1.

http:// это протокол. Он отображает, какой протокол браузер должен использовать. Обычно это HTTP-протокол или его безопасная версия - HTTPS. Интернет требует эти 2 протокола, но браузеры часто могут использовать и другие протоколы, например mailto: (чтобы открыть почтовый клиент) или ftp: для запуска передачи файлов, так что не стоит удивляться, если вы вдруг увидите другие протоколы.

****

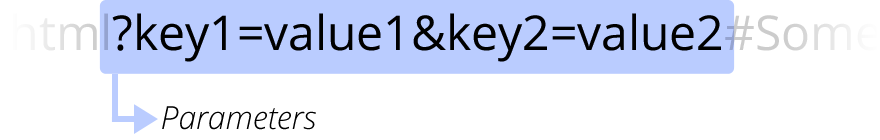
www.example.com это доменное имя. Оно означает, какой веб-сервер должен быть запрошен. В качестве альтернативы может быть использован и [IP-адрес](https://developer.mozilla.org/ru/docs/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C/IP_Address), но это делается редко, поскольку запоминать IP сложнее, и это не популярно в интернете.

****

:80 это порт. Он отображает технический параметр, используемый для доступа к ресурсам на веб-сервере. Обычно подразумевается, что веб-сервер использует стандартные порты HTTP-протокола (80 для HTTP и 443 для HTTPS) для доступа к своим ресурсам. В любом случае, порт - это факультативная составная часть URL.

****

/path/to/myfile.html это адрес ресурса на веб-сервере. В прошлом, адрес отображал местоположение реального файла в реальной директории на веб-сервере. В наши дни это чаще всего абстракция, позволяющая обрабатывать адреса и отображать тот или иной контент из баз данных.

****

?key1=value1&key2=value2 это дополнительные параметры, которые браузер сообщает веб-серверу. Эти параметры - список пар ключ/значение, которые разделены символом &. Веб-сервер может использовать эти параметры для исполнения дополнительных команд перед тем как отдать ресурс. Каждый веб-сервер имеет свои собственные правила обработки этих параметров и узнать их можно, только спросив владельца сервера.

****

#SomewhereInTheDocument это якорь на другую часть того же самого ресурса. Якорь представляет собой вид "закладки" внутри ресурса, которая переадресовывает браузер на "заложенную" часть ресурса. В HTML-документе, например, браузер может переместиться в точку, где установлен якорь; в видео- или аудио-документе браузер может перейти к времени, на которое ссылается якорь. Важно отметить, что часть URL после #, которая также известна как идентификатор фрагмента, никогда не посылается на сервер вместе с запросом.

2. Браузер и веб-сервер взаимодействуют по технологии клиент-сервер. После ввода адреса в строку адреса браузер формирует запрос к серверу. Сервер формирует страницу и передает ее браузеру. Браузер выводит страницу пользователю, который своими действиями формирует новый запрос.

3.

1. Стартовая строка (англ. Starting line) — определяет тип сообщения;
2. Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Text, application

Description automatically generated

Рис. 1. Структура протокола HTTP (дамп пакета, полученный сниффером Wireshark)

Стартовая строка HTTP

Cтартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать.

Стартовые строки различаются для запроса и ответа. **Строка запроса** выглядит так:

Метод URI HTTP/Версия протокола

Пример запроса:

GET /web-programming/index.html HTTP/1.1

Стартовая **строка ответа** сервера имеет следующий формат:

HTTP/Версия КодСостояния [Пояснение]

Например, на предыдущий наш запрос клиентом данной страницы сервер ответил строкой:

HTTP/1.1 200 Ok

Методы протокола

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами (Табл. 1). Названия метода чувствительны к регистру.

Таблица 1. Методы протокола HTTP

|  |  |
| --- | --- |
| **Метод** | **Краткое описание** |
| **OPTIONS** | Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера.  Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «\*». Запросы «OPTIONS \* HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.  Результат выполнения этого метода не кэшируется. |
| **GET** | Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»: GET /path/resource?param1=value1¶m2=value2 HTTP/1.1  Согласно стандарту HTTP, запросы типа GET считаются идемпотентными[4] — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.  Кроме обычного метода GET, различают ещё условный GET и частичный GET. Условные запросы GET содержат заголовки If-Modified-Since, If-Match, If-Range и подобные. Частичные GET содержат в запросе Range. Порядок выполнения подобных запросов определён стандартами отдельно. |
| **HEAD** | Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.  Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая. |
| **POST** | Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы.  В отличие от метода GET, метод POST не считается идемпотентным[4], то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).  При результатах выполнения 200 (Ok) и 204 (No Content) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.  Сообщение ответа сервера на выполнение метода POST не кэшируется. |
| **PUT** | Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки Content-\* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки 501 (Not Implemented).  Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствуют находящемуся по данному URI ресурсу.  Сообщения ответов сервера на метод PUT не кэшируются. |
| **PATCH** | Аналогично PUT, но применяется только к фрагменту ресурса. |
| **DELETE** | Удаляет указанный ресурс. |
| **TRACE** | Возвращает полученный запрос так, что клиент может увидеть, что промежуточные сервера добавляют или изменяют в запросе. |
| **LINK** | Устанавливает связь указанного ресурса с другими. |
| **UNLINK** | Убирает связь указанного ресурса с другими. |

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он не применим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Наиболее востребованными являются методы GET и POST — на человеко-ориентированных ресурсах, POST — роботами поисковых машин и оффлайн-браузерами.

**Прокси-сервер**

Прокси - это транзитный сервер, перенаправляющий HTTP-трафик. Прокси-серверы используются для ускорения выполнения запросов путем кэширования веб-страниц. В локальной сети применяется как межсетевой экран и средство управления HTTP-трафиком (например, для блокирования доступа к некоторым ресурсам). В Интернете прокси часто используют для анонимизации запросов - в этом случае веб-сервер получает ip-адрес прокси-сервера, а не реального клиента. В современных браузерах можно задать целый список прокси и переключаться между серверами из этого списка по мере необходимости (обычно такая возможность доступна через расширения или плагины браузера).

Коды состояния

**Код состояния** информирует клиента о результатах выполнения запроса и определяет его дальнейшее поведение. Набор кодов состояния является стандартом, и все они описаны в соответствующих документах RFC.

Каждый код представляется целым трехзначным числом. Первая цифра указывает на класс состояния, последующие - порядковый номер состояния (рис 1.). За кодом ответа обычно следует краткое описание на английском языке.

Введение новых кодов должно производиться только после согласования с IETF. Клиент может не знать все коды состояния, но он обязан отреагировать в соответствии с классом кода.

Применяемые в настоящее время классы кодов состояния и некоторые примеры ответов сервера приведены в табл. 2.

Таблица 2. Коды состояния протокола HTTP

|  |  |
| --- | --- |
| **Класс кодов** | **Краткое описание** |
| **1xx Informational**(Информационный) | В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. [Прокси-сервера](https://iit-web-lectures.readthedocs.io/ru/latest/www/http.html#proxy) подобные сообщения должны отправлять дальше от сервера к клиенту.  Примеры ответов сервера:  100 Continue (Продолжать)  101 Switching Protocols (Переключение протоколов)  102 Processing (Идёт обработка) |
| **2xx Success**(Успешно) | Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.  Примеры ответов сервера:  200 OK (Успешно).  201 Created (Создано)  202 Accepted (Принято)  204 No Content (Нет содержимого)  206 Partial Content (Частичное содержимое) |
| **3xx** Redirection (Перенаправление) | Коды статуса класса 3xx сообщают клиенту, что для успешного выполнения операции нужно произвести следующий запрос к другому URI. В большинстве случаев новый адрес указывается в поле Location заголовка. Клиент в этом случае должен, как правило, произвести автоматический переход (жарг. «редирект»).  Обратите внимание, что при обращении к следующему ресурсу можно получить ответ из этого же класса кодов. Может получиться даже длинная цепочка из перенаправлений, которые, если будут производиться автоматически, создадут чрезмерную нагрузку на оборудование. Поэтому разработчики протокола HTTP настоятельно рекомендуют после второго подряд подобного ответа обязательно запрашивать подтверждение на перенаправление у пользователя (раньше рекомендовалось после 5-го). За этим следить обязан клиент, так как текущий сервер может перенаправить клиента на ресурс другого сервера. Клиент также должен предотвратить попадание в круговые перенаправления.  Примеры ответов сервера:  300 Multiple Choices (Множественный выбор)  301 Moved Permanently (Перемещено навсегда)  304 Not Modified (Не изменялось) |
| **4xx Client Error**(Ошибка клиента) | Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.  Примеры ответов сервера:  401 Unauthorized (Неавторизован)  402 Payment Required (Требуется оплата)  403 Forbidden (Запрещено)  404 Not Found (Не найдено)  405 Method Not Allowed (Метод не поддерживается)  406 Not Acceptable (Не приемлемо)  407 Proxy Authentication Required (Требуется аутентификация прокси) |
| **5xx Server Error**(Ошибка сервера) | Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.  Примеры ответов сервера:  500 Internal Server Error (Внутренняя ошибка сервера)  502 Bad Gateway (Плохой шлюз)  503 Service Unavailable (Сервис недоступен)  504 Gateway Timeout (Шлюз не отвечает) |

Заголовки HTTP

**Заголовок HTTP** (HTTP Header) — это строка в HTTP-сообщении, содержащая разделённую двоеточием пару вида «параметр-значение». Формат заголовка соответствует общему формату заголовков текстовых сетевых сообщений ARPA (RFC 822). Как правило, браузер и веб-сервер включают в сообщения более чем по одному заголовку. Заголовки должны отправляться раньше тела сообщения и отделяться от него хотя бы одной пустой строкой (CRLF).

