БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Е. В. Кремень, Ю. А. Кремень

ВЕБ-КОНСТРУИРОВАНИЕ

Рекомендовано
Учебно-методическим объединением
по естественно-научному образованию
в качестве учебно-методического пособия для студентов
учреждений высшего образования,
обучающихся по специальности
«математика»

Учебное электронное издание

Минск, БГУ, 2023

ISBN 978-985-881-344-4

УДК 004.738.5(075.8)
ББК 32.973.202 773

Р е ц е н з е н т ы:
кафедра алгебры, геометрии и математического моделирования
Брестского государственного университета имени А. С. Пушкина
(заведующий кафедрой кандидат физико-математических наук,
доцент *А. Н. Сендер*);
доктор физико-математических наук, доцент *А. С. Кравчук*

Кремень, Е. В. Веб-конструирование [Электронный ресурс] : учеб.-метод. пособие / Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2023. – 1 электрон. опт. диск (CD-ROM).

Подробно рассмотрены вопросы использования языка разметки и каскадных таблиц стилей, их применение при верстке веб-страниц. Приводится много фрагментов кода, что существенно ускоряет усваивание материала, а также способствует более квалифицированному подходу к верстке страниц. Особое внимание уделяется использованию элементов JavaScript на стороне клиента.

Предназначено для студентов учреждений высшего образования, обучающихся по специальности «математика».

Минимальные системные требования:
PC, Pentium 4 или выше;
RAM 1 Гб; Windows XP/7/10;
Adobe Acrobat.

Оригинал-макет подготовлен в программе Microsoft Word.
Ответственный за выпуск *Т. М. Турчиняк*. Дизайн обложки *В. П. Явуз*.
Технический редактор *В. П. Явуз*. Компьютерная верстка *Е. В. Анискевич*.
Корректор *Н. А. Ракуть*.

Подписано к использованию 12.05.2023. Объем 12,2 МБ.

Белорусский государственный университет.
Управление редакционно-издательской работы.
Пр. Независимости, 4, 220030, Минск.
Телефон: (017) 259-70-70.
email: urir@bsu.by
http://elib.bsu.by/

Содержание

Введение

В настоящее время в связи со стремительным ростом популярности интернета и развитием компьютерной индустрии появилась необходимость в изучении технологий, связанных с созданием сайтов. Трудно найти область человеческой деятельности, в которой использование сети Интернет не принесло бы существенной пользы, поэтому резко повысился спрос на специалистов, способных эффективно работать в ней. Соответственно, появилась необходимость в обучении студентов современным информационным веб-технологиям и подготовке их к профессиональной практической работе в сети Интернет.

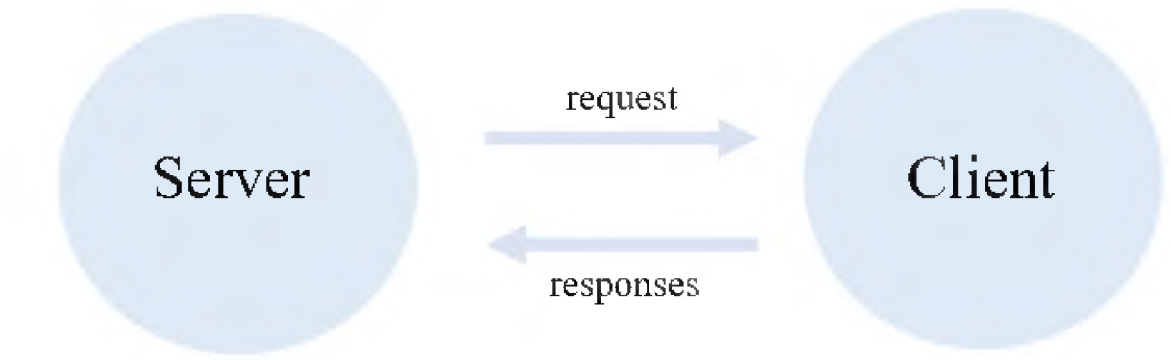
Основные понятия представления, организации и передачи информации в интернете

Рассмотрим, что происходит при просмотре веб-страницы. Компьютеры, подключенные к сети, называются клиентами и серверами.

Клиенты – это обычные пользователи, которые выходят в интернет со своих устройств, например компьютеров или телефонов, подключенных к Wi-Fi, с использованием программного обеспечения, доступного на этих устройствах, как правило браузеров.

Серверы – это компьютеры, которые хранят веб-страницы, сайты или приложения.

Упрощенная схема того, как они взаимодействуют, может выглядеть следующим образом:



Когда клиентское устройство пытается получить доступ к веб-странице, копия страницы загружается с сервера на клиентский компьютер для отображения в браузере пользователя.

Интернет-подключение – технология, позволяющая отправлять и принимать данные по сети.

TCP/IP (Transmission Control Protocol / Internet Protocol) – сетевая модель передачи данных, представленных в цифровом виде. Протокол являются коммуникационным и определяет, каким образом данные должны передаваться по сети.

DNS (Domain Name System) – система доменных имен. У любого устройства в сети есть свой числовой адрес, он называется **IP-адрес** (Internet Protocol address). Реальные веб-адреса представляют собой неудобные, незапоминающиеся строки, которые состоят из чисел, например 173.194.121.32 (псевдоним этого IP-адреса – доменное имя google.com). Этот набор чисел представляет собой уникальное местоположение в интернете. Запоминать такие адреса тяжело, поэтому используют DNS. DNS-сервера связывают буквенный веб-адрес, который вы вводите в браузере, например bsu.by, с реальным IP-адресом сайта.

URL (Uniform Resource Locator) – адрес ресурса в сети, определяет местонахождение и способ обращения к нему.

URN (Uniform Resource Name) – имя ресурса в сети, определяет только название ресурса, но не говорит, как к нему подключиться, и не включает в себя указания на местонахождение. Пример URN книги, идентифицируемой номером ISBN: urn: isbn: 5170224575.

URI (Uniform Resource Identifier) – имя и адрес ресурса в сети [**URN** + **URL**]. Например, адрес сайта – это URI: http://google.com.

URN идентифицирует ресурс по имени и отвечает на вопрос «что?». URL – указывает путь и метод доступа к ресурсу и отвечает на вопросы «где?» и «как?». По сути **URI** = **URN** + **URL**. При этом URN и URL – это частные случаи URI. Если провести аналогию с человеком, например Ивановым И. И., проживающим по адресу: г. Минск, ул. Мира, 28-14, то:

URN – это имя человека, Иванов И. И.;

URL – это адрес человека, г. Минск, ул. Мира, 28-14;

URI – это полные данные, Иванов И. И., г. Минск, ул. Мира, 28-14.

HTTP (HyperText Transfer Protocol, или протокол передачи гипертекста) – это протокол, который определяет язык для клиентов и серверов, чтобы они могли общаться друг с другом.

Пакеты. В основном, когда данные передаются через интернет, они отправляются в виде тысяч мелких кусочков, так что множество разных пользователей могут скачивать один и тот же сайт одновременно. Если бы сайты отправлялись одним большим куском, тогда только один пользователь мог бы скачать его за один раз, и это сделало бы пользование интернетом неэффективным.

Маршрутизатор – специализированное устройство, которое пересылает пакеты между различными сегментами сети на основе правил и таблиц маршрутизации.

Файлы компонентов сайта. Сайт состоит из нескольких различных файлов, которые бывают двух основных типов: файлы кода (HTML, CSS и JavaScript и др.) и материалы (изображения, музыка, видео, документы Word и PDF и т. д.).

Когда вы вводите веб-адрес в свой браузер, происходит следующая последовательность действий.

1. Браузер обращается к DNS-серверу и находит реальный адрес сервера, на котором расположен сайт.
2. Браузер посылает HTTP-запрос к серверу, запрашивая его отправить копию сайта для клиента. Это сообщение и все остальные данные, передаваемые между клиентом и сервером, передаются по интернет-соединению с использованием протокола TCP/IP.
3. Если сервер одобряет запрос клиента, сервер отправляет клиенту статус «200 ОК» и затем начинает отправку файлов сайта в браузер в виде небольших порций, называемых пакетными данными.
4. Браузер собирает маленькие куски (пакеты) в полноценный сайт и показывает его в браузере.

Упражнения

1. Ознакомьтесь с кодами ответов сервера клиенту https://developer.mozilla.org/ru/docs/Web/HTTP/Status или https://tools.ietf.org/html/rfc7231#section-6.5.1.
2. Для более глубокого понимания механизмов работы браузерных систем ознакомьтесь с описанием работы алгоритмов WebKit и Gecko https://www.html5rocks.com/ru/tutorials/internals/howbrowserswork/.

Фронтенд и бэкенд

Давно остались в прошлом времена, когда сайты представляли собой набор статических страниц с табличной разметкой. Веб-разработка не стоит на месте. Появились новые языки и технологии разработки сайтов. Сегодня это одно из самых перспективных и высокооплачиваемых IT-направлений.

Самая простейшая веб-страница в браузере представляется для пользователя как текст, стилизованный определенным образом. Веб-дизайнеры имеют доступ к сотням шрифтов различных размеров, цветов и даже алфавитов (например, испанский, японский, китайский), а браузеры могут точно отображать большинство из них. Веб-страницы могут также содержать изображения, видеоклипы и фоновую музыку. Они могут включать выпадающие меню, поля для поиска, активные ссылки на продукты или

другие страницы этого же веб-сайта, а также на внешние ресурсы. Многие веб-сайты поддерживают опции для настройки отображения веб-страницы согласно предпочтениям пользователя или физиологическим ограничениям, таким как плохое зрение, глухота или дальтонизм. Часто веб-страница может содержать подвижный контент, который пролистывается, в то время как ее остальная часть остается неподвижной.

Сложная и многоуровневая структура современных веб-приложений требует иерархического разделения процесса их разработки. Исторически этот процесс разделяется на две части: фронтенд (***front-end*** (клиентскую)) и бэкенд (***back-end*** (серверную)). Рассмотрим их более подробно. Эти термины имеют три различных значения.

Первое значение: **фронтенд – браузер**, **бэкенд – сервер**. В веб-разработке в качестве фронтенда выступают HTML-верстка, стили CSS и JavaScript, а в качестве бэкенда – серверная часть, которую программируют, например, на PHP, Java или ASP.net. Грубо говоря, все то, что исполняется на стороне клиента, – фронтенд, а на стороне сервера – бэкенд.

Второе значение: **фронтенд – статика**, **бэкенд – программный код**. В среде разработчиков высоконагруженных систем (highload-разработчиков) термином «фронтенд» называют ту программную часть, которая непосредственно «отдает» контент. Например, на больших проектах часто программную серверную часть представляют два веб-сервера: Apache и Nginx. Nginx принимает запросы и в случае статического файла (изображение, файл .css, .js или .xml) сразу же отдает его содержимое, а в случае, например, PHP-скрипта отправляет его к серверу Apache, который уже умеет обрабатывать PHP. Тут Nginx – это фронтенд, а Apache – бэкенд.

Третье значение: **фронтенд – открытые данные**, **бэкенд – административная часть**. Если речь идет о CMS (Content Management System, или система управления контентом), административную часть называют бэкендом, а «лицевую» часть сайта – фронтендом.

В рамках данного курса будем использовать первое значение: фронтенд – это все то, что исполняется на стороне клиента. А значит, к фронтенд-технологиям можно отнести весь пласт языков и технологий для создания веб-страниц. Базовыми из них являются HTML, CSS, JavaScript.

HTML

HTML (HyperText Markup Language) – язык гипертекстовой разметки документов. Размеченный документ определенным образом интерпретируется браузерами и отображается в человекочитаемой форме.

HTML позволяет:

- создавать обычные текстовые документы;
- структурировать и форматировать текстовые документы;
- соединять текстовые документы между собой через гиперссылки;
- вставлять в текстовые документы рисунки, графику, аудио- и видеоконтент.

Разберем термин HTML в деталях.

Гипертекст (HyperText) означает интерактивный текст, т. е. текст, с которым пользователь может взаимодействовать. Термин «гипертекст» придумал Теодор Нельсон в работе «A File Structure for the Complex, the Changing and the Indeterminate», опубликованной в 1965 г. Традиционный принцип расположения текста линейный: слово следует за словом, фраза за фразой, страница следует за страницей, читать нужно слева направо. Нельсон же предложил располагать информацию в узлах сети, связь между которыми не жесткая, а выбирается самим пользователем.

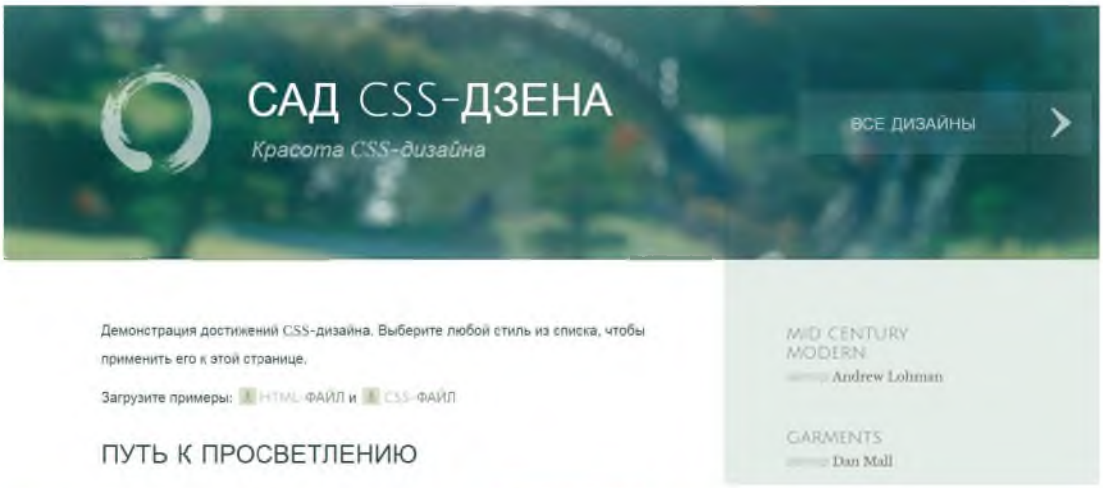
Разметка (Markup**)** обозначает возможности создавать контент в различных формах: заголовок, параграф, список, таблица, рисунок, ссылка и т. д., чтобы затем определять различные атрибуты для этих элементов контента.

Язык (Language) – это язык, который имеет свой собственный синтаксис, словарь и правила написания.

Первые веб-страницы целиком определял HTML. HTML5 отвечает только за отображение контента и структуру документа.

CSS

CSS (Cascading Style Sheets) – язык, описывающий внешний вид (оформление) и поведение HTML-элементов. Он позволяет применять стили выборочно к элементам в документах HTML. Дословно CSS переводится как «каскадные таблицы стилей». Пример влияния CSS на вид страницы хорошо иллюстрирует сайт «Сад CSS-дзена» http://www.csszen-garden.com/tr/ru/, который демонстрирует одну и ту же HTML-страницу, к которой по ссылкам внутри подключаются различные стили оформления. При этом представление контента меняется неузнаваемо. На сайте можно скачать эти CSS-файлы для последующего их использования. Ниже приведены несколько примеров оформления с сайта «Сад CSS-дзена».


САД CSS-ДЗЕНА
Красота CSS-дизайна
ВСЕ ДИЗАЙНЫ
Демонстрация достижений CSS-дизайна. Выберите любой стиль из списка, чтобы применить его к этой странице.
Загрузите примеры: HTML-ФАЙЛ и CSS-ФАЙЛ
ПУТЬ К ПРОСВЕТЛЕНИЮ
MID CENTURY MODERN
Andrew Lohman
GARMENTS
Dan Mall


DR. SHEA'S MIRACULOUS
Сад CSS-дзена
КРАСОТА CSS-ДИЗАЙНА
ПУТЬ
К ПРОСВЕТЛЕНИЮ


ВЫБЕРИТЕ ДИЗАЙН:
Mid Century Modern автор
Garments автор
Dan Mall
Steel автор
n Knoeller
Apothecary автор
Screen Filler автор
Fountain Kiss автор
A Robot Named Jimmy автор
mel:media
Verde Moderna автор
АРХИВЫ:
Следующие дизайны
Все дизайны
РЕСУРСЫ:
CSS для этого дизайна
Ресурсы по CSS
Ответы на вопросы
Отправить дизайн
Переводы
САД CSS-ДЗЕНА
КРАСОТА CSS-ДИЗАЙНА
ТАК ЧТО ЖЕ ЭТО ТАКОЕ?
CSS позволяет целиком и полностью контролировать оформление гипертекстового документа. Единственный способ проиллюстрировать этот принцип, чтобы вызвать восхищение — это дать кисти в руки тем, кто может создать красоту на основе существующей структуры. Дизайнеры и разработчики вместе сделали много для красоты веба, но мы всегда можем пойти дальше.

JavaScript

JavaScript (JS) – это полноценный динамический язык программирования, который применяется к HTML-документу и может обеспечить динамическую интерактивность на странице. Встроен во все браузеры. Язык применяется в совокупности с различными библиотеками (jQuery, LoDash и др.) и фреймворками (Backbone, ExtJS, AngularJS, Ember, NodeJS и др.), которые облегчают решение определенных, часто встречающихся задач. Кроме этого, уже разработано большое количество инструментов с использованием основного языка JavaScript. К ним относятся, в частности, программные интерфейсы приложения или API. Есть API, встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание HTML и установка CSS-стилей, захват видеопотока и манипуляция им, работа с веб-камерой пользователя или генерация 3D-графики. Есть сторонние API, которые позволяют разработчикам внедрять в свои сайты функциональность, разработанную другими программистами.

JavaScript Object Notation (JSON) – это формат обмена данными. По синтаксису этот формат напоминает JavaScript. JSON поддерживают множество языков программирования, но особенно он полезен для JavaScript-приложений, таких как веб-сайты и расширения для браузера.

jQuery

jQuery – библиотека JavaScript, поддерживающая взаимодействие JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM (Document Object Model), обращаться к его атрибутам и содержимому, манипулировать им. Также библиотека jQuery предоставляет удобный API для работы с AJAX.

AJAX

AJAX (Asynchronous JavaScript and XML – асинхронный JavaScript и XML) – это не технология сама по себе, а термин, который описывает подход к совместному использованию существующих технологий. AJAX включает HTML или XHTML, CSS, JavaScript, DOM, XML, XSLT и объект XMLHttpRequest. Когда эти технологии объединяются в модель AJAX, веб-приложения способны делать быстрые обновления интерфейса пользователя на странице без необходимости полной перезагрузки страницы браузером. Приложения работают быстрее и становятся более отзывчивыми к действиям пользователей.

Подробно остановимся на HTML, CSS и JavaScript.

Вспомогательные материалы для изучения HTML

Учебники:

- http://w3schools.com;
- http://htmlbook.ru/samhtml;
- https://developer.mozilla.org/ru/docs/Web/HTML;
- https://puzzleweb.ru.

Справочники:

- http://w3schools.com/tags/default.asp;
- http://htmlbook.ru/html;
- https://developer.mozilla.org/ru/docs/Web/HTML/Reference.

Для быстрого просмотра HTML-разметки и стилей без создания файлов удобно использовать песочницы (online sandbox):

- http://jsbin.com;
- http://jsfiddle.net;
- http://cssdeck.com;
- http://codepen.io.

Спецификация (стандарт) W3C: http://w3.org/TR/html.

Необходимые инструменты разработки

Существует огромный набор средств разработки сайтов.

Текстовый редактор

Для работы с контентом необходим редактор. Существует два типа редакторов: WYSIWYG и текстовые редакторы HTML. WYSIWYG является аббревиатурой от What You See Is What You Get («что вы видите, то и получаете»). Редакторы этого типа предоставляют интерфейс редактирования, который показывает, как выглядит код на рабочей веб-странице. WYSIWYG-редакторы не требуют знаний HTML, поэтому пользователю, не имеющему опыта программирования, гораздо легче начать работу. Однако при этом пользователь не контролирует генерируемый код, и, как следствие, код может быть неоптимальным, «тяжелым».

Текстовые редакторы HTML основаны на тексте. Чтобы пользоваться ими, нужны знания HTML. Текстовый редактор не всегда дает возможность предварительно посмотреть, как будет выглядеть живой сайт. Хороший текстовый редактор повышает эффективность работы, а также помогает избежать некоторых наиболее распространенных ошибок.

Текстовый редактор может быть очень простым, например Notepad (но лучше использовать, например, Notepad++ или VIM), а может обладать более продвинутыми возможностями, например Visual Studio Code, Sublime Text, Atom, GNU Emacs, Dreamweaver или WebStorm.

Так, Microsoft Word является прекрасным текстовым редактором для создания документов. Однако в качестве редактора для размещения текста на сайте он не совсем подходит, так как создает очень «грязный», «тяжелый» HTML-код. Сгенерированные элементы смотрятся вполне нормально, но код, созданный Word, можно записать намного аккуратнее, используя специализированный редактор. Это относится и к использованию редакторов Google Docs, OpenOffice, LibreOffice и некоторых других.

Какой HTML-редактор подходит именно вам, зависит от требований к продукту, который вы намерены создать с помощью HTML, от вашего текущего уровня знаний. В целом любой разработчик может пользоваться тем, что ему нравится, это не принципиально.

В курсе предлагается использовать программу Adobe Dreamweaver CC, которая является мощным и универсальным инструментом премиум-класса. Она обслуживает как бэкенд-, так и фронтенд-разработку. Имеются различные плагины к программе. Dreamweaver поддерживает как текстовые, так и WYSIWYG-методы работы с кодом. Таким образом, можно выбирать, работать с визуальным представлением страницы или классическим редактированием текста.

К недостаткам Adobe Dreamweaver CC можно отнести то, что это платный продукт, который распространяется на условии подписки.

Веб-браузеры

Для тестирования кода необходимо иметь несколько веб-браузеров. В настоящее время наиболее часто используются Firefox, Chrome, Opera, Safari, Microsoft Edge и Internet Explorer (IE). Необходимо также тестировать сайт на работу на мобильных устройствах и в любых старых браузерах, которые может использовать целевая аудитория сайта, например IE 6–8. Lynx подходит для того, чтобы увидеть, как сайт воспринимается слабовидящими пользователями.

Графический редактор

В веб-страницы включают изображения. Для их создания и обработки можно использовать графические редакторы, например Photoshop, GIMP, Paint.NET и др.

Система контроля версий

Сейчас активно используются репозитории кода с инструментарием контроля версий, позволяющие работать над проектом в команде, обмениваться кодом и избегать редакторских конфликтов, управлять файлами на сервере. Одним из наиболее популярных репозиториев является GitHub.

FTP-программа

FTP-программы позволяют загружать веб-страницы на сервер, например Cyberduck, Fetch и FileZilla.

Локальный веб-сервер

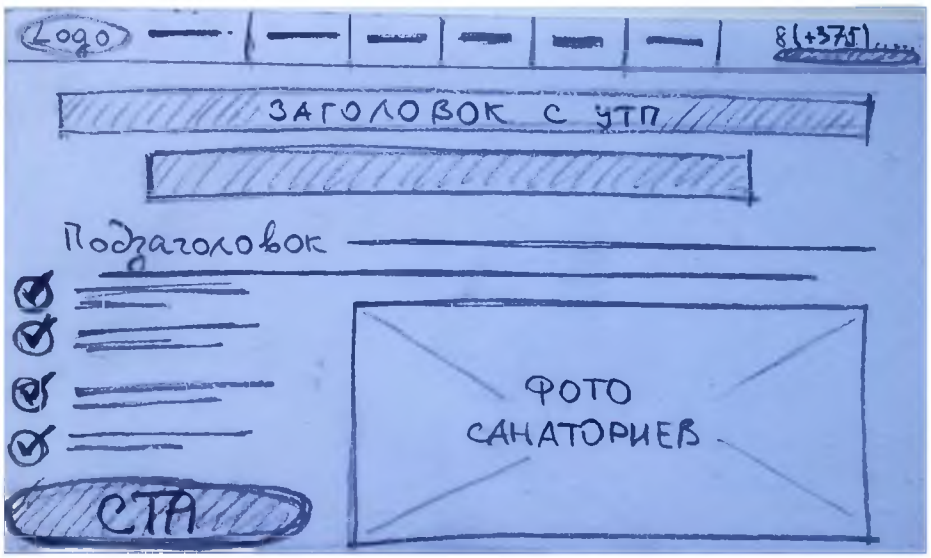
Некоторые страницы необходимо запускать на веб-сервере. Локальные серверы позволяют запускать свой сайт без использования хостинга, прямо на домашнем компьютере. Они используются в процессе разработки и тестирования. В качестве локального сервера можно использовать OpenServer, WampServer или XAMPP.

Упражнения

1. Установите на компьютере визуальный редактор Adobe Dreamweaver CC и Notepad++.
2. Для тестирования кода установите на компьютере несколько веб-браузеров (не менее пяти).
3. Найдите в интернете ресурс, позволяющий бесплатно тестировать веб-страницу на кросс-браузерность.
4. Укажите ссылку на ресурсе https://w3schools.com на страницу об HTML-элементах в обучающем курсе по HTML.
5. Найдите не менее пяти интернет-ресурсов с материалами для изучения HTML.

Планирование

Прежде чем создать сайт или веб-страницу, надо определиться, какую цель она будет реализовывать, что именно вы хотите, чтобы пользователь сделал на странице: прочитал информацию, нажал кнопку, заполнил форму и т. д. В зависимости от того, что пользователь должен будет сделать, нужно выстраивать дизайн страницы и подбирать контент. Под дизайном имеется в виду структура и оформление страницы. Нужно взять карандаш и бумагу и сделать примерный набросок страницы. Приведем пример наброска страницы для выбора путевки в санаторий.



Разработка сайта, как правило, начинается именно с набросков страниц на бумаге, по которым затем строятся цифровые макеты с использованием графических редакторов или веб-технологий.

Веб-команда крупных реальных проектов включает в себя также графических дизайнеров и дизайнера с опытом взаимодействия (user-experience (UX) designer). Графические дизайнеры работают над визуализацией веб-сайта. UX-дизайнеры разрабатывают концепцию того, как пользователи будут взаимодействовать с веб-сайтом.

Упражнения

1. Подумайте, что бы вы хотели разместить на своем персональном сайте и какие страницы вам необходимо будет создать для этого. Приведите список страниц.
2. Каким целям должен будет служить ваш персональный сайт? Какие действия, по вашему мнению, должны совершать пользователи на его страницах? Составьте эскизы всех страниц на бумаге и приведите список действий для каждой из них отдельно.
3. Составьте план структуры вашего персонального сайта. Какие именно страницы будут связаны ссылками? Для представления структуры сайта удобно воспользоваться любым онлайн-инструментом для составления ментальных карт.

Структура сайта на диске

Веб-сайт состоит из множества файлов: текстового контента, кода, стилей, медиаконтента и т. д. Для удобства структура папок сайта на локальном компьютере должна быть такой же, как и файловая структура опубликованного веб-сайта на сервере.

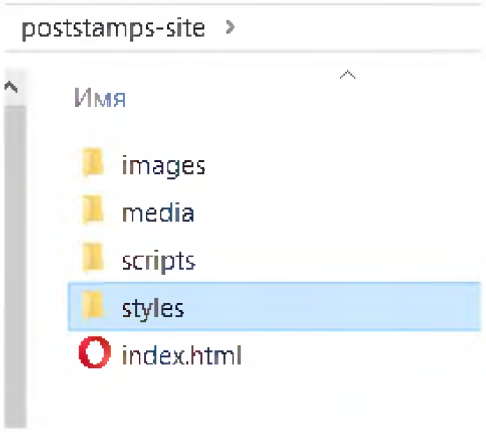
Все файлы сайта должны находиться в отдельной папке с именем проекта. Наиболее распространенные части, присутствующие в любом проекте сайта, – это HTML-файл главной страницы и папки, содержащие изображения, файлы стилей и файлы скриптов.

Файл index.html – это главная страница сайта. Его название зависит от требований сервера. Этот файл хранится прямо в корневой папке сайта.

Обычно создают папку images. В ней хранят все изображения, которые используются на сайте. Ее размещают также в корневой папке сайта. Там же создают папки media, styles, scripts. Папка media содержит медиаконтент сайта, styles – подключаемые таблицы стилей страниц, а scripts – весь JavaScript-код, используемый для добавления интерактивных функций на сайте.

Желательно называть папки и файлы полностью в нижнем регистре без пробелов. Регистр важен, поскольку, как правило, веб-серверы чувствительны к регистру.

Браузеры, веб-серверы и языки программирования не обрабатывают пробелы последовательно. Например, если использовать пробел в имени файла, то некоторые системы могут отнестись к данному имени файла как к двум именам. Некоторые серверы заменяют пробелы в имени файла на «%20» (символьный код для пробелов в URI), в результате чего могут пострадать ссылки. Лучше разделять слова дефисами или нижними подчеркиваниями. Однако нижние подчеркивания не всегда хорошо отображаются, поэтому my-file.html смотрится лучше, чем my_file.html. Следует учесть, что поисковая система Google рассматривает дефис как разделитель слов, а знак подчеркивания – нет. Поэтому лучше всего писать названия папок и файлов на английском языке в нижнем регистре без пробелов, разделяя слова дефисами. Это позволит в будущем сталкиваться с меньшим количеством проблем из-за совместимости программных продуктов.



Упражнение

Составьте план структуры на диске вашего персонального сайта.

