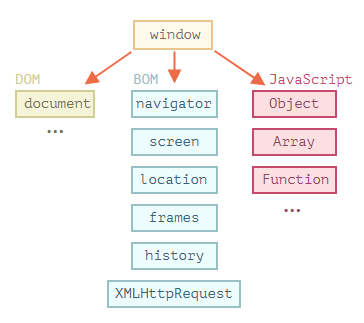
Язык JavaScript изначально был создан для веб-браузеров. Но с тех пор он значительно эволюционировал и превратился в кроссплатформенный язык программирования для решения широкого круга задач.

Сегодня JavaScript может использоваться в браузере, на веб-сервере или в какой-то другой среде, даже в кофеварке. Каждая среда предоставляет свой функционал, который спецификация JavaScript называет *окружением*.

Окружение предоставляет свои объекты и дополнительные функции, в дополнение базовым языковым. Браузеры, например, дают средства для управления веб-страницами. Node.js делает доступными какие-то серверные возможности и так далее.

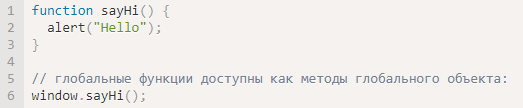
На картинке ниже в общих чертах показано, что доступно для JavaScript в браузерном окружении:



Как мы видим, имеется корневой объект window, который выступает в 2 ролях:

1. Во-первых, это глобальный объект для JavaScript-кода, об этом более подробно говорится в главе [Глобальный объект](https://learn.javascript.ru/global-object).
2. Во-вторых, он также представляет собой окно браузера и располагает методами для управления им.

Например, здесь мы используем window как глобальный объект:



А здесь мы используем window как объект окна браузера, чтобы узнать его высоту:



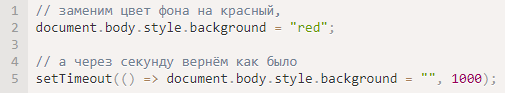
Существует гораздо больше свойств и методов для управления окном браузера. Мы рассмотрим их позднее.

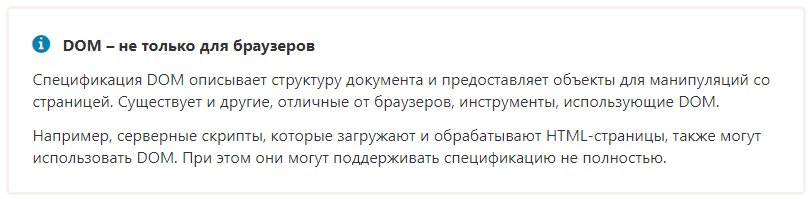
**DOM (Document Object Model)**

Document Object Model, сокращённо DOM – объектная модель документа, которая представляет все содержимое страницы в виде объектов, которые можно менять.

Объект document – основная «входная точка». С его помощью мы можем что-то создавать или менять на странице.

Например:





# **DOM-дерево**

Основой HTML-документа являются теги.

В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

Все эти объекты доступны при помощи JavaScript, мы можем использовать их для изменения страницы.

Например, document.body – объект для тега <body>.

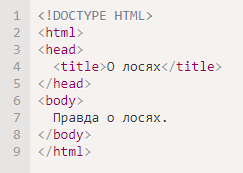
Если запустить этот код, то <body> станет красным на 3 секунды:



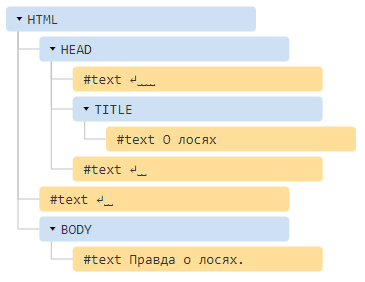
Это был лишь небольшой пример того, что может DOM. Скоро мы изучим много способов работать с DOM, но сначала нужно познакомиться с его структурой.

**Пример DOM**

Начнём с такого, простого, документа:



DOM – это представление HTML-документа в виде дерева тегов. Вот как оно выглядит:



Каждый узел этого дерева – это объект.

Теги являются *узлами-элементами* (или просто элементами). Они образуют структуру дерева: <html> – это корневой узел, <head> и <body> его дочерние узлы и т.д.

Текст внутри элементов образует *текстовые узлы*, обозначенные как #text. Текстовый узел содержит в себе только строку текста. У него не может быть потомков, т.е. он находится всегда на самом нижнем уровне.

Например, в теге <title> есть текстовый узел "О лосях".

Обратите внимание на специальные символы в текстовых узлах:

* перевод строки: ↵ (в JavaScript он обозначается как \n)
* пробел: ␣

Пробелы и переводы строки – это полноправные символы, как буквы и цифры. Они образуют текстовые узлы и становятся частью дерева DOM. Так, в примере выше в теге <head> есть несколько пробелов перед <title>, которые образуют текстовый узел #text (он содержит в себе только перенос строки и несколько пробелов).

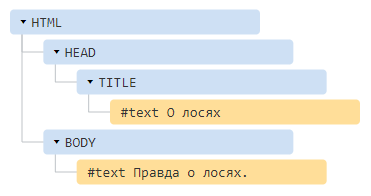
Существует всего два исключения из этого правила:

1. По историческим причинам пробелы и перевод строки перед тегом <head> игнорируются
2. Если мы записываем что-либо после закрывающего тега </body>, браузер автоматически перемещает эту запись в конец body, поскольку спецификация HTML требует, чтобы всё содержимое было внутри <body>. Поэтому после закрывающего тега </body> не может быть никаких пробелов.