Название параметра должно состоять минимум из одного печатного символа (ASCII-коды от 33 до 126). После названия сразу должен следовать символ двоеточия. Значение может содержать любые символы ASCII, кроме перевода строки (CR, код 10) и возврата каретки (LF, код 13).

Пробельные символы в начале и конце значения обрезаются. Последовательность нескольких пробельных символов внутри значения может восприниматься как один пробел. Регистр символов в названии и значении не имеет значения (если иное не предусмотрено форматом поля).

Пример заголовков ответа сервера:

Server: Apache/2.2.3 (CentOS)

Last-Modified: Wed, 09 Feb 201117:13:15 GMT

Content-Type: text/html; charset=UTF-8

Accept-Ranges: bytes

Date: Thu, 03 Mar 201104:04:36 GMT

Content-Length: 2945

Age: 51

X-Cache: HIT from proxy.omgtu

Via: 1.0 proxy.omgtu (squid/3.1.8)

Connection: keep-alive

200 OK

Все HTTP-заголовки разделяются на четыре основных группы:

1. General Headers (Основные заголовки) — должны включаться в любое сообщение клиента и сервера.
2. Request Headers (Заголовки запроса) — используются только в запросах клиента.
3. Response Headers (Заголовки ответа) — присутствуют только в ответах сервера.
4. Entity Headers (Заголовки сущности) — сопровождают каждую сущность сообщения.

4.

**<!DOCTYPE html>**

<htmllang="ru">

<head>

<metacharset="UTF-8">

<title>Моя первая страница</title>

</head>

<body>

</body>

</html>

## Парный тег head

Тег служит для хранения служебной информации. Здесь возможны самые разные сочетания тегов, которые подсказывают браузеру название страницы, описание, ключевые слова и так далее. Такая информация называется метаинформацией. В современном вебе она отвечает не только за служебную информацию для браузера, но и активно используется при продвижении сайта. Поисковые системы считывают всю эту информацию и на основе множества алгоритмов определяют место сайта при разных поисковых запросах.

Любые данные, которые указаны внутри тега head, не видны при отображении страницы в браузере. Это значит, что нет необходимости располагать там информацию, которая предназначена для отображения.

Хоть различной информации внутри head может быть множество, в этом уроке разберём несколько основных тегов, которые пригодятся при создании любой веб-страницы:

Внутри контейнера <head> могут размещаться следующие теги: [**<base>**](https://htmlweb.ru/html/base.php), **<link>**, **<meta>**,[**<script>**](https://htmlweb.ru/java/js.php), [**<style>**](https://htmlweb.ru/css/styles.php#gl_style), **<title>**.

Тег **<meta>** определяет метаданные, которые предназначены для браузеров и поисковых систем. Чаще всего атрибуты любого метатега сводятся к парам «имя=значение».

Тег **<link>** устанавливает связь с внешним файлом, который имеет отношение к текущему html-документу.

Глобальные стили CSS располагаются в контейнере **<style>...</style>**, расположенном в свою очередь в контейнере **<head>...</head>**.

Сценарий JavaScript встраивается в HTML-документ с помощью тега **<script>**.

Тег **<base>** задает базовый адрес (URL) для гипертекстовых ссылок документа, заданных в относительной форме. Кроме того, <base> позволяет способ открытия ссылок по умолчанию.

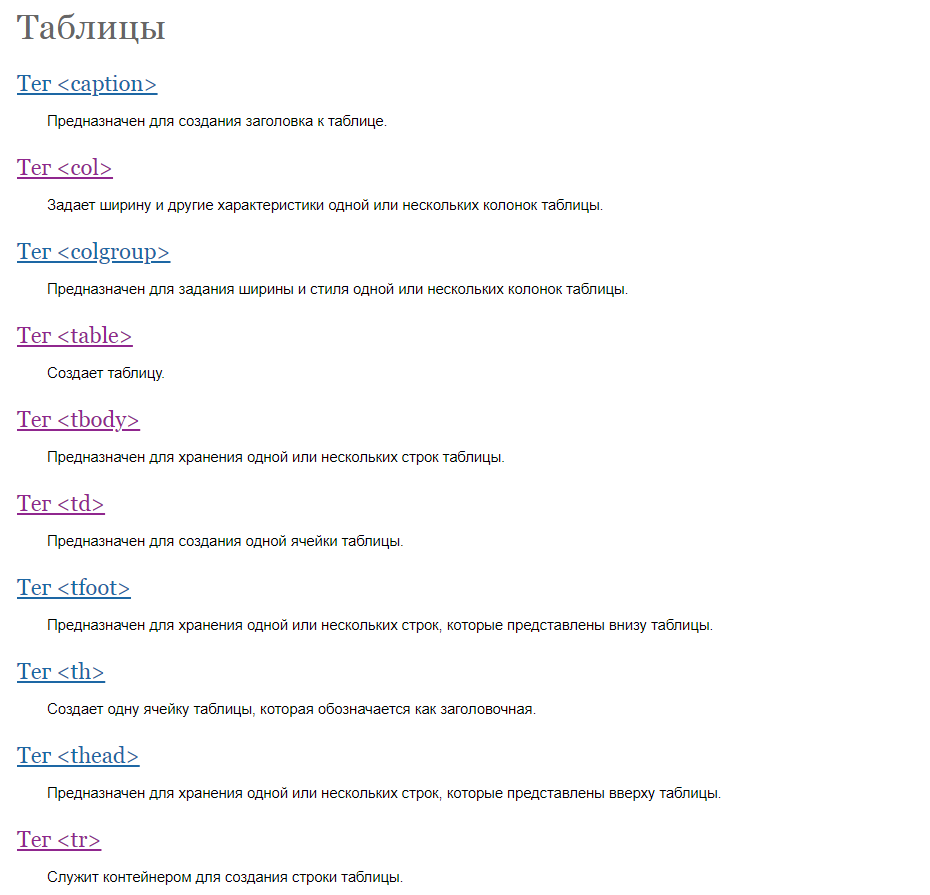
5. HTML. Основные блочные (block) теги. Теги таблицы

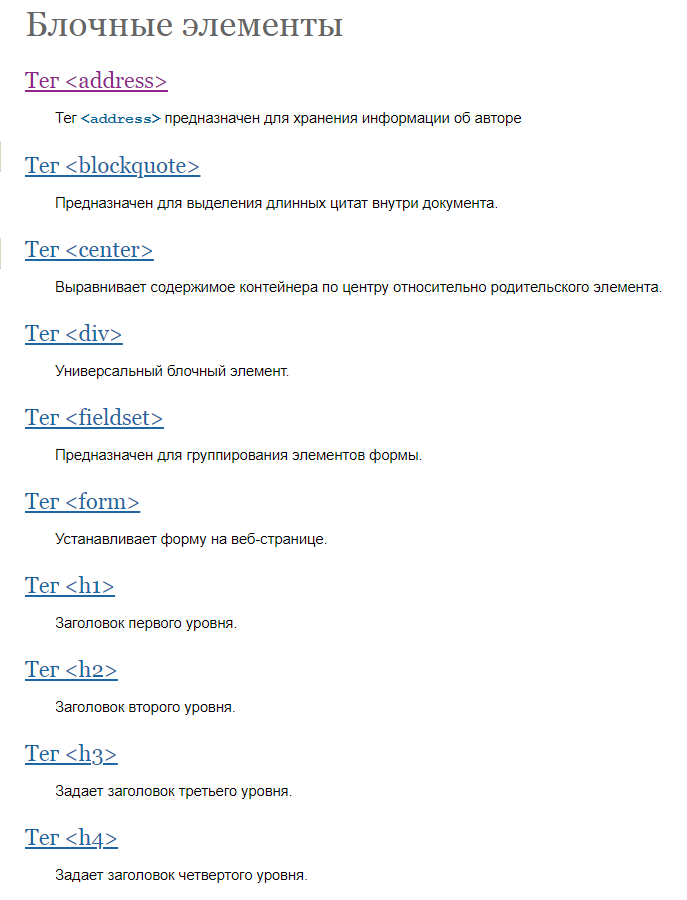
**Блочные элементы** представляют собой большие строительные блоки вашей веб-страницы. При отображении браузер автоматически добавляет разрыв строки до и после блочного элемента, при этом он занимает всю доступную ширину (по умолчанию отображается на веб-странице в виде прямоугольника), а высота блочного элемента вычисляется браузером автоматически, исходя из объема его содержимого. Текст в блочных элементах по умолчанию выравнивается по левому краю.

Обычно блочные элементы используются, чтобы разделить содержимое веб-страницы на логические блоки (например: верхний колонтитул (шапка сайта), меню, блок с контентом, нижний колонтитул (footer) и др.). Блочные элементы нельзя вкладывать в строчные, исключением являются только ссылки и ячейки таблиц, внутри которых по стандарту HTML5 могут быть расположены блочные элементы.