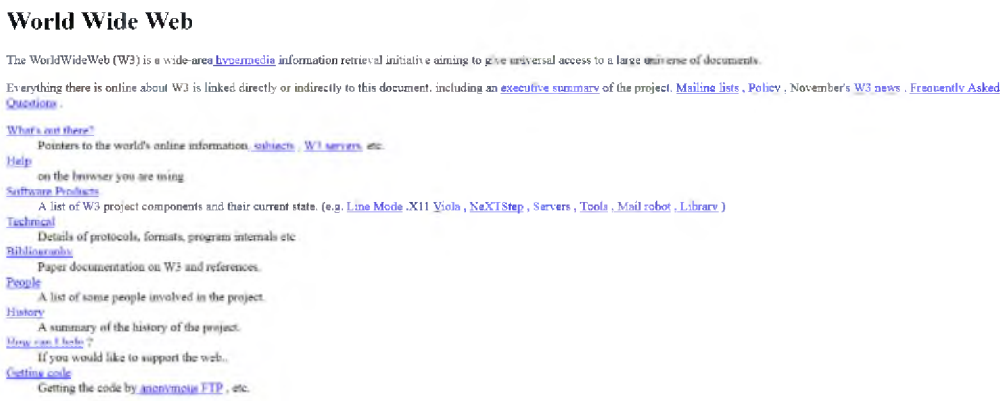
История возникновения HTML

В 60-е гг. XX в. американский ученый Джозеф Карл Робнетт Ликлайдер описал идею сети и назвал ее «Галактическая сеть». В 1969 г. американское агентство DARPA начало создавать экспериментальную сеть «с коммутацией пакетов». Ее назвали ARPANET. Коммутация пакетов – способ передачи данных по сети, когда информация делится на маленькие пакеты, которые отправляются независимо друг от друга. Это обеспечивает надежность, скорость и эффективность пересылки. В декабре 1970 г. в Network Working Group придумали протокол управления сетью, а в 1971–1972 гг. его реализовали в ARPANET. Благодаря этому появилась возможность создавать сетевые приложения. Первым приложением стала электронная почта в 1972 г. Интернет, каким мы его знаем, родился в 1989–1991 гг. в Швейцарии в стенах Европейского центра ядерных исследований (CERN) в результате работы команды во главе с Тимом Бернерсом-Ли и Робертом Кайо над созданием гипертекстовой информационной системы, способной связать бесконечное множество документов. В CERN были созданы все основные компоненты, из которых состоит Всемирная паутина:

- веб-сервер (назывался HTTPD);
- веб-клиент (первый интернет-браузер с именем World Wide Web – WWW);
- протокол HTTP для обмена между веб-сервером и веб-клиентом;
- язык HTML (стандарт представления веб-документов);
- система идентификации документов URI.

Результаты работы были представлены и выложены в глобальной сети для использования на бесплатной основе. Дальнейшей популяризации способствовала разработка браузера Mosaic, разработанного NCSA.

Первый в мире веб-сайт Бернерс-Ли создал по адресу http://info.cern.ch, теперь он хранится в архиве http://info.cern.ch/hypertext/WWW/TheProject.html. На этом сайте описывалось, что такое Всемирная паутина, как установить веб-сервер, как получить браузер и т. п. Этот сайт также являлся первым в мире интернет-каталогом, потому что позже Тим Бернерс-Ли разместил и поддерживал там список ссылок на другие сайты. Ниже приведен вид первой веб-страницы из архива.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project. Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Наибольшее распространение HTML получил в 1990-х гг., благодаря ему начался бурный рост популярности интернета. Первоначально под эгидой различных компаний язык развивался разными путями. Поскольку все разрабатываемые HTML-документы должны одинаково работать на различных платформах и в различных браузерах, то для унификации практических разработок в 1994 г. была создана специальная группа World Wide Web Consortium's HTML Working Group. Консорциум выпускает спецификации языка. Каждая новая версия HTML пытается добиться еще большей степени универсальности.

Хронология развития HTML:

Версия	Год
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

В настоящее время ведется разработка и апробирование HTML6.

Основное отличие HTML5 от предыдущих версий состоит в следующем:

- добавлены новые теги, в том числе семантические, например video, audio и ряд других;
- создан новый алгоритм парсинга для создания DOM-структуры документа;
- переопределены правила и семантика уже существовавших тегов HTML.

Фактически HTML5 можно считать не просто новой версией языка разметки для создания веб-страниц, а платформой для создания приложений, в том числе мобильных под Android, iOS, Windows Mobile и даже де-

сктопных приложений для обычных компьютеров (в частности, в операционной системе (ОС) Windows 8/8.1/10). Поэтому термин HTML5 используется чаще всего в двух значениях: либо как обновленный язык разметки гипертекста, либо как мощная платформа для создания веб-приложений, которая включает не только непосредственно язык разметки гипертекста, обновленный HTML, но и язык программирования JavaScript и каскадные таблицы стилей CSS3.

W3C. Стандарт HTML

Консорциум Всемирной паутины (World Wide Web Consortium, W3C) – независимая международная организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины. Консорциум возглавляет сэр Тимоти Джон Бернерс-Ли, автор множества разработок в области информационных технологий. Консорциум определяет стандарт HTML, CSS, HTTP, URI и др. в виде спецификаций. Текущую полную спецификацию на английском языке можно посмотреть по адресу https://www.w3.org/TR/html5/.

HTML разрабатывался с таким расчетом, чтобы все виды устройств: компьютеры с графическими дисплеями различного разрешения и глубины цвета, сотовые телефоны, переносные устройства, разговорные устройства, компьютеры с высокой и низкой тактовой частотой и т. д. – могли получать информацию из интернета. У консорциума есть собственный обучающий сайт: https://w3.org/ http://w3schools.com/html/default.asp.

Основные понятия

Введем некоторые понятия.

HTML-документ иначе еще называют HTML-страницей или веб-страницей. HTML-документы – это текстовые файлы с расширением .html или .htm, содержимое которых соответствует спецификации HTML.

Веб-сайт – это совокупность HTML-документов, взаимосвязанных темой, расположением и владельцем.

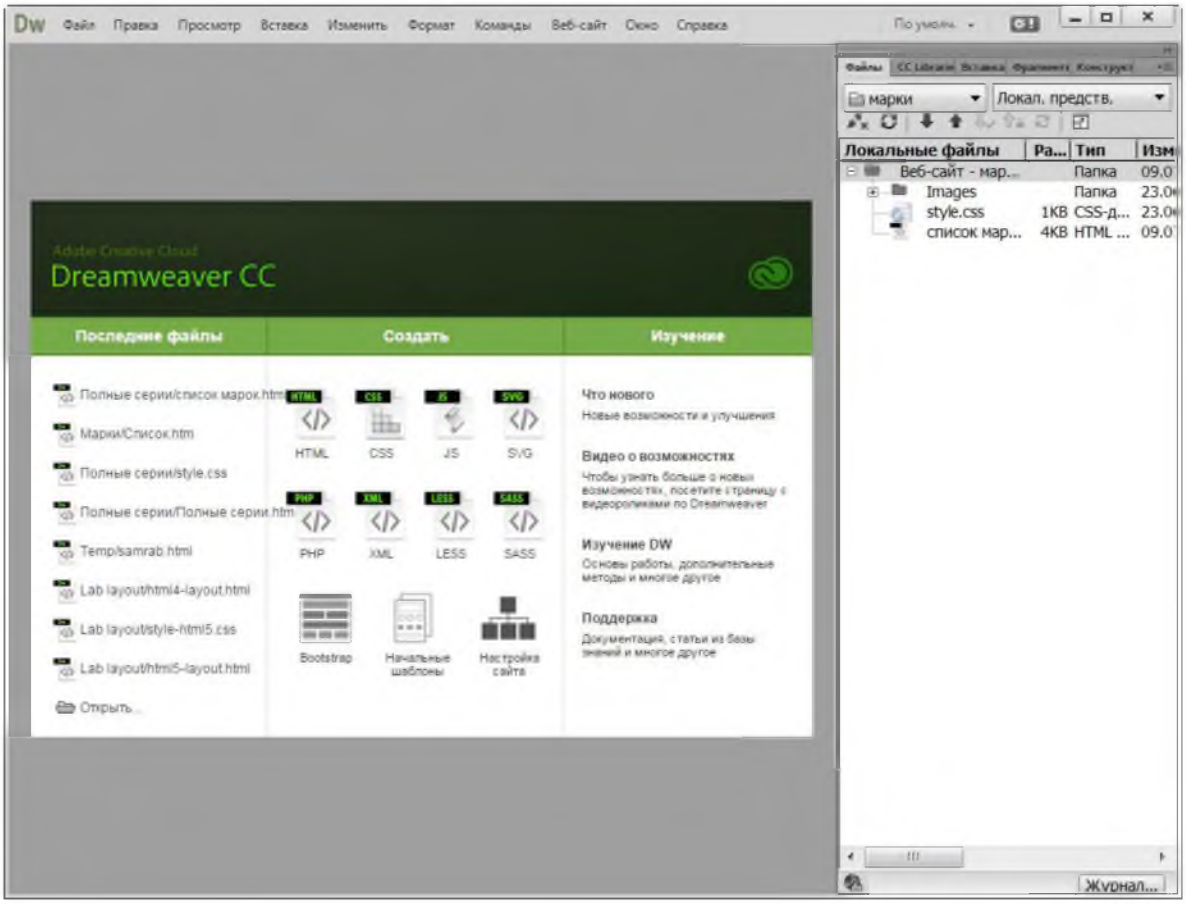
HTML определяет набор тегов или правил, согласно которым отображается текст в браузере. **HTML-теги**, или **HTML-элементы**, – это специальные дескрипторы, из которых состоит HTML-документ. HTML-документы могут быть созданы в любом текстовом редакторе и сохранены с расширением ***.htm*** или ***.html***. Самый простой вариант текстового редактора для создания HTML-файлов – это Windows Notepad. Однако удобнее использовать редакторы с подсветкой и подсказками синтаксиса, например:

- Notepad++ (http://notepad-plus-plus.org/);
- Notepad2 (http://flos-freeware.ch/);
- Sublime Text (http://sublimetext.com/);
- WebStorm (http://jetbrains.ru/products/webstorm/).

Для выполнения тестов, создания примеров удобно использовать песочницы, или online sandboxes:

- https://jsbin.com;
- https://jsfiddle.net;
- https://cssdeck.com;
- https://codepen.io.

Очень удобен визуальный редактор Adobe Dreamweaver CC.

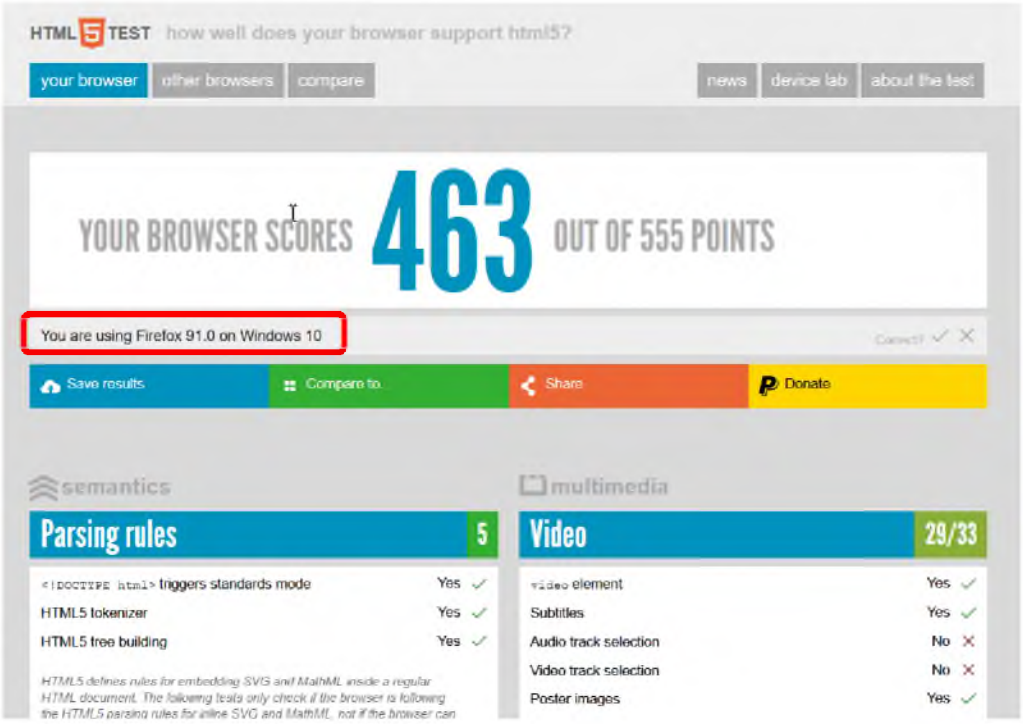


Поддержка браузерами последней версии HTML

Традиционно между спецификацией HTML5 и использованием этой технологии на практике всегда существовал разрыв. Браузеры работают на разных движках. Эти движки отвечают за загрузку, обработку, отображение и расчет данных. Каждый движок воспринимает информацию по-

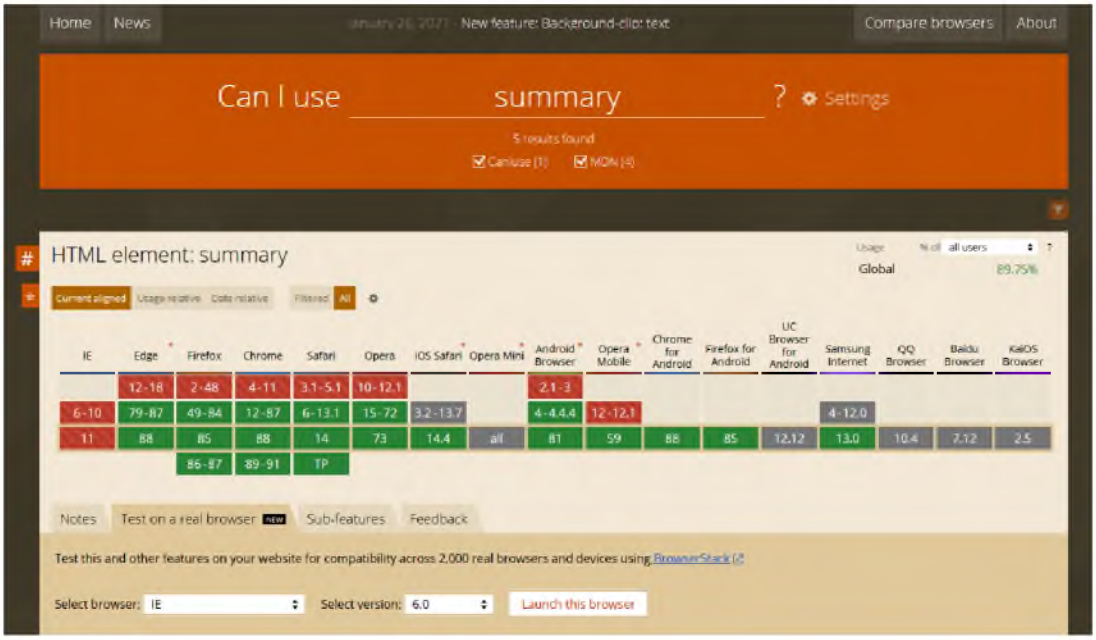
своему, а следовательно, по-разному интерпретирует HTML-теги и CSS-стили. Большинство браузеров начинают внедрять новые элементы и теги еще до того, как они официально войдут в спецификацию. Это приводит к тому, что к моменту попадания тега в спецификацию многие браузеры его уже поддерживают. К настоящему моменту последние версии популярных браузеров (Google Chrome, Firefox, Opera, IE 11, Microsoft Edge) поддерживают большинство возможностей HTML5. В то же время многие старые браузеры, например IE 8 и более ранние версии, их не поддерживают, а IE 9, 10 поддерживают лишь частично. При этом даже те браузеры, которые в целом поддерживают стандарты, могут не поддерживать какие-то отдельные функции, и это тоже надо учитывать в работе. Сайт редко одинаково выглядит во всех браузерах. В общем случае это и не нужно. Главное, чтобы не было критических ошибок.

Самым простым способом проверки кросс-браузерности сайта является ручная проверка. Нужно установить на компьютере три – пять основных браузеров и для начала просматривать веб-страницу в них. Следует также зайти на веб-страницу с планшета и мобильного телефона. Однако полноценный анализ кросс-браузерности возможен только с помощью специальных сервисов. Например, для проверки поддержки HTML5 конкретным браузером можно использовать сервис http://html5test.com. Для этого достаточно подключиться к интернету, запустить веб-браузер и зайти на сайт http://html5test.com/. Браузер будет автоматически проверен на совместимость с HTML5.



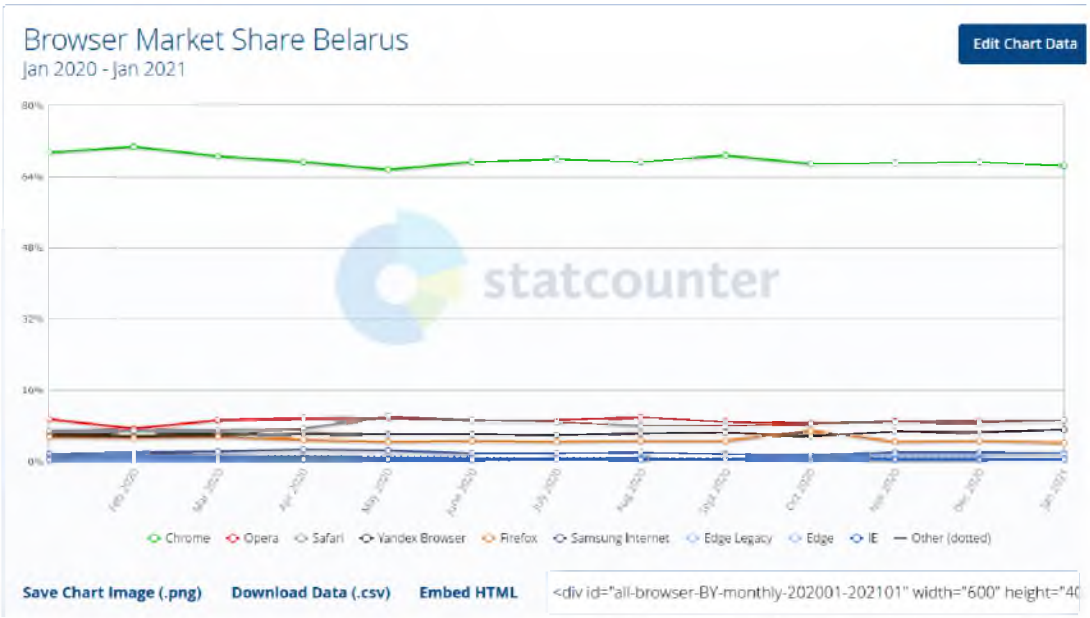
И с т о ч н и к: http://netler.ru/ikt/html5-test.htm.

Часто используют также сервис https://caniuse.com/. На этом сайте можно найти подробности поддержки HTML5-элементов во всех основных браузерах.



Поскольку некоторые элементы в различных браузерах имеют несколько отличающийся вид, то при разработке сайта надо учитывать, какие браузеры в основном используют посетители сайта.

Статистику популярности конкретных браузеров можно найти, например, на сайте https://gs.statcounter.com/. На этом сайте можно просмотреть популярность не только конкретного браузера, но и каждой из его версий, причем с выбором конкретного региона или страны и периода времени.



Упражнение

Используя любой из бесплатных сервисов, проведите анализ кросс-браузерности главной страницы сайта механико-математического факультета.

Нормализация стилей. Reset & Normalize

Каждый браузер имеет свой набор базовых стилей, которые он применяет к странице по умолчанию. Если создать страницу только с применением HTML без использования оформления и стилей, браузер все равно отобразит тег заголовка первого уровня <h1> большим размером и жирным начертанием, тег заголовка второго уровня <h2> – чуть меньшим размером, выделит текст в теге <i> курсивом, а текст в теге <u> сделает подчеркнутым и т. д. Это произойдет потому, что в каждом браузере уже есть свои стили для элементов, которые по умолчанию применяются к открываемым в нем страницам. Исторически сложилось так, что в различных браузерах эти правила немного отличаются. Раньше эти отличия были кардинальными и очень бросались в глаза, а сейчас они минимальны, но все же есть. Чтобы убрать различия и сделать по умолчанию отображение страницы во всех браузерах одинаковым, используют специальный CSS-файл: reset.css или normalize.css.

Первым исторически появился файл reset.css. Он содержит в себе перечисление всех возможных HTML-тегов и сбрасывает их значение на ноль, т. е. убирает все возможные отступы, делает шрифт одинаковым во всех тегах, сбрасывая все стили текста. Так что все заголовки и абзацы отображаются простым текстом, одним размером и без отступов. В результате получаем сброс стилей по умолчанию во всех браузерах.

Работает это так: сначала на странице подключается файл reset.css, и уже после него – свой файл со стилями. В итоге сбрасываются все стили, и уже потом в style.css задается собственное стилевое оформление. Таким образом достигается то, что все браузеры сбрасывают свои стили по умолчанию, и вся разметка будет основываться на тех стилях, которые будут указаны пользователем в style.css.

Скачать reset.css можно на сайте http://cssreset.com или http://meyerweb.com/eric/tools/css/reset/.

После того как на странице подключен reset.css, все стили приходится прописывать заново. Это не слишком удобно и занимает много времени. Поэтому на смену reset пришел другой инструмент – normalize.css. Normalize.css не сбрасывает все стили, а нормализует, приводит их к единому виду во всех браузерах.

Скачать normalize.css можно на сайте проекта https://necolas.github.io/normalize.css/ или на GitHub – https://github.com/necolas/normalize.css.

В настоящее время предпочтительно использовать normalize.css, поскольку он:

- сохраняет полезные стили по умолчанию;
- нормализует стили для широкого диапазона элементов;
- исправляет ошибки и общие несоответствия браузеров;
- повышает юзабильность;
- хорошо и подробно откомментирован;
- поддерживает мобильные устройства и HTML5.

Использовать normalize.css можно двумя способами.

Подход 1: можно использовать normalize.css как основу для базового CSS своего собственного проекта, добавляя свои стилевые правила и настраивая значения в соответствии с требованиями дизайна.

Подход 2: можно включить файл normalize.css в нетронутом виде в проект, а далее в своем стилевом файле, если это необходимо, переопределять значения по умолчанию.

Подключение файлов нормализации настроек:

```
/* Подключаем самым первым RESET.CSS или NORMALIZE.CSS,
как обычный CSS-файл */
<head>
  <link rel="stylesheet" href="css/normalize.css">
/* or <link rel="stylesheet" href="css/reset.css"> */
/* а далее - подключаем файл со своими стилями */
  <link rel="stylesheet" href="css/mystyles.css">
/* Теперь документ будет одинаков во всех браузерах */
<head>
```

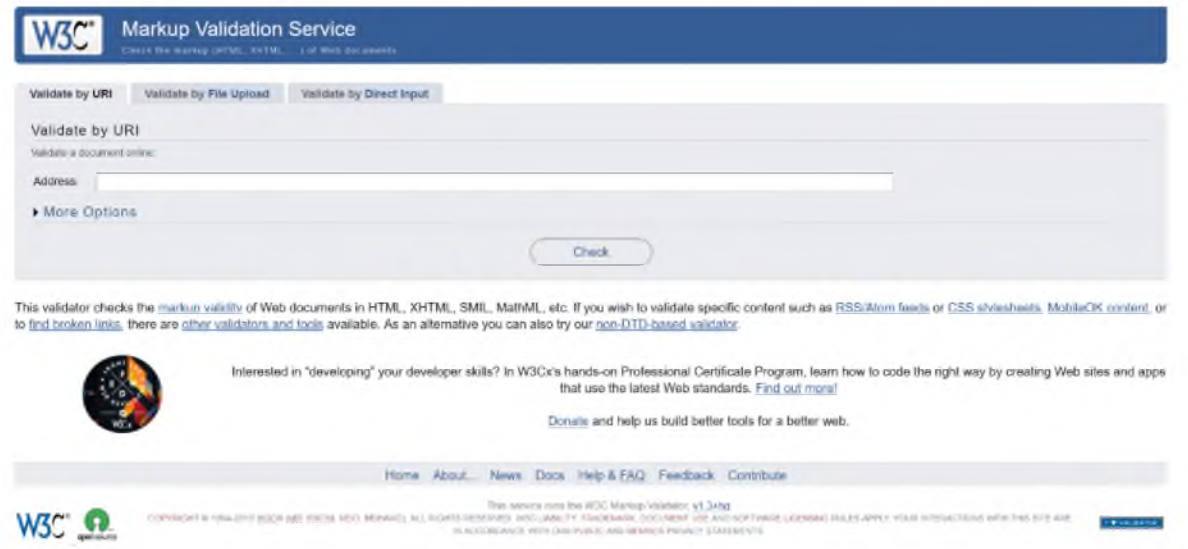
Упражнения

1. Создайте на основе HTML5 простую страницу о факультете, на котором учитесь. Контент можно взять, например, с сайта факультета.
2. На примере страницы, созданной в п. 1, покажите, как подключить к документу стилевой файл normalize.css двумя способами:
   а) с сайта проекта https://necolas.github.io/normalize.css/;
   б) из папки своего сайта.
3. На примере страницы, созданной в п. 1, покажите, как подключить к нему стилевой файл reset.css.

4. Не менее чем в пяти браузерах просмотрите, есть ли отличия при просмотре страниц, созданных с подключением reset.css в п. 2 и normalize.css в п. 1, *б*.

Валидация HTML-кода

Если в HTML-документе есть ошибки, то он все равно будет отображаться в браузере. Это происходит потому, что все браузеры имеют встроенные алгоритмы исправления ошибок и дополнения недостающего кода и в любом случае будут пытаться отобразить документ, руководствуясь своими алгоритмами. Однако в результате страница может отобразиться не так, как этого хотел разработчик. Корректность написания HTML-кода нужно проверять. Если он небольшой, то можно просто просмотреть весь код вручную и найти ошибки. Некоторые редакторы кода, например Adobe Dreamweaver CC, выводят сообщения об ошибках. Однако лучше всего проверять страницы в сервисе валидации разметки https://validator.w3.org/.



Его создал и поддерживает W3C. Сервис проверяет HTML-код и составляет отчет по ошибкам в нем. Можно проверить файл целиком или скопировать и ввести часть кода.

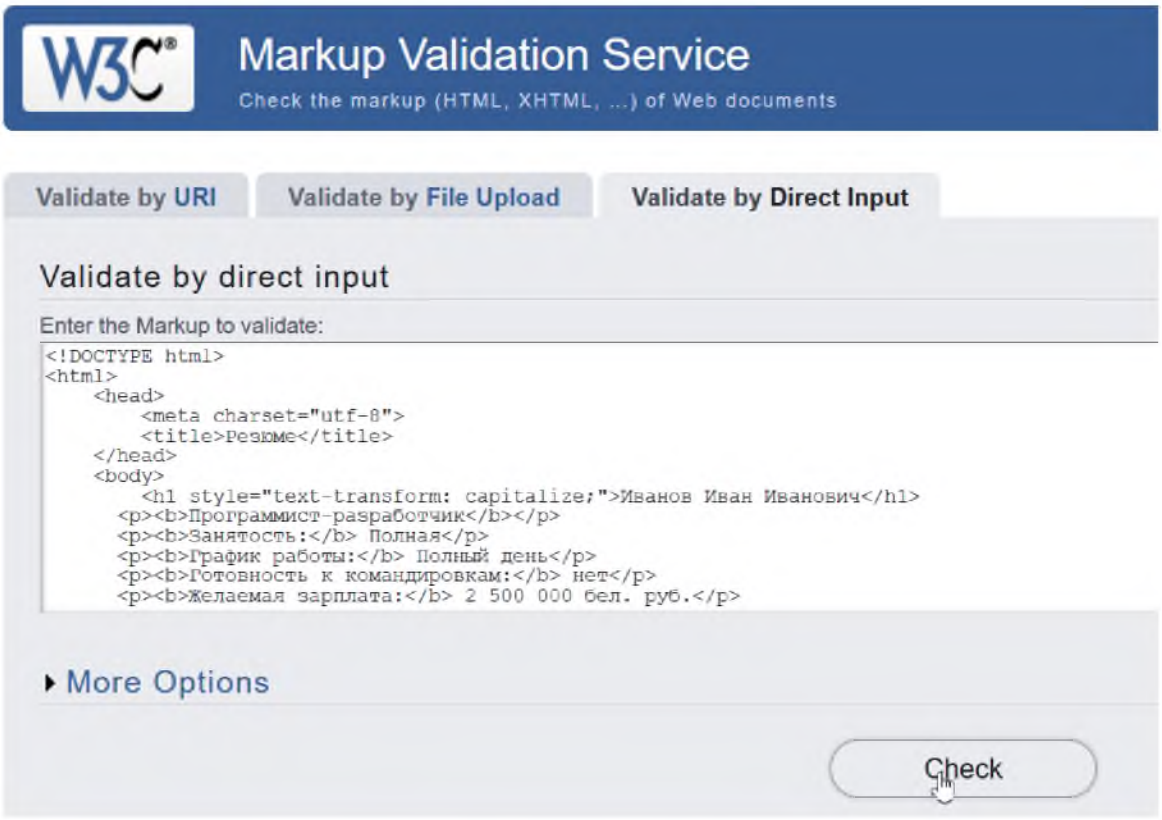
Для проверки кода надо:

1) открыть сервис валидации разметки в браузере;

2) перейти на вкладку Validate by Direct Input;

3) скопировать весь код документа (не только body) и вставить в место для ввода;

4) нажать на Check («Проверить»).



Упражнения

1. Проверьте на валидность главную страницу сайта механико-математического факультета.
2. Дан код, который позволяет встроить фрагмент карты на страницу. Проверьте его на валидность.

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <meta name="author" content="Иванов И.И.">
    <title>Мой университет на карте</title>
    <style>
      p {
        font-size: 1.5em;}
    </style>
  </head>
  <body>
    <p>Мой университет на карте. Именно тут я учусь</p>
    <iframe
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3
!1d2351.1278233003327!2d27.544860415679747!3d53.89393198
```

```
0097435!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!
1s0x46dbcfe70b729b83%3A0x7f694c68e4dfcd08!2z0L_RgNC-
0YHQv9C10LrRgiDQndC10LfQsNCy0LjRgdC40LzQvtGB0YLQuCA0LCDQ
nNC40L3RgdC6!5e0!3m2!1sru!2sby!4v1601449609394!5m2!1sru!
2sby"    width="600"    height="450"    frameborder="0"
style="border:0;" allowfullscreen="" aria-hidden="false"
tabindex="0"></iframe>
  </body>
</html>
```

Основные правила синтаксиса

Теги и элементы

HTML-теги образуют HTML-элементы и используются для указания браузеру, как он должен показывать и интерпретировать тот или иной контент.

HTML-элемент имеет следующую структуру:

открывающий тег + закрывающий тег + содержимое = элемент.

Большая часть тегов отображает вложенный контент, отдельные теги выполняют только служебные функции и не отображаются браузером. Названия тегов строго определены стандартом.