В остальных случаях всё просто – если в документе есть пробелы (или любые другие символы), они становятся текстовыми узлами дерева DOM, и если мы их удалим, то в DOM их тоже не будет.

Здесь пробельных текстовых узлов нет:



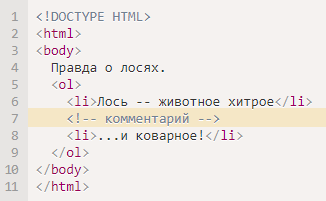


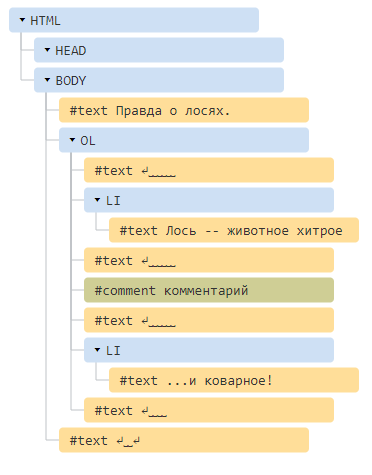


## Другие типы узлов

Есть и некоторые другие типы узлов, кроме элементов и текстовых узлов.

Например, узел-комментарий:





Здесь мы видим узел нового типа – *комментарий*, обозначенный как #comment, между двумя текстовыми узлами.

Казалось бы – зачем комментарий в DOM? Он никак не влияет на визуальное отображение. Но есть важное правило: если что-то есть в HTML, то оно должно быть в DOM-дереве.

**Все, что есть в HTML, даже комментарии, является частью DOM.**

Даже директива <!DOCTYPE...>, которую мы ставим в начале HTML, тоже является DOM-узлом. Она находится в дереве DOM прямо перед <html>. Мы не будем рассматривать этот узел, мы даже не рисуем его на наших диаграммах, но он существует.

Даже объект document, представляющий весь документ, формально является DOM-узлом.

Существует [12 типов узлов](https://dom.spec.whatwg.org/#node). Но на практике мы в основном работаем с 4 из них:

1. document – «входная точка» в DOM.
2. узлы-элементы – HTML-теги, основные строительные блоки.
3. текстовые узлы – содержат текст.
4. комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна в DOM для чтения JS.

**Взаимодействие с консолью**

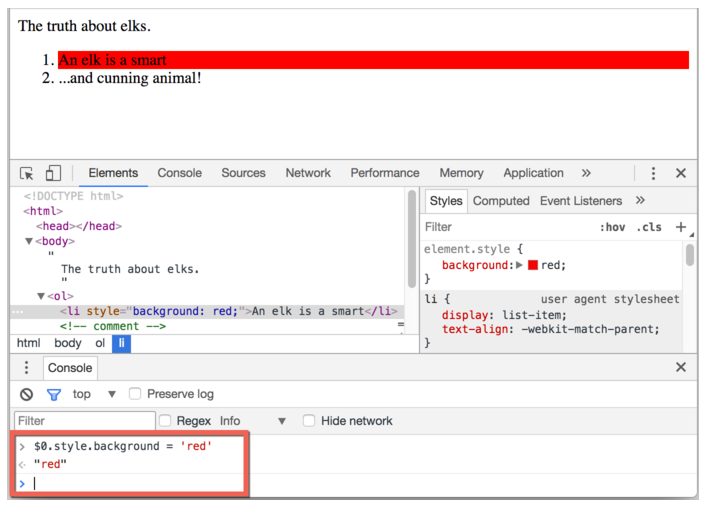
При работе с DOM нам часто требуется применить к нему JavaScript. Например: получить узел и запустить какой-нибудь код для его изменения, чтобы посмотреть результат. Вот несколько подсказок, как перемещаться между вкладками Elements и Console.

Для начала:

1. На вкладке Elements выберите первый элемент <li>.
2. Нажмите Esc – прямо под вкладкой Elements откроется Console.

Последний элемент, выбранный во вкладке Elements, доступен в консоли как $0; предыдущий, выбранный до него, как $1 и т.д.

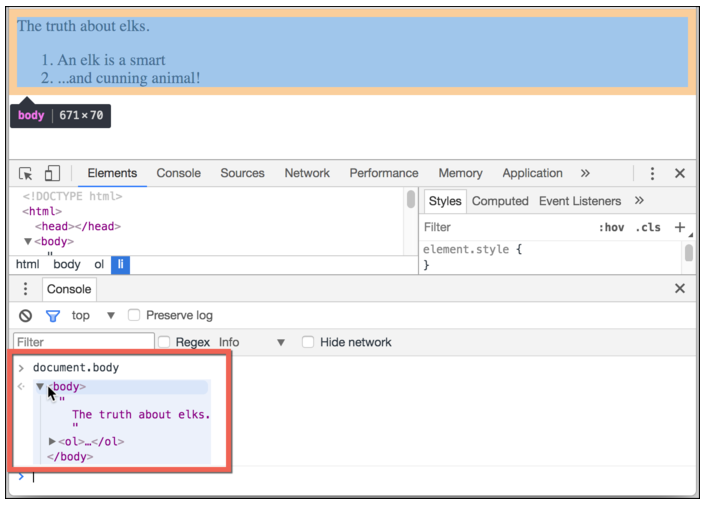
Теперь мы можем запускать на них команды. Например $0.style.background = 'red' сделает выбранный элемент красным, как здесь:



Это мы посмотрели как получить узел из Elements в Console.