**Строчные элементы** используются для разметки частей содержимого элементов. Ширина строчного элемента равна объему содержимого. В отличии от блочных элементов браузер не добавляет разрыв строки до и после строчного элемента, поэтому, если несколько строчных элементов идут подряд друг за другом, они располагаются на одной строке и переносятся на другую строку при необходимости. В большинстве случаев внутрь строчных элементов допустимо помещать другие строчные элементы, вставлять блочные элементы внутрь строчных запрещено.

Элемент <table> служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов <tr> и <td>. Внутри <table> допустимо использовать следующие элементы: <caption>, <col>, <colgroup>, <tbody>, <td>, <tfoot>, <th>, <thead> и <tr>.







6. HTML. Основные строчные (inline) теги. Тег iframe

Тег **<iframe>** создает плавающий фрейм, который находится внутри обычного документа, он позволяет загружать в область заданных размеров любые другие независимые документы.

Тег **<iframe>** является контейнером, содержание которого игнорируется браузерами, не поддерживающими данный тег. Для таких браузеров можно указать альтернативный текст, который увидят пользователи. Он должен располагаться между элементами **<iframe>** и **</iframe>**.

<a>,<br>,<input>,<label>,<span>,<select>,<textarea><img>

7.

## [Тег <button>](http://htmlbook.ru/html/button)

Создает на веб-странице кнопку.

## [Тег <fieldset>](http://htmlbook.ru/html/fieldset)

Предназначен для группирования элементов формы.

## [Тег <form>](http://htmlbook.ru/html/form)

Устанавливает форму на веб-странице.

## [Тег <input>](http://htmlbook.ru/html/input)

Позволяет создавать разные элементы интерфейса.

## [Тег <keygen>](http://htmlbook.ru/html/keygen)

Используется для генерации пары ключей — закрытого и открытого.

## [Тег <label>](http://htmlbook.ru/html/label)

Устанавливает связь между определенной меткой и элементом формы.

## [Тег <legend>](http://htmlbook.ru/html/legend)

Создает заголовок группы элементов формы.

## [Тег <optgroup>](http://htmlbook.ru/html/optgroup)

Группирует элементы списка.

## [Тег <option>](http://htmlbook.ru/html/option)

Определяет отдельные пункты списка, создаваемого с помощью контейнера select

## [Тег <output>](http://htmlbook.ru/html/output)

Определяет область для вывода, преимущественно с помощью скриптов.

## [Тег <select>](http://htmlbook.ru/html/select)

Создает раскрывающийся список.

## [Тег <textarea>](http://htmlbook.ru/html/textarea)

Создает поле для многострочного текста.

|  |  |  |
| --- | --- | --- |
| button | Кнопка. |  |
| checkbox | Флажки. Позволяют выбрать более одного варианта из предложенных. | Пиво  Чай  Кофе |
| file | Поле для ввода имени файла, который пересылается на сервер. |  |
| hidden | Скрытое поле. Оно никак не отображается на веб-странице. |  |
| image | Поле с изображением. При нажатии на рисунок данные формы отправляются на сервер. |  |
| password | Обычное текстовое поле, но отличается от него тем, что все символы показываются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль. |  |
| radio | Переключатели. Используются, когда следует выбрать один вариант из нескольких предложенных. | Пиво  Чай  Кофе |
| reset | Кнопка для возвращения данных формы в первоначальное значение. |  |
| submit | Кнопка для отправки данных формы на сервер. |  |
| text | Текстовое поле. Предназначено для ввода символов с помощью клавиатуры. |  |

8. Семантической называют верстку, использующую для структурирования html-документов теги, которые разделяют код на логические блоки (явно показывающие их роль и содержание в структуре web-страницы). Этот вид верстки стал доступен с появлением HTML5, его новые элементы играют большую роль в создание явной логической структуры документа.

Семантическая верстка отличается от блочной лишь элементами, используемыми при структуризации страницы, но в отличие от семантических тегов, используемых в семантической верстке, тег div, являющийся основным используемым элементом в блочной верстке, не передает четкий смысл о содержащемся в нем контенте, так как он является лишь контейнером для группировки контента.

Предлагаем немного подробнее рассмотреть элементы, которые были использованы в макете:

## <section>

[Тег <section>](https://puzzleweb.ru/html/tag_section.php) определяет раздел страницы, в котором размещается логически связанная информация, как правило, имеющая свой заголовок.

Каждый раздел может иметь свои собственные теги <header> и <footer>, включающие в себя свои собственные заголовки.

## <header>

[Тег <header>](https://puzzleweb.ru/html/tag_header.php) определяет верхнюю часть страницы или элемента. Он может содержать заголовки, но это не является обязательным. Есть лишь одно ограничение использования <header> он не должен содержать сам себя и элемент <footer>.

## <nav>

Для создания меню навигации на веб-странице в HTML5 был добавлен [тег <nav>](https://puzzleweb.ru/html/tag_nav.php). Он предназначен для содержания гиперссылок, ведущих на другие страницы или другую часть документа.

## <article>

[Тег <article>](https://puzzleweb.ru/html/tag_article.php) используется для разделения раздела страницы на логические блоки, блок должен быть связан с общим разделом, но иметь свой собственный смысл и быть отдельной логической единицей страницы.

## <aside>

Для создания небольшой области на странице, в стороне от общего содержимого, используется [тег <aside>](https://puzzleweb.ru/html/tag_aside.php). Он нужен для вынесения информации в отдельный блок, его содержимое обычно не относится к основной теме страницы, а лишь дает дополнительную информацию, к какой-либо ее части.

## <footer>

[Тег <footer>](https://puzzleweb.ru/html/tag_footer.php) используется для определения нижнего колонтитула страницы или элемента. Он не может содержать в себе <header> и определять у себя нижний колонтитул.

9.

Связанные стили

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, как правило, с расширением css, а для связывания документа с этим файлом применяется тег **<link>**. Данный тег помещается в контейнер **<head>**, как показано в примере 3.1.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Стили**</title>**

**<link** rel="stylesheet" href="http://htmlbook.ru/mysite.css"**>**

**<link** rel="stylesheet" href="http://www.htmlbook.ru/main.css"**>**

**</head>**

**<body>**

**<h1>**Заголовок**</h1>**

**<p>**Текст**</p>**

**</body>**

**</html>**

Значение атрибута тега **<link>** — rel остаётся неизменным независимо от кода, как приведено в данном примере. Значение href задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Заметьте, что таким образом можно подключать таблицу стилей, которая находится на другом сайте.

Содержимое файла mysite.css подключаемого посредством тега **<link>** приведено в примере 3.2.

**Пример 3.2. Файл со стилем**

**H1** {

color:#000080;

font-size:200%;

font-family: Arial, Verdana, sans-serif;

text-align: center;

}

**P** {

padding-left:20px;

}

Как видно из данного примера, файл со стилем не хранит никаких данных, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

Глобальные стили

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера **<style>**, как показано в примере 3.3.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Глобальные стили**</title>**

**<style>**

**H1** {

font-size:120%;

font-family: Verdana, Arial, Helvetica, sans-serif;

color:#333366;

}

**</style>**

**</head>**

**<body>**

**<h1>**Hello, world!**</h1>**

**</body>**

**</html>**

Внутренние стили

Внутренний или встроенный стиль является по существу расширением для одиночного тега используемого на текущей веб-странице. Для определения стиля используется атрибут style, а его значением выступает набор стилевых правил (пример 3.4).

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Внутренние стили**</title>**

**</head>**

**<body>**

**<p** style="font-size: 120%; font-family: monospace; color: #cd66cc"**>**Пример текста**</p>**

**</body>**

**</html>**

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль. В примере 3.5 применяется сразу два метода добавления стиля в документ.

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Подключение стиля**</title>**

**<style>**

**H1** {

font-size:120%;

font-family: Arial, Helvetica, sans-serif;

color: green;

}

**</style>**

**</head>**

**<body>**

**<h1** style="font-size: 36px; font-family: Times, serif; color: red"**>**Заголовок 1**</h1>**

**<h1>**Заголовок 2**</h1>**

**</body>**

**</html>**

Импорт CSS

В текущую стилевую таблицу можно импортировать содержимое CSS-файла с помощью команды @import. Этот метод допускается использовать совместно со связанными или глобальными стилями, но никак не с внутренними стилями. Общий синтаксис следующий.

@import url("имя файла") типы носителей;

@import "имя файла" типы носителей;

После ключевого слова @import указывается путь к стилевому файлу одним из двух приведенных способов — с помощью url или без него. В примере 3.6 показано, как можно импортировать стиль из внешнего файла в таблицу глобальных стилей.

**Пример 3.6. Импорт CSS**

HTML5CSS 2.1IECrOpSaFx

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Импорт**</title>**

**<style>**

@import url("**style**/**header**.css");

**H1** {

font-size:120%;

font-family: Arial, Helvetica, sans-serif;

color: green;

}

**</style>**

**</head>**

**<body>**

**<h1>**Заголовок 1**</h1>**

**<h2>**Заголовок 2**</h2>**

**</body>**

**</html>**

10. CSS. Селектор. Виды селекторов.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Селектор** - это часть CSS-правила, которая сообщает браузеру, к какому элементу (или элементам) веб-страницы будет применён стиль.