```
<body>
  … контент …
  <tagname   attr1="value1"  attr2="value2">
    … контент …
  </tagname>
```

Открывающий тег состоит из названия элемента, например em, помещенного внутри угловых скобок: . Данный тег служит признаком начала элемента и с этого момента начинает влиять на следующее после него содержимое.

Закрывающий тег выглядит так же, как и открывающий, но содержит слеш перед названием тега, например . Он служит признаком конца элемента. Текст, заключенный между открывающим тегом и закрывающим , будет отображаться курсивным начертанием. Теги бывают парными и непарными, или одиночными. Пропуски закрывающих тегов являются типичной ошибкой новичков, которая может приводить к неопределенным результатам. Каждый парный тег должен быть закрыт. Если не закрыть парный тег, то тег, скорее всего, отобразится, и отобразится правильно, поскольку все браузеры имеют встроенные алгоритмы исправления ошибок и дополнения недостающего кода. Браузер пытается отобразить любой документ, применяя эти алгоритмы, но не всегда они

обрабатывают код так, как хотел бы этого автор страницы. Поэтому всегда лучше самому делать разметку четко и досконально, не полагаясь на волю браузера.

Непарные (одиночные) теги

Одиночные теги не имеют закрывающего тега. Схема одиночных (непарных) тегов выглядит следующим образом:

```
<tagname   attr1="value1"  attr2="value2">
<tagname   attr1="value1"  attr2="value2"/>
```

Как правило, непарные теги используются для вставки какого-либо контента в то место, где они находятся, например изображения.

```
<img
  src="../Марки/Полные
серии/Images/Kyrgyzstan/Кыргызстан 1994 серия 7м Горные
цветы.jpg" alt="Горные цветы">
```

Атрибуты

После названия тега через пробел могут быть перечислены его атрибуты или параметры. Атрибуты содержат дополнительную информацию об элементе, которая, не должна отображаться в содержимом элемента.

Например, атрибут id позволяет дать элементу имя, которое в дальнейшем позволяет обращаться к этому конкретному элементу из JavaScript, а атрибут class может быть использован для назначения элементам, имеющим одинаковый атрибут class, определенных стилевых правил.

Атрибут может быть один, или их может несколько, или они могут вовсе отсутствовать. Могут быть обязательными или опциональными. После названия атрибута ставится знак равно, а затем в кавычках записывается его значение. Если атрибутов несколько, то они перечисляются через пробел.

```
<тег параметр1="значение" параметр2="значение">...</тег>
```

Например:

```
<font face="Arial, Helvetica, sans-serif" size="5"
color="blue">В тексте установлен тип шрифта, размер шрифта
и его цвет</font>
```

Кавычки могут использоваться как двойные, так и одинарные, но типы открывающих и закрывающих кавычек должны совпадать.

```
<font face='Arial, Helvetica, sans-serif' size='5'
color='blue'>В тексте установлен тип шрифта, размер шрифта
и его цвет</font>
```

Если значение атрибута без пробелов не заключить в кавычки, то, скорее всего, атрибут воспримется браузером верно. Если значение атрибута включает пробелы, то без кавычек браузер его не сможет интерпретировать правильно.

```
<!--Работает хорошо-->
<img src="My Penguins.jpg">
<!--Неверно, но все равно браузеры отображают-->
<img src=Penguins.jpg>
<!--Неверно и не работает-->
<img src=My Penguins.jpg>
```

Если значение атрибута опционально (определено) и состоит из одного слова, то кавычки можно опускать.

```
<font size=5 color=blue>В тексте установлен размер шрифта
и его цвет</font>
```

Если значение атрибута совпадает с его именем, то значение атрибута можно опускать. Например, `<ul compact="compact">` можно сократить до `<ul compact>`. Определенные браузеры даже требуют минимизации некоторых атрибутов, например таких, как compact, ismap, checked, nowrap, noshade, nohref. Порядок перечисления параметров в любом теге не влияет на результат отображения элемента.

```
<!--И с таким порядком перечисления атрибутов работает хорошо-->
<img alt="Пингвины" height="768" width="1024" src="My Penguins.jpg" />
```

Названия атрибутов предопределены стандартами. Исключение составляют атрибуты data-*.

Значения атрибутов могут:

- быть предопределенными (атрибут target тега <a> имеет значения из списка _self, _blank, _parent, _top);
- свободно задаваться разработчиком (для атрибутов id, class, src, href, title…).

В именах атрибутов нельзя использовать пробелы, кавычки, знак больше (>), слеш (/) и равно (=), а также любые символы, не определенные в Юникоде. В значениях атрибутов допустимо писать текст и спецсимволы, за исключением амперсанда (&), который должен заменяться на &.

Регистр написания тегов

HTML нечувствителен к регистру, поэтому названия тегов и их параметров можно записывать как прописными, так и строчными буквами.

```
<font size="5">Размер текста 5</font><br>
<FONT size="5">Размер текста 5</FONT><br>
<font SIZE="5">Размер текста 5</font><BR>
```

Стандарт HTML5 не требует написания тегов в нижнем регистре, но консорциум W3C **рекомендует использование нижнего регистра в HTML** и требует нижний регистр для более жестких типов документов, таких как XHTML.

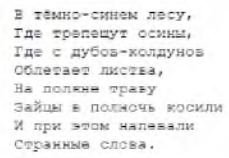
Пробельные символы

Контент внутри парных тегов

При выводе документа на экран пробелы и пустые строки не сохраняются, за исключением текста, заключенного в теги <pre>. Таким образом, в файле HTML любая последовательность подряд стоящих пробелов, символов табуляции и пустых строк эквивалентна одному пробелу.

```
<pre>В темно-синем лесу,
Где трепещут осины,
Где с дубов-колдунов
Облетает листва,
На поляне траву
Зайцы в полночь косили
И при этом напевали
Странные слова.</pre>
```

```
А нам все равно,
А нам все равно,
Пусть боимся мы волка и сову.
Дело есть у нас –
В самый жуткий час
Мы волшебную
Косим трын-траву.
```



А нам всё равно, А нам всё равно, Пусть боимся мы волка и сову. Дело есть у нас – В самый жуткий час Мы волшебную Косим трын-траву.

Как видно из рисунка, в результате выполнения первая часть кода, которая была заключена в тег <pre>, сохранила исходное форматирование. Во второй части контента все слова выведены в строку через один пробел.

Внутри тега между его параметрами

Внутри тега между его параметрами допустимо ставить любое число пробельных символов, в том числе перенос строк.

```
<img src="My Penguins.jpg"
     alt="Пингвины"
     height="768"
     width="1024">
```

Внутри тега в рамках одного атрибута

Внутри тега в рамках одного атрибута вставлять переходы на новую строку и тем самым разрывать значения атрибутов нельзя. Это не будет правильно отображаться браузерами. Например, следующий тег не будет отображен правильно.

```
<img src="My
Penguins.jpg" alt="Пингвины" height="200"/>
```

На этапе создания страницы и ее тестирования рекомендуется писать код классической программистской «лесенкой», принятой во всех языках программирования, где отступы отображают вложенность элементов. Гораздо легче разобраться, что происходит в коде, если он удобно отформатирован, а не просто собран вместе. Рекомендуется каждый вложенный элемент сдвигать на два пробела относительно элемента, в котором он находится. Многие редакторы при написании HTML-кода сами автоматически устанавливают отступы соответственно вложенности элементов. Можно использовать любое форматирование, в том числе различное количество пробелов для отступа, но лучше придерживаться одного стиля.

Вложенные элементы

Можно вкладывать элементы внутрь других элементов – это называется вложенностью.

Блочные и строчные элементы

Существует две важные категории элементов в HTML: элементы блочного уровня и строчные элементы.

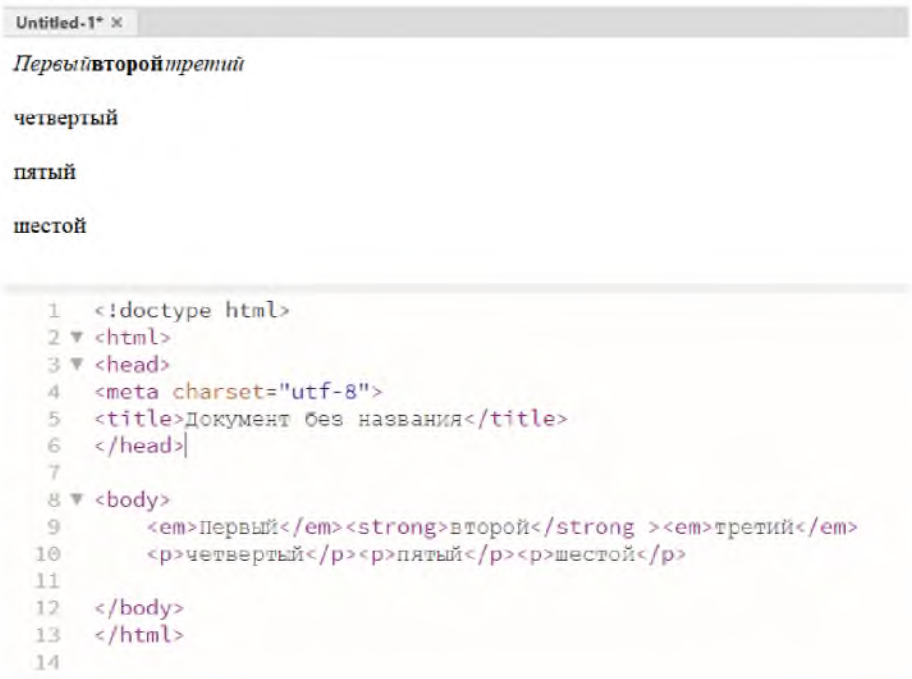
Элементы блочного уровня формируют видимый блок на странице. Они окажутся на новой строке после любого контента, который шел до них, и любой контент после них также окажется на новой строке. Чаще всего элементами блочного уровня бывают структурные элементы страницы, представляющие собой, например, параграфы (абзацы), списки, меню навигации, футеры, или подвалы, и т. п. Элементы блочного уровня не вкладываются в строчные элементы, но иногда могут вкладываться в другие элементы блочного уровня.

Строчные элементы содержатся в элементах блочного уровня и окружают только малые части содержимого документа. Строчные элементы не приводят к появлению новой строки в документе: они обычно встречаются внутри абзаца текста, например элемент <a> (ссылка) или акцентирующие элементы вроде или .

Резюмируя все вышесказанное, можно сказать:

- блочные элементы занимают все доступное пространство по ширине;
- строчные элементы ведут себя как текст – выстраиваются в ряд по горизонтали и переносятся на следующую строчку, если не хватает места.

Посмотрите на следующий пример.



 и – это строчные элементы, поэтому первые три элемента находятся на одной строке друг с другом без пробелов между ними. <p> – это элемент блочного уровня, так что каждый элемент находится на новой строке, с пространством выше и ниже каждого (этот интервал определяется CSS-оформлением по умолчанию, которое браузеры применяют к абзацам).

Следует отметить, что разделение элементов только на блочные и строчные использовалось в спецификации HTML до версии 4.01 включительно. Такое разделение не совсем точно описывало различные типы содержимого, поэтому в спецификации HTML5 модель содержимого была расширена, благодаря чему каждый элемент может принадлежать нулю, одной категории или более. Выделяют следующие общие категории:

- метасодержимое;
- потоковое содержимое;
- секционное содержимое;
- заголовочное содержимое;
- текстовое содержимое;
- встроенное содержимое;
- интерактивное содержимое.

Иерархия вложенности тегов

Существует определенная иерархия вложенности тегов, например, метатеги должны находиться только внутри тега <head>. Если теги между собой равноценны в иерархии связи, то их последовательность не имеет значения. Так, можно поменять местами теги <title> и <meta>, что никак не отразится на конечном результате.

Если несколько тегов вложены друг в друга, то порядок их закрытия должен быть обратным к порядку их открытия. Например:

```
<p>В тег абзаца
  <span> вложен span
    <em>и курсив</em>
  </span>
</p>
```

Правило вложенности тегов

Парные теги (контейнеры) могут содержать не только текст, но и другие теги. При этом действует одно правило: теги должны закрываться в порядке, обратном тому, в котором они открывались, т. е. теги вкладываются, но не пересекаются. Обратите внимание на расположение тегов в следующем примере.

Неправильно: <tag1> <tag2> <tag3> </tag2> </tag1> </tag3>.

Правильно: <tag1> <tag2> <tag3> </tag3> </tag2> </tag1>.

Чтобы запомнить это правило, представьте, что теги – это матрешки. Можно положить одну в другую, но нельзя одну половинку поместить внутри, а другую – снаружи. Это правило не относится к пустым тегам, которые попросту не имеют закрывающей пары.

Вложенные элементы называются дочерними, а те, в которые они вложены, – родительскими. Это относительное свойство, поскольку каждый узел может являться одновременно дочерним для одного и родителем для других. Элемент верхнего уровня (не имеющий родителей) называется корневым. В HTML корневым называется элемент <html>, внутри которого располагаются элементы <body> и <head>. Само присутствие открывающего тега <html> и закрывающего </html> является необязательным и их можно опустить, но, согласно спецификации, не стоит этого делать. Если теги находятся на одном уровне, то они называются ***сиблингами***.

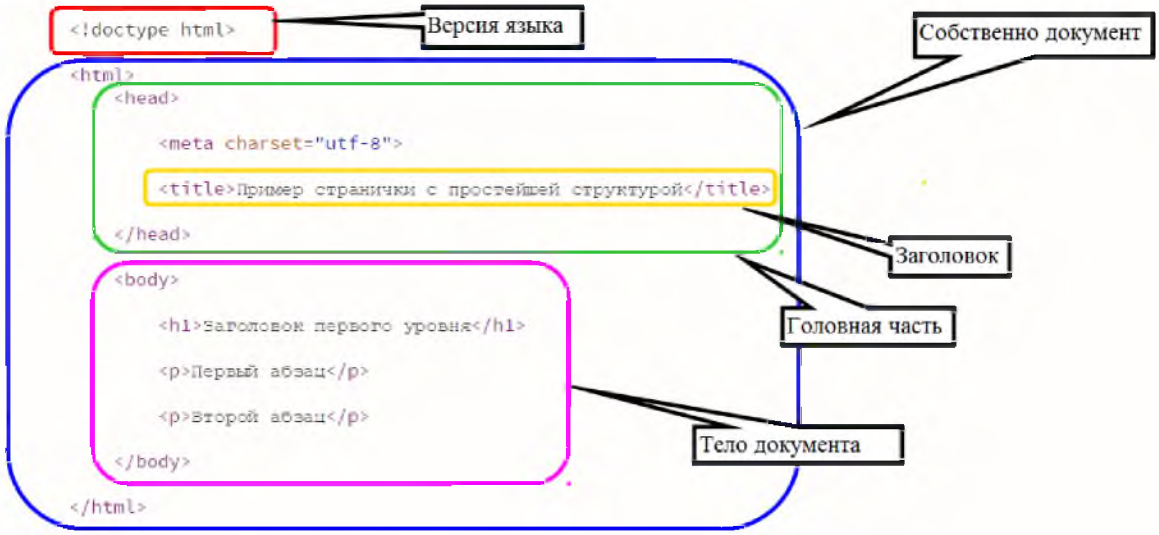
В данном примере тег <html> корневой, элементы <body> и <head> – сиблинги. Сиблингами также являются, например, <h1>, <p> и <table>. Для тега для отображения таблицы <table> теги строк <tr> являются дочерними, а для них, в свою очередь, дочерними являются элементы <th> и <td>.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>test</title>
  </head>
  <body>
    <h1>Успеваемость</h1>
    <p>Группа 01</p>
    <table width="200" border="1">
      <tbody>
        <tr>
          <td> </td>
          <td>c++</td>
          <td>java</td>
        </tr>
        <tr>
          <td>Иванов</td>
          <td>7</td>
          <td>8</td>
        </tr>
        <tr>
          <td>Петров</td>
          <td>8</td>
          <td>6</td>
```

```
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

Структура документа

HTML-документ формально делится на три части. В первой, необязательной части содержится информация об используемой версии языка HTML и правилах его форматирования. Вторая и третья части должны быть обязательно заключены в тег <html>, т. е. сам HTML-документ начинается с открывающего тега <html> и заканчивается закрывающим тегом </html>. Во второй части размещается заголовок документа, а в третьей – его тело. HTML-документ обязательно должен содержать теги <html>, <head>, <body>. Ниже приведен пример простейшего HTML-документа.



Результат выполнения:

Заголовок первого уровня

Первый абзац

Второй абзац

Элемент <!doctype html> объявляет тип документа.

Элемент <html> иногда называют «корневым элементом». Он является контейнером для всех других элементов, за исключением тега <!doctype html>. Для упрощения работы поисковых систем и браузеров в тег <html> включают атрибут lang, который содержит информацию о языке веб-страницы.

Элемент <head> выступает в качестве контейнера для всего содержимого, которое необходимо включить в HTML-документ, но нет надобности показывать посетителям страницы. Он включает такие вещи, как ключевые слова и описание страницы для поисковых машин, CSS для стилизации контента, объявление поддерживаемого набора символов и многое другое.

<meta charset="utf-8"> устанавливает в качестве символьной кодировки для документа UTF-8, который включает большинство символов из множества языков. По существу, страница с такой кодировкой сможет отобразить любой текстовый контент.

Элемент <title> устанавливает заголовок страницы, который появляется во вкладке браузера, загружающей эту страницу. Также это заглавие используется при описании страницы, когда ее сохраняют в закладках или избранном.

Элемент <body> содержит весь контент, который будет показан пользователям.

Это минимальный шаблон, в профессиональных проектах используют более развернутые варианты. Один из примеров такого шаблона можно скачать по адресу https://html5boilerplate.com/. Этот шаблон предлагает готовую общую структуру папок и файлов будущего проекта, готовый HTML5-шаблон и основные конфигурации для взаимодействия с сервером.

Очень популярен фреймворк Bootstrap (http://getbootstrap.com/, http://mybootstrap.ru/) – свободный набор инструментов для создания сайтов и веб-приложений. Bootstrap включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

У новичков часто возникает вопрос: «Зачем придавать структуру текстовому контенту?» Если разместить на странице весь текстовый контент без разбивки на элементы, то в браузере текст будет выглядеть сплошным. Это может иметь негативные последствия.

1. Пользователи, просматривающие веб-страницу, быстро сканируют ее в поисках подходящего контента, часто просматривая только заголовки. Если они не смогут увидеть ничего полезного в течение нескольких секунд, то, скорее всего, уйдут со страницы.
2. Поисковые системы, индексирующие страницу, считают содержание заголовков важными ключевыми словами для влияния на ранжирование поиска страницы. Без заголовков страница будет плохо работать с точки зрения SEO.

3. Слабовидящие люди часто не читают веб-страницы, а слушают их. Это делается с помощью программы чтения с экрана. Такое программное обеспечение предоставляет способы быстрого доступа к текстовому контенту. Среди различных используемых методов программы-ридеры предоставляют схему документа, считывая заголовки и позволяя своим пользователям быстро находить нужную им информацию. Если заголовки недоступны, пользователи будут вынуждены прослушивать весь документ.
4. Чтобы стилизовать контент с помощью CSS или сделать его интересным с помощью JavaScript, нужно, чтобы элементы обертывали соответствующий контент, тогда CSS и JavaScript смогут эффективно работать.

Информация об используемой версии языка HTML

В самом начале HTML-документа, перед секцией заголовка, располагается элемент <!doctype>. Он предназначен для указания типа текущего документа и правил форматирования документа. Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, поскольку сам HTML существует в нескольких версиях, кроме того, имеется XHTML (Extensible HyperText Markup Language – расширенный язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису. Чтобы браузер «не путался» и понимал, согласно какому стандарту отображать веб-страницу, необходимо в первой строке кода задавать <!doctype>. Указание типа документа, точной версии языка, при помощи которого он создан, в исходнике документа не обязательно, но считается хорошим стилем программирования. Это позволяет браузеру точно определить тип документа, проверить его структуру на соответствие указанным правилам и соответственно отобразить его в соответствии с этими правилами. Если тип документа не указан явно, то он определяется автоматически по расширению файла и некоторым ключевым словам. Правила написания HTML-кода объединены в файлы, содержащие описание типа документа, так называемые DTD-файлы (*Document Type Definition*).

Тег <!doctype>, как и все остальные, нечувствителен к регистру.

HTML5	
<!doctype html>	HTML5
HTML 4.01	
<!doctype HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	Строгий синтаксис, не включает в себя устаревшие теги и теги управления фреймами

<!doctype HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Переходный синтаксис, включает в себя в том числе и устаревшие теги и их атрибуты
<!doctype html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">	Включает в себя все теги и их атрибуты, а также теги управления фреймами
XHTML 1.0	
<!doctype html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	Строгий синтаксис XHTML
<!doctype html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">	Переходный синтаксис XHTML
<!doctype html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">	Документ написан на XHTML и содержит фреймы
<!doctype html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">	Документ написан на XHTML
<!doctype html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">	XHTML Basic 1.1

Когда HTML-документ загружается на компьютере пользователя, то с сайта https://w3.org/ загружается соответствующий DTD-файл, согласно которому затем проверяется правильность HTML-документа.

W3C-спецификацию doctype можно посмотреть на сайте https://w3.org/QA/2002/04/valid-dtd-list.html.

Секция заголовка <head>

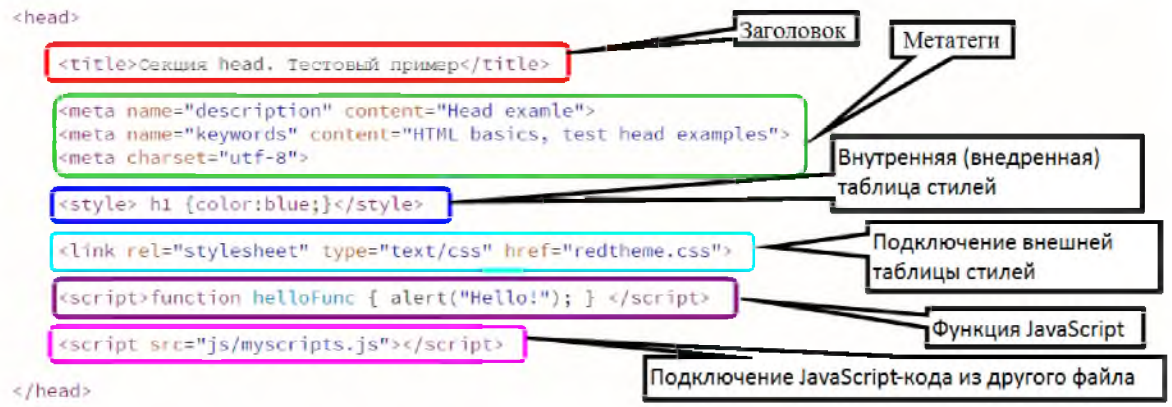
Секция заголовка <head> может содержать служебную информацию для описания и индексирования документа поисковыми машинами, подключения ресурсов из внешних источников, скрипты, стили. Теги секции <head> не отображается веб-обозревателями, исключение составляет <title>.

Внутри контейнера <head> допускается размещать следующие элементы:

- <title> – определяет заголовок вкладки браузера, в которой открыта страница;
- <meta> – содержит информацию для браузера и поисковых машин;

- <style> – содержит внутренние CSS-стили для элементов страницы;
- <link> – подключает внешние CSS-стили (.css) и другие ресурсы;
- <script> – является контейнером для JavaScript-кода и подключает внешние файлы со скриптами (.js);
- <noscript> – показывает свое содержимое, если браузер не поддерживает работу со скриптами или их поддержка отключена пользователем;
- <base> – определяет URL-базу для всех относительных URL на странице и значение target по умолчанию для гиперссылок.

Ниже на рисунке приведен пример типичной секции head.



<title>

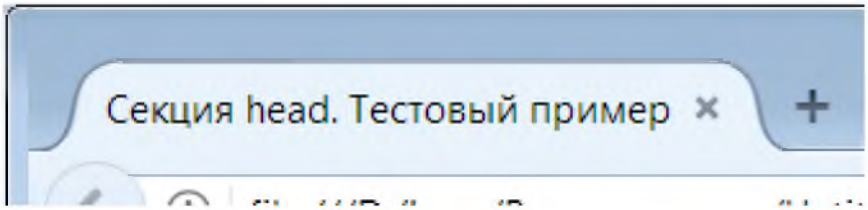
Элемент <title> может отображаться только как название окна в браузере. В документе может быть только один тег <title>. Если тег <title> отсутствует, то документ не будет считаться валидным.

Элемент <title>:

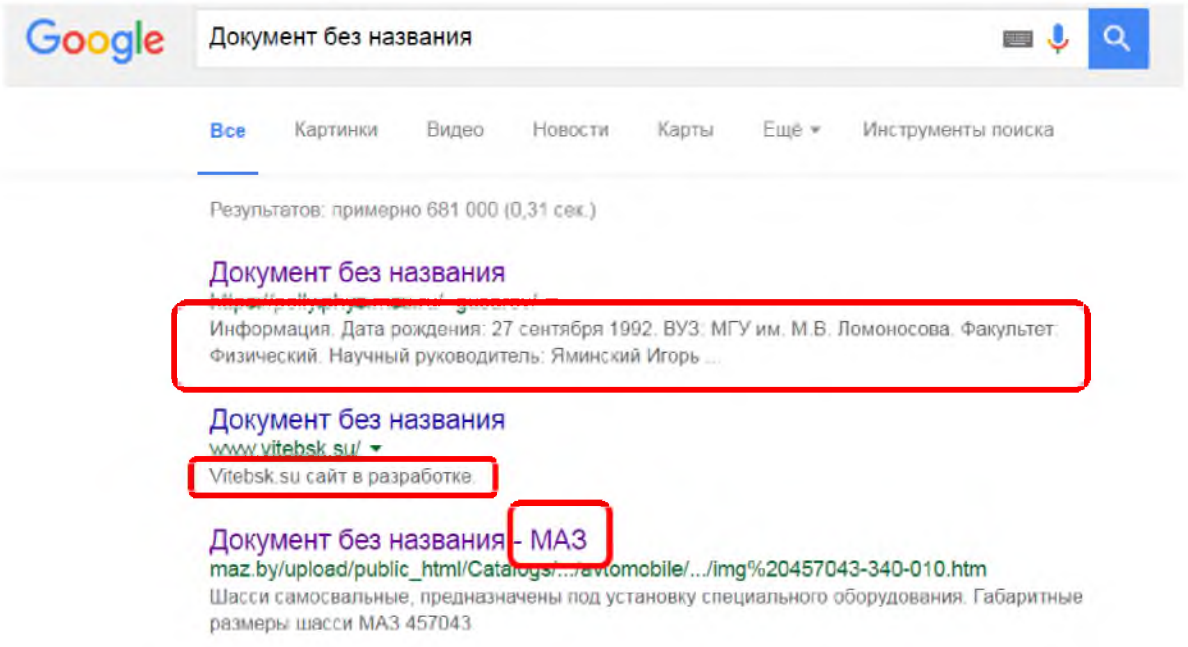
- определяет заголовок на панели инструментов браузера;
- дает название для страницы, когда она добавляется в «Избранное»;
- отображает заголовок страницы в результатах поисковой системы.

Тег <title> используется поисковыми машинами для идентификации содержимого документа, и поэтому необходимо давать осмысленные заголовки. Так, вместо заголовка «Введение», который не дает достаточно информации о документе, надо записать, например, «Введение в высшую математику». В идеале в заголовок должны войти все ключевые слова, по которым предполагается продвигать страницу в поисковых системах, а лучше целый запрос. Название должно быть не более 70 символов: поисковые машины не воспринимают более длинный текст. Кроме того, название, превышающее этот размер, может не поместиться в строку заголовка окна браузера.