Есть и обратный путь: если есть переменная node, ссылающаяся на DOM-узел, можно использовать в консоли команду inspect(node), чтобы увидеть этот элемент во вкладке Elements.

Или мы можем просто вывести DOM-узел в консоль и исследовать «на месте», как document.body ниже:



Это может быть полезно для отладки.

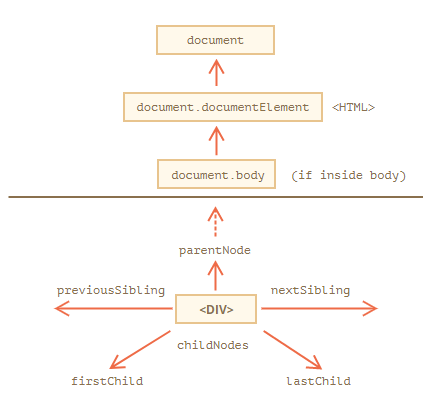
Инструменты разработчика браузера отлично помогают в разработке: мы можем исследовать DOM, пробовать с ним что-то делать и смотреть, что идёт не так.

# **Навигация по DOM-элементам**

DOM позволяет нам делать что угодно с элементами и их содержимым, но для начала нужно получить соответствующий DOM-объект.

Все операции с DOM начинаются с объекта document. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу.

Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



## Сверху: documentElement и body

Самые верхние элементы дерева доступны как свойства объекта document:

**<html> = document.documentElement**

Самый верхний узел документа: document.documentElement. В DOM он соответствует тегу <html>.

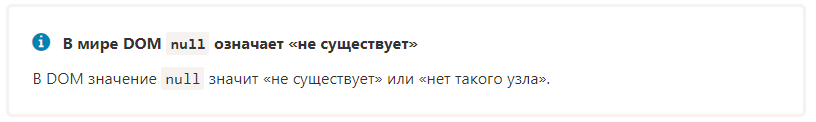
**<body> = document.body**

Другой часто используемый DOM-узел – узел тега <body>: document.body.

**<head> = document.head**

Тег <head> доступен как document.head.



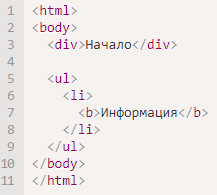


**Дети: childNodes, firstChild, lastChild**

Здесь и далее мы будем использовать два принципиально разных термина:

* **Дочерние узлы (или дети)** – элементы, которые являются непосредственными детьми узла. Другими словами, элементы, которые лежат непосредственно внутри данного. Например, <head> и <body> являются детьми элемента <html>.
* **Потомки** – все элементы, которые лежат внутри данного, включая детей, их детей и т.д.

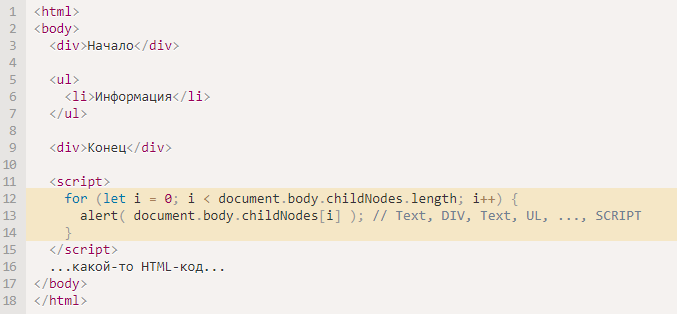
В примере ниже детьми тега <body> являются теги <div> и <ul> (и несколько пустых текстовых узлов):



…А потомки <body>– это и прямые дети <div>, <ul> и вложенные в них: <li> (потомок <ul>) и <b> (потомок <li>) – в общем, все элементы поддерева.

**Коллекция childNodes содержит список всех детей, включая текстовые узлы.**

Пример ниже последовательно выведет детей document.body:



Обратим внимание на маленькую деталь. Если запустить пример выше, то последним будет выведен элемент <script>. На самом деле, в документе есть ещё «какой-то HTML-код», но на момент выполнения скрипта браузер ещё до него не дошёл, поэтому скрипт не видит его.

**Свойства firstChild и lastChild обеспечивают быстрый доступ к первому и последнему дочернему элементу.**

Они, по сути, являются всего лишь сокращениями. Если у тега есть дочерние узлы, условие ниже всегда верно:



Для проверки наличия дочерних узлов существует также специальная функция elem.hasChildNodes().

### **DOM-коллекции**

Как мы уже видели, childNodes похож на массив. На самом деле это не массив, а коллекция – особый перебираемый объект-псевдомассив.

И есть два важных следствия из этого:

1. Для перебора коллекции мы можем использовать for..of:



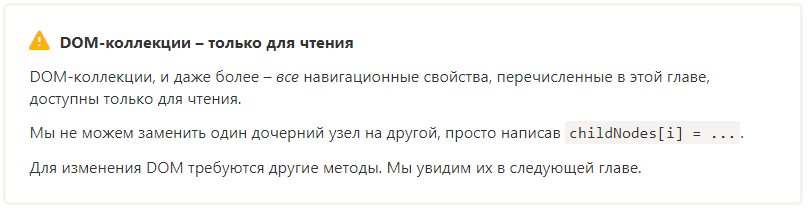
Это работает, потому что коллекция является перебираемым объектом (есть требуемый для этого метод Symbol.iterator).