1. [**Базовые (селекторы тегов)**](https://goldbusinessnet.com/uchebnik-css/css-selektory-tegov-class-klassy-id-identifikatory/#tag-selector) — самый простой для понимания, название которого совпадает с именем соответствующего тега в ХТМЛ коде документа.

Возьмем в качестве примера горизонтальную линию как элемент дизайна страницы сайта. Мы уже знаем, что эта линия создается с [помощью тега hr](https://goldbusinessnet.com/osnovy-html/html-tegi-p-br-hr-atributy/#hr), который прописывается в HTML-коде. Если вы перейдете по только что данной ссылке, то сможете отследить, как меняется внешний вид этого элемента при применении атрибутов тега hr, которые задаются в ХТМЛ-документе вместе с ним.

Однако, как я уже говорил, современные тенденции направлены на разгрузку HTML кода за счет использования оформления с помощью стилей CSS, представленных в отдельном файле. Поэтому приведенные в той статье атрибуты тега hr (width, size, color, noshade) заменяем на свойства стилей, **прописанные в style.css**:

|  |
| --- |
| hr { |
| border: 0; | |

|  |
| --- |
| width: 350px; |
| height: 10px; |

|  |  |
| --- | --- |
| background-color: blue; | |
| } |

2. [**Классы (class) и идентификаторы (id)**](https://goldbusinessnet.com/uchebnik-css/css-selektory-tegov-class-klassy-id-identifikatory/#id-class) — имеют некоторую общность, поскольку являются атрибутами тегов, к которым они добавляются в HTML-коде, и одновременно селекторами, кои и указываются в файле CSS, каждый со своим набором свойств.

Cвязь элементов, описываемых тегами с классами и идентификаторами, осуществляется тем, что параметрами id и class являются названия селекторов. Примеры ниже.

Тег с классом «text-x» в HTML-коде:

|  |
| --- |
| <p class="text-x">Текстовый фрагмент красного цвета.</p> |

Стили для этого класса в CSS (впереди обязательно нужно поставить точку):

|  |
| --- |
| .text-x {color: red; font-size: 14px} |

То же самое для идентификаторов ([пример с контейнером див](https://goldbusinessnet.com/osnovy-html/html-tegi-div-i-span-elementy-blochnoj-verstki-sajta/)). HTML:

|  |
| --- |
| <div id="area">Блок с зеленым текстом, окруженный рамкой</div> |

CSS (перед идентификатором ставится значок решетки):

|  |
| --- |
| #area {color: green; border: 2px solid #d6c99a} |

Последующие ниже варианты могут содержать как селекторы тегов, так и классы с идентификаторами.

3. [**Дочерние селекторы**](https://goldbusinessnet.com/uchebnik-css/dochernie-kontekstnye-css-selektory/#child-selector) — как известно, в HTML различают родительские элементы (теги) и дочерние (их потомки), которые в них вложены. Весьма логично, что в CSS точно такое же разделение существует между соответствующими этим элементам селекторами.

Правило дочернего селектора указывает, что свойства стилей будут применены к конкретному элементу (в ниже следующем примере это тег гиперссылки **a**) только в том случае, если он является *непосредственным потомком* (1-й уровень вложенности) определенного родителя (**li**):

|  |
| --- |
| li > a {font-weight: normal; font-size: 80%; background: #fff} |

4. [**Контекстные селекторы**](https://goldbusinessnet.com/uchebnik-css/dochernie-kontekstnye-css-selektory/#сontext-selector) — очень напоминают дочерние селекторы с той лишь разницей, что здесь свойства CSS будут применены ко всем потомкам относительно указанного родителя, вне зависимости от степени вложенности (родительский селектор в правиле стилей идет первым и отделен от дочернего пробелом):

|  |
| --- |
| p a {color: green; font-size: 14px} |

5. [**Соседние селекторы**](https://goldbusinessnet.com/uchebnik-css/sosednie-css-selektory-universalnyj-selektor-i-ix-primenenie/#neighboring-selectors) — данное CSS правило справедливо для элементов, описанных тегами, которые идут друг за другом в HTML-документе, причем, не имеют родственных отношений, иными словами, не вложены друг в друга. Соответствующие им селекторы отделяются друг от друга знаком «+», причем, свойства будут применены ко второму элементу лишь в том случае, если он располагается непосредственно рядом с первым:

|  |
| --- |
| b + i {color: red} |

6. [**Универсальный селектор**](https://goldbusinessnet.com/uchebnik-css/sosednie-css-selektory-universalnyj-selektor-i-ix-primenenie/#universal-selector) — применяется в тех случаях, когда нужно задать единый стиль оформления для тех или иных элементов на веб-странице, расположенных в различных ее областях. Например, определить вид и размер шрифта. Этот селектор обозначается звездочкой:

|  |
| --- |
| \* {font-family: Verdana,sans-serif; font-size: 14px} |

7. [**Селекторы атрибутов**](https://goldbusinessnet.com/uchebnik-css/css-selektory-atributov/) — позволяет присваивать одни и те же стили всем элементам на веб-странице, имеющим одинаковые атрибуты:

|  |
| --- |
| [title] {color: green} |

Либо определенным элементам:

|  |
| --- |
| p[title] {color: green} |

[Селекторы псевдокласса](#Селекторы_псевдокласса)

Псевдокласс класса CSS - это ключевое слово, добавленное в селектор, который задает особое состояние выбранного элемента. Например [: hover](file:////ru/docs/Web/CSS/:%20hover) применит стиль, когда пользователь наводит на элемент, указанный селектором.

Псевдо-классы вместе с псевдоэлементами позволяют применять стиль к элементу не только по отношению к содержанию дерева документов, но и по отношению к внешним факторам, таким как история навигатора ([:visited](https://developer.mozilla.org/ru/docs/Web/CSS/:visited), для примера), статус его содержимого (наподобии [:checked](https://developer.mozilla.org/ru/docs/Web/CSS/:checked) на некоторых элементах формы) или положение мыши (наподобии [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) который позволяет узнать, находится ли мышь над элементом или нет).

Синтаксис

selector:pseudo-class{

property: value;

}

## 11. CSS. Наследование и каскадирование свойств. Приоритет правил.

## Наследование

Начнем с самой простой для понимания концепции CSS. Суть ее состоит в том, что стили, присвоенные некоторому элементу, **наследуются** всеми потомками (вложенными элементами), если они не переопределены явно. Например, размер шрифта и его цвет достаточно применить к дескриптору body чтобы все элементы внутри имели те же свойства. Но, для заголовков h1-h6 размер шрифта не будет присвоен, потому что у браузера для них есть своя таблица стилей по умолчанию, а наследованные стили имеют самый низкий приоритет. Аналогичная ситуация с цветом ссылок.

Таким образом, наследование позволяет сократить таблицу CSS. Но в то же время если стилей много, то отследить какой родительский элемент установил некоторое свойство, становится довольно сложно.

## Каскадирование

Правила каскадирования позволяют разрешать ситуации, когда для одного элемента прописано несколько стилей. **Каскадирование**основано на присвоении некоторого приоритета каждому правилу. Авторские таблицы стилей имеют самый большой приоритет, меньший — пользовательские, самый низкий — таблицы стилей по умолчанию браузера. У пользователя есть возможность переопределить авторское правило путем добавления флага !important к своему.

Правила каскадирования определяют следующие приоритеты:

1. пользовательские стили, отмеченные !important
2. авторские стили, отмеченные !important
3. авторские стили
4. пользовательские стили
5. стили по умолчанию

После каскадирования правила упорядочиваются на основе специфичности селекторов.

12. Блочная модель

Любой блочный элемент состоит из набора свойств, подобно капустным листам накладываемых друг на друга. Основой блока выступает его контент (это может быть текст, изображение и др.), ширина которого задается свойством width, а высота через height; вокруг контента идут поля (padding), они создают пустое пространство от контента до внутреннего края границ; затем идут собственно сами границы (border) и завершают блок отступы (margin), невидимое пустое пространство от внешнего края границ. Порядок влияния этих свойств на блок четко определён и не может быть нарушен.

Возможно, вы об этом не знаете, но у каждого HTML-элемента есть блок, который его окружает. Этот блок состоит из нескольких слоёв, которыми можно независимо управлять с помощью CSS. Это позволяет разместить элементы относительно друг друга и оформить их разными способами.

Вот из чего состоят слои:

* Ширина — ширина площади содержимого элемента. Для блочных элементов значение по умолчанию равно 100%. У строчных элементов ширина зависит от содержимого.
* Высота — определяет высоту элемента. Как правило, она зависит от внутреннего содержимого, но при желании можно указать конкретную высоту. Опять же, это работает только с блочными элементами.
* Границы — границы есть у каждого элемента, даже если вы их не видите. У них может быть разный размер, цвет и оформление.
* Отступы — они определяют расстояние между границей элемента и его содержимым. Их можно использовать, например, для того, чтобы текст внутри элемента оставался читаемым.
* Поля — они определяют расстояние между границей элемента и тем, что его окружает.

Display - Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе.