```
<title>Секция head. Тестовый пример</title>
```



Если страница создается в какой-либо системе управления контентом, то CMS часто автоматически вносит стандартный ***<title>***. Если забыть его переопределить, то может случиться, что страница будет продвигаться в поисковой машине по такому автоматически созданному заголовку. Пример неудачного продвижения сайтов по фразе «Документ без названия»:



К тегу <title> можно обратиться через DOM-модель документа (https://w3schools.com/jsref/dom_obj_title.asp).

Например, получить содержимое тега <title> можно следующим образом:

```
var x = document.getElementsByTagName("TITLE")[0].text;
```

Следующий код позволяет создать тег <title> и добавить в него содержимое:

```
var x = document.createElement("TITLE");
var t = document.createTextNode("HTML DOM Objects");
x.appendChild(t);
document.head.appendChild(x);
```

Упражнения

1. Создайте веб-страницу с вашим резюме. Продумайте, что должен содержать тег <title> страницы с вашим резюме на вашем же персональном сайте для наилучшего ее продвижения в поисковике и удобства пользователей. Проверьте, правильно ли вы выбрали заголовок. Для этого наберите в поисковике Google запрос, по которому вы бы хотели, чтобы находилось ваше резюме, и посмотрите выдачу. Если в выдаче не будут присутствовать страницы с аналогичными резюме, то содержимое <title> вы подобрали неверно.
2. В созданный документ динамически, используя JavaScript, добавьте тег <title>. Содержимое тега возьмите из п. 1.

<meta>

Метатеги содержат служебную информацию, не отображаемую в окне браузера, или метаданные. Метаданные – это информация о данных, в данном случае о документе HTML. Метаданные не отображаются на странице, но обрабатываются поисковой машиной (ключевые слова) и используются браузером (информация о том, как отображать страницу) или другими веб-службами.

Метаэлементы обычно используются для указания описания страниц, автора документа, последнего изменения и т. д. В частности, метаданные могут представлять собой собственные форматы социальных сетей или других сайтов, созданные для предоставления им специальной информации, которую они могут использовать.

Разрешается использовать более чем один метатег. Все они должны быть размещены в контейнере <head>.

Пример определения описания страницы:

```
<meta name="description" content="Head example">
```

Пример определения автора страницы:

```
<meta name="author" content="Ivanov Ivan">
```

Пример обновления документа каждые 30 секунд:

```
<meta http-equiv="refresh" content="30">
```

К тегу <meta> можно обратиться через DOM-модель документа. Например, следующим образом можно получить ссылку на первый метатег:

```
var x = document.getElementsByTagName("META")[0];
```

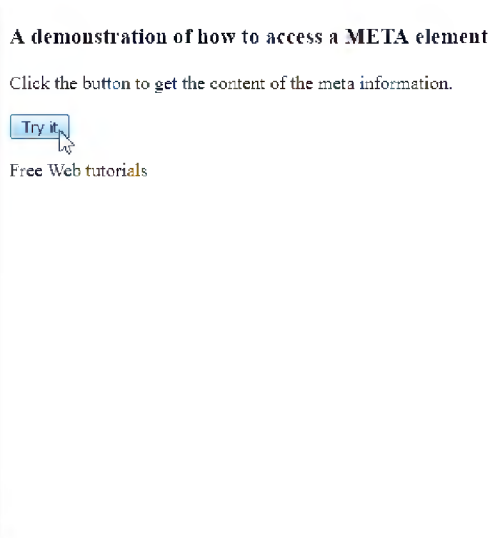
Можно динамически создать метатег определения описания страницы:

```
var x = document.createElement("META");
```

```
x.setAttribute("name", "description");
x.setAttribute("content", "Free Web tutorials");
document.head.appendChild(x);
```

Или следующим образом:

```
<!DOCTYPE html>
<html>
<head>
<meta name="description" content="Free Web tutorials">
</head>
<body>

<h3>A demonstration of how to access a META element</h3>

<p>Click the button to get the content of the meta information.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
    var x = document.getElementsByTagName("META")[0].content;
    document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```



У тега <meta> есть несколько атрибутов.

name

Атрибут name указывает имя для информации, которая хранится в атрибуте content. Часто используется для указания имени автора. В этом случае атрибуту name присваивается значение author, и поисковые системы смогут найти нужный сайт по имени автора.

```
<meta name="author" content="Ivanov, Иванов Иван">
```

Может содержать краткое описание сайта, используемое поисковой машиной для индексирования. В этом случае атрибуту name присваивается значение description.

```
<meta name="description" content="The best homepage">
```

Часто description используют, чтобы повлиять на выбор сниппета поисковой машиной. Согласно результатам исследования американского SEO-специалиста Эвана Холла на первой странице выдачи Google выводит метатег description для сниппета примерно в 30 % случаев.

Метатег может содержать ключевые слова, используемые поисковым роботом. В этом случае атрибуту name присваивается значение keywords. Длина содержимого метатегов keywords не должна превышать 1000 символов. В настоящее время в плане продвижения практически не дает эффекта, но слова, входящие в content, включаются в общее число при подсчете повторений слов на странице.

```
<meta name="keywords" content="META, HTML, WWW, Web, паутина, поиск, определение, рекомендации, примеры использования, учебник, руководство, справка">
```

Метатег может указывать на один из пакетов программного обеспечения, используемых для создания документа (не используется на страницах, созданных вручную). В этом случае атрибуту name присваивается значение generator.

```
<meta name="generator" content="Joomla! 1.5 - Open Source Content Management">
```

В HTML5 был введен метод, позволяющий контролировать видимую область на веб-странице с помощью тега <meta>. В этом случае name устанавливается значение viewport. Ниже приведен пример настройки области просмотра, чтобы сайт выглядел хорошо на всех устройствах.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
```

http-equiv

Атрибут http-equiv служит альтернативой атрибуту name. Используется так же, как и атрибут name. Атрибут http-equiv может применяться для имитации заголовка ответа HTTP. При отображении страницы браузер будет следовать инструкциям, заданным в атрибуте, при отсутствии их в ответе сервера.

Может принимать значения content-type, default-style или refresh.

Например, content-type указывает тип документа:

```
<meta http-equiv = "content-type" content = "text / html; charset = UTF-8">
```

Атрибут http-equiv, определенный как default-style, указывает на таблицу стилей по умолчанию.

```
<meta http-equiv="default-style" content="the document's preferred stylesheet ">
```

Данный атрибут может использоваться для автоматического обновления HTML-документа через определенное количество секунд. В этом случае атрибуту http-equiv присваивается значение refresh.

Пример перезагрузки HTML-документа через каждые 7 с:

```
<META http-equiv="refresh" content="7">
```

Пример перенаправления пользователя по истечении 7 с на сайт bsu.by:

```
<meta http-equiv="refresh" content="7";
        url="http://www.bsu.by>"
```

content

Атрибут content указывает значение, связанное с атрибутом http-equiv или name.

charset

Атрибут определяет кодировку символов, которая должна быть использована на компьютере пользователя при просмотре данного HTML-документа.

```
<meta charset="utf-8">
```

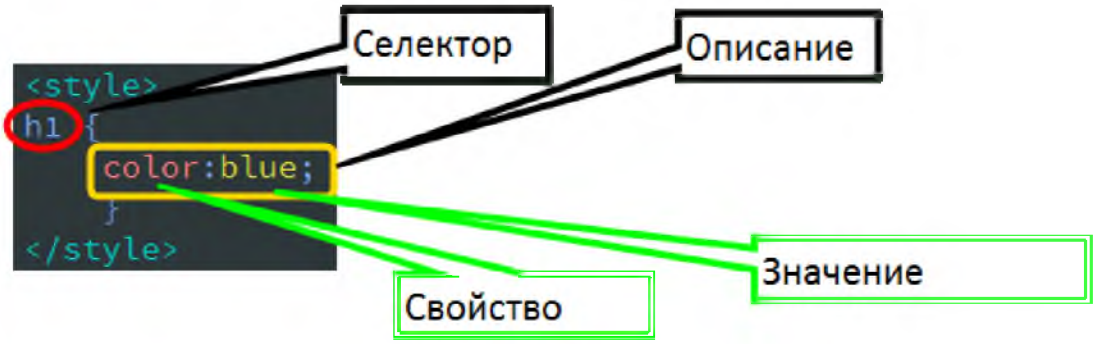
Упражнения

1. В поисковике Google введите запрос «информационный портал». Откройте страницы сайтов, попавших в топ-3, и проанализируйте содержимое атрибута content в теге <meta> с атрибутом name="description". Как содержимое атрибута content соотносится со сниппетом страницы? Термином «сниппет» называют небольшие отрывки текста из найденной поисковой машиной страницы сайта, использующиеся в качестве описания ссылки в результатах поиска. Встречается ли текст из сниппета в контенте страницы, который предназначен для пользователей? Почему? На каждой из трех открытых ранее страниц информационных порталов перейдите по ссылке на главную новость. Оцените содержимое атрибута content в теге <meta> с атрибутом name="description" уже для веб-страниц конкретных новостей. Посмотрите, с содержимым каких тегов совпадает значение атрибута content. На основании своих наблюдений сделайте выводы о том, что именно должно быть записано в атрибуте content в теге <meta> с атрибутом name="description".
2. Оцените с точки зрения продвижения в поисковике Google, правильно ли написаны теги <meta> с атрибутом name="description" на страницах сайта, который вы чаще всего посещаете.

<style>

Внутренние, или внедренные, таблицы стилей CSS расположены внутри HTML-документа, в его головной части, в теге <style>. Тег <style> используется для определения информации стиля для одной HTML-страницы. Каждый HTML-документ может содержать несколько тегов <style>.

На рисунке ниже приведена схема размещения стиля в теге <style>.



<link>

Тег устанавливает связь с внешними ресурсами, например, присоединяет стилевые файлы, файлы шрифтов, иконки и т. д. Элемент <link> не содержит контента, тег содержит только атрибуты. Синтаксис тега следующий:

```
<head>
<link атрибуты>
</head>
```

Тег имеет следующие атрибуты:

- href содержит путь к связываемому файлу;
- hreflang указывает язык текста в связанном документе;
- media определяет устройство, на котором будет отображаться связанный документ;
- rel определяет отношения между текущим документом и файлом, на который делается ссылка;
- sizes указывает размер иконок для визуального отображения;
- type содержит MIME-тип данных подключаемого файла.

Примеры:

```
<link rel="stylesheet" href="ie.css">
<link rel="shortcut icon"
href="http://mysite.by/apple.ico">
```

Тег <link> также может содержать глобальные атрибуты.

<script>

Тег <script> предназначен для описания скриптов, может содержать ссылку на программу или ее текст на определенном языке. Обычное использование JavaScript – это манипулирование изображениями, проверка правильности заполнения формы и динамические изменения содержимого. Скрипты могут располагаться во внешнем файле и связываться

с любым HTML-документом. Такой подход позволяет использовать одни и те же общие функции на многих веб-страницах и ускоряет их загрузку, так как внешний файл кэшируется при первой загрузке и скрипт вызывается быстрее впоследствии.

Тег <script> может располагаться в заголовке или теле HTML-документа в неограниченном количестве. В большинстве случаев местоположение кода скрипта никак не сказывается на работе программы. Однако скрипты, которые должны выполняться в первую очередь, обычно помещают в заголовок документа.

Пример тега <script> в теле документа:

```
<!doctype html>
<html>
  <body>
    <p id="demo"></p>
    <script>
      document.getElementById("demo").innerHTML = "Hello
JavaScript!";
    </script>
</body>
</html>
```

В результате на страницу в абзац с именем demo будет выведен текст "Hello JavaScript!". Пример нескольких подключений тега ***<script>*** в различных местах документа дан ниже.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Документ без названия</title>
 <!--Скрипт в head с необязательным атрибутом type.
Выполняется немедленно. -->
    <script>
      document.write("Hello javascript from head!")
    </script>
<!--Скрипт в head без атрибута type. Выполняется
немедленно. -->
    <script>
      alert("Hello!!!")
    </script>
<!--Скрипт присоединяет внешний файл сценария через
атрибут src в head. Есть атрибут defer скрипт выполняется,
когда страница закончит разбор. Без defer не выполнится,
так как тег body, к которому обращается скрипт, еще не
сформирован. -->
```

```
    <script src="script5.js" defer>
  </script>
  </head>
  <body>
    <p id="demo"></p>
    <p id="demo1"></p>
<!--Скрипт присоединяет внешний файл сценария через
атрибут src в body. -->
    <script src="script3.js" defer></script>
<!--Скрипт в body. Выполняется, как встретится на
странице. -->
    <script>
      document.getElementById("demo").innerHTML = "Hello
JavaScript from body!";
    </script>
  </body>
</html>
```

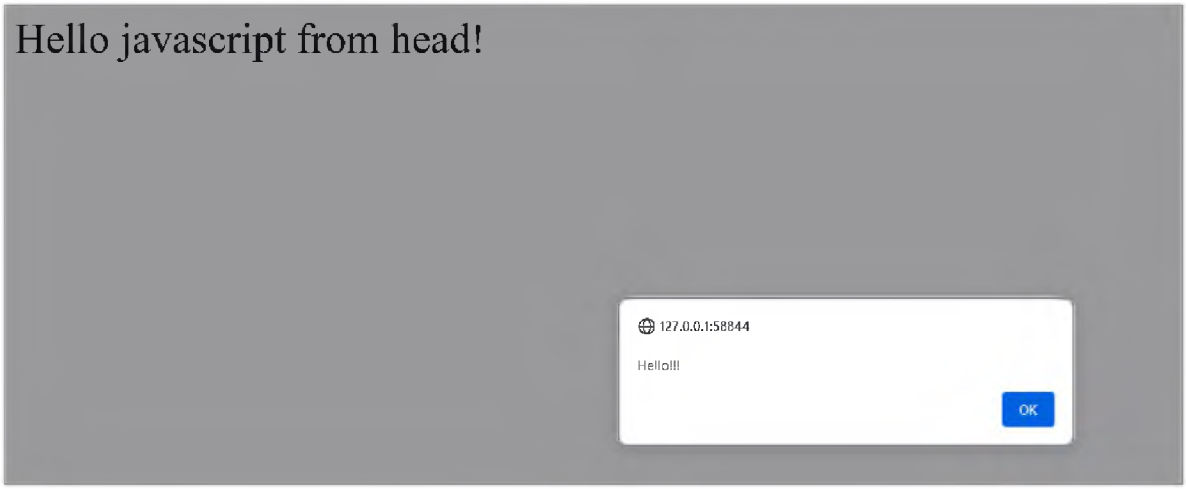
Содержимое файла script5.js:

```
var x = document.getElementsByTagName("BODY")[0];
    x.style.backgroundColor = "cyan";
```

Содержимое файла script3.js:

```
document.getElementById("demo1").style.fontSize = "45px";
document.getElementById("demo1").style.color = "red";
document.getElementById("demo1").innerHTML = "This text
comes from an external script. Call from body.";
```

Результат выполнения:

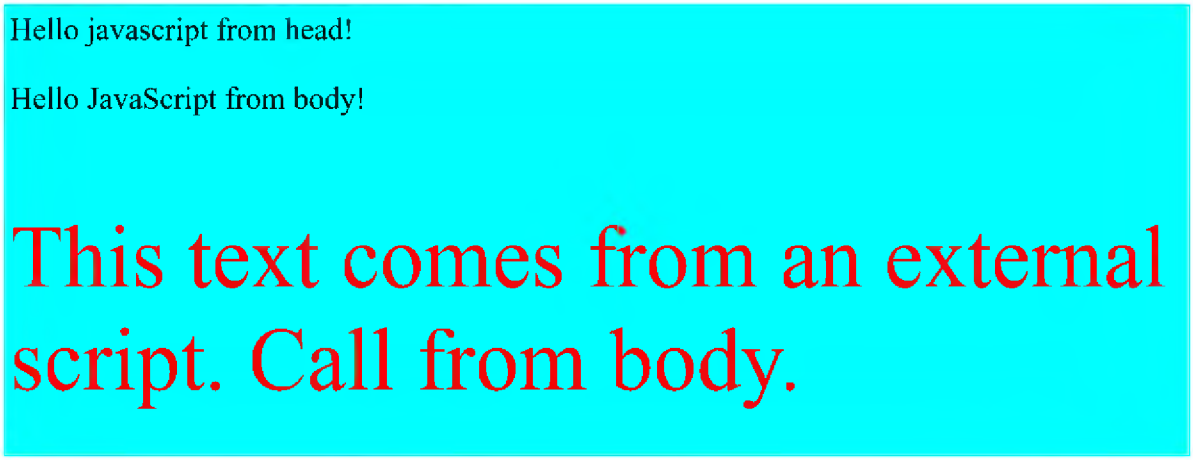


Выполнились два первых скрипта из <head>. Затем в <head> присоединяется внешний файл сценария script5.js через атрибут src.

```
var x = document.getElementsByTagName("BODY")[0];
    x.style.backgroundColor = "cyan";
```

Изменяется цвет фона в документе. Есть атрибут defer – скрипт выполняется, когда страница закончит разбор. Без defer не выполнится, так как тег body, к которому обращается скрипт, еще не сформирован.

В теле документа встречается скрипт script3.js, который изменяет цвет и размер шрифта в параграфе с id="demo1" и вписывает туда текст. Скрипт, размещенный перед закрывающим тегом </body>, выполняется, как встретится на странице.



Рассмотрим атрибуты тега <script>.

src

Содержит адрес подключаемого файла сценария.

async

Используется только для внешних скриптов и указывает, что скрипт выполняется асинхронно. Это означает, что скрипт из указанного в атрибуте src файла будет выполняться без ожидания загрузки и отображения веб-страницы. В то же время и страница не ожидает результата выполнения скрипта, а продолжает загружаться как обычно.

Пример.

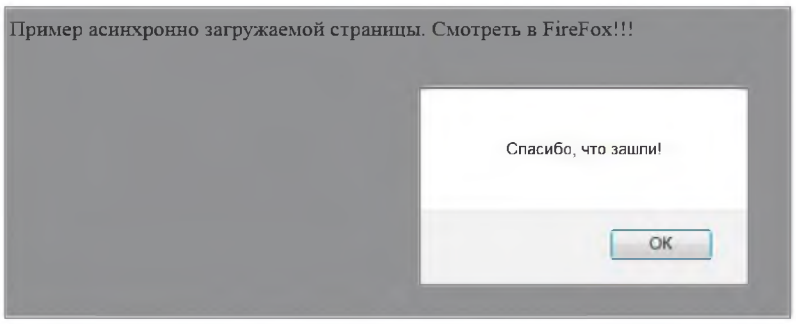
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>async</title>
    <script async src="script2.js"></script>
</head>
  <body>
    <p>Пример асинхронно загружаемой страницы. Смотреть
в FireFox!!!</p>
```

```
  </body>
</html>
```

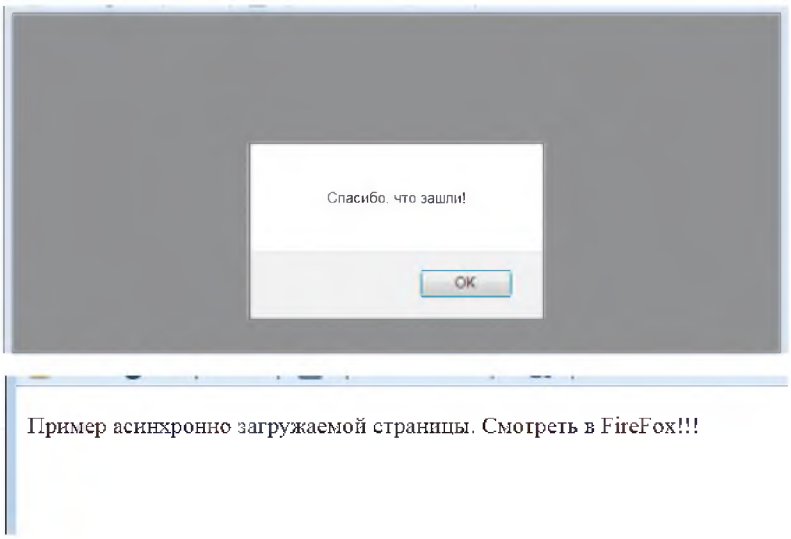
Содержимое script2.js:

```
alert("Спасибо, что зашли!");
```

Результат выполнения с атрибутом async:



Результат выполнения без атрибута async:



Как видно из примера, если скрипт выполняется с атрибутом async, то текст на странице и всплывающее окно видны одновременно.

charset

Задает кодировку символов, используемую в подключаемом файле сценария.

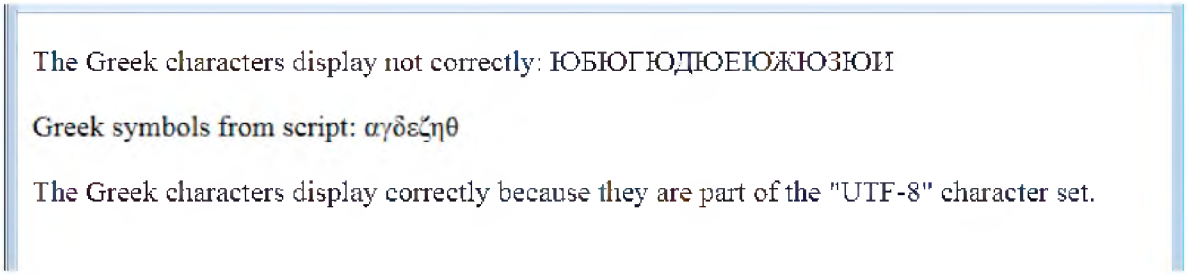
```
<!doctype html>
<html>
  <body>
    <p>The Greek characters display not correctly:
αγδεζηθ</p>
    <script charset="UTF-8" src="demoScriptCharset.js">
    </script>
```

```
    <p>The Greek characters display correctly because they
are part of the "UTF-8" character set.</p>
  </body>
</html>
```

Текст файла demoScriptCharset.js:

```
document.write("Greek symbols from script: αγδεζηθ");
```

Результат выполнения:



Как видно из результата выполнения, в первом абзаце греческие символы отображаются неправильно, потому что в документе не прописан метатег для указания кодировки UTF-8 <meta charset="utf-8">, а греческие символы являются частью набора символов UTF-8.

При работе скрипта греческие символы отображаются правильно, потому что в теге <script> задается кодировка символов UTF-8 через атрибут charset="UTF-8".

defer

При наличии defer скрипт будет выполняться, только когда страница закончит синтаксический анализ. Применяется только для внешних скриптов, т. е. он должен использоваться только в том случае, если присутствует атрибут ***src***.

Существует несколько способов выполнения внешнего скрипта:

- если указан атрибут async, то скрипт будет выполняться без ожидания загрузки и отображения веб-страницы;
- если async отсутствует и присутствует отсрочка defer, то скрипт будет выполняться, когда страница закончит синтаксический анализ;
- если async и defer, то скрипт будет выполняться немедленно, как встретился на странице, до того как браузер продолжит разбор страницы.

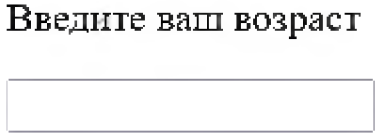
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Тег SCRIPT, атрибут defer</title>
    <script defer src="script1.js"></script>
  </head>
```

```
  <body>
    <form action="handler.php">
    <p>Введите ваш возраст</p>
    <p><input type="text" name="textField"></p>
    </form>
  </body>
</html>
```

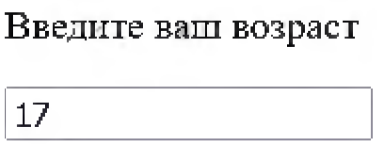
Текст файла script1.js:

```
document.forms[0].textField.value = 17;
```

В данном случае без атрибута defer скрипт должен был выполниться до загрузки тела документа, т. е. когда форма еще не определена, поэтому он не мог выполниться и в поле формы будет пусто.



Defer позволяет отложить выполнение скрипта до тех пор, пока вся страница не будет загружена, поэтому в поле формы будет проставлено 17.



type

Задает тип скрипта между тегами <script> и </script>.

Для JavaScript тип – «application / javascript». Это же значение стоит по умолчанию, поэтому атрибут является необязательным и устаревшим. Встречается только в очень старом коде.

```
<!--Скрипт в head с необязательным атрибутом type.
Выполняется немедленно. -->
<script type="application/javascript">
  document.write("Hello javascript from head!")
</script>
```

<noscript>

Тег <noscript> определяет альтернативный контент для пользователей, которые отключили сценарии в браузере или пользуются браузером, который не поддерживает javaScript.

Тег <noscript> может быть помещен как в секцию <head>, так и в <body>. Если <noscript> размещается в секции <head>, то он может содержать только элементы <link>, <style> и <meta>.

Содержимое внутри тега <noscript> будет отображаться, если JavaScript не поддерживается или его выполнение отключено в браузере пользователя.

```
<!doctype html>
<html>
  <body>
    <script>document.write("Привет мир!");</script>
    <noscript>Извините, ваш браузер не поддерживает
JavaScript!</noscript>
    <p>Этот текст будет виден только в том случае, если
JavaScript не поддерживается или отключен в браузере.</p>
  </body>
</html>
```

<base>

Элемент <base> определен внутри контейнера <head> и указывает базовый URL для всех относительных URL-адресов в документе. В документе может быть только один элемент <base>.

Атрибуты тега <base>

href

Тег <base> предназначен для документов, в которых используется относительный адрес и которые могут переноситься в другую папку или даже на другой компьютер без потери связи. Браузер ищет тег <base>, определяет полный адрес документа и корректно загружает его. Например, если адрес документа указан как <base href="http://www.megasite.by/news/">, то при добавлении рисунков достаточно использовать относительный адрес <img src="images/elephant.gif">. При этом полный путь к изображению будет http://www.megasite.by/news/images/elephant.gif, что позволяет браузеру всегда находить графический файл, независимо от того, где находится текущая веб-страница.

target

В теге <base> можно указать имя окна или фрейма, куда будет загружаться документ, открываемый по ссылке. В зависимости от значения атрибута документ может быть открыт:

- _blank – в новом окне или вкладке;
- _self – в том же окне (значение по умолчанию);
- _parent – в родительском окне;
- _top – в полноэкранном окне;
- framename – в именованном фрейме.

Пример (ссылка откроется в новой вкладке):

```
<!doctype html>
<html>
  <head>
    <base target="_blank">
    <meta charset="utf-8">
  </head>
  <body>
    <a href="Examples for presentation html5 1
a.htm">Новый документ</a>
  </body>
</html>
```

Комментарии в html-документе. <!--…-->

В HTML, как и в большинстве языков программирования, есть возможность писать комментарии в коде. Комментарии игнорируются обозревателем и не видны пользователю, их добавляют для того, чтобы пояснить, как работает написанный код, что делают отдельные его части и т. д. Это удобно и полезно, если вы возвращаетесь к коду, который давно не видели или хотите передать кому-то другому. Комментарии задаются между маркерами <!-- и -->.

Пример использования комментариев:

```
<html><!--Начало документа-->
  <head>
    <meta charset="utf-8">
    <!--Заголовок-->
    <title> Пример использования комментариев </title>
  </head>
  <body><!--Начало тела документа-->
    <!--Внутри тега комментарий поместить нельзя-->
    <!--А собственно текста в документе нет-->
  </body><!--Конец тела документа-->
</html><!--Конец документа-->
```

Тег <body>

Тег <body> представляет контейнер для содержания веб-страницы или контента, отображаемого в окне браузера. Внутри тега <body> располагаются теги, которые определяют контент документа и его структуру.

Все атрибуты тега <body> являются устаревшими, вместо них рекомендуется использовать стили и применять их к тегу <body>. Тег <body> также поддерживает атрибуты событий.

Наиболее часто используемые в теле документа теги:

- <p> – параграф, текстовый абзац;
- <a> – гиперссылка;
- <h1> … <h6> – заголовки разных уровней;
- <img> – рисунок;
-
 – перевод строки;
- – универсальный блочный контейнер;
- – универсальный строчный контейнер.

Параграф <p>

Большинство структурированных текстов состоят из абзацев и заголовков. Тег <p> предназначен для выделения параграфа или абзаца. Применение тега <p> приводит к переводу на новую строку до тега и после него. Браузеры автоматически добавляют некоторое пространство (margin) до и после каждого элемента <p>. Большинство браузеров отобразят элемент <p> со следующими значениями по умолчанию:

```
p {
    display: block;
    margin-top: 1em;
    margin-bottom: 1em;
    margin-left: 0;
    margin-right: 0;
}
```

Тег <p> поддерживает глобальные атрибуты в HTML и атрибуты событий.

Глобальные атрибуты в HTML

Глобальные атрибуты могут применяться ко всем тегам, поэтому они выделены в отдельную группу, чтобы не повторять их при описании каждого тега. Справочник глобальных атрибутов: https://www.w3schools.com/tags/ref_standardattributes.asp.

accesskey

Атрибут accesskey позволяет получить доступ к элементу с помощью сочетания клавиш с заданной в атрибуте буквой или цифрой.

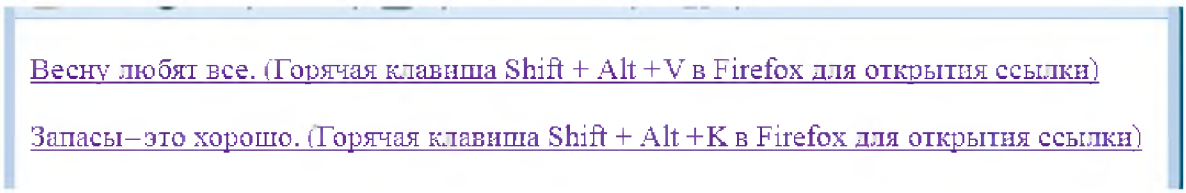
Браузеры при этом используют различные комбинации клавиш.

Браузер	Windows	Linux	Mac
Chrome	Alt + accesskey	Alt + accesskey	Ctrl + Alt + accesskey
Edge	Alt + accesskey	не определено	не определено
Internet Explorer	Alt + accesskey	не определено	не определено
Firefox	Alt + Shift + accesskey	Alt + Shift + accesskey	Ctrl + Alt + accesskey
Safari	Alt + accesskey	не определено	Ctrl + Alt + accesskey
Opera	Opera 15 or newer: [Alt] + accesskey Opera 12.1 or older: [Shift] + [Esc] + accesskey		

Обычно элемент при использовании комбинации клавиш получает фокус, но конкретное действие зависит от применяемого тега. К примеру, для ссылки (тега <a>) произойдет переход по ссылке; для текстовых полей курсор переместится с поле ввода и можно будет вводить текст; для флажков (<input type="checkbox">) будет поставлена галочка и т. д.

