового свойства, например: arr.test = 5.

Создание «дыр», например: добавление arr[0], затем arr[1000] (между ними ничего нет).

Заполнение массива в обратном порядке, например: arr[1000], arr[999] и т.д.

Массив следует считать особой структурой, позволяющей работать с упорядоченными данными. Для этого массивы предоставляют специальные методы. Массивы тщательно настроены в движках JavaScript для работы с однотипными упорядоченными данными, поэтому, пожалуйста, используйте их именно в таких случаях. Если вам нужны произвольные ключи, вполне возможно, лучше подойдёт обычный объект {}.

[Эффективность](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "effektivnost)

Методы push/pop выполняются быстро, а методы shift/unshift – медленно.

Почему работать с концом массива быстрее, чем с его началом? Давайте посмотрим, что происходит во время выполнения:

fruits.shift(); // удаляем первый элемент с начала

Просто взять и удалить элемент с номером 0 недостаточно. Нужно также заново пронумеровать остальные элементы.

Операция shift должна выполнить 3 действия:

Удалить элемент с индексом 0.

Сдвинуть все элементы влево, заново пронумеровать их, заменив 1 на 0, 2 на 1 и т.д.

Обновить свойство length .

Чем больше элементов содержит массив, тем больше времени потребуется для того, чтобы их переместить, больше операций с памятью.

То же самое происходит с unshift: чтобы добавить элемент в начало массива, нам нужно сначала сдвинуть существующие элементы вправо, увеличивая их индексы.

А что же с push/pop? Им не нужно ничего перемещать. Чтобы удалить элемент в конце массива, метод pop очищает индекс и уменьшает значение length.

Действия при операции pop:

fruits.pop(); // удаляем один элемент с конца

Метод pop не требует перемещения, потому что остальные элементы остаются с теми же индексами. Именно поэтому он выполняется очень быстро.

Аналогично работает метод push.

[Перебор элементов](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "perebor-elementov)

Одним из самых старых способов перебора элементов массива является цикл for по цифровым индексам:

let arr = ["Яблоко", "Апельсин", "Груша"];

for (let i = 0; i < arr.length; i++) {

alert( arr[i] );

}

Но для массивов возможен и другой вариант цикла, for..of:

let fruits = ["Яблоко", "Апельсин", "Слива"];

// проходит по значениям

for (let fruit of fruits) {

alert( fruit );

}

Цикл for..of не предоставляет доступа к номеру текущего элемента, только к его значению, но в большинстве случаев этого достаточно. А также это короче.

Технически, так как массив является объектом, можно использовать и вариант for..in:

let arr = ["Яблоко", "Апельсин", "Груша"];

for (let key in arr) {

alert( arr[key] ); // Яблоко, Апельсин, Груша

}

Но на самом деле это – плохая идея. Существуют скрытые недостатки этого способа:

Цикл for..in выполняет перебор всех свойств объекта, а не только цифровых.

В браузере и других программных средах также существуют так называемые «псевдомассивы» – объекты, которые выглядят, как массив. То есть, у них есть свойство length и индексы, но они также могут иметь дополнительные нечисловые свойства и методы, которые нам обычно не нужны. Тем не менее, цикл for..in выведет и их. Поэтому, если нам приходится иметь дело с объектами, похожими на массив, такие «лишние» свойства могут стать проблемой.

Цикл for..in оптимизирован под произвольные объекты, не массивы, и поэтому в 10-100 раз медленнее. Увеличение скорости выполнения может иметь значение только при возникновении узких мест. Но мы всё же должны представлять разницу.

В общем, не следует использовать цикл for..in для массивов.

[Немного о «length»](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "nemnogo-o-length)

Свойство length автоматически обновляется при изменении массива. Если быть точными, это не количество элементов массива, а наибольший цифровой индекс плюс один.

Например, единственный элемент, имеющий большой индекс, даёт большую длину:

let fruits = [];

fruits[123] = "Яблоко";

alert( fruits.length ); // 124

Обратите внимание, что обычно мы не используем массивы таким образом.

Ещё один интересный факт о свойстве length – его можно перезаписать.

Если мы вручную увеличим его, ничего интересного не произойдёт. Зато, если мы уменьшим его, массив станет короче. Этот процесс необратим, как мы можем понять из примера:

let arr = [1, 2, 3, 4, 5];

arr.length = 2; // укорачиваем до двух элементов

alert( arr ); // [1, 2]

arr.length = 5; // возвращаем length как было

alert( arr[3] ); // undefined: значения не восстановились

Таким образом, самый простой способ очистить массив – это arr.length = 0;.