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| block | Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега **<span>**, заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого. |
| inline | Элемент отображается как встроенный. Использование блочных тегов, таких как **<div>** и **<p>**, автоматически создает перенос и показывает содержимое этих тегов с новой строки. Значение inline отменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент. |
| inline-block | Это значение генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно встроенному элементу. Фактически такой элемент по своему действию похож на встраиваемые элементы (вроде тега **<img>**). При этом его внутренняя часть форматируется как блочный элемент, а сам элемент — как встроенный. |

Позиционирование элементов

<http://htmlbook.ru/samlayout/blochnaya-verstka/pozitsionirovanie-elementov>

Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, перпендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет  z-index. Это свойство работает только для элементов, у которых значение position задано как absolute, fixed или relative.

### Синтаксис

z-index: число | auto | inherit

### Значения

В качестве значения используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже. Хотя спецификация и разрешает использовать отрицательные значения z-index, но такие элементы не отображаются в браузере Firefox до версии 2.0 включительно.

Кроме числовых значений применяется auto — порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что их родительский элемент. Значение inherit указывает, что оно наследуется у родителя.

14. Обтекания элементов

Как правило, все блоки и элементы на веб-странице в браузере появляются в том порядке, в каком они определены в коде html. Однако CSS предоставляет специальное свойство **float**, которое позволяет установить обтекание элементов, благодаря чему мы можем создать более интересные и разнообразные по своему дизайну веб-страницы.

Это свойство может принимать одно из следующих значений:

* left: элемент перемещается влево, а все содержимое, которое идет ниже его, обтекает правый край элемента
* right: элемент перемещается вправо
* none: отменяет обтекание и возвращает объект в его обычную позицию

При применении свойства **float** для стилизуемых элементов, кроме элемента img, рекомендуется установить свойство width.

Иногда возникает необходимость запретить обтекания. Подобная задача может быть актуальна, если какой-то блок должен переноситься вниз на новую строку, а не обтекать плавающий элемент. Например, футер, как правило, должен находиться строго внизу и растягиваться по всей ширине страницы. Если же перед футером находится плавающий элемент, то футер может обтекать этот элемент, что не желательно.

Для запрета обтекания элементов в CSS применяется свойство **clear**, которое указывает браузеру, что к стилизуемому элементу не должно применяться обтекание.

Свойство clear может принимать следующие значения:

* left: стилизуемый элемент может обтекать плавающий элемент справа. Слева же обтекание не работает
* right: стилизуемый элемент может обтекать плавающий элемент только слева. А справа обтекание не работает
* both: стилизуемый элемент может обтекать плавающие элементы и относительно них смещается вниз
* none: стилизуемый элемент ведет себя стандартным образом, то есть принимает участие в обтекании справа и слева

Псевдоклассы и псевдоэлементы

С помощью псевдоклассов в CSS можно устанавливать стиль для уже существующих элементов веб-страницы, который будет применяться в случае каких-то действий пользователя. Псевдоэлементы же отличаются тем, что могут определять стиль несуществующего содержимого, а также четко не обозначенных элементов.

### Популярные псевдоклассы CSS

Вы можете придавать элементам стиль, который будет зависеть от состояния этих элементов. Вот список некоторых псевдоклассов:

* **:link** – этот псевдокласс задает стиль ссылкам, по которым пользователь еще не перешел;
* **:visited** – этот же, наоборот, применяет стиль к уже посещенным ссылкам;
* **:hover** – определяет стиль элемента, когда на него наведен курсор (может применяться не только к ссылкам);
* **:active** – задает стиль активной ссылке (то есть, в момент клика по ней);
* **:focus** – применяет стиль к элементу при фокусировке на нем (например, при установке курсора в строку поиска);
* **:not()** – этот полезный псевдокласс позволяет выбрать и стилизовать только те элементы, которые не содержат селектор, указанный в скобках.

Как записываются псевдоклассы CSS? Необходимо добавить стиль с названием необходимого элемента + имя псевдокласса. Пример:

a:link {

color: red;

}

a:hover {

color: #26A65B;

}

a:visited {

color: #CCC;

}

В записанном стиле сказано, что обычные, непосещённые ссылки должны быть красного цветаa:link {color: red;}, ссылка при наведении должна менять цвет на другойa:hover {color: #26A65B;}, а посещённая ссылка должна иметь третий цветa: visited {color: #CCC;}.

### Популярные псевдоэлементы CSS

Если псевдоклассы записываются с одним двоеточием, то псевдоэлементы – с двумя. Это было внедрено в CSS3 для того, чтобы различать псевдоклассы и псевдоэлементы между собой. Однако раньше этой разницы не существовало и с псевдоэлементами использовалось одно двоеточие. Сейчас браузеры поддерживают оба варианта написания (но не для всех случаев). Рассмотрим некоторые псевдоэлементы:

* **::after** – используется вместе со свойством **content** и позволяет вывести необходимые данные после содержимого элемента;
* **::before** – выполняет похожую функцию, что и предыдущий, только выводит данные перед содержимым элемента;
* **::selection** – этот псевдоэлемент распознается браузерами только при использовании двух двоеточий и позволяет установить цвет и фон для текста, который выделен пользователем;
* **::first-letter** – используется для изменения стиля первого символа в тексте элемента;
* **::first-line** – используется для изменения стиля первой строки текста элемента.

Пример использования псевдоэлементов:

blockquote::before {

content: "«";

}

blockquote::after {

content: "»";

}

blockquote::selection {

color: #C8F7C5;

background-color: #1E824C;

}

Мы написали стиль для длинных цитат, который добавляет кавычки «ёлочки» в начале и в конце содержимого тега **<blockquote>**, а также изменяет цвет и фон выделенного пользователем текста цитаты.

Import

Внешняя таблица стилей может быть подключена к веб-странице с помощью HTML тега [**<link>**](https://basicweb.ru/html/tag_link.php), или с использованием встроенного в CSS правила **@import**, которое по большому счету делает тоже самое. Оба варианта позволяют присоединить таблицы стилей к веб-странице.

Основное отличие правила **@import** заключается в том, что с использованием этого правила вы можете присоединить одну внешнюю таблицу стилей к другой.

Media

Наряду с типами носителей в CSS3 включена поддержка различных технических параметров устройств, на основе которых требуется загружать те или иные стили. К примеру, можно определить смартфон с максимальным разрешением 640 пикселов и для него установить одни стилевые свойства, а для остальных устройств другие. Также можно выявить различные характеристики вроде наличия монохромного экрана, ориентации (портретная или альбомная) и др. Все характеристики легко комбинируются, поэтому допустимо задать стиль только для устройств в альбомной ориентации с заданным разрешением экрана.

Возможности медиа-запросов не ограничиваются выявлением мобильных устройств, с их помощью можно создавать адаптивный макет. Такой макет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Медиа-запросы ограничивают ширину макета и при достижении этого значения (к примеру за счёт уменьшения окна или при просмотре на устройстве с указанным размером) уже применяется другой стиль.

Синтаксис

Все запросы начинаются с правила [@media](http://htmlbook.ru/css/media), после чего следует условие, в котором используются типы носителей, логические операторы и медиа-функции. Типы носителей перечислены в табл. 1.

|  |  |
| --- | --- |
| *Табл. 1. Типы носителей и их описание* | |
| **Тип** | **Описание** |
| all | Все типы. Это значение используется по умолчанию. |
| braille | Устройства, основанные на системе Брайля, которые предназначены для чтения слепыми людьми. |
| embossed | Принтеры, использующие для печати систему Брайля. |
| handheld | Смартфоны и аналогичные им аппараты. |
| print | Принтеры и другие печатающие устройства. |
| projection | Проекторы. |
| screen | Экран монитора. |
| speech | Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры. |
| tty | Устройства с фиксированным размером символов (телетайпы, терминалы, устройства с ограничениями дисплея). |
| tv | Телевизоры. |

Пользовательские шрифты

Правило @font-face позволяет определить настройки шрифтов, а также загрузить специфичный шрифт на компьютер пользователя.

### Синтаксис

@font-face { свойства шрифта }

### Значения

Внутри конструкции @font-face может находиться набор свойств для изменения параметров шрифта (font-family, font-size, font-style и др.), а также ссылка на шрифтовой файл. Ссылка записывается в виде src: url(URI), где URI — относительный или абсолютный путь к файлу.

**Пример**

HTML5CSS2.1CSS3IECrOp 11Op 12SaFx

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**@font-face**</title>**

**<style>**

@**font**-face {

font-family: Pompadur;/\* Имя шрифта \*/

src:**url(**fonts/pompadur.ttf**)**;/\* Путь к файлу со шрифтом \*/

}

**P**{

font-family: Pompadur;

}

**</style>**

15. Анимация

CSS transitions позволяют сделать изменения CSS-свойств плавно и в течение некоторого времени. Таким образом, вы получаете возможность контролировать процесс перехода элемента от одного состояния к другому. Начнем с простейшего примера и продолжим дальше.

При наведении курсора на квадрат плавно изменяется цвет фона.

Теперь подробнее рассмотрим, как управлять переходами в CSS. У нас на вооружении есть всего 5 свойств, которые позволяют контролировать transition-анимацию:

* transition-property;
* transition-duration;
* transition-timing-function;
* transition-delay;
* transition;

**transition-property** — указывает список свойств, которые будут анимироваться; свойства, которые здесь не указаны, будут изменяться обычным образом. Можно анимировать все свойства для конкретного элемента, указав значение all. Если вы не указали ни одного свойства, то по умолчанию используется значение all.