Атрибут accesskey может принимать значения цифр от 0 до 9 или латинских букв (a–z). Значения по умолчанию нет. В HTML5 может применяться к любым тегам, но не всегда в этом есть смысл.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Атрибут accesskey</title>
  </head>
  <body>
    <p>
     <a href="images/Vesna.jpg" accesskey="v">Весну любят
все. (Горячая клавиша Shift + Alt + V в Firefox для
открытия ссылки) </a>
    </p>
    <p>
    <a href="images/kats.jpg" accesskey="k">Запасы - это
хорошо. (Горячая клавиша Shift + Alt + K в Firefox для
открытия ссылки)</a>
    </p>
  </body>
</html>
```



Обычно стараются сделать горячую клавишу для элементов уникальной на странице. Если на странице несколько элементов имеют одинаковый ключ доступа, то поведение браузеров будет отличаться:

- в IE и Firefox будет активирован следующий элемент с одинаковой горячей клавишей доступа;
- в Chrome и Safari будет активирован последний элемент с одинаковой горячей клавишей доступа;
- в Opera будет активирован первый элемент с одинаковой горячей клавишей доступа.

id

Атрибут id содержит уникальный (в пределах документа) идентификатор для тега. Его чаще всего используют при указании стиля в таблице стилей CSS и для определения тега в JavaScript через HTML DOM для управления этим элементом.

Правила именования для id HTML5:

- должен содержать хотя бы один символ;
- не должен содержать пробелов;
- нечувствителен к регистру.

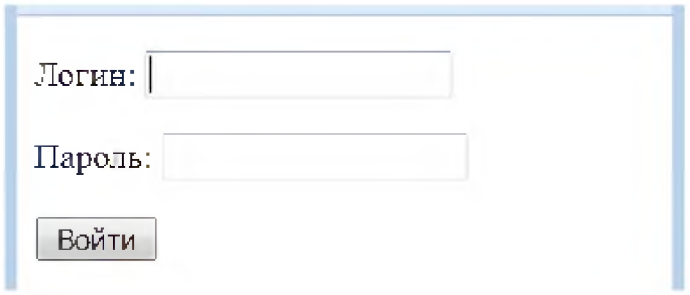
В следующем примере показано использование id для указания стиля и для определения тега в JavaScript через DOM. Абзац имеет уникальный id, по которому присоединяем стиль, позволяющий абзацу выглядеть как ссылка: меняем цвет шрифта на синий, вводим подчеркивание текста и меняем вид курсора над абзацем. Скрипт showForm() будет вызываться при щелчке по абзацу с id="link" и будет скрывать абзац и делать видимой форму с id="auth".

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>hidden</title>
    <!--Использование id для указания стиля-->
    <style>
     #link {
            cursor: pointer;
            color: blue;
```

```
            text-decoration: underline;
          }
    </style>
  <!--Использование id для определения тега в JavaScript через DOM-->
    <script>
     function showForm() {
      document.getElementById("auth").hidden = false;
      document.getElementById("link").hidden = true;   }
    </script>
  </head>
  <body>
    <!--Абзац благодаря присоединенному стилю будет выглядеть как ссылка-->
    <p  id="link"  onclick="showForm()">Авторизация на сайте</p>
    <form id="auth" hidden>
     <p><label>Логин:          <input          name="user" required></label></p>
     <p><label>Пароль: <input name="pass" type="password" required></label></p>
     <p><input type="submit" value="Войти"></p>
    </form>
  </body>
</html>
```

Результат при загрузке страницы:

После щелчка:



class

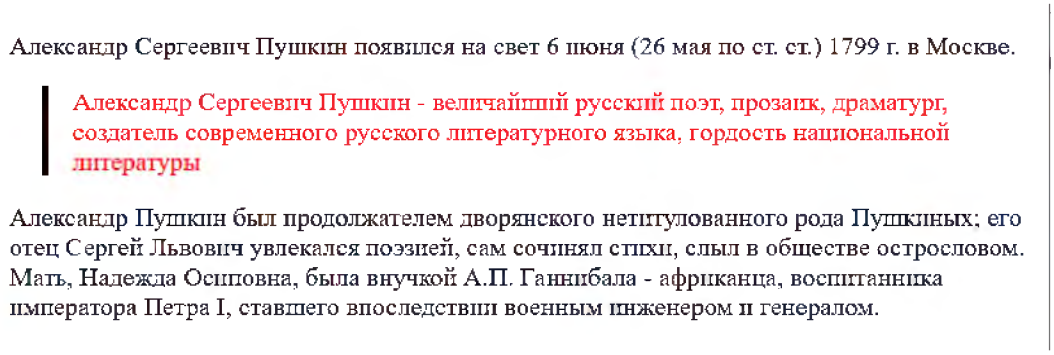
Задает стилевой класс, который позволяет связать определенный тег со стилевым оформлением. Может использоваться также в JavaScript для

определения элементов через DOM и внесения изменений в элементы с указанным классом. В значении допускается указывать сразу несколько классов, разделяя их между собой пробелом. Имена классов могут содержать в себе латинские буквы (A–Z, a–z), цифры (0–9), символ дефиса (-) и подчеркивания (_). Использование русских букв в классах недопустимо. Имена классов нечувствительны к регистру.

Пример демонстрирует присоединение стилевого правила по классу:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Атрибут class</title>
    <style>
      p.attantion { /* Абзац с классом attantion*/
      color: red; /* Красный цвет текста */
      margin-left: 20px; /* Отступ слева */
      border-left: 4px solid black; /* Граница слева */
  padding-left: 15px; /* Расстояние от линии до текста */}
    </style>
  </head>
  <body>
    <p>Александр Сергеевич Пушкин появился на свет 6 июня (26 мая по ст. ст.) 1799 г. в Москве.</p>
    <p class="attantion">Александр Сергеевич Пушкин – величайший русский поэт, прозаик, драматург, создатель современного русского литературного языка, гордость национальной литературы</p>
    <p>Александр Пушкин был продолжателем дворянского нетитулованного рода Пушкиных; его отец Сергей Львович увлекался поэзией, сам сочинял стихи, слыл в обществе острословом. Мать, Надежда Осиповна, была внучкой А.П.Ганнибала – африканца, воспитанника императора Петра I, ставшего впоследствии военным инженером и генералом.</p>
  </body>
</html>
```

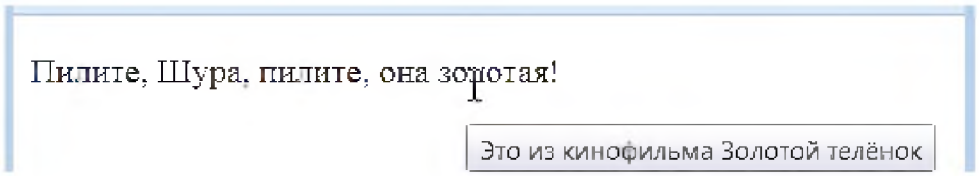
Результат отображения:



title

Указывает дополнительную информацию об элементе в виде всплывающей подсказки при наведении указателя мыши на элемент. Вид подсказки зависит от браузера, настроек операционной системы и не может быть изменен с помощью HTML-кода или стилей. Значением атрибута может быть любая текстовая строка. Строка должна заключаться в двойные или одинарные кавычки.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Атрибут title</title>
  </head>
  <body>
    <p title="Это из кинофильма Золотой теленок">Пилите,
Шура, пилите, она золотая!</p>
  </body>
</html>
```



Описание из атрибута title отображается только при наведении курсора, а значит, люди, работающие только с клавиатурой, не увидят информацию, которую содержит title. Если эта информация действительно важна для удобства использования страницы, то ее можно поместить в обычный текст.

tabindex

Атрибут tabindex устанавливает порядок получения фокуса при переходе между элементами с помощью клавиши Tab. Переход происходит от меньшего значения к большему, например от 1 к 2, затем к 3 и т. д. При этом строгая последовательность не важна, допускается пропускать какие-то числа и начинать с любой цифры. Если значения tabindex у элементов совпадают, тогда учитывается их порядок появления в коде. Заблокированные элементы, у которых установлен атрибут disabled, не участвуют в переходе и фокус не получают.

Переход к элементам, у которых не задан атрибут tabindex или его значение равно 0, происходит после всех «нумерованных» элементов в том порядке, как они указаны в коде.

В качестве значения может использоваться любое целое положительное число. Значение по умолчанию 0. Число должно быть указано в кавычках.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Атрибут tabindex</title>
  </head>
  <body>
   <p>Нажмите кнопку Tab для перехода между элементами</p>
   <p><button>Шестой</button></p>
   <p><button>Седьмой</button></p>
   <p><button tabindex="5">Пятый</button></p>
   <p><button tabindex="1">Первый</button></p>
   <p><button tabindex="3">Третий</button></p>
   <p><button tabindex="2">Второй</button></p>
   <p><button tabindex="4">Четвертый</button></p>
  </body>
</html>
```

hidden

Скрывает содержимое элемента от просмотра. По умолчанию атрибут выключен. Может использоваться, чтобы пользователь не мог видеть элемент до тех пор, пока не будет выполнено какое-либо другое условие, например выбор флажка. Затем через JavaScript можно удалить атрибут hidden и сделать элемент видимым. Пример применения тот же, что и для атрибута id.

lang

Можно установить язык для страницы, добавив атрибут lang в открывающий HTML-тег, например:

```
<html lang="en-US">
```

или

```
<html lang="ru">
```

Если указан язык страницы, то страница будет более эффективно индексироваться поисковыми системами, поскольку она будет правильно отображаться в языковых результатах. Некоторые слова пишутся одинаково на нескольких языках, но произносятся по-разному. Например, слово «шесть» пишется одинаково как на французском, так и на английском языке, но произносится по-разному. Если указан язык

страницы, то пользователи, использующие программы, читающие страницы вслух, всегда услышат слова, произнесенные правильно.

Можно также указать язык для части документа. Браузер использует значение параметра для правильного отображения некоторых национальных символов.

```
<p>Пример на японском: <span lang="jp">ご飯が熱い。
</span>.</p>
```

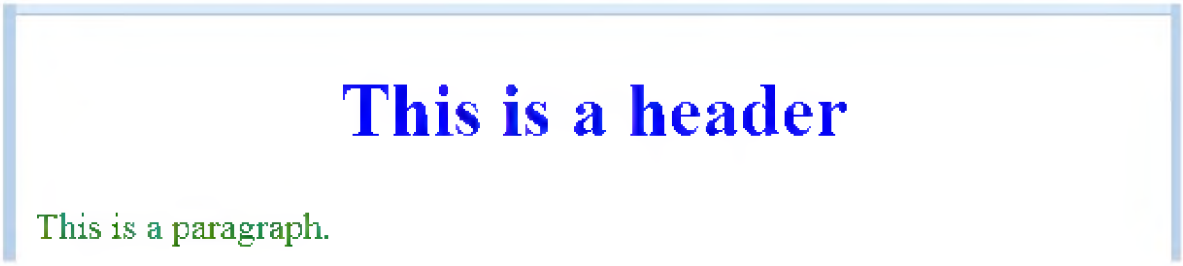
Текст документа может быть набран на одном языке, но содержать вставки на других языках, которые могут различаться по правилам оформления текста. Например, для русского, немецкого и английского языков характерны разные кавычки, в которые берется цитата. Чтобы указать язык, на котором написан текст внутри текущего элемента, и применяется атрибут lang. Браузер использует его значение для правильного отображения некоторых символов. Значения по умолчанию нет.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      q:lang(de) {
      quotes:  "\201E"  "\201C";  /*  Вид  кавычек  для
немецкого языка */}
      q:lang(en) {
      quotes:  "\201C"  "\201D";  /*  Вид  кавычек  для
английского языка */}
   q:lang(fr), q:lang(ru) { /* Вид кавычек для русского
и французского языка */
     quotes: "\00AB" "\00BB";
      }
    </style>
  </head>
  <body>
    <p>Цитата на французском языке: <q lang="fr">Ce que
femme veut, Dieu le veut</q>.</p>
    <p>Цитата  на  немецком:  <q  lang="de">Der  Mensch,
versuche die Gotter nicht</q>.</p>
    <p>Цитата на английском: <q lang="en">To be or not to
be</q>.</p>
    <p>Цитата на белорусском: <q lang="be">Хто без навукі,
той як бязрукі</q>.</p>
  </body>
</html>
```

style

Применяется для определения стиля элемента с помощью правил CSS. В качестве значений указываются стилевые правила: вначале следует имя стилевого свойства, затем через двоеточие его значение. Стилевые свойства разделяются между собой точкой с запятой. Встроенная таблица стилей, определенная через атрибут style, переопределит любой набор стилей в глобальном масштабе, например стили, указанные в теге <style> или во внешней таблице стилей.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h1  style="color:blue;text-align:center">This  is  a
header</h1>
    <p style="color:green">This is a paragraph.</p>
  </body>
</html>
```



contenteditable

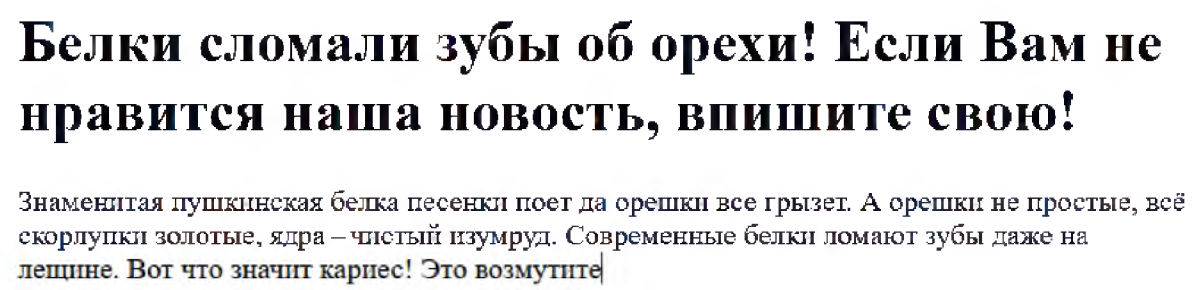
Сообщает, что элемент доступен для редактирования пользователем, т. е. можно удалять текст и вводить новый. Также работают стандартные команды вроде отмены, вставки текста из буфера и др.

Синтаксис: contenteditable="true | false".

Значение true включает режим редактирования, а false запрещает редактирование элемента. Вместо true допустимо указывать пустое значение (contenteditable="") или вообще его не писать (contenteditable). По умолчанию наследует значение родителя. Атрибут добавлен в HTML5.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
```

```
  <body>
    <h1 contenteditable="true">Белки сломали зубы об орехи! Если Вам не нравится наша новость, впишите свою!</h1>
    <p contenteditable="true">Знаменитая пушкинская белка песенки поет
да орешки все грызет. А орешки не простые, всё скорлупки золотые, ядра – чистый изумруд. Современные белки ломают зубы даже на лещине. Вот что значит кариес!</p>
  </body>
</html>
```



dir

Задает направление и отображение текста – слева направо или справа налево. Браузеры обычно самостоятельно различают направление текста, если он задан в кодировке Unicode, но с помощью атрибута dir можно указать, в каком направлении отображать текст. Для арабских и еврейских символов приоритетным является направление, заложенное в Unicode, поэтому на них атрибут dir действовать не будет. Может принимать значение ltr или rtl. Значение по умолчанию – ltr. При нем текст отображается как обычно – слева направо. При значении rtl текст инвертируется и отображается справа налево.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p dir="rtl">Директору ОАО "Рога и копыта" О.И. Бендеру</p>
    <p>Прошу зачислить меня на работу.</p>
  </body>
</html>
```

Директору ОАО "Рога и копыта" О.И. Бендеру

Прошу зачислить меня на работу.

draggable

Указывает, перетаскивается элемент или нет. В HTML5 поддерживается механизм Drag&Drop с небольшим применением JavaScript.

spellcheck

Указывает браузеру, должен ли элемент проверять орфографию и грамматику в тексте. Несмотря на то что атрибут можно устанавливать практически для всех элементов, результат будет заметен только для полей форм <input>, <textarea>, а также редактируемых элементов, если у них установлен атрибут contenteditable.

Браузер может не поддерживать проверку орфографии, или она может быть отключена. Браузер Firefox не выключает проверку орфографии для редактируемых полей, у которых установлен атрибут contenteditable.

Синтаксис:

```
<element spellcheck="true|false">
```

Атрибут добавлен в HTML5.

```
<!doctype html>
<html>
  <body>
    <p   contenteditable="true"   spellcheck="true">Этот
абзац можно редактировать. Попробуй измени текст.</p>
профессия:      <input      type="text"      name="fname"
spellcheck="true">
  </body>
</html>
```

Этот абзац можно редактировать. Попробуй измени текст. Дает карова молоко.

профессия: валитель автобуса

translate

Атрибут указывает, следует переводить содержимое элемента или нет. Синтаксис:

```
<element translate="yes|no">
```

В качестве примера можно набрать на странице следующий код и просмотреть страницу через Google Translate (https://translate.google.ru).

```
<p translate="no">Don't translate this!</p>
<p>This can be translated to any language.</p>
```

Атрибут добавлен в HTML5.

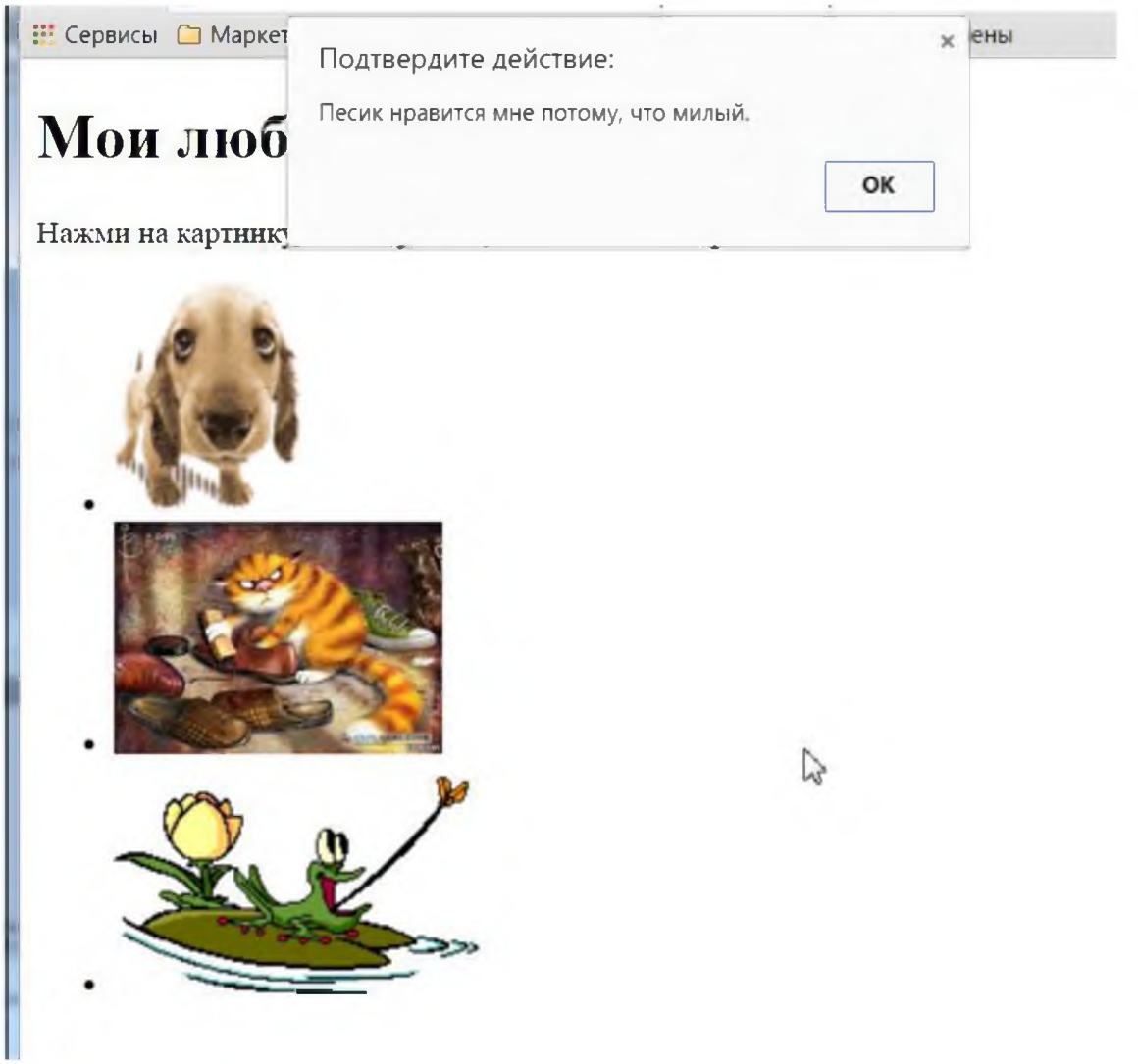
data- *

Используется для хранения конфиденциальных данных на странице или в приложении.

Атрибут data- * состоит из двух частей: имени и значения. Имя атрибута не должно содержать заглавные буквы и может быть длиной по крайней мере в один символ после префикса "data-". Значение атрибута может быть любой строкой. Пользовательские атрибуты с префиксом "data-" будут полностью игнорироваться браузером, поэтому никак в нем не выводятся. Однако атрибут может использоваться в работе скриптов, а также для оформления элементов через CSS. Атрибут добавлен в HTML5.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script>
      function showDetails(image) {
        var animalType = image.getAttribute("data-image-
type");
        alert(image.getAttribute("id") + " нравится мне
потому, что " + animalType + ".");
      }
    </script>
  </head>
  <body>
    <h1>Мои любимые картинки</h1>
    <p>Нажми на картинку, чтобы узнать, за что мне она
нравится</p>
    <ul>
      <li onclick="showDetails(this)" id="Песик" data-
image-type="милый"><img src="images/pes.gif"
height="120"></li>
      <li onclick="showDetails(this)" id="Кот" data-
image-type="обиженный и злой"><img
src="images/zloj_kot.jpg" height="120"></li>
      <li onclick="showDetails(this)" id="Лягушка" data-
image-type="голодная"><img src="images/frog.gif"
height="120"></li>
```

```
    </ul>
  </body>
</html>
```



Атрибуты событий

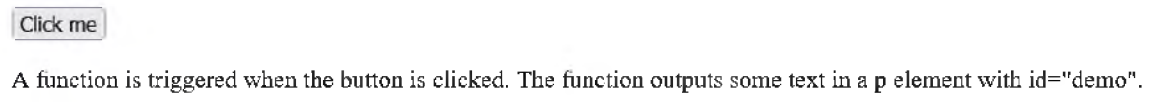
В HTML есть возможность запускать различные действия в браузере при наступлении определенных событий, например запускать JavaScript-код, когда пользователь нажимает на кнопку.

Глобальных атрибутов событий или событийных атрибутов, которые можно добавлять к элементам HTML для определения наступления событий, довольно много. Их можно поделить на несколько групп.

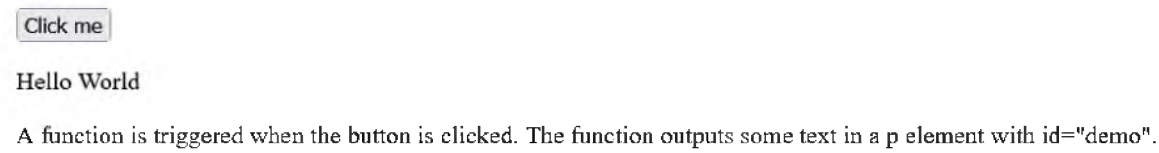
Ниже приведен пример шаблона для обработки события (в данном случае onclick). В значении атрибута стоит вызов функции JavaScript, а может стоять любой фрагмент JavaScript-кода.

```
<!doctype html>
<html>
  <body>
    <button onclick="myFunction()">Click me</button>
    <p id="demo"></p>
    <p>A function is triggered when the button is clicked.
The function outputs some text in a p element with
id="demo".</p>
    <script>
      function myFunction() {
        document.getElementById("demo").innerHTML =
"Hello World";
      }
    </script>
  </body>
</html>
```

Первоначально абзац с атрибутом id, равным demo, не имеет текста.



При нажатии на кнопку в абзац с атрибутом id, равным demo, добавляется текст.



Подробно об атрибутах событий можно почитать в HTML-справочнике: https://www.w3schools.com/tags/ref_eventattributes.asp.

Атрибуты событий окна

Перечислим атрибуты событий окна:

- onafterprint – скрипт запускается после печати документа;
- onbeforeprint – скрипт запускается до того, как документ будет напечатан;
- onbeforeunload – скрипт запускается, когда документ будет выгружен;
- onerror – скрипт запускается при возникновении ошибки;
- onhashchange – скрипт запускается, когда произошли изменения в якорной части URL-адреса;

• onload – скрипт запускается, когда страница полностью загружена, включая содержание, изображения, стилевые файлы и внешние скрипты;

• onmessage – скрипт запускается при запуске сообщения;

• onoffline – скрипт запускается, когда браузер начинает работать в автономном режиме;

• ononline – скрипт запускается, когда браузер начинает работать в Интернете;

• onpagehide – сценарий запускается, когда пользователь уходит со страницы;

• onpageshow – сценарий запускается, когда пользователь переходит на страницу;

• onpopstate – сценарий запускается при изменении истории окна;

• onresize – сценарий запускается при изменении размера окна браузера;

• onstorage – сценарий запускается при обновлении области веб-хранилища;

• onunload – сценарий запускается, когда страница выгружается (или окно браузера закрывается).

Атрибуты событий формы

• onblur – сценарий запускается, когда элемент теряет фокус;

• onchange – сценарий запускается, когда изменяется значение элемента;

• oncontextmenu – сценарий запускается при вызове контекстного меню;

• onfocus – сценарий запускается, когда элемент получает фокус;

• oninput – сценарий запускается, когда пользователь начинает ввод в элемент;

• oninvalid – сценарий запускается, если значение элемента невалидно;

• onreset – сценарий запускается при нажатии кнопки Reset в форме;

• onsearch – сценарий запускается, когда пользователь что-то пишет в поле поиска (для <input = "search">);

• onselect – сценарий запускается после того, как какой-либо текст был выбран в элементе;

• onsubmit – сценарий запускается при отправке формы.

Атрибуты событий клавиатуры

• onkeydown – сценарий запускается, когда пользователь нажимает клавишу;

• onkeypress – сценарий запускается, когда пользователь нажимает клавишу;

• onkeyup – сценарий запускается, когда пользователь отпускает клавишу.

Атрибуты событий мыши

• onclick – сценарий запускается, когда на элементе происходит щелчок мышью;

• ondblclick – сценарий запускается, когда на элементе происходит двойной щелчок мышью;

• onmousedown – сценарий запускается при нажатии кнопки мыши на элементе;

• onmousemove – сценарий запускается, когда указатель мыши движется над элементом;

• onmouseout – сценарий запускается, когда указатель мыши перемещается за границы элемента;

• onmouseover – сценарий запускается, когда указатель мыши перемещается над элементом;

• onmouseup – сценарий запускается, когда кнопка мыши отпускается над элементом;

• onwheel – сценарий запускается, когда колесико мыши прокручивается над элементом.

Атрибуты событий при перетаскивании

• ondrag – сценарий запускается, когда курсор двигается при перетаскивании;

• ondragend – сценарий запускается, когда пользователь отпускает курсор мыши в процессе перетаскивания;

• ondragenter – сценарий запускается, когда перетаскиваемый элемент достигает конечного элемента;

• ondragleave – сценарий запускается, когда курсор мыши покидает пределы перетаскиваемого элемента;

• ondragover – сценарий запускается, когда курсор мыши наведен на элемент при перетаскивании;

• ondrop – сценарий запускается, когда происходит drop (бросание) элемента;

• dragstart – сценарий запускается, когда пользователь начинает перетаскивание элемента.

Атрибуты событий в буфере обмена

- oncopy – сценарий запускается, когда пользователь копирует содержимое элемента;
- oncut – сценарий запускается, когда пользователь вырезает содержимое элемента;
- onpaste – сценарий запускается, когда пользователь вставляет некоторый контент из буфера обмена в элемент.

Атрибуты медиасобытий

- onabort – сценарий запускается при прерывании;
- oncanplay – сценарий запускается, когда файл достаточно буферизирован и готов начать воспроизведение;
- oncanplaythrough – сценарий запускается, когда файл может быть воспроизведен до конца без паузы для буферизации;
- oncuechange – сценарий запускается, когда происходит изменение метки в элементе <track>;
- ondurationchange – сценарий запускается при изменении длины носителя;
- onemptied – сценарий запускается, когда файл становится недоступен;
- onended – сценарий запускается, когда медиа достигает конца (полезное событие для таких сообщений, как «спасибо за прослушивание»);
- onerror – сценарий запускается, когда возникает ошибка при загрузке файла;
- onloadeddata – сценарий запускается при загрузке медиаданных;
- onloadedmetadata – сценарий запускается, когда загружаются метаданные (например, размеры и продолжительность);
- onloadstart – сценарий запускается, когда файл начинает загружаться;
- onpause – сценарий запускается, когда воспроизведение приостанавливается либо пользователем, либо программно;
- onplay – сценарий запускается, когда носитель готов начать воспроизведение;
- onplaying – сценарий запускается при начале воспроизведения медиа;
- onprogress – сценарий запускается, когда браузер находится в процессе получения медиаданных;
- onratechange – сценарий запускается каждый раз при изменении скорости воспроизведения (например, когда пользователь переключается в режим замедленного воспроизведения или быстрого перехода вперед);

• onseeked – сценарий запускается, когда атрибут seek установлен равным false, что указывает на то, что поиск закончился;

• onseeking – сценарий запускается, когда атрибут seek установлен равным true, что указывает на то, что поиск активен;

• onstalled – сценарий запускается, когда браузер не сможет получить данные мультимедиа по любой причине;

• onsuspend – сценарий запускается, когда при извлечении медиаданных происходит остановка до того, как они будут полностью загружены, по любой причине;

• ontimeupdate – сценарий запускается, когда позиция проигрывания меняется (например, когда пользователь быстро переходит в другую точку);

• onvolumechange – сценарий запускается при изменении громкости;

• onwaiting – сценарий запускается, когда носитель приостановлен, но ожидается его возобновление.

Разные атрибуты

• onshow – сценарий запускается, когда элемент <menu> отображается как контекстное меню;

• ontoggle – сценарий запускается, когда пользователь открывает или закрывает элемент <details>.