[new Array()](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "new-array)

Существует ещё один вариант синтаксиса для создания массива:

let arr = new Array("Яблоко", "Груша", "и тд");

Он редко применяется, так как квадратные скобки [] короче. Кроме того, у него есть хитрая особенность.

Если new Array вызывается с одним аргументом, который представляет собой число, он создаёт массив без элементов, но с заданной длиной.

Давайте посмотрим, как можно оказать себе медвежью услугу:

let arr = new Array(2); // создастся ли массив [2]?

alert( arr[0] ); // undefined! нет элементов.

alert( arr.length ); // length 2

Как мы видим, в коде, представленном выше, в new Array(number) все элементы равны undefined.

Чтобы избежать появления таких неожиданных ситуаций, мы обычно используем квадратные скобки, если, конечно, не знаем точно, что по какой-то причине нужен именно Array.

[Многомерные массивы](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "mnogomernye-massivy)

Массивы могут содержать элементы, которые тоже являются массивами. Это можно использовать для создания многомерных массивов, например, для хранения матриц:

let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // 5, центральный элемент

[toString](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "tostring)

Массивы по-своему реализуют метод toString, который возвращает список элементов, разделённых запятыми.

Например:

let arr = [1, 2, 3];

alert( arr ); // 1,2,3

alert( String(arr) === '1,2,3' ); // true

Давайте теперь попробуем следующее:

alert( [] + 1 ); // "1"

alert( [1] + 1 ); // "11"

alert( [1,2] + 1 ); // "1,21"

Массивы не имеют ни Symbol.toPrimitive, ни функционирующего valueOf, они реализуют только преобразование toString, таким образом, здесь [] становится пустой строкой, [1] становится "1", а [1,2] становится "1,2".

Когда бинарный оператор плюс "+" добавляет что-либо к строке, он тоже преобразует это в строку, таким образом:

alert( "" + 1 ); // "1"

alert( "1" + 1 ); // "11"

alert( "1,2" + 1 ); // "1,21"

[Не сравнивайте массивы при помощи ==](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "ne-sravnivayte-massivy-pri-pomoschi)

В JavaScript, в отличие от некоторых других языков программирования, массивы не следует сравнивать при помощи оператора ==.

У этого оператора нет специального подхода к массивам, он работает с ними, как и с любыми другими объектами.

Давайте ещё раз напомним правила:

Два объекта равны друг другу == только в том случае, если они ссылаются на один и тот же объект.

Если один из аргументов == является объектом, а другой – примитивом, то объект преобразуется в примитив, как описано в главе [Преобразование объектов в примитивы](https://learn.javascript.ru/object-toprimitive).

…За исключением null и undefined, которые равны == друг другу и ничему больше.

Оператор строгого равенства === ещё проще, так как он не преобразует типы.

Итак, если мы всё же сравниваем массивы с помощью ==, то они никогда не будут одинаковыми, если только мы не сравним две переменные, которые ссылаются на один и тот же массив

Например:

alert( [] == [] ); // false

alert( [0] == [0] ); // false

Технически эти массивы являются разными объектами. Так что они не равны. Оператор == не выполняет поэлементное сравнение.

Сравнение с примитивами также может дать, казалось бы, странные результаты:

alert( 0 == [] ); // true

alert('0' == [] ); // false

Здесь, в обоих случаях, мы сравниваем примитив с объектом массива. Таким образом, массив [] преобразуется в примитив с целью сравнения и становится пустой строкой ''.

Затем продолжается процесс сравнения с примитивами, как описано в главе [Преобразование типов](https://learn.javascript.ru/type-conversions):

// после того, как [] был преобразован в ''

alert( 0 == '' ); // true, так как '' преобразуется в число 0

alert('0' == '' ); // false, нет преобразования типов, разные строки

Так как же сравнить массивы?

Это просто: не используйте оператор ==. Вместо этого сравните их поэлементам в цикле или используя методы итерации, описанные в следующей главе.

[Итого](https://learn.javascript.ru/array?ysclid=lf7vrsxd4x589374369" \l "itogo)

Массив – это особый тип объекта, предназначенный для работы с упорядоченным набором элементов.

Объявление:

// квадратные скобки (обычно)

let arr = [item1, item2...];

// new Array (очень редко)

let arr = new Array(item1, item2...);

Вызов new Array(number) создаёт массив с заданной длиной, но без элементов.