Пример:

transition-property: width;

**transition-duration** — задаёт значение продолжительности анимации, время можно указывать в секундах или миллисекундах.

Пример:

transition-duration: 1s;

**transition-timing-function** — временная функция, указывает точки ускорения и замедления за определенный период времени для контроля изменения скорости анимации. Проще говоря, с помощью этого свойства можно указать поведение для анимации. Например, мы можем ускорить анимацию в начале и замедлить в конце, либо наоборот. Для того, чтобы задать процесс анимации используются кривые Безье. Вообще, transition-timing-function позволяет сделать очень много разных поведений для анимаций, это целая тема для статьи, поэтому здесь мы не будем углубляться.

Пример:

transition-timing-function: cubic-bezier(0, 0, 1, 1);

**transition-delay** — задаёт задержку времени до начала анимации, можно указывать в секундах или миллисекундах.

Пример:

transition-delay: 500ms;

**transition** — это общее свойство, которое позволяет перечислить первые четыре свойства в порядке: property, duration, timing-function, delay.

Пример:

transition: background-color 1s cubic-bezier(0, 0, 1, 1) 500ms;

У нас получилась вот такая простая анимация: при наведении мышкой на квадрат изменяется ширина; продолжительность анимации одна секунда; анимация воспроизводится по линейной функции; задержка перед началом анимации 500 миллисекунд.

С помощью CSS transitions можно анимировать почти все свойства и создавать много интересных, красивых, функциональных и даже сложных анимаций, которые будут дополнять и улучшать ваш проект. Например, я сделал вот такой Material-FAB на чистом CSS, используя transitions:

## CSS animations

CSS animations позволяют делать более сложные анимации, нежели CSS transitions. Весь секрет в @keyframes. Правило @keyframes позволяет создавать анимацию с помощью набора ключевых кадров, то есть описывает состояние объекта в определенный момент времени. Давайте рассмотрим простой пример.

Наш круг ожил и он как будто пульсирует.

Есть 9 свойств, которые позволяют контролировать CSS animations:

* animation-name;
* animation-duration;
* animation-timing-function;
* animation-delay;
* animation-iteration-count;
* animation-direction;
* animation-play-state;
* animation-fill-mode;
* animation;

**animation-name** — здесь указвается имя анимации, которое связывает правило @keyframes с селектором.

Пример:

animation-name: my-animation;

@keyframes my-animation {

0% { opacity: 0; }

100% { opacity: 1; }

}

**animation-iteration-count** — задаёт количество повторов анимации, значение по умолчанию 1. Значение infinite означает, что анимация будет проигрываться бесконечно.

Пример:

animation-iteration-count: 2;

**animation-direction** — задаёт направление анимации.

Пример:

animation-direction: reverse;

**animation-play-state** — данное свойство управляет остановкой и прогрыванием анимации. Есть два значения, running (анимация проигрывается, по умолчанию) и paused (останавливает анимацию).

Пример:

animation-play-state: paused;

**animation-fill-mode** — устанавливает, какие CSS-свойства будут приминены к объекту до или после анимации. Может принимать такие значения:

* none — анимируемые CSS-свойства применятся к объекту только во время воспроизведения анимации, по окончании объект возвращается в исходное состояние;
* forwards — анимируемые CSS-свойства применятся к объекту по окончании воспроизведения анимации;
* backwards — анимируемые CSS-свойства применятся к объекту до начала воспроизведения анимации;
* both — анимируемые CSS-свойства применятся к объекту и до начала, и после окончания воспроизведения анимации;

Пример:

animation-fill-mode: backwards;

Свойства **animation-duration**, **animation-timing-function**, **animation-delay** работают так же, как аналогичные свойства в CSS transitions, о которых я писал раньше, поэтому не буду повторяться.

Этапы отрисовки

### [Document Object Model](https://developer.mozilla.org/ru/docs/Web/Performance/Critical_rendering_path#Document_Object_Model)

Построение DOM инкрементально. Ответ в виде HTML превращается в токены, которые превращаются в узлы (nodes), которые формируют DOM дерево. Простейший узел начинается с startTag-токена и заканчивается токеном endTag. Узлы содержат всю необходимую информацию об HTML элементе, соответствующем этому узлу. Узлы (nodes) связаны с Render Tree с помощью иерархии токенов: если какой-то набор startTag и endTag-токенов появляется между уже существующим набором токенов, мы получаем узел (node) внутри узла (node), то есть получаем иерархию дерева DOM.

Чем больше количество узлов (node) имеет приложение, тем дольше происходит формирование DOM tree, а значит дольше происходит обработка критических этапов рендеринга. Измеряйте! Несколько лишних узлов (node) не сделают погоды, но [divitis](https://en.wiktionary.org/wiki/divitis) обязательно приведет к подвисаниям.

### [CSS Object Model](https://developer.mozilla.org/ru/docs/Web/Performance/Critical_rendering_path#CSS_Object_Model)

DOM несет в себе всё содержимое страницы. CSSOM содержит все стили страницы, то есть данные о том, как стилизовать DOM. CSSOM похож на DOM, но всё же отличается. Если формирование DOM инкрементально, CSSOM - нет. CSS блокирует рендер: браузер блокирует рендеринг страницы до тех пор, пока не получит и не обработает все CSS-правила. CSS блокирует рендеринг, потому что правила могут быть перезаписаны, а значит, необходимо дождаться построения CSSOM, чтобы убедиться в отсутствии дополнительных переопределений.

У CSS имеются свои правила валидации токенов. Помните, что C в CSS означает "Cascade". CSS правила ниспадают каскадом. Иными словами, когда парсер преобразует токены в узлы (nodes), вложенные узлы наследуют стили от родительских. Инкрементальная обработка недоступна для CSS, потому что набор следующих правил может перезаписать предыдущие. Объектная модель CSS (CSSOM) строится по мере парсинга CSS, но она не может быть использована для построения дерева рендера (render tree), потому что может оказаться так, что следующий набор правил может сделать какой-либо из узлов дерева невидимым на экране. Это может привести к лишнему вызову компоновки и перерасчета стилей.

Говоря о производительности селекторов (selector), наименее специфичные селекторы срабатывают быстрее. Например, .foo {} сработает быстрее .bar .foo {}. В первом случае, условно, понадобится одна операция, чтобы найти элемент .foo, во втором случае, сначала будут найдены все .foo, апотом**браузер пройдет вверх** по дереву в поисках родительского элемента .bar. Более специфичные селекторы требуют от браузера большего количества работы, но эти проблемы, вероятно, не стоят их оптимизации.

Если вы измерите время, требуемое на парсинг CSS, вы будете удивлены тем, как быстро работают браузеры. Более специфичные правила более затратны, потому что требуют обхода большего числа узлов в DOM дереве, но эта дороговизна обходится довольно дёшево, особенно в сравнении с другими узкими местами производительности. Сначала измеряйте. Потом оптимизируйте, если это действительно необходимо. Вероятно, специфичность селекторов не то, что действительно затормаживает ваше приложение. Когда дело доходит до оптимизации CSS, улучшение производительность селекторов ускоряет рендеринг лишь на микросекунды. Существуют другие [пути оптимизации CSS](https://developer.mozilla.org/en-US/docs/Learn/Performance/CSS_performance), такие как минификация, разделение CSS-файлов на разные файлы на освове media-queries.

### [Дерево рендера (Render Tree)](#Дерево_рендера_Render_Tree)

Дерево рендера охватывает сразу и содержимое страницы, и стили: это место, где DOM и CSSOM деревья комбинируются в одно дерево. Для построения дерева рендера браузер проверяет каждый узел (node) DOM, начиная от корневого (root) и определяет, какие CSS правила нужно присоединить к этому узлу.

Дерево рендера охватывает только видимое содержимое. Например, секция head (в основном) не содержит никакой видимой информации, а потому может не включаться в дерево. Кроме того, если у какого-то узла стоит свойство display: none,  оно так же не включается в дерево (как и потомки этого узла).

### [Компоновка (Layout)](#Компоновка_Layout)

В тот момент, когда дерево рендера (render tree) построено, становится возможным этап компоновки (layout). Компоновка зависит от размеров экрана. Этот этап определяет, где и как на странице будут спозиционированы элементы и каковы связи между элементами.

Что насчет ширины элемента? Блочные элемент по определению имеют ширину в 100% от ширины их родителя. Элемент с шириной в 50% будет иметь ширину в два раза меньше родительской. Если не указано иного, элемент body имеет ширину 100%, то есть 100% ширины родителя - видимой области viewport (окна документа).

Мета тэг viewport, который вы можете указать в Head страницы, определяет ширину видимой области и влияет на компоновку. Без этого тэга браузеры используют ширину "по умолчанию", которая обычно составляет 960px. В браузерах, открывающихся по умолчанию в полноэкранном режиме, например, в браузере телефона, установка тега <meta name="viewport" content="width=device-width"> установит ширину видимой области в 100% от ширины экрана устройства, вместо того, чтобы использовать ширину по умолчанию. Эта ширина (device-width) изменяется каждый раз, когда пользователь поворачивает телефон. Это приводит к запуску этапа компоновки. Равно как и при изменении размеров окна в обычном браузере.