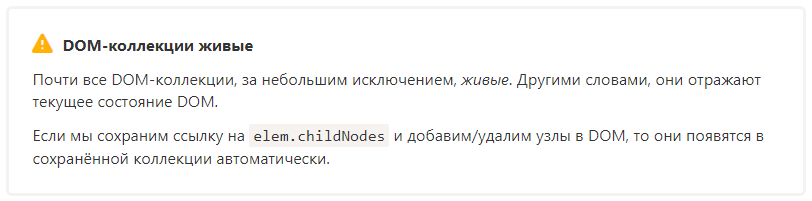
1. Методы массивов не будут работать, потому что коллекция – это не массив:

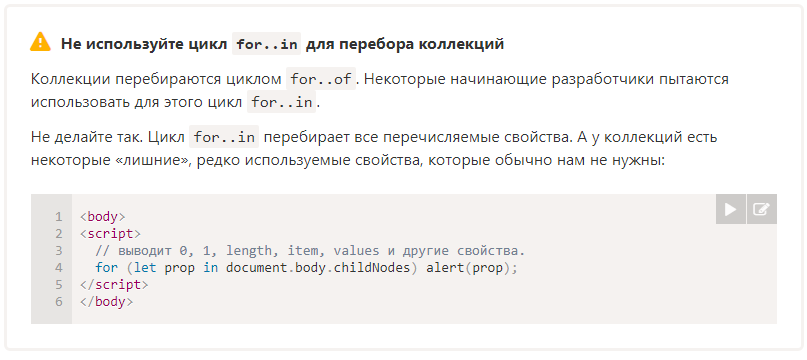


Первый пункт – это хорошо для нас. Второй – бывает неудобен, но можно пережить. Если нам хочется использовать именно методы массива, то мы можем создать настоящий массив из коллекции, используя Array.from:









## Соседи и родитель

Соседи – это узлы, у которых один и тот же родитель.

Например, здесь <head> и <body> соседи:

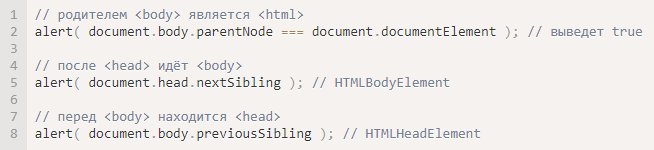


* говорят, что <body> – «следующий» или «правый» сосед <head>
* также можно сказать, что <head> «предыдущий» или «левый» сосед <body>.

Следующий узел того же родителя (следующий сосед) – в свойстве nextSibling, а предыдущий – в previousSibling.

Родитель доступен через parentNode.

Например:

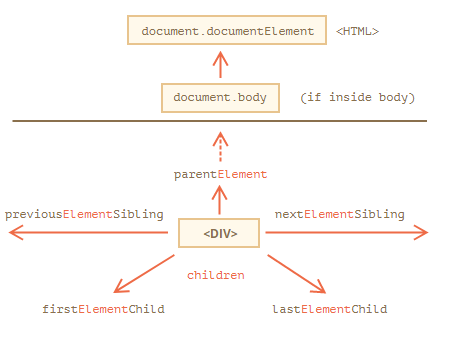


## Навигация только по элементам

Навигационные свойства, описанные выше, относятся ко всем узлам в документе. В частности, в childNodes находятся и текстовые узлы и узлы-элементы и узлы-комментарии, если они есть.

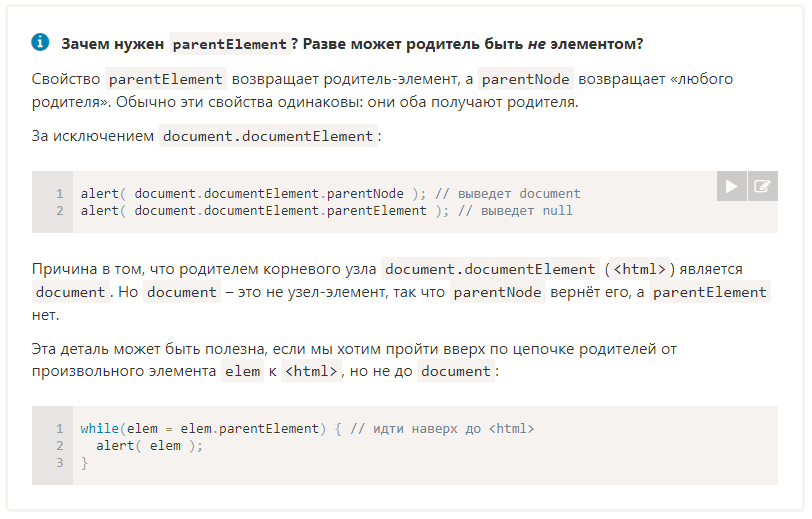
Но для большинства задач текстовые узлы и узлы-комментарии нам не нужны. Мы хотим манипулировать узлами-элементами, которые представляют собой теги и формируют структуру страницы.

Поэтому давайте рассмотрим дополнительный набор ссылок, которые учитывают только узлы-элементы:

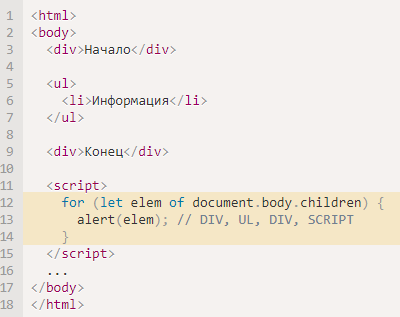


Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово Element:

* children – коллекция детей, которые являются элементами.
* firstElementChild, lastElementChild – первый и последний дочерний элемент.
* previousElementSibling, nextElementSibling – соседи-элементы.
* parentElement – родитель-элемент.