Свойство length отражает длину массива или, если точнее, его последний цифровой индекс плюс один. Длина корректируется автоматически методами массива.

Если мы уменьшаем length вручную, массив укорачивается.

Получение элементов:

Мы можем получить элемент по его индексу, например arr[0].

Также мы можем использовать метод at(i) для получения элементов с отрицательным индексом, для отрицательных значений i, он отступает от конца массива. В остальном он работает так же, как arr[i], если i >= 0.

Мы можем использовать массив как двустороннюю очередь, используя следующие операции:

push(...items)добавляет items в конец массива.

pop() удаляет элемент в конце массива и возвращает его.

shift() удаляет элемент в начале массива и возвращает его.

unshift(...items) добавляет items в начало массива.

Чтобы пройтись по элементам массива:

for (let i=0; i<arr.length; i++) – работает быстрее всего, совместим со старыми браузерами.

for (let item of arr) – современный синтаксис только для значений элементов (к индексам нет доступа).

for (let i in arr) – никогда не используйте для массивов!

Мы вернёмся к массивам и изучим другие методы добавления, удаления, выделения элементов и сортировки массивов в главе: [Методы массивов](https://learn.javascript.ru/array-methods).

Циклы while и for

При написании скриптов зачастую встаёт задача сделать однотипное действие много раз.

Например, вывести товары из списка один за другим. Или просто перебрать все числа от 1 до 10 и для каждого выполнить одинаковый код.

Для многократного повторения одного участка кода предусмотрены циклы.

Циклы for…of и for…in

Небольшое объявление для продвинутых читателей.

В этой статье рассматриваются только базовые циклы: while, do..while и for(..;..;..).

Если вы пришли к этой статье в поисках других типов циклов, вот указатели:

См. [for…in](https://learn.javascript.ru/object#forin) для перебора свойств объекта.

См. [for…of](https://learn.javascript.ru/array#perebor-elementov) и [Перебираемые объекты](https://learn.javascript.ru/iterable) для перебора массивов и перебираемых объектов.

В противном случае, продолжайте читать.

[Цикл «while»](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "tsikl-while)

Цикл while имеет следующий синтаксис:

while (condition) {

// код

// также называемый "телом цикла"

}

Код из тела цикла выполняется, пока условие condition истинно.

Например, цикл ниже выводит i, пока i < 3:

let i = 0;

while (i < 3) { // выводит 0, затем 1, затем 2

alert( i );

i++;

}

Одно выполнение тела цикла по-научному называется итерация. Цикл в примере выше совершает три итерации.

Если бы строка i++ отсутствовала в примере выше, то цикл бы повторялся (в теории) вечно. На практике, конечно, браузер не позволит такому случиться, он предоставит пользователю возможность остановить «подвисший» скрипт, а JavaScript на стороне сервера придётся «убить» процесс.

Любое выражение или переменная может быть условием цикла, а не только сравнение: условие while вычисляется и преобразуется в логическое значение.

Например, while (i) – более краткий вариант while (i != 0):

let i = 3;

while (i) { // когда i будет равно 0, условие станет ложным, и цикл остановится

alert( i );

i--;

}

Фигурные скобки не требуются для тела цикла из одной строки

Если тело цикла состоит лишь из одной инструкции, мы можем опустить фигурные скобки {…}:

let i = 3;

while (i) alert(i--);

[Цикл «do…while»](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "tsikl-dowhile)

Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

do {

// тело цикла

} while (condition);

Цикл сначала выполнит тело, а затем проверит условие condition, и пока его значение равно true, он будет выполняться снова и снова.

Например:

let i = 0;

do {

alert( i );

i++;

} while (i < 3);

Такая форма синтаксиса оправдана, если вы хотите, чтобы тело цикла выполнилось хотя бы один раз, даже если условие окажется ложным. На практике чаще используется форма с предусловием: while(…) {…}.

[Цикл «for»](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "tsikl-for)

Более сложный, но при этом самый распространённый цикл — цикл for.

Выглядит он так:

for (начало; условие; шаг) {

// ... тело цикла ...