На производительность компоновки (layout) непосредственно влияет DOM - чем больше узлов (nodes) в вашем документе, тем больше времени понадобится на перерасчет позиций и размеров всех элементов. Компоновка может стать узким местом, ведущим к зависаниям, особенно если выполняет одновременно со скроллом или другой анимацией. И хотя задержка 20мс при загрузке или переориентации экрана может быть приемлемой, это все равно может привести к подвисаниям при анимации и скролле. Каждый раз, когда дерево рендера (render tree) модифицируется, например, из-за добавления узла (node), его модификации или при изменении стилей box-модели, запускается компоновка.

Для уменьшения частоты и продолжительности этого этапа, группируйте обновления экрана и избегайте анимации свойств, связанных с box-моделью элементов.

### [Отрисовка (Paint)](#Отрисовка_Paint)

Последний этап в нашем списке - отрисовка (paint) пикселей на экране. Когда дерево рендера (render tree) создано, компоновка (layout) произошла, пиксели могут быть отрисованы. При первичной загрузке документа (onload) весь экран будет отрисован. После этого только необходимые к обновлению части экрана будут перерисовываться, так как браузер старается оптимизировать процесс отрисовки, избегая ненужной работы. Так, если у вас в документе есть два элемента, перерисовываться будет только тот, который вы изменили. Время отрисовки зависит от того, какой тип обновления применился к дереву рендера (render tree). И хотя отрисовка - это очень быстрый процесс, и он, вероятно, слабо влияет на производительность, очень важно помнить, что оба этапа - компоновка (layout) и отрисовка (paint) должны работать сообща и укладываться в частоту обновления кадров. Каждое CSS-свойство, применяемое к любому узлу (node) увеличивает время отрисовки, но полное удаление стиля, способное сэкономить вам 0.001мс, вряд ли даст вам желаемый результат, но зато с легкостью ухудшит пользовательский опыт

**Transform**

Трансформирует элемент, в частности, позволяет его масштабировать, вращать, сдвигать, наклонять, а также комбинировать виды трансформаций.

### Синтаксис

transform: <функция> [<функция>]\* | none

### Значения

функция

Функция трансформации.

none

Отменяет действие трансформации.

### Функции трансформации

#### matrix

Задаёт [матрицу преобразований](http://htmlbook.ru/blog/matritsa-preobrazovanii).

#### rotate

Поворот элемента на заданный угол относительно точки трансформации, задаваемой свойством transform-origin.

transform: rotate(<[угол](http://htmlbook.ru/css/value/angle)>)

#### scale

Масштаб элемента по горизонтали и вертикали.

transform: scale(sx[, sy]);

Значение больше 1 увеличивает масштаб элемента, меньше 1 — уменьшает масштаб.

#### scaleX

Масштабирует элемент по горизонтали.

transform: scaleX(sx);

#### scaleY

Масштабирует элемент по вертикали.

transform: scaleY(sy);

#### skewX

Наклоняет элемент на заданный угол по вертикали.

transform: skewX(<[угол](http://htmlbook.ru/css/value/angle)>)

#### skewY

Наклоняет элемент на заданный угол по горизонтали.

transform: skewY(<[угол](http://htmlbook.ru/css/value/angle)>)

#### translate

Сдвигает элемент на заданное значение по горизонтали и вертикали.

transform: translate(tx[, ty])

#### translateX

Сдвигает элемент по горизонтали на указанное значение. Положительное значение сдвигает вправо, отрицательное влево.

transform: translateX(tx)

#### translateY

Сдвигает элемент по вертикали на указанное значение. Положительное значение сдвигает вниз, отрицательное вверх.

transform: translateY(ty)

**JS**

|  |
| --- |
| <script type="text/javascript"> |

|  |  |
| --- | --- |
| 06 | for(var i=1; i<=3; i++) { |

|  |  |
| --- | --- |
| 07 | alert("Из шляпы достали "+i+" кролика!") |

|  |  |
| --- | --- |
| 08 | } |

|  |  |
| --- | --- |
| 09 | </script> |

Обычно javascript-код вообще не пишут в HTML, а подключают отдельный файл со скриптом, вот так:

|  |
| --- |
| <script src="/my/script.js"></script> |

Обычно javascript стараются отделить от собственно документа.

Для этого его помещают внутрь тега HEAD, а в теле страницы по возможности оставляется чистая верстка.

|  |
| --- |
| <script type="text/javascript"> |

|  |  |
| --- | --- |
|  | for(var i=1; i<=3; i++) { |

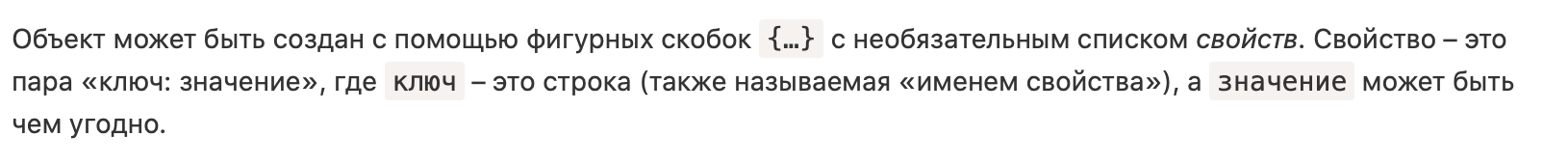
|  |  |
| --- | --- |
|  | alert("Из шляпы достали "+i+" кролика!") |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | </script> |

Graphical user interface, application

Description automatically generated



Graphical user interface, text, application, email

Description automatically generated

Сравнения

Graphical user interface, text, application, email

Description automatically generated

# 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.

DOM – это представление HTML-документа в виде дерева тегов.

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

События

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

Вот список самых часто используемых DOM-событий, пока просто для ознакомления:

**События мыши:**

* click – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).
* contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши.
* mouseover / mouseout – когда мышь наводится на / покидает элемент.
* mousedown / mouseup – когда нажали / отжали кнопку мыши на элементе.
* mousemove – при движении мыши.

**События на элементах управления:**

* submit – пользователь отправил форму <form>.
* focus – пользователь фокусируется на элементе, например нажимает на <input>.

**Клавиатурные события:**

* keydown и keyup – когда пользователь нажимает / отпускает клавишу.

**События документа:**

* DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

**CSS events:**

* transitionend – когда CSS-анимация завершена.

Существует множество других событий. Мы подробно разберём их в последующих главах.

## [Обработчики событий](https://learn.javascript.ru/introduction-browser-events" \l "obrabotchiki-sobytiy)

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия пользователя.

Есть несколько способов назначить событию обработчик. Сейчас мы их рассмотрим, начиная с самого простого.

### [Использование атрибута HTML](https://learn.javascript.ru/introduction-browser-events" \l "ispolzovanie-atributa-html)

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.

Например, чтобы назначить обработчик события click на элементе input, можно использовать атрибут onclick, вот так:

<input value="Нажми меня"onclick="alert('Клик!')"type="button">

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

Обратите внимание, для содержимого атрибута onclick используются одинарные кавычки, так как сам атрибут находится в двойных. Если мы забудем об этом и поставим двойные кавычки внутри атрибута, вот так: onclick="alert("Click!")", код не будет работать.

Атрибут HTML-тега – не самое удобное место для написания большого количества кода, поэтому лучше создать отдельную JavaScript-функцию и вызвать её там.

Следующий пример по клику запускает функцию countRabbits():

<script>

function countRabbits(){

for(let i=1; i<=3; i++){

alert("Кролик номер "+ i);

}

}

</script>

<input type="button"onclick="countRabbits()"value="Считать кроликов!">

Как мы помним, атрибут HTML-тега не чувствителен к регистру, поэтому ONCLICK будет работать так же, как onClick и onCLICK… Но, как правило, атрибуты пишут в нижнем регистре: onclick.

### [Использование свойства DOM-объекта](https://learn.javascript.ru/introduction-browser-events" \l "ispolzovanie-svoystva-dom-obekta)

Можно назначать обработчик, используя свойство DOM-элемента on<событие>.

К примеру, elem.onclick:

<input id="elem"type="button"value="Нажми меня!">

<script>

elem.onclick=function(){

alert('Спасибо');

};

</script>

Если обработчик задан через атрибут, то браузер читает HTML-разметку, создаёт новую функцию из содержимого атрибута и записывает в свойство.

# Делегирование событий

Всплытие и перехват событий позволяет реализовать один из самых важных приёмов разработки – делегирование.

Идея в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому, мы ставим один обработчик на их общего предка.

Из него можно получить целевой элемент event.target, понять на каком именно потомке произошло событие и обработать его.

Идея делегирования событий роста. Вместо прикрепления обработчиков событий прямо к кнопкам, мы делегируем отслеживание этого события родительскому элементу <div id="buttons">. Когда нажимается кнопка, обработчик, назначенный родительскому элементу ловит всплывающее событие (помните раздел про распространение событий?).

Использование делегирования событий требует 3 шагов:

**Шаг 1. Определить общего родителя элементов для отслеживания событий**  
В примере ниже <div id="buttons"> является общим родителем для кнопок.

**Шаг 2. Прикрепить к родительскому элементу обработчик событий**  
document.getElementById('buttons').addEventListener('click', handler) прикрепляет обработчик событий к родителю кнопок. Этот обработчик также реагирует на нажатия на кнопки, так как *события нажатий на кнопки всплывают по всем элементам-предкам* (благодаря распространению событий).