Изменим один из примеров выше: заменим childNodes на children. Теперь цикл выводит только элементы:



## Ещё немного ссылок: таблицы

До сих пор мы описывали основные навигационные ссылки.

Некоторые типы DOM-элементов предоставляют для удобства дополнительные свойства, специфичные для их типа.

Таблицы – отличный пример таких элементов.

**Элемент <table>**, в дополнение к свойствам, о которых речь шла выше, поддерживает следующие:

* table.rows – коллекция строк <tr> таблицы.
* table.caption/tHead/tFoot – ссылки на элементы таблицы <caption>, <thead>, <tfoot>.
* table.tBodies – коллекция элементов таблицы <tbody> (по спецификации их может быть больше одного).

**<thead>, <tfoot>, <tbody>** предоставляют свойство rows:

* tbody.rows – коллекция строк <tr> секции.

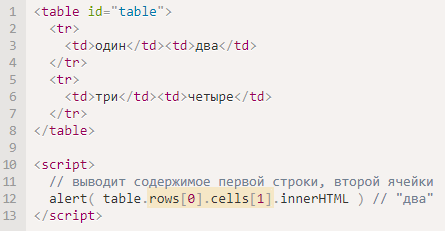
**<tr>:**

* tr.cells – коллекция <td> и <th> ячеек, находящихся внутри строки <tr>.
* tr.sectionRowIndex – номер строки <tr> в текущей секции <thead>/<tbody>/<tfoot>.
* tr.rowIndex – номер строки <tr> в таблице (включая все строки таблицы).

**<td> and <th>:**

* td.cellIndex – номер ячейки в строке <tr>.

Пример использования:



## Чем может быть полезна библиотека jQuery?

Во время создания сайтов используются три языка — HTML, CSS, Javascript. Первый применяется для разметки страниц, второй нужен для стилизации, а третий — добавляет ресурсу интерактивности. При прокрутке страниц или кликах по кнопкам может воспроизводиться анимация либо другие действия, заданные с помощью Javascript.

Что же такое Jquery? Это сторонняя библиотека для JS, помогающая уменьшить количество необходимого кода, параллельно увеличив функционал языка. Библиотека кроссбраузерная, поэтому код будет корректно отображаться в любом современном браузере.

Чтобы увидеть, насколько Jquery сокращает количество необходимого кода, посмотрите на такой простой пример:

#Javascript

document.getElementById("someBlock").innerHTML = "Hello, World!";

#Jquery

$("#someBlock").html("Hello, World!");

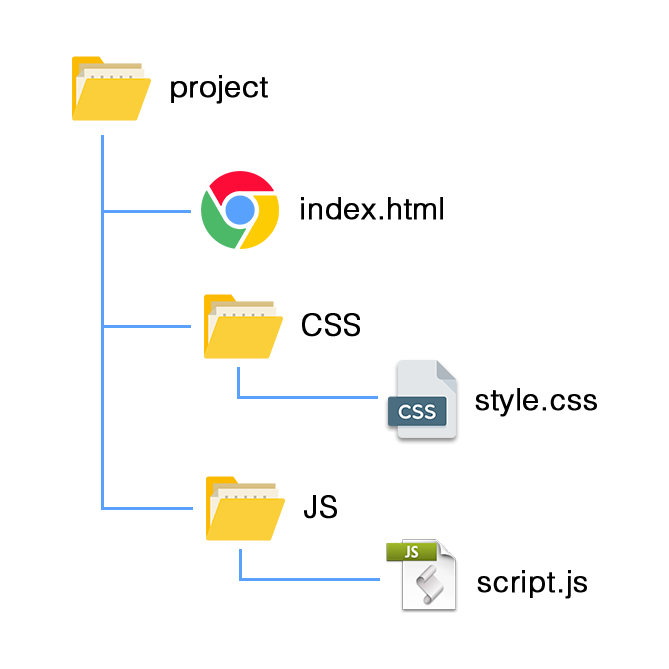
Разница получается существенная. Что вам потребуется для воплощения наших советов в жизнь? Во-первых, нужно иметь базовые знания HTML, CSS, понимать принцип работы селекторов. Во-вторых, пригодятся базовые навыки программирования. С Jquery можно работать, даже не зная хорошо Javascript. Однако, нужно понимать, что такое переменные, типы данных. Да и логическое мышление будет не лишним вместе со знаниями основ математики.

## Как установить и настроить JQuery?

По своему принципу библиотека является JS файлом. Его нужно подключить с помощью тега вашей HTML-странички. Это можно сделать двумя способами:

* Скачать скрипт с [официального сайта](https://jquery.com/) и добавить в папку проекта;
* Использовать ссылку на файл из CDN. Такое подключение скрипта будет работать быстрее локального.

Для начала нам нужно создать папку для проекта. Внутри мы размещаем файл HTML-разметки index.html. Также нам понадобится каталог css для стилевого файла style.css, а также js – для script.js. Структура папки проекта будет такой:



HTML-страничку нужно разметить. Самый простой каркас для сайта выглядит вот так:

<!doctype html>

<html lang="en">

<head>

<title>Jquery Demo</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

</body>

</html>

Перед закрывающим тегом <body> стоит вставить тег <script> cо ссылкой на CDN, а также второй <script> с вашим JS-файлом. Теперь ваш файл будет выглядеть вот так:

<!doctype html>

<html lang="en">

<head>

<title>Jquery Demo</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"

integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="

crossorigin="anonymous">

</script>

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

</body>

</html>

Обратите внимание, что пользовательский скрипт подключается после библиотеки, иначе он не будет работать. Если использовать загруженный скрипт Jquery, то файл jquery.min.js поместите в папку js. После этого вы получите полный доступ к API библиотеки.