Упражнения

1. Добавьте к абзацу `<p>Сегодня обещают дождь.</p>` подсказку с текстом «Не забудьте взять зонт».
2. Добавьте возможность перехода по ссылке в Chrome с помощью сочетания клавиш Alt + G:

```
<p><a href="https://www.google.by/">Мой любимый
поисковик</a></p>
```

3. Внесите изменения в код таким образом, чтобы при нажатии клавиши Tab в фокус попадало сначала поле для ввода логина, затем поле для ввода пароля, затем кнопка «Войти» и только после этого – ссылка новости.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>hidden</title>
  <body>
```

```
    <p>Если Вы не хотите авторизоваться, то можете
просмотреть
      <a href="http://www.onliner.by/">новости</a>
    </p>
    <p>Авторизация на сайте</p>
    <form>
      <p><label>Логин: <input name="user"
required></label></p>
      <p><label>Пароль: <input name="pass"
type="password" required></label></p>
      <p><input type="submit" value="Войти"></p>
    </form>
  </body>
</html>
```

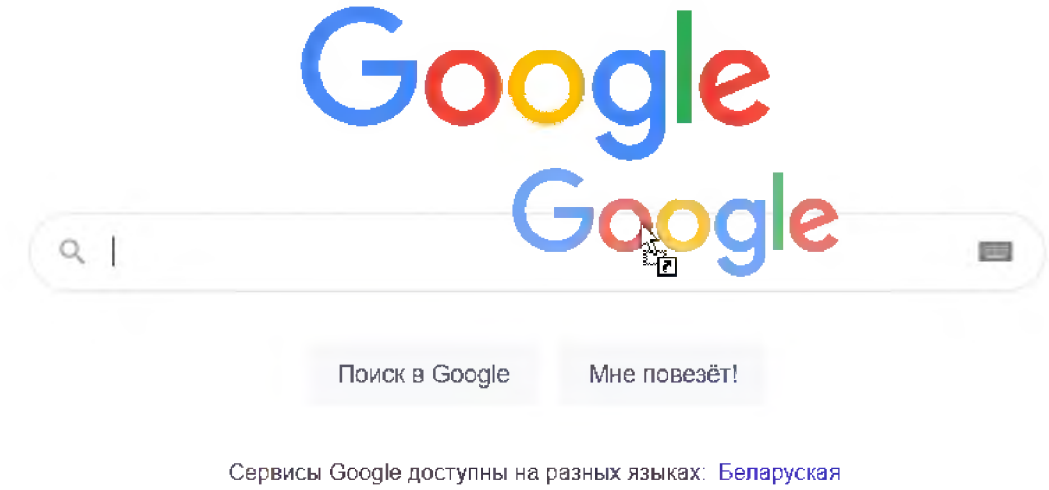
4. Изображение ниже не существует. Укажите альтернативный текст «Первое сентября на ММФ» для изображения

```
<img src="images/sept.jpg" width="135" height="50">
```

5. На страничке перечислите в столбик все цвета радуги. Рядом с названием цвета разместите кнопку с надписью «Показать». По щелчку по кнопке название цвета должно выводиться на фоне такого же цвета. При этом надпись на кнопке должна смениться на «Спрятать». Повторный щелчок по кнопке должен привести к тому, что фон уберется, а надпись на кнопке опять сменится на «Показать». Для нахождения объекта с определенным классом используйте метод getElementsByClassName.
6. Сделайте страничку для написания диктантов с возможностью проверять ошибки и выставлять оценку.

Перетаскивание объектов Drag and Drop

В стандарте HTML5 есть API, который позволяет реализовать эффект Drag&Drop: https://www.w3.org/html/wg/spec/dnd.html. Он дает возможность с помощью специальных событий контролировать захват элемента на странице мышью и его перемещение в новое положение. По умолчанию перемещаться могут только ссылки, изображения и выделенные фрагменты. Если начать перетаскивать их, появится фантомная копия, которая будет следовать за курсором. Ниже приведен пример появления фантомной копии при попытке перемещения картинки в поисковике Google.



Чтобы добавить возможность перетаскивания другим элементам (не ссылкам, изображениям или выделенным фрагментам), нужно задать атрибуту draggable значение true:

```
<p draggable="true" id="dragtarget">Переместии меня!</p>
```

Уже сейчас перетаскивание доступно для элементов, но пока это выражается только в появлении фантомной копии.

Существует система, которая позволяет коду реагировать на события в момент их возникновения. Браузеры дают возможность делать это путем регистрации функций как обработчиков заданных событий.

События Drag&Drop

События Drag&Drop, с помощью которых можно контролировать процесс перетаскивания:

- dragstart – срабатывает в момент начала перетаскивания элемента;
- dragenter – срабатывает, когда перетаскиваемый элемент перемещается в область, куда он может быть перемещен;
- dragover – срабатывает каждые несколько сотен миллисекунд, пока перетаскиваемый элемент находится над зоной, в которую может быть сброшен;
- dragleave – срабатывает в момент перетаскивания, когда курсор мыши покидает пределы перетаскиваемого элемента;
- drag – срабатывает каждые несколько сотен миллисекунд, пока элемент перетаскивается;
- drop – срабатывает в тот момент, когда элемент уже перемещен, если он может быть перемещен в текущую зону. Событие сработает только при успешном завершении операции перетаскивания. Например,

событие не сработает, если пользователь отменит перетаскивание нажатием Esc или не донесет элемент до цели;

- dragend – срабатывает, когда действие перетаскивания завершено, независимо от того, было оно успешным или нет. Это событие не запускается при перетаскивании файла в браузер с рабочего стола.

Добавить обработчики событий можно различными способами.

В HTML:

```
<element ondragstart="myScript">
```

В JavaScript:

```
object.ondragstart = function(){myScript};
```

В JavaScript с помощью метода addEventListener():

```
object.addEventListener("dragstart", myScript);
```

Перетаскивание без использования атрибутов событий

Допустим, на странице есть два прямоугольных блока. В левом из них размещен текст. Необходимо реализовать перетаскивание текста из блока в блок. Будем делать это поэтапно.

Пусть есть код:

```
<!doctype html>
<html>
  <head>
    <style>
    .droptarget {
        float: left;
        width: 120px;
        height: 35px;
        margin: 15px;
        padding: 10px;
        border: 1px solid #aaaaaa;
      }
    </style>
  </head>
<body>
  <p>Демонстрация перетаскивания тега параграфа между
прямоугольниками:</p>
  <div class="droptarget">
    <p     draggable="true"     id="dragtarget">Перемести
меня!</p><!--1.  Добавляем  возможность  перетаскивания
через атрибут draggable-->
  </div>
  <div class="droptarget"></div>
  <p      style="clear:both;"><strong>Замечание:</strong>
события  при  перетягивании  объектов  не  поддерживаются
```

```
Internet Explorer 8 и более ранними версиями, а также
Safari 5.1 и более ранними версиями.</p>
  <p id="demo"></p>
</body>
</html>
```

Пока выделенный элемент не перемещается, отображается только фантомная копия при попытке перемещения.

Первым добавляется прослушиватель события dragstart.

Можно добавить прослушиватель для события dragstart с использованием атрибута ondragstart.

```
<p            draggable="true"            id="dragtarget"
ondragstart="event.dataTransfer.setData('text/plain',
'Перемести меня!')">
Перемести меня!</p>
```

Можно также добавить прослушиватель событий для целевого объекта (eventTarget), используя метод addEventListener(), регистрирующий обработчик события, для которого он будет вызываться при возникновении события.

```
target.addEventListener(тип_события, обработчик);
```

Целевым объектом target может быть объект Element, Document, Window или любой другой объект, поддерживающий события, например такой, как XMLHttpRequest.

Первое, что надо сделать в обработчике события dragstart, – это связать данные с перетаскиванием. Надо передать в обработчик событие, которое происходит в DOM. В общем случае событие может быть вызвано действием пользователя, например щелчком по кнопке мыши или касанием клавиатуры, или сгенерировано API-интерфейсами для представления хода выполнения асинхронной задачи. Его также можно запустить программно.

Свойство dataTransfer события – это то самое место, где реализуется перетаскивание. Оно содержит часть данных, отправляемых при выполнении этого действия. Объект dataTransfer устанавливается в событии dragstart, а считывается и обрабатывается в событии drop. Принимает два параметра: строку, которая объявляет формат второго параметра, и фактические передаваемые данные. При вызове функции event.dataTransfer.setData(format, data) содержанию объекта data назначается MIME-тип format. Свойство event.target содержит элемент, на котором сработало событие. Это не

тот элемент, к которому был привязан обработчик этого события, а именно самый глубокий тег, на который непосредственно был, к примеру, совершен клик. Например, если вы нажмете на p, то event.target будет p.

```
event.dataTransfer.setData("Text", event.target.id);
```

dataTransfer позволяют получать доступ к данным перетаскиваемого элемента и изменять их.

Наиболее важные свойства:

- dataTransfer.effectAllowed = value – возвращает тип доступного действия: none, copy, copyLink, copyMove, link, linkMove, move, all или uninitialized;
- dataTransfer.setData(format, data) – добавляет данные в нужном формате;
- dataTransfer.clearData(format) – убирает данные;
- dataTransfer.setDragImage(element, x, y) – устанавливает изображение для перетаскивания с координатами курсора (0, 0 – левый верхний угол);
- data = dataTransfer.getData(format) – возвращает данные;

Кроме того, изменим форматирование фантомной копии, сделаем копию объекта более прозрачной.

```
event.target.style.opacity = "0.4";
```

А также сопроводим комментарием начало перетаскивания. Комментарий будем выводить в элемент с id "demo".

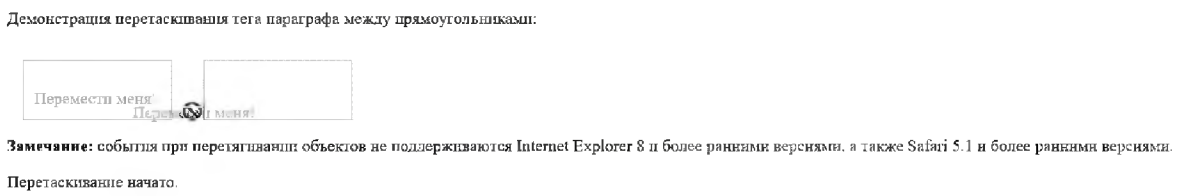
```
document.getElementById("demo").innerHTML =
"Перетаскивание начато";
```

В итоге получим следующий код:

```
<script>
//событие -    dragStart: возникает на перетаскиваемом
элементе во время начала перетаскивания.
 document.addEventListener("dragstart", function(event) {
 //Метод setData добавляет данные в нужном формате
    event.dataTransfer.setData("Text", event.target.id);
 //Выводит некоторый текст при начале перетаскивания
элемента в элемент p с id "demo"
    document.getElementById("demo").innerHTML =
"Перетаскивание начато.";
 // Изменение прозрачности перетаскиваемого элемента
    event.target.style.opacity = "0.4";
});
</script>
```

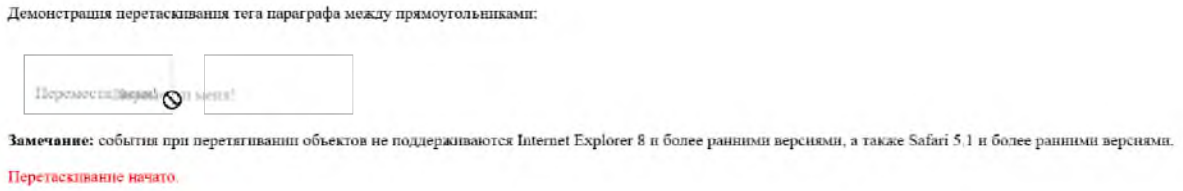
Этот код добавляем в тег <script> перед закрывающим тегом </body>.

Зрительно пока изменилось только то, что при перетаскивании текст становится более серым и в информационный абзац выводится информация о начале перетаскивания.



Событие drag возникает на перетаскиваемом элементе во время операции перетаскивания. Никаких дополнительных действий во время этого события не требуется. Поэтому при перетаскивании элемента p будем с целью привлечения внимания только изменять цвет выводимого текста в элемент p с id "demo".

```
document.addEventListener("drag", function(event) {
    document.getElementById("demo").style.color = "red";
});
```

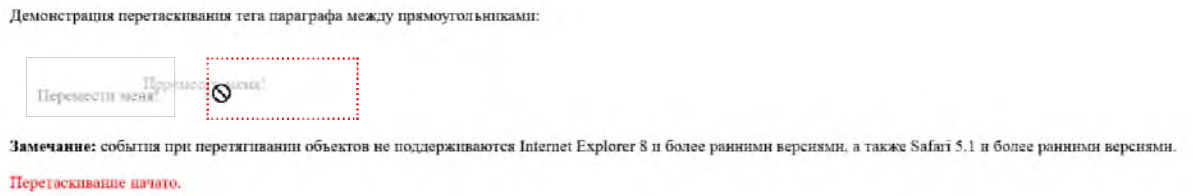


Событие dragleave возникает на элементе, из которого мы перетаскиваем элемент, в тот момент, когда указатель мыши с перетаскиваемым элементом покидает его. Сбрасываем стиль границы покидаемого элемента.

```
document.addEventListener("dragleave", function(event) {
    if ( event.target.className == "droptarget" ) {
        event.target.style.border = "";
}
});
```

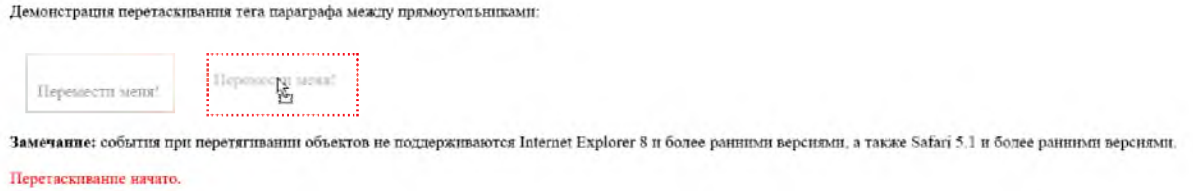
Событие dragenter возникает на элементе, в который мы перетаскиваем, когда указатель мыши с перетаскиваемым элементом оказывается над ним. При появлении перемещаемого элемента над элементом-приемником изменим стиль границы элемента-приемника.

```
document.addEventListener("dragenter", function(event) {
    if ( event.target.className == "droptarget" ) {
        event.target.style.border = "3px dotted red";
    }
});
```



При успешном перемещении элемент должен оказаться на новом месте. Однако по умолчанию большинство областей на странице недоступны для сброса. Чтобы создать область, в которую элементы могут быть сброшены, необходимо слушать событие dragover или drop на нужном элементе и отменять действие по умолчанию с помощью метода preventDefault. Тогда стандартное поведение будет переопределено – перетаскивание и сброс в эту область станут возможными. Для большинства типов событий обработчики событий вызываются до того, как сработает действие по умолчанию.

```
document.addEventListener("dragover", function(event) {
    event.preventDefault();
});
```

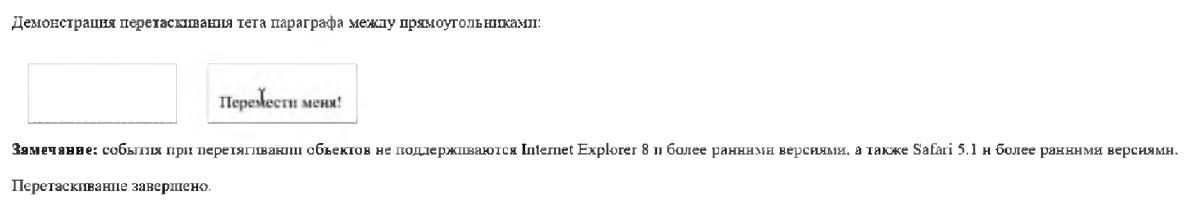


Событие dragEnd происходит, когда пользователь отпускает курсор мыши в процессе перетаскивания. Возникает на перетаскиваемом элементе, когда перетаскивание завершается, причем не важно, успешно или нет. В обработчике этого события выводим соответствующий текст в элемент p с id "demo" и восстанавливаем прозрачность перетаскиваемого элемента.

```
document.addEventListener("dragend", function(event) {
    document.getElementById("demo").innerHTML =
"Перетаскивание завершено.";
    event.target.style.opacity = "1";
});
```

Событие drop возникает на элементе, в который мы перетаскиваем, в тот момент, когда пользователь отпускает на нем указатель мыши с перетаскиваемым элементом, т. е. перетаскивание завершается успешно. В обработчике отменяем действие по умолчанию с помощью метода preventDefault, поскольку по умолчанию большинство областей на странице недоступны для сброса. Затем возвращаем исходный стиль перемещаемому объекту, элементу-приемнику (цвет и стиль рамки прямоугольника). После этого надо получить перетаскиваемые данные с помощью метода dataTransfer.getData() и добавить их в элемент-приемник.

```
document.addEventListener("drop", function(event) {
//preventDefault - предотвращение обработки данных по
умолчанию в браузере
    event.preventDefault();
    if ( event.target.className == "droptarget" ) {
//Сбросить цвет выходного текста
        document.getElementById("demo").style.color = "";
//Сбросить цвет рамки DIV
        event.target.style.border = "";
//Получить перетаскиваемые данные с помощью метода
dataTransfer.getData ()
//Перетаскиваемые данные — это идентификатор
перетаскиваемого элемента ("drag1")
        var data = event.dataTransfer.getData("Text");
//Добавить перетянутый элемент в элемент-приемник
event.target.appendChild(document.getElementById(data));
    }
});
```



Перетаскивание работает.

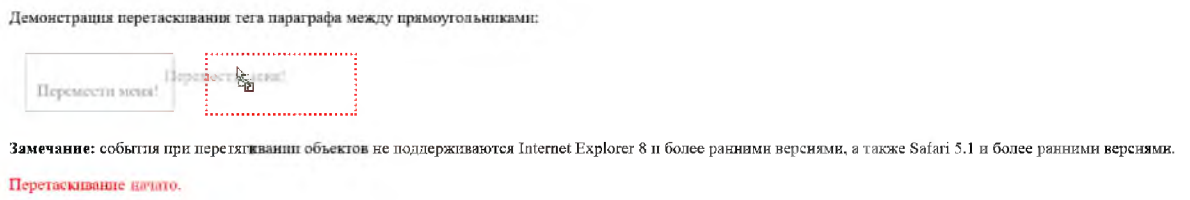
Полный код примера:

```
<!doctype html>
<html>
  <head>
    <style>
      .droptarget {
          float: left;
          width: 120px;
```

```
            height: 35px;
            margin: 15px;
            padding: 10px;
            border: 1px solid #aaaaaa;
        }
    </style>
  </head>
  <body>

    <p>Демонстрация перетаскивания тега параграфа между прямоугольниками:</p>

    <div class="droptarget">
      <p    draggable="true"    id="dragtarget">Перемести меня!</p><!--1.  Добавляем  возможность  перетаскивания через атрибут draggable-->
    </div>

    <div class="droptarget"></div>

    <p    style="clear:both;"><strong>Замечание:</strong> события  при  перетягивании  объектов  не  поддерживаются Internet  Explorer  8  и  более  ранними  версиями,  а  также Safari 5.1 и более ранними версиями.</p>

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

    <script>
//событие -    dragStart: возникает на перетаскиваемом элементе во время начала перетаскивания.
document.addEventListener("dragstart", function(event) {
//Метод setData добавляет данные в нужном формате
    event.dataTransfer.setData("Text", event.target.id);

//Выводит  некоторый  текст  при  начале  перетаскивания элемента в элемент p с id "demo"
    document.getElementById("demo").innerHTML = "Перетаскивание начато.";

    // Изменение прозрачности перетаскиваемого элемента
    event.target.style.opacity = "0.4";
});
```

```
//событие - drag: возникает на перетаскиваемом элементе
во время операции перетаскивания.
//При перетаскивании элемента p изменяется цвет выводимого
текста в элемент p с id "demo"
document.addEventListener("drag", function(event) {
    document.getElementById("demo").style.color = "red";
});

//событие - dragEnd: пользователь отпускает курсор мыши
в процессе перетаскивания. Возникает на перетаскиваемом
элементе когда перетаскивание завершается.
//Выводится некоторый текст, когда закончите
перетаскивание элемента, в элемент p с id "demo"
// и восстанавливается прозрачность перетаскиваемого
элемента
document.addEventListener("dragend", function(event) {
    document.getElementById("demo").innerHTML =
"Перетаскивание завершено.";
    event.target.style.opacity = "1";
});

/*------Цепочка событий при перетаскивании в элементе-
источнике-------*/

//событие - dragEnter: возникает на элементе, в который
мы перетаскиваем, когда указатель мыши с перетаскиваемым
элементом оказывается над ним.
//изменяется стиль границы элемента-приемника
document.addEventListener("dragenter", function(event) {
    if ( event.target.className == "droptarget" ) {
        event.target.style.border = "3px dotted red";
    }
});

//Для большинства типов событий обработчики событий
вызываются до того, как сработает действие по умолчанию.
//Если обработчик не хочет, чтобы это действие
происходило, он может вызвать метод preventDefault объекта
события.
// По умолчанию данные/элементы нельзя перетаскивать
в другие элементы. Чтобы разрешить перемещение, запрещаем
обработку элемента по умолчанию.
//событие - dragOver: курсор мыши наведен на элемент при
перетаскивании. Возникает на элементе, в который мы
```

```
перетаскиваем, когда указатель мыши с перетаскиваемым
элементом находится над ним.
document.addEventListener("dragover", function(event) {
    event.preventDefault();
});

//событие - dragLeave: возникает на элементе, в который
мы перетаскиваем, в тот момент, когда указатель мыши
с перетаскиваемым элементом покидает его,
//изменяем стиль границы элемента-приемника
document.addEventListener("dragleave", function(event) {
    if ( event.target.className == "droptarget" ) {

        event.target.style.border = "";
    }
});

// событие - drop. Возникает на элементе, в который мы
перетаскиваем, в тот момент, когда пользователь отпускает
на нем указатель мыши с перетаскиваемым элементом.
document.addEventListener("drop", function(event) {
     //preventDefault - предотвращение обработки данных по
умолчанию в браузере
    event.preventDefault();
    if ( event.target.className == "droptarget" ) {
         //Сбросить цвет выходного текста
        document.getElementById("demo").style.color = "";
         //Сбросить цвет рамки DIV
        event.target.style.border = "";
         //Получить перетаскиваемые данные с помощью
метода dataTransfer.getData ()
         //Перетаскиваемые данные — это идентификатор
перетаскиваемого элемента ("drag1")
        var data = event.dataTransfer.getData("Text");
         //Добавить перетянутый элемент в элемент-
приемник
event.target.appendChild(document.getElementById(data));
    }
});
</script>

</body>
</html>
```



Перетаскивание с использованием атрибутов событий

Использование глобальных атрибутов событий позволяет не заботиться о добавлении прослушивателей событий, что существенно упрощает код. Ниже приведен пример заказа в кафе. Пользователь имеет возможность перетащить изображения понравившихся блюд из блока «Можно заказать» в блок «Ваш заказ». Также реализована возможность перетаскивания из блока «Ваш заказ» в блок «Можно заказать», которой пользователь может воспользоваться, если он передумал заказывать выбранное ранее блюдо.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Документ без названия</title>
  <style>
      h2 {
    font-size: 3.5em;
    color: #FF0000;
    font-size:60px;
    text-align: center;
    }
    #fig1, #fig2 {
        float: left;
        width: 344px;
        height: 600px;
        margin: 100px;
        padding: 20px;
        border: 2px solid black;
    }
    figcaption{
        font-size:40px;
        text-align: center;
    }
```

```
    img{
        width: 100px;
        height: 100px;
        margin: 10px;
    }
    </style>
<script>
function allowDrop(ev) {//отменяем действие по умолчанию
над объектом, куда бросаем
    ev.preventDefault();
}

function dragStart(ev) {//Заносим данные для переноса
    ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {//отменяем действие по умолчанию над
объектом, куда бросаем, извлекаем из dataTransfer
переносимый объект и добавляем его в нужное место
    ev.preventDefault();
    var data = ev.dataTransfer.getData("text");

ev.target.appendChild(document.getElementById(data));
}
</script>

</head>

<body>
    <h2>Сделайте свой заказ</h2>
        <figure     id="fig1"     ondrop="drop(event)"
ondragover="allowDrop(event)" >
            <figcaption>Можно заказать</figcaption>

            <img src="img/kartoska100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag1" width="100" height="100">
            <img src="img/bigmac100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag2" width="100" height="100">
            <img src="img/kola100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag3" width="100" height="100">
            <img src="img/pizza100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag4" width="100" height="100">
```

```
            <img src="img/sok100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag5" width="100" height="100">
            <img src="img/sushi100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag6" width="100" height="100">
            <img src="img/chicken100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag7" width="100" height="100">
            <img src="img/fruit100x100.jpg"
draggable="true" ondragstart="dragStart(event)"
id="drag8" width="100" height="100">

        </figure>


        <figure id="fig2" ondrop="drop(event)"
ondragover="allowDrop(event)">
            <figcaption>Ваш заказ</figcaption>
        </figure>

</body>
</html>
```

Как видно из примера, для реализации простого перетаскивания объектов, если нет необходимости в добавлении дополнительных эффектов, достаточно написания обработчиков событий dragstart, dragover и drop.

Перемещение копии объекта с заданием собственного значка перемещения вместо фантомной копии

В левой части страницы находится блок «Можно заказать», где размещены изображения блюд, которые можно заказать, возле каждого блюда указана его цена.

Пользователь имеет возможность перетащить изображения понравившихся блюд из блока «Можно заказать» в блок «Ваш заказ» (только изображение, а не текст с ценой!). При этом само блюдо из блока «Можно заказать» не удаляется, что дает возможность заказать несколько одинаковых блюд. Нельзя изменить заказ, т. е. переместить изображение из блока «Ваш заказ» в блок «Можно заказать». Поскольку заранее нельзя предугадать, насколько большим будет заказ, в блоке «Ваш заказ» устанавливается вертикальная полоса прокрутки. При перемещении копии объекта задается собственный значок перемещения вместо фантомной копии.

Перемещение копии объекта и замена фантомной копии другим изображением достигнуты через использование свойств объекта dataTransfer.

Объект dataTransfer (https://www.w3.org/TR/2011/WD-html5-20110113/dnd.html) обладает свойствами, которые создают визуальную подсказку для пользователей в процессе перетаскивания. Их также можно использовать для управления реакцией каждой цели перетаскивания на определенный тип данных.

Так, например, dataTransfer.effectAllowed (https://docs.w3cub.com/-dom/datatransfer/effectallowed/) определяет эффект, разрешенный для операции перетаскивания. Это свойство может принимать следующие значения: none, copy, copyLink, copyMove, link, linkMove, move, all и uninitialized. Операция копирования используется, чтобы указать, что перетаскиваемые данные будут скопированы из текущего местоположения в место перетаскивания. Операция перемещения используется для указания того, что перетаскиваемые данные будут перемещены, а операция связывания – для указания того, что между исходным и местом перетаскивания будет создана некоторая форма связи или соединения. Это свойство должно быть установлено в dragstart, dragenter и dragover, чтобы установить желаемый эффект перетаскивания для источника перетаскивания.

Свойство dataTransfer.dropEffect (https://docs.w3cub.com/dom/data-transfer/dropeffect/) управляет реакцией, которую пользователь получает во время событий dragenter и dragover, поэтому оно также должно быть установлено в dragenter и dragover. Когда перетаскиваемый объект наводится на целевой элемент, указатель браузера принимает вид, соответствующий типу предполагаемой операции (например, копирование, перенос и т. д.). Свойство может принимать следующие значения: none, copy, link, move.

Когда происходит перетаскивание, из цели перетаскивания (элемента, в котором происходит событие dragstart) создается полупрозрачное изображение (фантомное изображение) и следует за указателем мыши во время перетаскивания. Этот фантом создается автоматически, но, если требуется настраиваемое изображение, метод void dataTransfer.setDragImage (img, xOffset, yOffset) можно использовать для установки настраиваемого изображения (https://docs.w3cub.com/dom/datatransfer/setdragimage/). Параметры x и y – это смещения изображения относительно указателя мыши. Чтобы центрировать изображение, можно использовать значения, равные половине ширины и высоты изображения.

```
var dragIcon = document.createElement('img');
dragIcon.src = 'logo.png';
dragIcon.width = 100;
e.dataTransfer.setDragImage(dragIcon, -10, -10);
```

Ниже приведен код примера.

```
<head>
 <meta charset="UTF-8" />
```

```
   <style>
h2 {
font-size: 3.5em;
color: #FF0000;
font-size:60px;
text-align: center;
}
#figure1{
    float: left;
    width: 244px;
    height: 520px;
    margin: 100px;
    padding: 20px;
    border: 2px solid black;
}
#figure2 {
    float: left;
    width: 300px;
    height: 520px;
    margin: 100px;
    padding: 20px;
    border: 2px solid black;
}
figcaption{
     font-size:40px;
     text-align: center;
}
img{
     width: 100px;
    height: 100px;
     margin: 10px;
}
</style>
 <script>
(new Image()).src = 'to_order50x50.jpg';//создание
видимой копии объекта
     /* var dragIcon = document.createElement('img');
     dragIcon.src = 'to_order50x50.jpg';*/
function allowDrop(ev) {//отменяем действие по умолчанию
над объектом, куда бросаем
    ev.preventDefault();
       //var data = ev.dataTransfer.getData("text");
      ev.dataTransfer.effectAllowed =
"copy";//effectAllowed еще не в стандарте, поэтому
добавляем drop-копию вручную
```

```
      ev.dataTransfer.dropEffect = "copy";
}
function dragStart(ev) {//Заносим данные для переноса
     ev.dataTransfer.setData("text", ev.target.id);
     var dragIcon = new Image();
     dragIcon.src = 'to_order50x50.jpg';
     ev.dataTransfer.setDragImage(dragIcon, -10, -10);
     ev.dataTransfer.effectAllowed =
"copy";//effectAllowed еще не в стандарте, поэтому
добавляем drop-копию вручную
}
function drop(ev) {//отменяем действие по умолчанию над
объектом, куда бросаем, извлекаем из dataTransfer
переносимый объект и добавляем его в нужное место
    ev.preventDefault();//
    var data = ev.dataTransfer.getData("text");
     //effectAllowed еще не в стандарте,
     //ev.target.appendChild(document.getElementById(data
));
     //поэтому добавляем drop-копию вручную
     var nodeCopy =
document.getElementById(data).cloneNode(true);
     nodeCopy.id = "newId"; // нельзя использовать то же
самое id
     ev.target.appendChild(nodeCopy);
}
</script>
</head>
<body>
<h2>Сделайте свой заказ</h2>
<figure id="figure1">
     <figcaption>Можно заказать</figcaption>
     <p><img src="kartoska100x100.jpg" draggable="true"
ondragstart="dragStart(event)"   width="100"
height="100" id="img1">5 рублей</p>
     <p><img src="bigmac100x100.jpg" draggable="true"
ondragstart="dragStart(event)" id="drag2" width="100"
height="100">6 рублей</p>
     <p><img src="kola100x100.jpg" draggable="true"
ondragstart="dragStart(event)" id="drag3" width="100"
height="100">5 рублей</p>
</figure>
```

```
<figure id="figure2" ondrop="drop(event)"
ondragover="allowDrop(event)" style="overflow-y: scroll"
>
     <figcaption>Ваш заказ</figcaption>
</figure>
</body>
</html>
```

Упражнения

Создайте HTML-страницы с играми. При реализации используйте перемещение изображений.