}

Давайте разберёмся, что означает каждая часть, на примере. Цикл ниже выполняет alert(i) для i от 0 до (но не включая) 3:

for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2

alert(i);

}

Рассмотрим конструкцию for подробней:

| часть |  |  |
| --- | --- | --- |
| начало | let i = 0 | Выполняется один раз при входе в цикл |
| условие | i < 3 | Проверяется перед каждой итерацией цикла. Если оно вычислится в false, цикл остановится. |
| тело | alert(i) | Выполняется снова и снова, пока условие вычисляется в true. |
| шаг | i++ | Выполняется после тела цикла на каждой итерации перед проверкой условия. |

В целом, алгоритм работы цикла выглядит следующим образом:

Выполнить начало

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ (Если условие == true → Выполнить тело, Выполнить шаг)

→ ...

То есть, начало выполняется один раз, а затем каждая итерация заключается в проверке условия, после которой выполняется тело и шаг.

Если тема циклов для вас нова, может быть полезным вернуться к примеру выше и воспроизвести его работу на листе бумаги, шаг за шагом.

Вот в точности то, что происходит в нашем случае:

// for (let i = 0; i < 3; i++) alert(i)

// Выполнить начало

let i = 0;

// Если условие == true → Выполнить тело, Выполнить шаг

if (i < 3) { alert(i); i++ }

// Если условие == true → Выполнить тело, Выполнить шаг

if (i < 3) { alert(i); i++ }

// Если условие == true → Выполнить тело, Выполнить шаг

if (i < 3) { alert(i); i++ }

// ...конец, потому что теперь i == 3

Встроенное объявление переменной

В примере переменная счётчика i была объявлена прямо в цикле. Это так называемое «встроенное» объявление переменной. Такие переменные существуют только внутри цикла.

for (let i = 0; i < 3; i++) {

alert(i); // 0, 1, 2

}

alert(i); // ошибка, нет такой переменной

Вместо объявления новой переменной мы можем использовать уже существующую:

let i = 0;

for (i = 0; i < 3; i++) { // используем существующую переменную

alert(i); // 0, 1, 2

}

alert(i); // 3, переменная доступна, т.к. была объявлена снаружи цикла

[Пропуск частей «for»](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "propusk-chastey-for)

Любая часть for может быть пропущена.

Для примера, мы можем пропустить начало если нам ничего не нужно делать перед стартом цикла.

Вот так:

let i = 0; // мы уже имеем объявленную i с присвоенным значением

for (; i < 3; i++) { // нет необходимости в "начале"

alert( i ); // 0, 1, 2

}

Можно убрать и шаг:

let i = 0;

for (; i < 3;) {

alert( i++ );

}

Это сделает цикл аналогичным while (i < 3).

А можно и вообще убрать всё, получив бесконечный цикл:

for (;;) {

// будет выполняться вечно

}

При этом сами точки с запятой ; обязательно должны присутствовать, иначе будет ошибка синтаксиса.

[Прерывание цикла: «break»](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "preryvanie-tsikla-break)

Обычно цикл завершается при вычислении условия в false.

Но мы можем выйти из цикла в любой момент с помощью специальной директивы break.

Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока посетитель их вводит, а затем – выдаёт:

let sum = 0;

while (true) {

let value = +prompt("Введите число", '');

if (!value) break; // (\*)

sum += value;

}

alert( 'Сумма: ' + sum );

Директива break в строке (\*) полностью прекращает выполнение цикла и передаёт управление на строку за его телом, то есть на alert.

Вообще, сочетание «бесконечный цикл + break» – отличная штука для тех ситуаций, когда условие, по которому нужно прерваться, находится не в начале или конце цикла, а посередине или даже в нескольких местах его тела.

[Переход к следующей итерации: continue](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "continue)

Директива continue – «облегчённая версия» break. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно true).

Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

Например, цикл ниже использует continue, чтобы выводить только нечётные значения:

for (let i = 0; i < 10; i++) {

// если true, пропустить оставшуюся часть тела цикла

if (i % 2 == 0) continue;

alert(i); // 1, затем 3, 5, 7, 9

}

Для чётных значений i, директива continue прекращает выполнение тела цикла и передаёт управление на следующую итерацию for (со следующим числом). Таким образом alert вызывается только для нечётных значений.

Директива continue позволяет избегать вложенности

Цикл, который обрабатывает только нечётные значения, мог бы выглядеть так:

for (let i = 0; i < 10; i++) {

if (i % 2) {

alert( i );

}

}

С технической точки зрения он полностью идентичен. Действительно, вместо continue можно просто завернуть действия в блок if.

Однако мы получили дополнительный уровень вложенности фигурных скобок. Если код внутри if более длинный, то это ухудшает читаемость, в отличие от варианта с continue.