### **Как использовать Jquery?**

Библиотека использует объектную модель документа DOM для взаимодействия с тегами HTML-страницы. Структуру DOM можно увидеть с помощью отладчика в браузере. Его можно открыть с помощью клавиши F12. Вы увидите дерево HTML-разметки — это и есть DOM. Каждый тег является объектом, с которым JS может взаимодействовать. Все объекты расположены в виде древовидной структуры. Каждый вложенный элемент или потомок является веткой корневого тега <html>. С помощью языка Javascript можно вносить изменения в разметку.

Код страницы можно увидеть еще и с помощью клика правой кнопкой мыши. Выберите пункт «Просмотр кода страницы». Код будет очень похож на тот, который показывал отладчик, но это не так. Через «Просмотр кода страницы» вы видите разметку, написанную в HTML-файле, а отладчик показывает его в виде DOM-структуры. Она динамична, поскольку JS может менять ее. А в разметку скрипт изменений вносить не может.

Тег <html> является наружной нодой в структуре DOM, поэтому это объект document. Для управления содержимым страницы с помощью Jquery, нужно обратиться к данному объекту. Внутри файла scripts.js напишите вот такой код:

$(document).ready(function() {

//тело функции

});

В тело функции можно писать код библиотеки. Jquery проверит его, ведь он будет выполнен только если DOM сможет загрузить скрипты. Иногда они могут не выполняться до тех пор, пока не загрузится разметка и стили.

Давайте потренируемся применять Jquery на примере вывода всем известной фразы Hello, World! Ее нужно выводить в какой-либо тег, который мы и создадим внутри файла index.html:

...

<body>

<p id="demo"></p>

</body>

...

Вызов Jquery осуществляется с помощью символа $. Действие библиотеки обычно определяется методом. Jquery получает доступ к объектной модели документа DOM посредством CSS-синтаксиса. Вызов библиотеки будет выглядеть вот так:

$(“selector”).method();

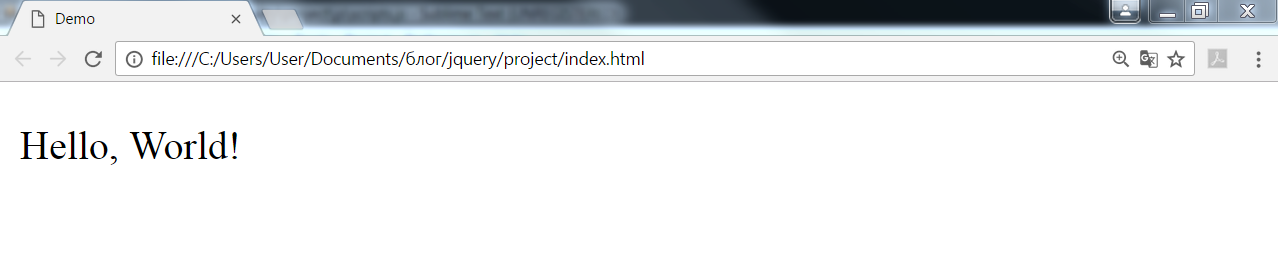
В нашем случае в качестве селектора будет выступать идентификатор demo. Обратиться к нему можно через #demo. Мы будем использовать метод .html() чтобы внести изменения в разметку страницы: например, вставить желаемый текст. В скрипт пропишите:

$(document).ready(function() {

$("#demo").html("Hello, World!");

});

Сохраните изменения в файле, нажав Ctrl+S и откройте в браузере файл index.html. На экране появится надпись Hello, World!



Посмотреть изменения в DOM можно нажав Ctrl+Shift+I. Вы увидите, что в ваш тег

прописалась фраза:

<p id="demo">Hello, World!</p>

Но если открыть в браузере исходный код с помощью Ctrl+U, то нашей фразы внутри разметки не будет. Вы увидите HTML-код без форматирования:

<p id="demo"></p>

### **Какие можно селекторы использовать?**

Они используются для того, чтобы сообщить библиотеке Jquery с какими элементами на странице нужно работать. Большая часть селекторов похожа на таковые в CSS с некоторыми отличиями. Обращение к селектору такое:

$(“selector”)

Вы можете использовать двойные кавычки, следуя спецификации библиотеки, либо же ставить одинарные. Чаще всего используются вот такие селекторы:

* $(this) или Current – это выбор текущего элемента;
* $(“\*”) либо Wildcard применяется для выбора всех элементов, расположенных на странице;
* $(“#id”) - обращение к уникальному идентификатору;
* $(“.class”) - таким образом можно выбрать несколько блоков с одинаковым классом;
* $(“div”) служит для обращения к тегу <div>;
* $(“li:first-child”) обратится к первому элементу <li> в списке <ul>;
* $(“[type='text']”) - это обращение по атрибуту.

### **Знакомство с событиями в Jquery**

В предыдущем примере с выводом на экран фразы Hello, World! наш код запускался сразу при загрузке страницы. Взаимодействия с пользователем не требовалось, поэтому текст можно было бы просто вписать в разметку, не используя библиотеку Jquery. Однако она может гораздо больше, например, делать вывод сообщения после нажатия на кнопку.

Первым делом нужно будет в файл разметки index.html добавить кнопку с помощью тега <button>.

...