1. Загадка Эйнштейна. Полагают, что эту задачу Эйнштейн придумал еще в детстве. Говорят, Эйнштейн считал, что лишь два процента населения земного шара способны оперировать в уме закономерностями, связанными сразу с пятью признаками. Поэтому задача использовалась Эйнштейном для проверки кандидатов в ассистенты на способность к логическому мышлению. Как частное следствие этого, приведенная головоломка может быть решена без использования бумаги лишь теми, кто принадлежит этим двум процентам.

Условие задачи: пять разных человек в пяти разных домах разного цвета выращивают пять разных видов ягод, выращивают пять разных видов животных, пьют пять разных видов напитков. Вопрос: кто выращивает рыбок?

Подсказки:

- норвежец живет в первом доме;
- англичанин живет в красном доме;
- зеленый дом находится левее белого;
- датчанин пьет чай;
- тот, кто выращивает малину, живет рядом с тем, кто выращивает кошек;
- тот, кто живет в желтом доме, выращивает клубнику;
- немец выращивает вишни;
- тот, кто живет в центре, пьет молоко;
- сосед того, кто выращивает малину, пьет воду;
- тот, кто выращивает крыжовник, выращивает птиц;
- швед выращивает собак;
- норвежец живет рядом с синим домом;
- тот, кто выращивает лошадей, живет в синем доме;
- тот, кто выращивает голубику, пьет какао;
- в зеленом доме пьют кофе.

Для поэтапного решения задачи достаточно заполнить следующими данными 25 позиций.

Национальность: норвежец, англичанин, датчанин, немец, швед.

Цвет дома: красный, зеленый, белый, желтый, синий.

Ягоды: малина, клубника, вишня, крыжовник, голубика.

Животные: кошки, птицы, собаки, лошади, рыбки.

Напиток: чай, молоко, вода, какао, кофе.

По сути, используя подсказки, надо заполнить следующую таблицу.

Номер дома	1	2	3	4	5
Национальность					
Цвет дома					
Ягоды					
Животные					
Напиток					

Разместите на странице изображения людей в национальных костюмах, домов различного цвета, ягод, животных и напитков, упомянутых в условии задачи. Номера домов и данные левого столбца таблицы выведите на странице в виде текста. Разместите 25 блоков для имитации таблицы. Пользователь должен иметь возможность перетаскивать изображения людей в национальных костюмах в блоки, расположенные на одной линии с текстом «Национальность», изображения домов разного цвета – в блоки, расположенные на одной линии с текстом «Цвет дома», и т. д. Далее на странице расположите кнопку «Ответить». При нажатии на эту кнопку проверьте решение и выдайте соответствующее сообщение.

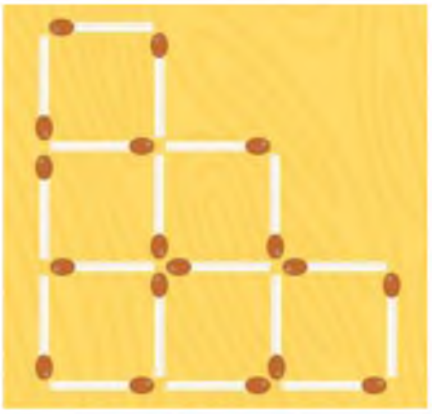
2. При обучении младших школьников используют задачи-головоломки на магические квадраты. По условию дается квадратная таблица, часть ячеек которой заполнены цифрами. Решить магический квадрат означает заполнить пустые ячейки так, чтобы сумма чисел по любой горизонтали, по вертикалям и диагоналям была одинаковой.

Условие задачи: создайте HTML-страницу с задачей-головоломкой «Магический квадрат». Пользователь должен иметь возможность перетащить изображения цифр в магический квадрат.

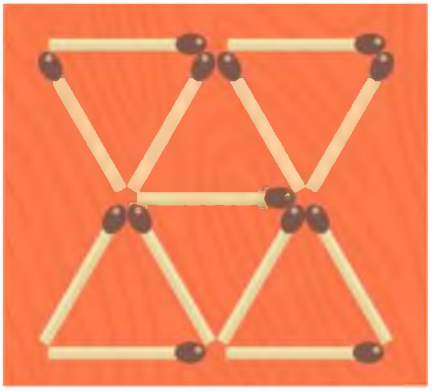
3. Головоломка из спичек. Такие головоломки используются при обучении младших школьников. По условию задачи дается неверное равенство 5 – 5 + 9 = 6, сложенное из спичек. Требуется переложить одну спичку таким образом, чтобы получилось верное равенство. Места, куда можно переместить спички, обозначьте изображением контура спички, например как на рисунке ниже.

4. Головоломка из спичек. Такие головоломки используются при обучении младших школьников. По условию задачи дается неверное равенство 9 – 5 = 8, сложенное из спичек. Требуется переложить одну спичку таким образом, чтобы получилось верное равенство. Места, куда можно переместить спички, обозначьте изображением контура спички, например как на рисунке в задаче 3.

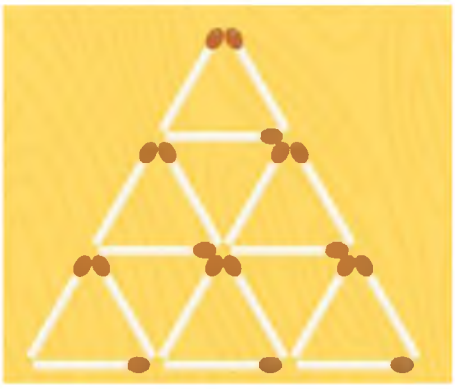
5. Задача со спичками. Нужно убрать две спички так, чтобы осталось четыре равных квадрата. Пользователь должен иметь возможность переместить убираемую спичку, например в корзину. При необходимости пользователь может взять спичку и перетащить ее из корзины на место.



6. Задача со спичками. Требуется убрать три спички так, чтобы осталось только три треугольника. Пользователь должен иметь возможность переместить убираемую спичку, например в корзину. При необходимости пользователь может взять спичку и перетащить ее из корзины на место.



7. Задача со спичками. Уберите пять спичек так, чтобы осталось только пять равных треугольников. Пользователь должен иметь возможность переместить убираемую спичку, например в корзину. При необходимости пользователь может взять спичку и перетащить ее из корзины на место.

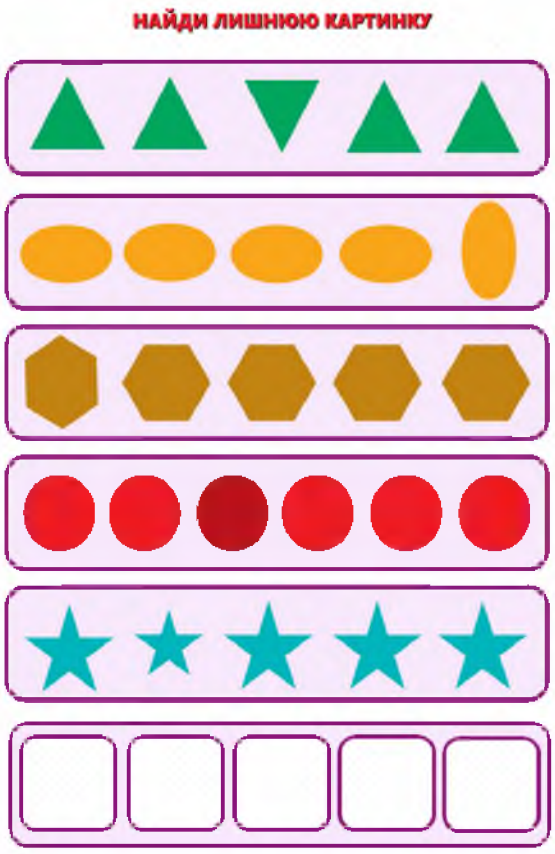


8. Перетяни карточки. Разместите на странице изображения предметов, в которых есть однотипные части, которые можно посчитать. Например, аквариумы с рыбками, клетки с мышками, корзинки с фруктами или, как на картинке ниже, пиццы с определенным числом одинаковых элементов начинки. Суть игры состоит в том, что пользователь должен переместить в блок под изображением предмета карточку с соответствующей цифрой. Должна быть возможность вернуть карточку с цифрой на первоначальное место и перетянуть под изображение предмета карточку с другой цифрой.

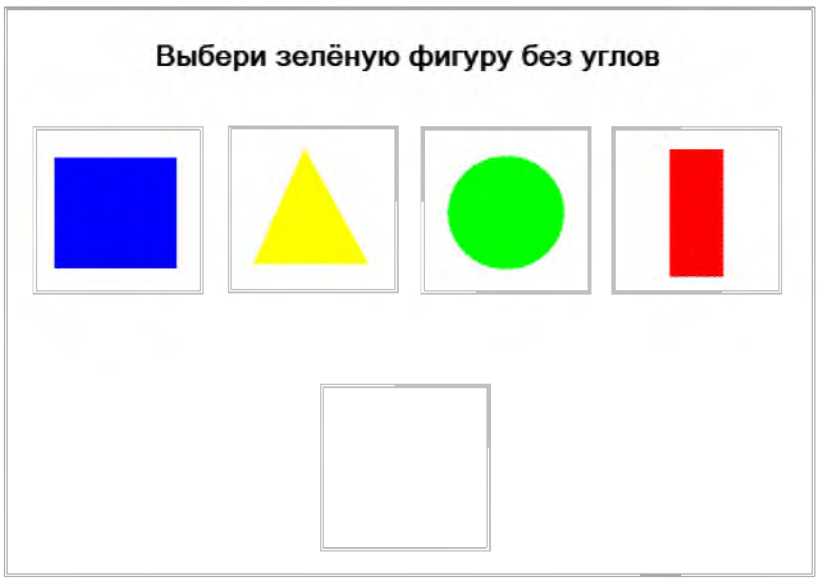
9. Продолжи логическую цепочку. Логические цепочки для детей развивают логическое мышление, внимание и память ребенка. Выстраивая логические цепочки, ребенок учится мыслить самостоятельно, проявлять инициативу. Расположить несколько изображений в ряд по некоторому правилу – непростая задача. Надо разгадать секрет каждого ряда и правильно продолжить последовательность рисунков.

Условие задачи: расположите на странице пять рядов изображений. В каждом ряду разместите пять изображений и оставьте место для шестого. Картинки в каждом ряду должны быть расположены согласно какому-либо правилу. Внизу страницы расположите на выбор изображения, которые можно добавлять. Примерный интерфейс приводится на картинке ниже.

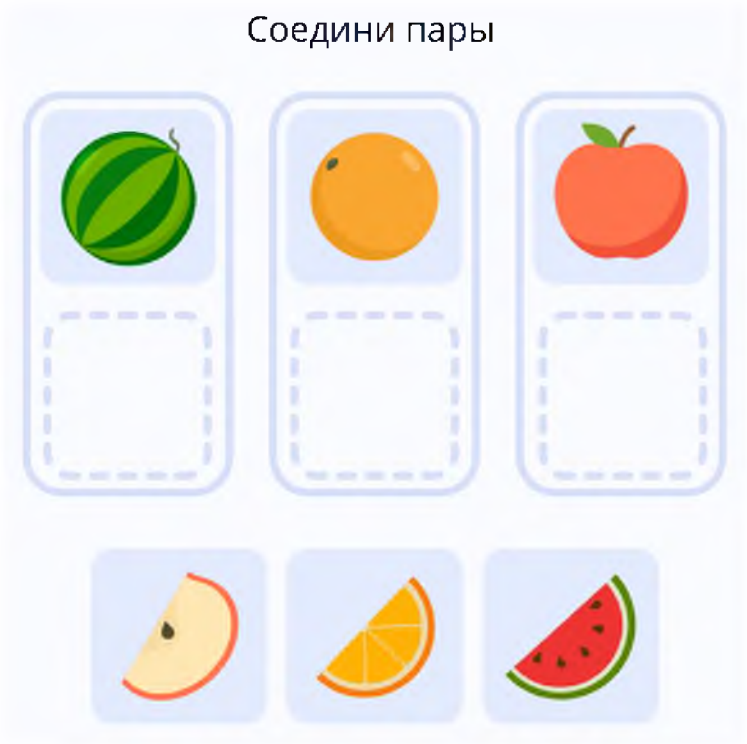
10. Найди лишнюю картинку. Условие задачи: разместите на странице пять рядов изображений. В каждом ряду разместите пять изображений: четыре одинаковых и одно очень похожее на них, но с небольшими отличиями. Пользователь должен иметь возможность убрать из ряда любое изображение. Примерный интерфейс приводится на картинке ниже.



11. Выбор по описанию. Из ряда геометрических фигур разного цвета нужно выбрать и перетянуть в пустой контейнер требуемую геометрическую фигуру соответствующего цвета. Пользователь должен иметь возможность перетянуть из ряда любое изображение и если он передумает, то вернуть картинку на место, выбрать и перетянуть другое изображение. Примерный интерфейс приводится на картинке ниже.



12. Соедини пары. Расположите на HTML-странице несколько блоков. В каждый блок поместите, например, фрукт и оставьте место для изображения части фрукта. Ниже в блоке поместите изображения частей фруктов. Цель задачи – поместить в каждый блок изображения части уже имеющегося там фрукта. Таким образом можно соединять и другие пары, например животных и их детенышей. Пользователь должен иметь возможность перетянуть из нижнего ряда любое изображение и поставить его в пару. Если пользователь изменит свое мнение, то он должен иметь возможность вернуть картинку на место или поставить ее в другую, пока не сформированную пару, а на освободившееся место выбрать и перетянуть другое изображение. Примерный интерфейс приводится на картинке ниже.



Заголовки <h1> – <h6>

В HTML-документах существует возможность выделения заголовков шести уровней. Все они отличаются величиной отступа и размером шрифта. Для их обозначения используются теги <h1>, <h2>, <h3>, <h4>, <h5>, <h6>.

Поисковые системы используют заголовки для индексации структуры и содержания веб-страниц. Кроме того, чтобы убедиться, что именно данная страница нужна пользователю, он не читает ее содержимое, а бегло просматривает заголовки на странице. Важно использовать заголовки, чтобы показать структуру и суть документа.

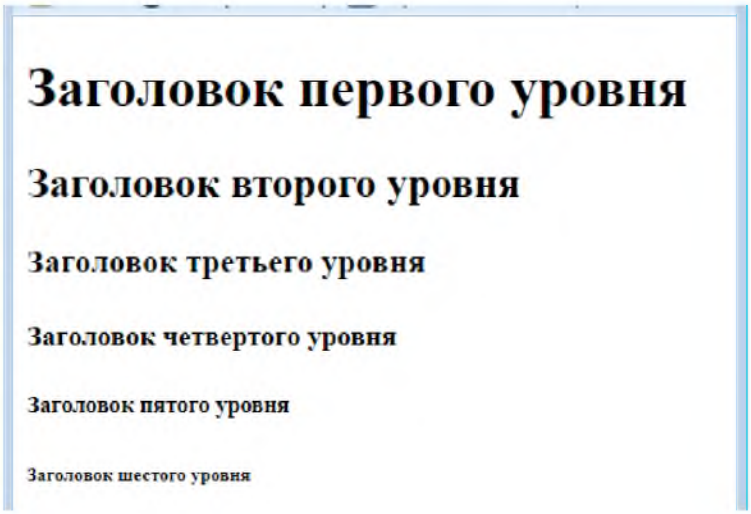
Поисковые системы рекомендуют использовать тег <h1> один раз на странице для указания самого важного. Для менее важного используются заголовки <h2>. Их может быть несколько. Теги <h3> уже не столь сильно учитываются поисковыми системами и используются скорее в декоративных целях. <h4>, <h5>, <h6> для поисковой системы несут еще меньшую нагрузку и не влияют на выдачу.

Из шести доступных уровней заголовка лучше использовать не более трех на странице. Документы с большим количеством уровней, т. е. с глубокой иерархией, становятся громоздкими и трудными для навигации. В таких случаях рекомендуется распределять контент по нескольким страницам, если это возможно.

Поскольку поисковые системы используют заголовки для индексации структуры и содержания веб-страниц, то не следует использовать заголовки только с оформительскими целями, чтобы сделать шрифт текста большим или жирным.

Пример использования тегов заголовков:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Заголовки</title>
</head>
<body>
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
<h4>Заголовок четвертого уровня</h4>
<h5>Заголовок пятого уровня</h5>
<h6>Заголовок шестого уровня</h6>
</body>
</html>
```



Упражнения

1. Замените теги параграфов, где это нужно, на теги заголовков таким образом, чтобы текст стал наиболее понятен как для пользователей, так и для поисковых машин.

```
<!doctype html>
<html>
<body>
<p>ТЕСТЫ И КОНКУРСЫ</p>
<p>Конкурс KP.BY: пришли рецепт с фото своего любимого блюда – и выиграй микроволновку или пылесос!</p>
<p>Хотите порадовать себя или своих близких новенькой бытовой техникой? Участвуйте в нашем конкурсе и выигрывайте приятные призы!</p>
<p>Конкурс на kp.by: «Тайная жизнь моего питомца»</p>
<p>Присылайте снимки ваших любимцев – котиков, собачек, кроликов, хомячков, морских свинок… Да хоть игуан! И выигрывайте приз: упаковку корма, билеты в кино и игрушечных героев «Тайной жизни домашних животных»</p>
</body>
</html>
```

2. Откройте одну из страниц сайта, который, с вашей точки зрения, наиболее успешен и интересен. Проанализируйте эту страницу с точки зрения использования тегов заголовков.
3. Проведите аудит сайта механико-математического факультета с точки зрения правильности использования тегов заголовков.

Ссылки <a>

Гиперссылка – это тот элемент, который, собственно, и обеспечил популярность интернету. Для создания ссылок, или установления связи между одной веб-страницей и другим ресурсом, в HTML предназначен тег <a>. Когда курсор пользователя находится над ссылкой, он приобретает вид маленькой руки с поднятым указательным пальцем. Для перехода по ссылке достаточно сделать щелчок по ней. Синтаксис ссылки следующий:

```
<a href="url"> link text </a>
```

Ссылка задает адрес документа, на который следует перейти. В зависимости от значения параметра href тег <a> устанавливает переход на другую страницу или якорь на этой же странице.

В качестве ссылки может использоваться адрес документа любого типа, поэтому результат перехода по ссылке зависит от типа целевого файла. Если веб-браузер не знает, как отображать или обрабатывать файл,

он спрашивает пользователя, хочет ли тот открыть файл (в этом случае обязанность открытия или обработки файла передается в соответствующее локальное приложение на устройстве) или сохранить его. Архивы (файлы с расширением .zip или .rar) будут сохраняться на локальный диск.

По умолчанию ссылки во всех браузерах отображаются следующим образом:

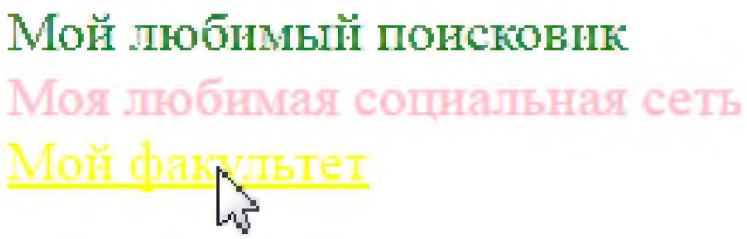
- непосещенная ссылка подчеркнута и выделена синим цветом;
- посещенная ссылка подчеркнута и выделена фиолетовым цветом;
- активная ссылка подчеркнута и выделена красным цветом.

Используя таблицы стилей CSS, можно переопределить стили:

- ***:link*** – для непосещенных ссылок;
- ***:visited*** – для ссылок, которые уже посещались;
- ***:hover*** – для ссылок в момент, когда они находятся в фокусе, но не активизированы, например когда указатель мыши находится над ссылкой, но щелчка мыши не произошло;
- ***:active*** – для ссылок в момент активизации, а именно между моментами, когда пользователь нажимает кнопку мыши и отпускает ее.

Например:

```
<style>
a:link {
    color:green;
    background-color:transparent;
    text-decoration:none
}
a:visited {
    color:pink;


}
a:hover {
    color:red;
    background-color:transparent;
    text-decoration:underline
}
a:active {
    color:yellow;
    background-color:transparent;
    text-decoration:underline
}
</style>
```



background-color:transparent – это прозрачный цвет фона. Поскольку это значение по умолчанию, данное свойство можно не указывать.

В большинстве браузеров тег <a> будет отображаться по умолчанию со следующими стилями:

```
a:link, a:visited {
    color: /* значение браузера по умолчанию */;
    text-decoration: underline;
    cursor: auto;
}
a:link:active, a:visited:active {
    color: /* значение браузера по умолчанию */;
}
```

Тег ***<a>*** поддерживает все глобальные атрибуты и атрибуты событий.

Атрибут href

Адрес документа, на который следует перейти, задается атрибутом href.

Возможные значения href:

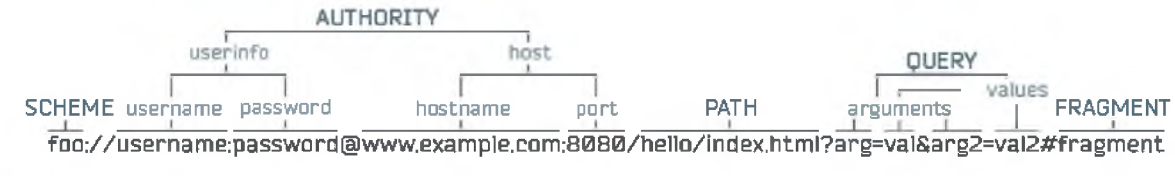
- абсолютный адрес, например другого сайта: href=http://www.example.com/default.htm;
- сокращенная ссылка. Относительный адрес, например внутренняя страница этого же сайта: href="default.htm";
- закладка на странице: href="#top";
- другой протокол: https://, ftp://, mailto:, file: и т. д.;
- скрипт: href="javascript:alert('Hello');";
- ссылка на электронную почту: href=mailto:your@email.

Адреса могут быть представлены различным образом. Ссылки могут быть полными и сокращенными. Полные ссылки содержат полный, реальный адрес сетевого ресурса, на который ссылаются, или URL. Чаще всего используются при необходимости получения доступа к информации, расположенной на другом сайте. Полные ссылки работают независимо от имени сайта или веб-страницы, где прописана ссылка.

Полные ссылки. URL

URL-адрес – синоним веб-адреса. Он может состоять из слов (w3schools.com) или IP-адреса (192.68.20.50). Большинство людей пользуются словами, потому что имена легче запомнить, чем числа.

Графически схему URL можно представить в следующем виде.



Рассмотрим каждый ее элемент более подробно.

Scheme – это название протокола передачи данных: http:// (незашифрованный протокол передачи данных), htpps:// (безопасный, зашифрованный протокол передачи данных), ftp:// (протокол передачи файлов) или file:// (файл на вашем компьютере). Далеко не всегда все параметры используются. Например, для доступа к какой-либо веб-странице по протоколу http://, как правило, схема не указывается.

Userinfo – имя пользователя и пароль, используемые для доступа к ресурсу. Как правило, авторизуются уже на сайте и имя пользователя и пароль также не указываются.

@ – разделитель между хостом и парой «логин – пароль». В случае если логин-пароль не указывается, то разделитель можно точно так же не указывать.

Hostname – это имя сервера.

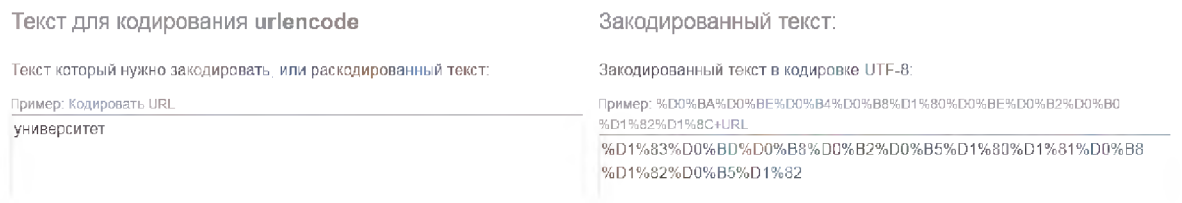
Port – это порт. Если он не указан, то используется порт по умолчанию (80 или 8080).

Path – это сетевой путь к HTML-документу на сервере.

Query, или запрос, следует за знаком вопроса и состоит из пар «имя параметра и его значение». Пары разделяются значком амперсанда (&).

Fragment может стоять за значком шарпа (#) и содержит якорь или ссылку в пределах этого же документа.

Стандарт URL документирован в RFC1738: http://www.ietf.org/rfc/rfc1738. Он позволяет использовать в ссылках только ограниченный набор символов: латинские буквы, цифры и лишь некоторые знаки препинания. Если нужно использовать в URL символы кириллицы, или иероглифы, или, скажем, специфические символы французского языка, то не входящие в стандарт символы должны быть перекодированы особым образом. В интернете достаточно много сервисов для кодировки и раскодировки URL-строк. Например, http://www.code-net.ru/services/urlencode-urldecode/.



Сокращенные ссылки

Сокращенные ссылки – это ссылки, реальный адрес которых формируется на основе адреса текущего документа или корня сайта. Чаще всего используются для доступа к информации в пределах сайта. В сокращенных ссылках название протокола, имя сервера и номер порта опускаются.

Сокращенные ссылки бывают двух типов: абсолютные и относительные.

```
<!doctype html>
<html>
  <body>
    <p>An absolute URL:
      <a href="http://www.w3schools.com">W3Schools</a>
    </p>
    <p>A relative URL:
     <a href="tag_a.asp">The a tag</a>
    </p>
  </body>
</html>
```

An absolute URL: W3Schools

A relative URL: The a tag

Абсолютные ссылки

Абсолютные ссылки задают путь к необходимому файлу относительно корневой папки сайта. Они начинаются с обязательного символа слэш (/), который указывает веб-серверу, что путь будет отсчитываться относительно корневой папки сайта. Например, адрес `/article5.htm` указывает на файл artile5.htm, хранящийся в корневой папке сайта, а адрес `/pages2/article21.htm` – на файл article21.htm, хранящийся в папке pages2, вложенной в корневую папку сайта.

Абсолютный адрес всегда будет указывать на одно и то же местоположение, независимо от того, где он используется.

Относительные ссылки

Относительные ссылки задают путь к необходимому файлу относительно текущей страницы сайта. Они не содержат в начале символа слэша («/»). Например, адрес `article7.htm` указывает на файл article7.htm, хранящийся в той же папке сайта, в которой хранится и файл текущего документа. Адрес `pages2/article21.htm` указывает на файл article21.htm, хранящийся в папке pages2, вложенной в папку, в которой хранится текущий документ, а адрес `../article4.htm` – на файл article4.htm, хранящийся в папке, в которую вложена папка, в которой хранится текущий документ. Две точки в начале адреса обозначают папку предыдущего уровня вложенности. Адрес `//google.com` указывает на домен в текущем протоколе: если мы находимся по адресу, который начинается с http, то ссылка будет вести на http://google.com.

Относительный адрес будет указывать на различные места в зависимости от того, где находится файл, в котором он используется

Если в коде HTML-документа в секции <head> присутствует тег <base>, который определяет URL-базу для всех относительных URL на странице, то полный адрес вызываемого по ссылке сетевого ресурса будет строиться на основе сокращенного адреса, исчисляемого от значения этого атрибута.

Рекомендуется использовать относительные пути к файлам. В этом случае веб-страницы не будут привязаны к текущему базовому адресу. Все ссылки будут работать корректно как на локальном компьютере (localhost), так и при размещении на сервере в интернете, даже если домен в будущем будет изменен.

Кроме этого, использование относительных URL-адресов более эффективно. Когда используется абсолютный URL-адрес, браузер начинает поиск реального местоположения сервера и запрашивает адрес у DNS, затем он переходит на этот сервер и находит файл, который запрашивается. При использовании относительного URL-адреса браузер сразу ищет файл, который запрашивается, на том же сервере.

Якоря или закладки

Иногда возникает необходимость перехода не к началу документа, а к его конкретному фрагменту. Причем это может быть переход в пределах одного достаточно объемного файла или к фрагменту в другом ресурсе. Для реализации такого перехода используются якоря.

Главное отличие ссылки от якоря в том, что с помощью ссылки осуществляется переход по заданному адресу и помечается то место, куда может быть осуществлен такой переход.

Для перехода к якорю необходимо:

1) создать закладку с атрибутом id:

```
<h2 id="C2">Chapter 2</h2>
```

2) добавить ссылку на закладку в рамках той же страницы. Для этого в атрибуте href ссылки нужно первым символом поставить #, а затем указать имя якоря:

```
<a href="#C2">Chapter 2</a>
```

Например:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p>table of contents</p>
    <p><a href="#C1">Chapter 1</a></p>
    <p><a href="#C2">Chapter 2</a></p>
    <p><a href="#C3">Chapter 3</a></p>
    <p><a href="#C4">Chapter 4</a></p>
    <h2 id="C1">Chapter 1</h2>
    <p>This chapter explains bla bla bla</p>
    <h2 id="C2">Chapter 2</h2>
    <p>This chapter explains bla bla bla</p>
    <h2 id="C3">Chapter 3</h2>
    <p>This chapter explains bla bla bla</p>
    <h2 id="C4">Chapter 4</h2>
    <p>This chapter explains bla bla bla</p>
  </body>
</html>
```

Ссылки на электронную почту

Можно создавать ссылки или кнопки, которые при нажатии открывают новое исходящее сообщение электронной почты, а не ссылку на ресурс или страницу. Для этого используется элемент <a> и mailto: – адрес почты. Ссылка на адрес электронной почты будет работать, только если установлен почтовый клиент.