**Шаг 3. Использовать event.target для выбора целевого элемента**  
Когда кнопка нажата, функция-обработчик вызывается с аргументом: объектом event. Свойство event.target обращается к элементу, на котором произошло событие (в нашем примере этот элемент – кнопка):

*// ...*

.addEventListener('click', event => {

if (event.target.className === 'buttonClass') {

console.log('Click!');

}

});

Кстати, на элемент к которому прикреплён сработавший обработчик события, указывает event.currentTarget. В нашем примере event.currentTarget указывает на элемент <div id="buttons">.

**Глобальный объект**

Глобальный объект предоставляет переменные и функции, доступные в любом месте программы. По умолчанию это те, что встроены в язык или среду исполнения.

В браузере он называется window

В браузере глобальные функции и переменные, объявленные с помощью var (не let/const!), становятся свойствами глобального объекта:

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

## [Что такое AJAX?](https://learn.javascript.ru/ajax-intro" \l "chto-takoe-ajax)

AJAX (аббревиатура от «**A**synchronous **J**avascript **A**nd **X**ml») – технология обращения к серверу без перезагрузки страницы.

За счёт этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.

Несмотря на то, что в названии технологии присутствует буква X (от слова XML), использовать XML вовсе не обязательно. Под AJAX подразумевают любое общение с сервером без перезагрузки страницы, организованное при помощи JavaScript.

Diagram

Description automatically generatedTable

Description automatically generatedTimeline

Description automatically generated with medium confidence

Timeline

Description automatically generated

Первая строка — это XML декларация. Здесь определяется версия XML (1.0) и кодировка файла.

Регулярные выражения

<https://learn.javascript.ru/regexp-methods>

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Regular_Expressions>

Работа со строками

<https://learn.javascript.ru/string>

php

строки

<https://www.php.net/manual/ru/language.types.string.php>

## Массивы[¶](https://www.php.net/manual/ru/language.types.array.php#language.types.array)

На самом деле массив в PHP - это упорядоченное отображение, которое устанавливает соответствие между значением и ключом. Этот тип оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно массив, список (вектор), хэш-таблицу (являющуюся реализацией карты), словарь, коллекцию, стэк, очередь и, возможно, что-то еще. Так как значением массива может быть другой массив PHP, можно также создавать деревья и многомерные массивы.

Объяснение этих структур данных выходит за рамки данного справочного руководства, но вы найдете как минимум один пример по каждой из них. За дополнительной информацией вы можете обратиться к соответствующей литературе по этой обширной теме.

### Синтаксис[¶](https://www.php.net/manual/ru/language.types.array.php#language.types.array.syntax)

#### **Определение при помощи [array()](https://www.php.net/manual/ru/function.array.php)**[**¶**](https://www.php.net/manual/ru/language.types.array.php#language.types.array.syntax.array-func)

Массив (тип array) может быть создан языковой конструкцией [array()](https://www.php.net/manual/ru/function.array.php). В качестве параметров она принимает любое количество разделенных запятыми пар key => value (ключ => значение).

array(

key =>value,

key2 =>value2,

key3 =>value3,

...

)

Запятая после последнего элемента массива необязательна и может быть опущена. Обычно это делается для однострочных массивов, то есть array(1, 2) предпочтительней array(1, 2, ). Для многострочных массивов с другой стороны обычно используется завершающая запятая, так как позволяет легче добавлять новые элементы в конец массива.

**ООП**

<https://code.tutsplus.com/ru/tutorials/basics-of-object-oriented-programming-in-php--cms-31910>

Модель исключений (exceptions) в PHP похожа с используемыми в других языках программирования. Исключение можно сгенерировать (выбросить) при помощи оператора [throw](https://www.php.net/manual/ru/language.exceptions.php), и можно перехватить (поймать) оператором [catch](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.catch). Код генерирующий исключение, должен быть окружен блоком [try](https://www.php.net/manual/ru/language.exceptions.php), для того, чтобы можно было перехватить исключение. Каждый блок [try](https://www.php.net/manual/ru/language.exceptions.php) должен иметь как минимум один соответствующий ему блок [catch](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.catch) или [finally](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.finally).

В случае, если выброшено исключение, для которого нет блока [catch](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.catch) в текущей функции, это исключение будет "всплывать" по стеку вызова, пока не будет найден подходящий блок [catch](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.catch). При этом, все встреченные блоки [finally](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.finally) будут исполнены. Если стек вызовов раскрутится до глобальной области видимости, не встретив подходящего блока [catch](https://www.php.net/manual/ru/language.exceptions.php" \l "language.exceptions.catch), программа завершит работу с фатальной ошибкой, если только у вас не настроен глобальный обработчик исключений.

Генерируемый объект должен принадлежать классу [Exception](https://www.php.net/manual/ru/class.exception.php) или наследоваться от [Exception](https://www.php.net/manual/ru/class.exception.php). Попытка сгенерировать исключение другого класса приведет к фатальной ошибке PHP.

**Обработка форм и фильтрация данных**

<https://habr.com/ru/post/143035/>

PHP. Обработка форм и фильтрация данных. Борьба с XSS-уязвимостью

Динамические веб-сайты активно работают с данными, полученными от пользователей. То есть по сути, почти вся информация на сайте была добавлена туда самими пользователями. Хороший пример такого сайта — это форум. Любой форум на 99.9% состоит из опубликованной там пользователями информации.

Но каждый раз, когда пользователям предоставляются какие-либо возможности, нужно быть готовыми к использованию этих возможностей не по назначению. Так, формы для публикации контента могут использовать не для отправки безобидных текстов, а для заражения вашего сайта вредоносным кодом.

Фильтрация данных

Любую информацию, полученную от пользователя, обязательно надо фильтровать перед выводом в шаблоне! Фильтрация означает применение к этой информации набора правил, которые очистят и подготовят её к публикации на сайте. Мы фильтруем информацию, чтобы недопустить появления на нашем сайте XSS-уязвимости.

XSS-уязвимость

XSS — это вид уязвимости, которая свойственна веб-приложениям.

Данная атака на сайт состоит в том, что в выдаваемую страницу внедряется вредоносный JS-код. Это становится возможным из-за недостаточной фильтрации данных, полученных от пользователя.

Принцип атаки

1. На сайте есть форма для публикации сообщения.

2. Через эту форму хакер вместо простого текста отправляет JS-код.

3. Сообщение публикуется на странице, которая доступна всем посетителям.

4. Вредоносный код выполняется для каждого посетителя этой страницы.

5. Вредоносный скрипт вредит пользователям сайта. Например, крадёт их куки.

Замена опасных символов

Проблема ясна. Надо обезопасить страницу, отфильтровав информацию из формы. Но как фильтровать, чтобы не потерять текст, но при этом лишить хакера возможности нам навредить?

Тут помогут HTML-мнемоники.

Мнемоника — это кодовое представление символа в HTML, который начинается со знака амперсанда «&» и завершается точкой с запятой «;».

Теги <script> состоят из треугольных скобок, а значит, если их заменить на мнемоники, то такой текст больше не будет трактоваться браузером как HTML-тег. Мнемоники часто используются, когда надо показать пользователю фрагмент с HTML-кодом на странице. Заглянув в исходный код такой страницы можно увидеть, что вместо кавычек и скобок в этом HTML-коде находятся мнемоники.

Есть и другой вариант фильтрации: попросту вырезать все теги из текста. Нет тегов — нет проблемы.

Функция фильтрации htmlspecialchars

Перейдём к практике. В PHP-сценарии добавим вызов функции, которая для переданной строки выполнит фильтрацию и заменит все опасные символы в ней на подходящие HTML-мнемоники. Такая функция называется htmlspecialchars.

Вот как выглядит её работа:

<?php

$text = "<script><script>"; // эту строку мы получили от пользователя

$safe\_str = htmlspecialchars($text); // отфильтрованная, безопасная строка

print($safe\_str); // узнаём, что получилось

Результат работы этого сценария: &lt;script&gt;&lt;/script&gt;

Никогда не забывайте использовать функцию htmlspecialchars при выводе информации от пользователей в шаблоне!

pdo

<https://prowebmastering.ru/php-pdo-start.html>

sql – in

<https://htmlacademy.ru/tutorial/php/sql-injections>

суперглобальные массивы

<http://on-line-teaching.com/php/global.arrays.htm>

загрузка файлов

<https://metanit.com/web/php/5.4.php>

<http://on-line-teaching.com/php/upload.files.htm>

<http://on-line-teaching.com/php/cookies.htm>

<http://on-line-teaching.com/php/sessions.htm>

**Конфиденциальность** (confidentiality) — это гарантия того, что секретные данные будут доступны только тем пользователям, которым этот доступ разрешен; такие пользователи называются легальными, или авторизованными.

**Доступность** (availability) — это гарантия того, что авторизованные пользователи всегда получат доступ к данным.

**Целостность** (integrity) — это гарантия сохранности данными правильных значений, которая обеспечивается запретом неавторизованным пользователям каким-либо образом изменять, модифицировать, разрушать или создавать данные.

Graphical user interface

Description automatically generated with medium confidence

<http://juice-health.ru/computers/encryption/468-methods-data-encryption>

<https://bitte.net.ua/blog/chto-takoy-cms-vidyi-osobennosti-rabotyi/>