<body>

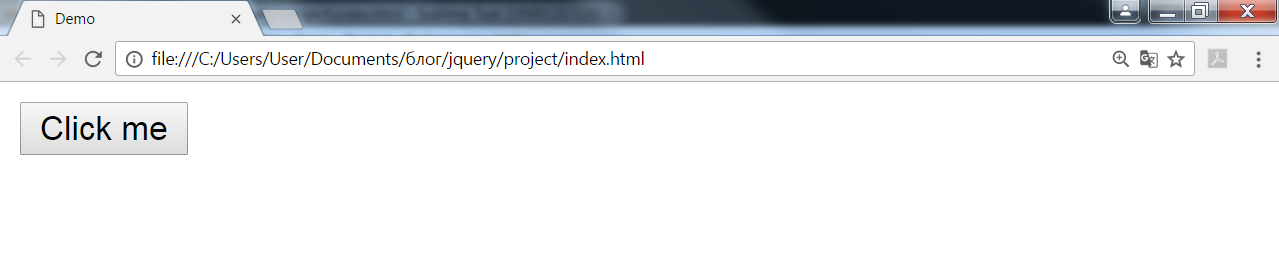
<button id="button">Click me</button>

<p id="demo"></p>

</body>

...

Вот что у вас получится, если открыть страничку в браузере:



Чтобы настроить действие на клик, будет использовать одноименный метод click(). С его помощью можно вызвать функцию вывода надписи Hello, World!

$(document).ready(function(){

$("#button").click();

});

Мы присвоили кнопке идентификатор button, к которому можно обратиться посредством селектора $(“#button”). Метод click() способен отслеживать сделанные клики. Внутри него нужно создать функцию для вывода надписи:

function(){

$("#demo").html("Hello, World!");

}

У нас получится вот такой код внутри файла скриптов:

$(document).ready(function() {

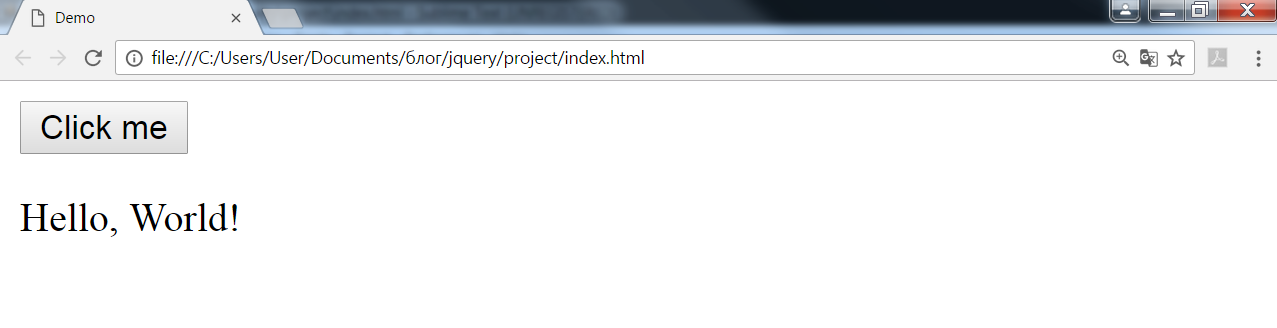
$("#button").click(function() {

$("#demo").html("Hello, World!");

});

});

Не забудьте сохранить изменения, а потом обновить страницу в браузере, чтобы увидеть результат. Теперь при клике мышью по кнопке на экран будет выводиться надпись Hello, World!



Событием называют любое взаимодействие пользователя с браузером посредством ввода данных с клавиатуры или мышкой.

Какие же события используются чаще всего? Это:

* hover() применяется для выполнения действия в момент наведения курсора на любой элемент. Можно также использовать специальные методы для мыши — mouseenter() и mouseleave(). Первый отвечает за наведение курсора, а второй — за покидание;
* click() следит за кликами на странице. Событие может вам пригодиться для создания выпадающего меню;
* scroll() выполняется во время прокрутки экрана. События можно использовать для того, чтобы, например, показывать изображения при скроллинге странички;
* submit() нужен для выполнения действий при заполнении формы;
* keydown() отслеживает нажатия клавиш. Таким образом вы сможете закрыть диалоговое окно нажатием кнопки Esc.

При помощи данных событий вы сможете сделать действительно интерактивный контент для вашего ресурса.

### **Какие можно использовать эффекты?**

Их обычно применяют для создания анимации на странице. Эффекты могут работать вместе с событиями. Мы с вами создадим наплывающий элемент, которым можно открывать и закрывать. В таком случае нам потребуются два класса — для открытия, закрытия. Предыдущие теги кнопки и параграф можно удалить из index.html. Взамен напишите вот такой код:

...

<body>

<button class="button">Open</button>

<section class="hidden">

<button class="button">Close</button>

</section>

</body>

...