Нельзя использовать break/continue справа от оператора „?“

Обратите внимание, что эти синтаксические конструкции не являются выражениями и не могут быть использованы с тернарным оператором ?. В частности, использование таких директив, как break/continue, вызовет ошибку.

Например, если мы возьмём этот код:

if (i > 5) {

alert(i);

} else {

continue;

}

…и перепишем его, используя вопросительный знак:

(i > 5) ? alert(i) : continue; // continue здесь приведёт к ошибке

…то будет синтаксическая ошибка.

Это ещё один повод не использовать оператор вопросительного знака ? вместо if.

[Метки для break/continue](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309" \l "metki-dlya-break-continue)

Бывает, нужно выйти одновременно из нескольких уровней цикла сразу.

Например, в коде ниже мы проходимся циклами по i и j, запрашивая с помощью prompt координаты (i, j) с (0,0) до (2,2):

for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

let input = prompt(`Значение на координатах (${i},${j})`, '');

// Что если мы захотим перейти к Готово (ниже) прямо отсюда?

}

}

alert('Готово!');

Нам нужен способ остановить выполнение, если пользователь отменит ввод.

Обычный break после input лишь прервёт внутренний цикл, но этого недостаточно. Достичь желаемого поведения можно с помощью меток.

Метка имеет вид идентификатора с двоеточием перед циклом:

labelName: for (...) {

...

}

Вызов break <labelName> в цикле ниже ищет ближайший внешний цикл с такой меткой и переходит в его конец.

outer: for (let i = 0; i < 3; i++) {

for (let j = 0; j < 3; j++) {

let input = prompt(`Значение на координатах (${i},${j})`, '');

// если пустая строка или Отмена, то выйти из обоих циклов

if (!input) break outer; // (\*)

// сделать что-нибудь со значениями...

}

}

alert('Готово!');

В примере выше это означает, что вызовом break outer будет разорван внешний цикл до метки с именем outer.

Таким образом управление перейдёт со строки, помеченной (\*), к alert('Готово!').

Можно размещать метку на отдельной строке:

outer:

for (let i = 0; i < 3; i++) { ... }

Директива continue также может быть использована с меткой. В этом случае управление перейдёт на следующую итерацию цикла с меткой.

Метки не позволяют «прыгнуть» куда угодно

Метки не дают возможности передавать управление в произвольное место кода.

Например, нет возможности сделать следующее:

break label; // не прыгает к метке ниже

label: for (...)

Директива break должна находиться внутри блока кода. Технически, подойдет любой маркированный блок кода, например:

label: {

// ...

break label; // работает

// ...

}

…Хотя в 99.9% случаев break используется внутри циклов, как мы видели в примерах выше.

К слову, continue возможно только внутри цикла.

[Итого](https://learn.javascript.ru/while-for?ysclid=lf7vti3zzn981852309#itogo)

Мы рассмотрели 3 вида циклов:

while – Проверяет условие перед каждой итерацией.

do..while – Проверяет условие после каждой итерации.

for (;;) – Проверяет условие перед каждой итерацией, есть возможность задать дополнительные настройки.

Чтобы организовать бесконечный цикл, используют конструкцию while (true). При этом он, как и любой другой цикл, может быть прерван директивой break.

Если на данной итерации цикла делать больше ничего не надо, но полностью прекращать цикл не следует – используют директиву continue.

Обе этих директивы поддерживают метки, которые ставятся перед циклом. Метки – единственный способ для break/continue выйти за пределы текущего цикла, повлиять на выполнение внешнего.

Заметим, что метки не позволяют прыгнуть в произвольное место кода, в JavaScript нет такой возможности.

Первое правило Junior Front-End разработчика, и разработчика вообще - это выполнение задач от руководства или заказчиков.  
Использование конкретных инструментов - jQuery, JavaScript, Angular, Brainf\*ck - задается требованиями конкретного руководителя или заказчика, либо остается на усмотрение разработчика (если пишем с нуля), который в любом случае смотрит на задачу, и никогда не определяется числом голосов в интернет-холиварах, где всегда одни за легкое и простое, другие за глубокое, универсальное, но тяжелое, и все друг друга ненавидят!  
  
Поэтому, чтобы понять куда двигаться дальше, необходимо изучить эти самые требования руководства и заказчиков, никто в здравом уме не будет против того, что вы лично поинтересуетесь, чем они занимаются и как к ним устроиться, а не ограничитесь текстом вакансии или "треда" на фриланс-бирже.