Вот теперь нам как раз и пригодится ранее созданный файл стилей style.css. Для того, чтобы спрятать элемент с классом hidden, пропишите для него свойство display со значением none. Мы расположим наш тег <section> в центре экрана для наглядности. В файл style.css пропишите:

.hidden {

display: none;

position: fixed;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

height: 200px;

width: 200px;

background: #b1b1b1;

}

Чтобы переключать видимость элемента с none на block нам потребуется метод toggle(). Его мы и пропишем в нашем файле scripts.js:

$(document).ready(function() {

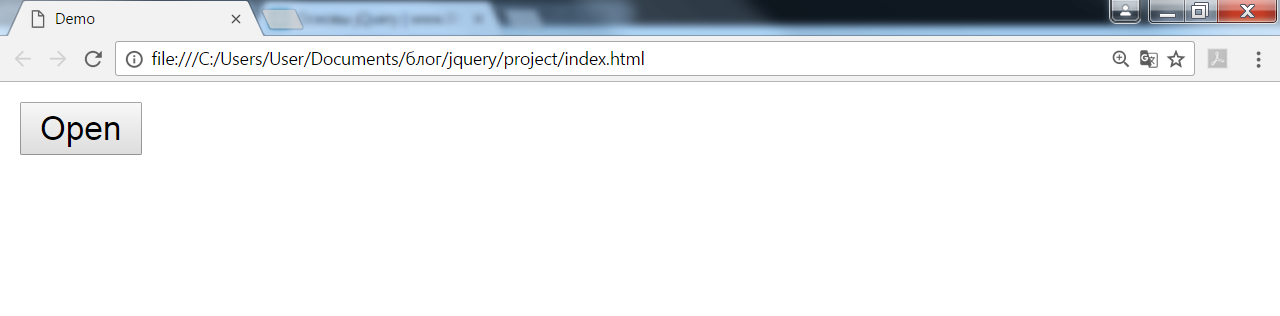
$(".button").click(function() {

$(".hidden ").toggle();

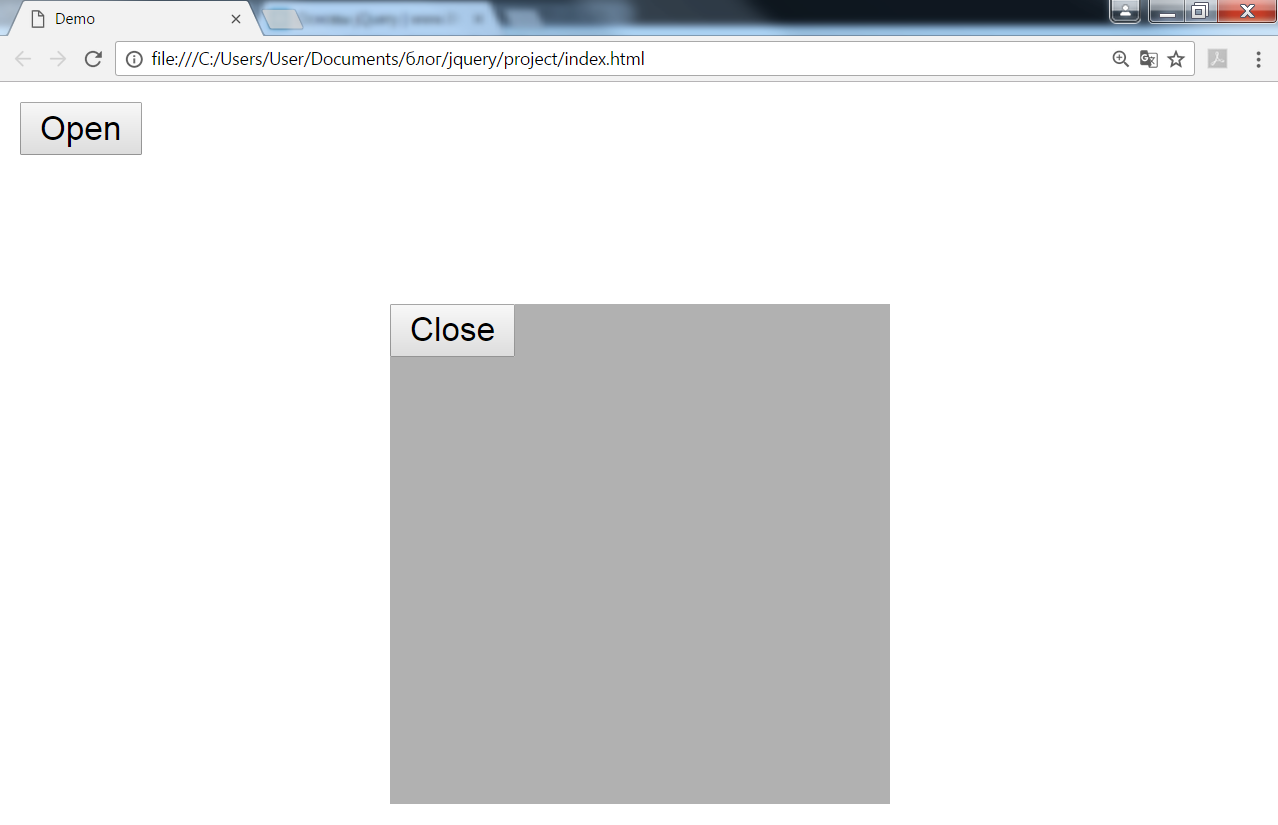
});

});

Не забудьте обновить файл index.html. Вот что у нас получится до клика по кнопке:



И после клика:



Вам не обязательно применять именно метод toggle(), взамен можете использовать slideToggle() либо fadeToggle(). С их помощью можно будет сделать интересные эффекты.

Какие из них самые востребованные? Это:

* toggle() управляет видимостью элементов. Метод может скрывать hide() или показывать show() элементы;
* slideToggle() меняет видимость с эффектом скольжения. С ним вместе работают методы slideUp() и slideDown();
* fadeToggle() управляет видимостью за счет прозрачности. Данный метод связывается с fadeIn(), а также fadeOut();
* animate() воспроизводит кастомные анимационные эффекты.

Главное запомнить, что события библиотеки Jquery могут обеспечивать взаимодействие пользователя с ресурсом, а эффекты управляют элементами, когда вышеописанное действие уже происходит.