Для создания гиперссылки вызова почтовой программы с целью написания письма используется `<a href="mailto:your@email">`, т. е. в атрибуте href сначала указывается ключевое слово mailto, затем выводится двоеточие и указывается адрес e-mail. Например:

```
<a href="mailto:IvanovIvan@bsu.by"> Отослать сообщение
</a>
```

Если необходим ввод пароля, то пароль помещается между именем пользователя и @.

```
<a   href="mailto:IvanovIvan:password@bsu.by">   Отослать
сообщение </a>
```

Адрес электронной почты не является обязательным для заполнения. Если в поле href оставить только mailto:, то почтовой программой откроется новое исходящее сообщение, в поле получателя будет пусто. Это можно использовать для кнопки «Поделиться».

```
<a href="mailto:"> Поделиться </a>
```

Самыми простыми и часто используемыми элементами mailto являются subject (тема письма), cc (кому отправить копию сообщения, все получатели письма увидят список получателей), bcc (скрытый адрес получателя, никто из получателей не будет видеть полный список получателей письма) и body (текст сообщения).

Значение каждого поля должно быть написано в URL-кодировке (https://ru.wikipedia.org/wiki/URL). В ней все зарезервированные символы ASCII заменяются на символ % и следующие две шестнадцатеричные цифры соответствующего значения в наборе символов ISO-8859-1. Так, например, пробелы между словами должны быть заменены на %20, чтобы гарантировать, что браузер будет отображать текст правильно.

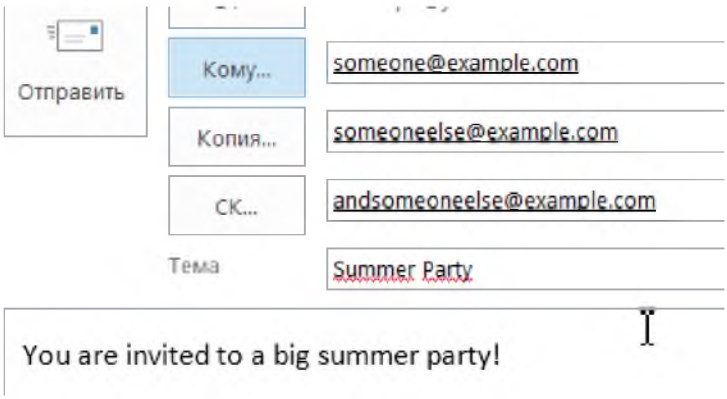
Для вывода темы сообщения:

```
<a
href="mailto:someone@example.com&subject=Hello%20again">
Пишите мне </a>
```

Ссылка сразу на несколько электронных адресов с темой и встроенным сообщением:

```
<a
href="mailto:someone@example.com?cc=someoneelse@example.
com&bcc=andsomeoneelse@example.com&subject=Summer%20Part
y&body=You%20are%20invited%20to%20a%20big%20summer%20par
ty!">Send mail!</a>
```

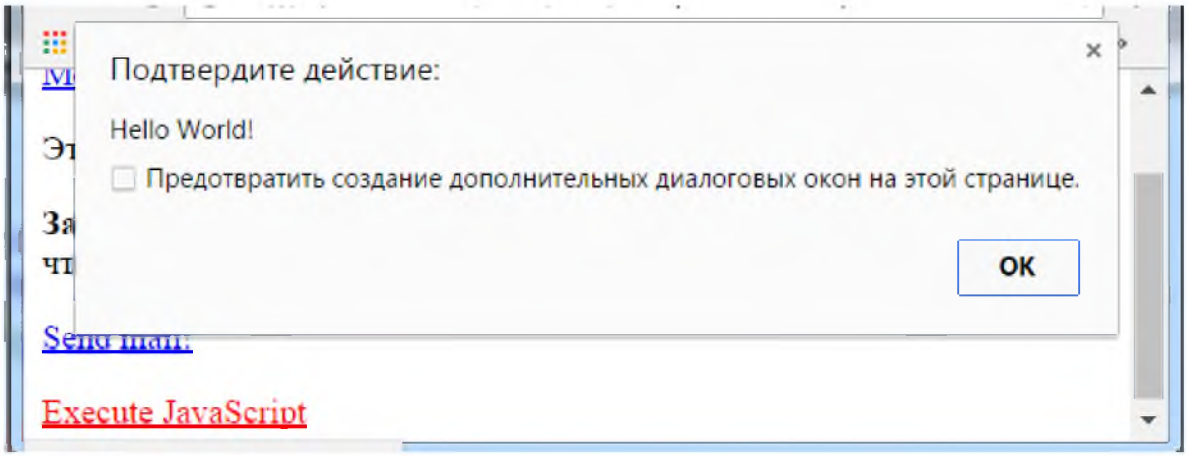
Причем адресат someoneelse@example видит всех получателей, andsomeoneelse@example.com не видит других получателей, кроме себя. Обратите внимание, что знак вопроса (?) используется для разделения основного адреса и дополнительных полей, амперсанд (&) – для разделения каждого поля mailto: URL.



Ссылки на JavaScript

В качестве адреса в ссылке можно разместить JavaScript.

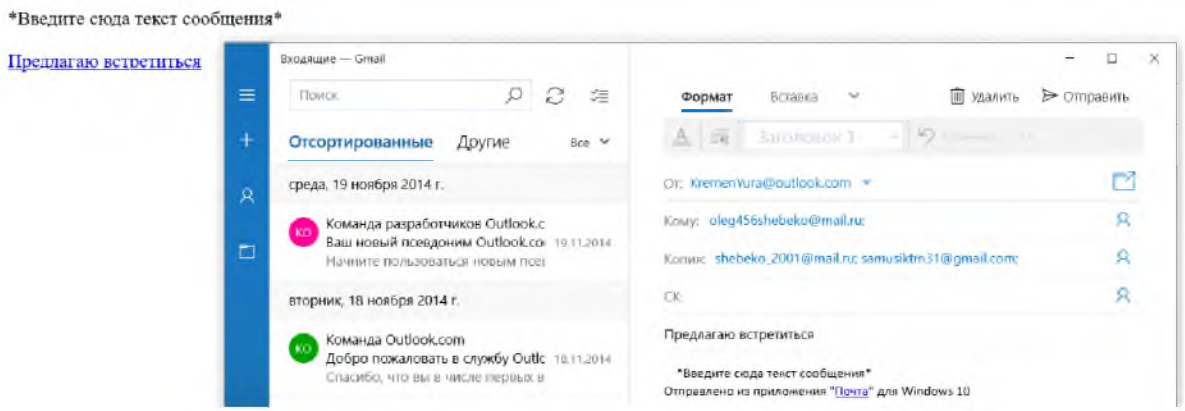
```
<a    href="javascript:alert('Hello    World!');">Execute
JavaScript</a>
```



Если на веб-странице есть редактируемый абзац и на этой же странице размещена ссылка, то можно у нее при помощи JavaScript изменить атрибут href. Поскольку событие onclick для ссылки происходит непосредственно перед тем, как произойдет переход по URL, указанному в атрибуте href ссылки, то вполне можно прописать в href любой адрес, а в событии onclick подменить этот адрес на нужный. Щелкнув по ссылке, формируем href как ссылку на электронную почту. Тема сообщения: «Предлагаю встретиться». В качестве получателей указаны несколько адресов. Встроенное сообщение берется из редактируемого абзаца.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Учеба на факультете</title>
    <script>
      function send(event) {
```

```
event.target.href='mailto:oleg456shebeko@mail.ru?cc=sheb
eko_2001@mail.ru,samusiktm31@gmail.com&subject=Предлагаю
встретиться&body='+document.getElementById('message').te
xtContent;
//document.getElementById("a1").href='mailto:oleg456sheb
eko@mail.ru?cc=shebeko_2001@mail.ru,samusiktm31@gmail.co
m&subject=Предлагаю
встретиться&body='+document.getElementById('message').te
xtContent;
      }
    </script>
  </head>
  <body>
    <p contenteditable="true" id="message">
    *Введите сюда текст сообщения*
    </p>
    <a  href=""  id="a1"  onClick="send(event)">Предлагаю
встретиться</a>
  </body>
</html>
```

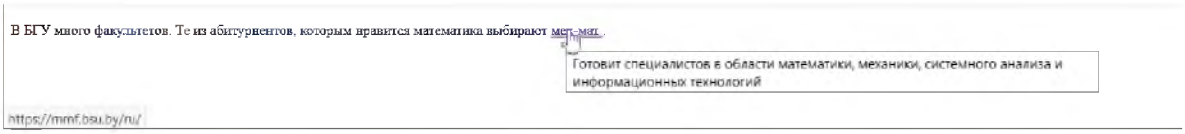


Атрибут title

Использование всплывающих подсказок для ссылок стало хорошим тоном. Подсказки отображают различные полезные сведения о ссылке, например о том, какую информацию содержит страница.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Документ без названия</title>
  </head>
  <body>
```

```
    <p>В БГУ много факультетов. Те из абитуриентов,
которым нравится математика, выбирают
      <a  href="https://mmf.bsu.by/ru/"  title="Готовит
специалистов в области математики, механики, системного
анализа и информационных технологий">мех-мат</a>.
    </p>
  </body>
</html>
```



Описание из атрибута title отображается только при наведении курсора, а значит, люди, использующие клавиатурные элементы управления для навигации по веб-страницам, не увидят его. Поэтому если информация из описания ссылки действительно важна для удобства использования страницы, то ее нужно разместить так, чтобы она была доступна для всех пользователей, например в обычный текст на странице.

Атрибут target

По умолчанию новый документ загружается в текущее окно браузера, однако это свойство можно изменить с помощью атрибута target.

Атрибут target содержит имя окна или фрейма, куда браузер будет загружать документ. Target может иметь одно из следующих значений:

- _blank – открывает документ в новом окне или вкладке;
- _self – открывает документ в том же окне/вкладке, где был произведен клик по ссылке;
- _parent – открывает документ в родительском окне;
- _top – отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то этот параметр работает как _self;
- framename – открывает связанный документ в именованном фрейме.

В данном примере ссылка будет загружена в новой вкладке браузера.

```
<p>Open link in a new window or tab: <a
href="http://www.w3schools.com"  target="_blank">Visit
W3Schools!</a></p>
```

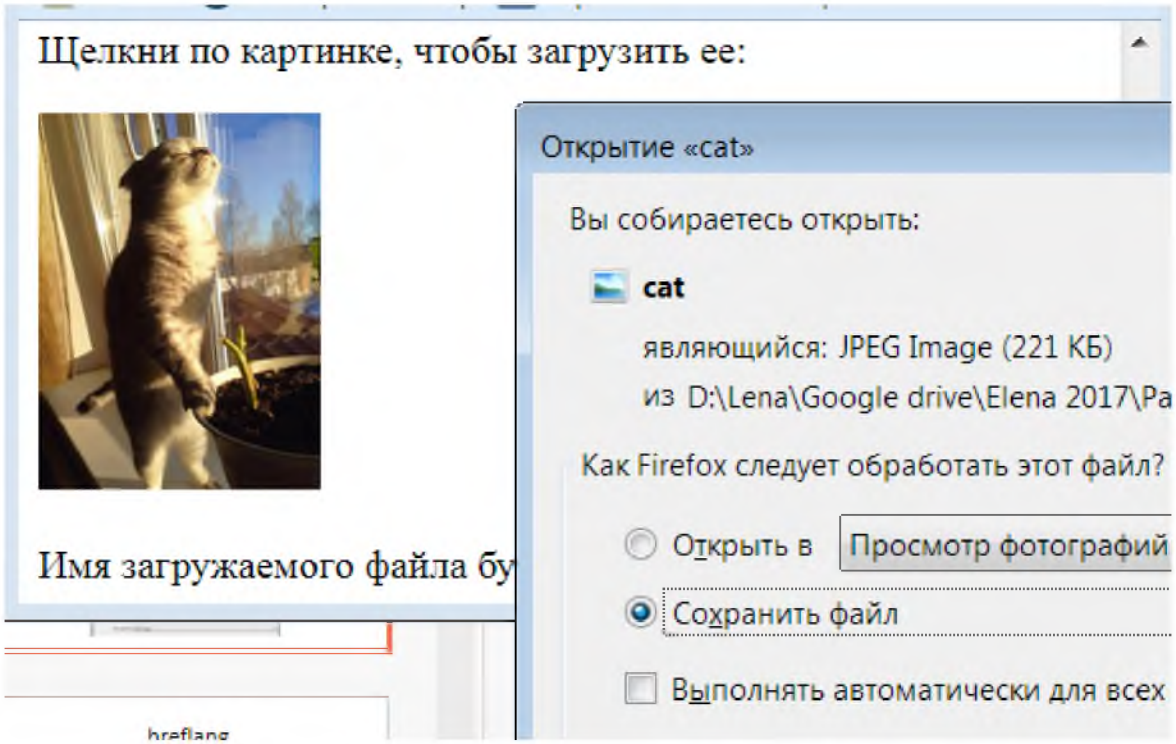
Атрибут download

Атрибут указывает, что объект, указанный в href, будет загружен, когда пользователь нажмет на гиперссылку. Атрибут download используется, только если установлен атрибут href.

В качестве значения атрибута указывают имя загруженного файла. Расширение файла указывать необязательно, браузер автоматически определит правильное расширение файла и добавит его в файл (.img, .pdf, .txt, .html и т. д.).

Если значение download не указано, то используется исходное имя файла.

```
<p>Щелкни по картинке, чтобы загрузить ее:<p>
<a href="images/Vesna.jpg" download="cat">
  <img border="0" src="images/Vesna.jpg" alt="Кот на подоконнике" height="150">
</a>
<p>Имя загружаемого файла будет cat.jpg вместо Vesna.jpg</p>
```

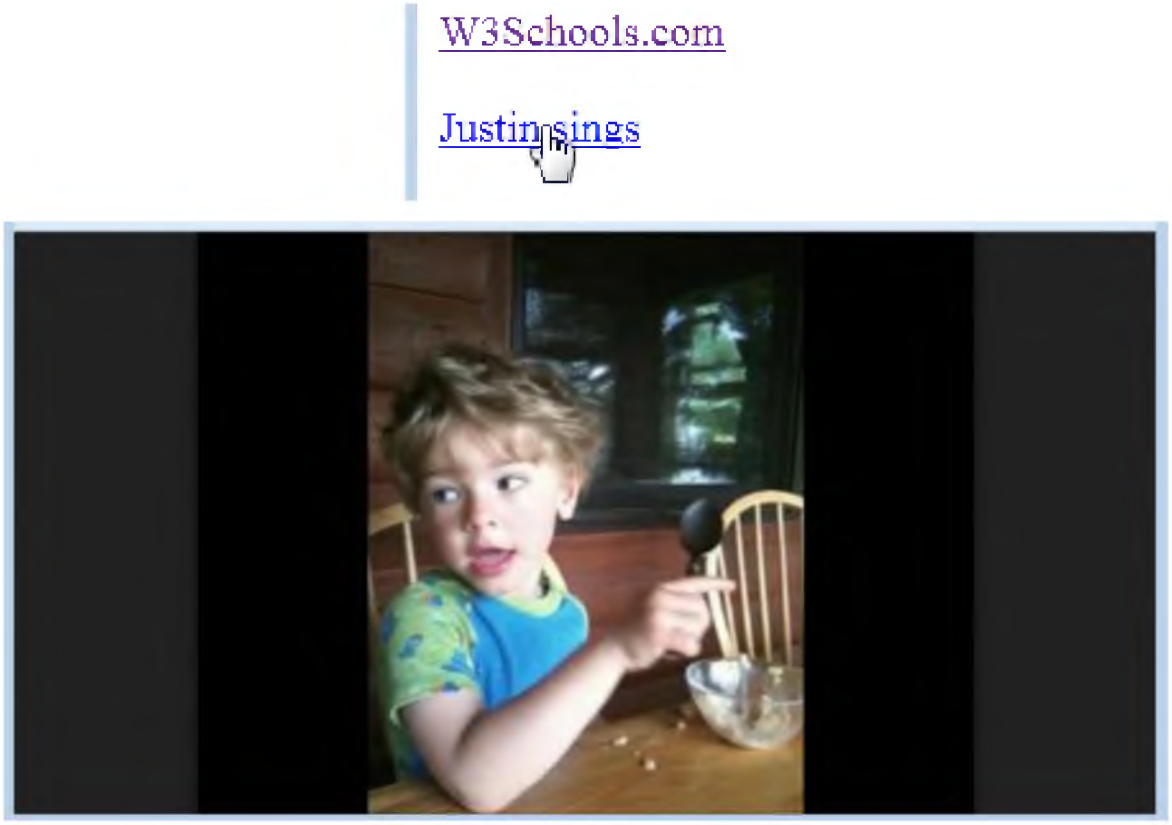


Атрибут type

Содержит подсказку о типе содержимого, находящегося по целевому адресу гиперссылки, например название media через запятую. Атрибут является рекомендательным. Подсказка позволяет браузерам решать, использовать указанный механизм для открытия ресурса или получить содержимое с предупреждением о том, что это содержимое имеет тип, не поддерживаемый пользовательским агентом. Программисты, использующие этот атрибут, сами отвечают за соответствие указанного типа реальному содержимому, находящемуся по гиперссылке.

Список стандартных медиатипов можно посмотреть по адресу http://www.iana.org.

```
<p>
  <a href="http://www.w3schools.com"
type="text/html">W3Schools.com
  </a>
</p>
<p>
  <a href="media/justin.mp4" type="video/mp4">Justin
sings
  </a>
</p>
```



Атрибут hreflang

Атрибут hreflang указывает на язык связанного документа. Значением атрибута является двухбуквенный код языка. Посмотреть коды языков можно, например, на сайте W3C: https://www.w3schools.com/tags/ref_language_codes.asp. Этот атрибут используется, только если установлен атрибут href. Атрибут является рекомендательным, браузер сам решает, использовать его или нет.

```
<a hreflang="en"
href="http://www.w3schools.com">W3Schools.com</a>
```

Атрибут media

Атрибут media определяет, для каких медиаустройств оптимизирован связанный документ. Этот атрибут используется для указания того, что целевой URL предназначен для специальных устройств, например iPhone, устройств воспроизведения речи или принтеров.

Атрибут является рекомендательным, браузер сам решает, использовать его или нет.

```
<a href="att_a_media.asp?output=print" media="print and (resolution:300dpi)">Open media attribute page for print</a>
```

Список значений можно найти на сайте W3C: https://www.w3schools.com/tags/att_a_media.asp.

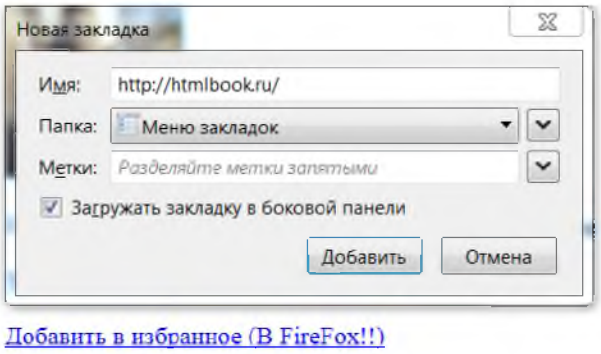
Атрибут rel

Определяет прямую связь между ресурсом, на который указывает ссылка, и текущим документом. Значением этого атрибута является список разделенных пробелами типов ссылок:

- alternate – ссылка на альтернативное представление документа (т. е. печать страницы, переведенная страница или зеркало);
- author – ссылка на автора документа;
- bookmark – постоянная ссылка на раздел или запись;
- external – указывает, что внешний ссылочный документ не является частью того же сайта, что и текущий документ;
- help – ссылка на документ со справкой;
- license – ссылка на страницу с лицензионным соглашением или авторскими правами;
- next – ссылка на следующую страницу или раздел;
- nofollow – поисковый бот не должен следовать по этой ссылке;
- noreferr – не передавать по ссылке HTTP-заголовки;
- noopener – без использования rel="noopener" новая страница открывается с получением прав на объект window с помощью window.opener, что является уязвимостью;
- prev – ссылка на предыдущую страницу или раздел;
- search – ссылка на поиск;
- tag – указывает, что метка (тег) имеет отношение к текущему документу.

Однако есть еще несколько значений, не внесенных в стандарт, но поддерживаемых браузерами, например sidebar.

```
<a    href="http://htmlbook.ru"    rel="sidebar">Добавить
в избранное (В FireFox!!)</a>
```



Что может быть ссылкой?

Ссылка необязательно должна быть текстовой. В качестве ссылки может использоваться любой HTML-элемент, например картинка, элемент списка или блок. В следующем примере как ссылка работают и текст, и картинка.

```
<a href="http://vangogh-world.ru">
  <img  src="images/Vangog.png"  alt="Vangog"  width="50"
style="border:0;">
  Мир Ван Гога
</a>
```

Рекомендации по написанию хороших ссылок

1. Пользователи часто бегло просматривают страницу, не читая каждое слово, и их глаза будут привлечены к тексту, который выделяется, например ссылкам. Им будет полезно описание того, куда ведет ссылка, поэтому лучше не писать «жми сюда», а выражаться более конкретно, например для того, чтобы скачать песню, – «скачать “Охота на волков”» или для того, чтобы перейти к подробному описанию тура, – «Горящий тур в Египет» и т. д.
2. Поисковые системы используют текст ссылки для индексирования файлов, поэтому рекомендуется включать ключевые слова в текст ссылки, чтобы эффективно описывать, куда ведет ссылка.

3. Не нужно включать адрес ссылки в ее текст. URL-адреса выглядят сложными, а звучат еще сложнее, когда программы чтения с экрана читают их по буквам.

4. Не пишите «ссылка» или «ссылки на» в тексте ссылки – это лишнее. Программы чтения с экрана сами это проговаривают для ссылок. На экране пользователи также видят, что является ссылкой, потому что ссылки, как правило, выделяются оформлением.

5. Следите за тем, чтобы текст ссылки был как можно короче. Длинный текст ссылки особенно раздражает пользователей программ чтения с экрана, которым придется услышать все, что написано.

6. Если ссылка указывает не на другую страницу, а, например, на медиаконтент, добавляйте описание. Допустим, есть ссылка на файл, нажав на которую можно загрузить документ PDF или Word, или открыть просмотр видео, прослушивание аудиофайла, или перейти на страницу с другим, неожиданным для пользователя результатом (всплывающее окно или загрузка Flash-фильма). Тогда, если используется соединение с низкой пропускной способностью и пользователь нажмет на ссылку без описания, начнется загрузка большого файла; или если не установлен Flash-плеер, а пользователь по ссылке перейдет на страницу с Flash-контентом и т. д.

Примеры ссылок с описанием:

```
<p><a href="http://www.example.com/large-report.pdf">
  Скачать отчет о продажах (PDF, 10MB)
</a></p>

<p><a href="http://www.example.com/video-stream/">
  Посмотреть видео (HD качество)
</a></p>

<p><a href="http://www.example.com/car-game">
  Играть в гонки (необходим Flash)
</a></p>
```

7. Старайтесь, чтобы у ссылок, указывающих на разные страницы, был разный текст. Иначе могут возникнуть проблемы у пользователей программ чтения с экрана, которые часто вызывают список ссылок: несколько ссылок, одинаково помеченных как «нажмите здесь», будут путать.

Упражнение

1. На странице со своим резюме вставьте ссылку на сайт факультета.
2. Вставьте ссылку на электронную почту на два адреса с темой сообщения и текстом сообщения.

3. Сделайте страницу с историей факультета и при помощи якорей – навигацию по странице.
4. Сделайте ссылки на странице относительными.
5. Вставьте на страницу ссылку, которая будет открываться в новой вкладке.
6. Измените стиль ссылки так, чтобы она не была подчеркнута.
7. Сделайте картинку ссылкой.
8. В нижней части документа вставьте ссылку на начало страницы в виде изображения стрелочки вверх.

Изображения <img>

Спецификация HTML5 рекомендует рассматривать рисунки на страницах подобно рисункам в книге. Другими словами, изображения отделяются от текста, но на них в тексте делаются ссылки. При этом изображения размещаются как плавающие объекты, т. е. их вставляют в нужное место в тексте, вместо того чтобы закреплять возле конкретного слова или элемента. Часто изображения снабжаются подписями, которые закрепляются возле рисунков.

Тег ***<img>*** предназначен для вставки изображений, чаще всего в формате .gif, .jpeg или .png. Поскольку формат .gif поддерживает анимированные изображения, то их размещение на странице тоже возможно. Синтаксис вставки анимированных изображений такой же, как и для неанимированных.

```
<p>Тег img предназначен для вставки изображений формате GIF, JPEG или PNG.</p>
<img src="pes.gif" width="138" height="175" alt="Щенок">
<img src="images/Злюка.jpg" width="166" height="185" alt="Кот">
<img src="images/html and css.png" width="287" height="175" alt="Логотип">
```

Как видно, первое изображение находится в том же каталоге, что и страница, а второе и третье – в подкаталоге images. Первое изображение анимированное.

Технически изображения не вставляются на страницу. Тег ***<img>*** выделяет место на странице для воспроизведения изображения, а само изображение вставляется в HTML-документ в виде ссылок на внешний файл.

Тег ***<img>*** пуст, он содержит только атрибуты и не имеет закрывающего тега. Тег ***<img>*** имеет два обязательных атрибута: ***src*** и ***alt***.

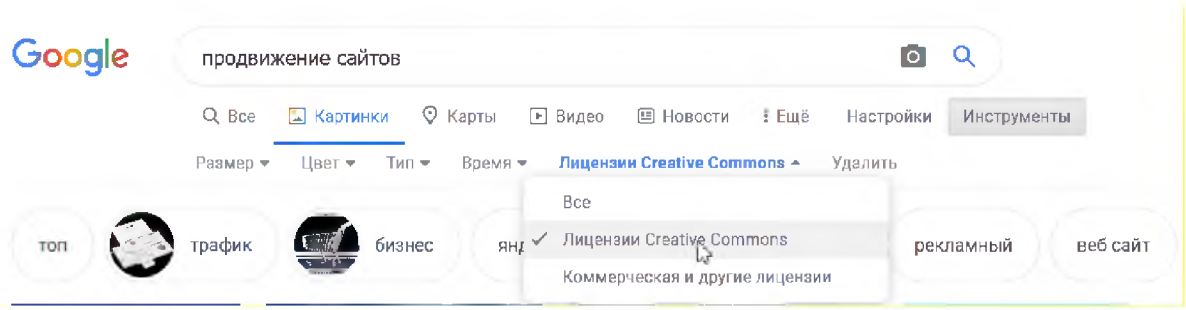
В большинстве браузеров тег ***<img>*** будет отображаться по умолчанию как встраиваемый блок:

```
img {
    display: inline-block;
}
```

Тег ***<img>*** используют для вставки изображений на страницу лишь в том случае, если изображение несет смысловую нагрузку на странице. Если же картинка является только элементом дизайна, то принято использовать CSS-свойство background-image и другие свойства background-*, которые применяются для контроля размещения фонового изображения.

Авторские права

Большинство изображений в интернете, использованных в «Google Картинках» имеют авторские права. Для снижения вероятности нарушения авторских прав используйте фильтр лицензии Google: «Картинки → Инструменты → Лицензии Creative Commons».



Изображения можно размещать на своих веб-страницах только в том случае, если:

- вы владеете авторскими правами на изображение;
- у вас есть письменное разрешение владельца изображения;
- изображение находится в открытом доступе и это можно доказать.

Нарушение авторских прав является незаконным и может повлечь судебные иски.

Помещение на странице в своем атрибуте src ссылки на изображение, размещенное на чужом сайте, называется называется «хотлинкинг» (от англ. hotlinking – ‘горячая ссылка’). При этом:

- происходит кража пропускной способности чужого сайта, на котором размещено изображение;
- вы не имеете контроля над изображением, изображение может быть удалено или перемещено;
- это замедляет вашу страницу;
- это также незаконно.

Атрибут src

Обязательный параметр, определяющий адрес файла с картинкой. Если указано только имя файла, то браузер ожидает найти изображение в той же папке, что и веб-страницы. Чаще всего изображения хранят в отдельной папке – в этом случае нужно указать имя папки в атрибуте src. Некоторые веб-сайты хранят изображения со своих страниц на серверах изображений, в качестве адреса можно указать любой адрес в интернете.

Поисковые системы просматривают имена изображений и учитывают их при индексации по поисковам запросам. Поэтому необходимо давать изображениям осмысленные имена, например, Puppy.jpg предпочтительнее, чем img25.jpg.

```
<p>Атрибут src.</p>
<img src="pes.gif" width="138" height="175" alt="Щенок">
<img src="images/Злюка.jpg" width="166" height="185"
alt="Кот">
<img src="https://encrypted-
tbn3.gstatic.com/images?q=tbn:ANd9GcQJKBYYroxSU30zpj6RKh
6SQDfDZfqtURqZwlk8bCs-jeMoZ7hd" width="287" height="175"
alt="Логотип">
```

В данном примере первое изображение находится в той же папке, что и страница. Второе – в папке images, которая размещена в той же директории, что и страница. Третье – на стороннем сайте, не являющемся хранилищем изображений, имеет место «хотлинкинг».

Атрибут alt

Обязательный параметр. Содержит текстовое описание картинки, которое отображается в браузере:

- пока картинка грузится при медленном соединении;
- когда в браузере отключен режим автоматической загрузки отображений;

- когда есть ошибка в атрибуте src;
- если пользователь использует специализированный браузер для чтения с экрана;
- если браузер не поддерживает формат данного изображения.

Некоторые люди до сих пор используют текстовые браузеры, такие как Lynx, которые вместо изображений отображают текст из атрибута alt.

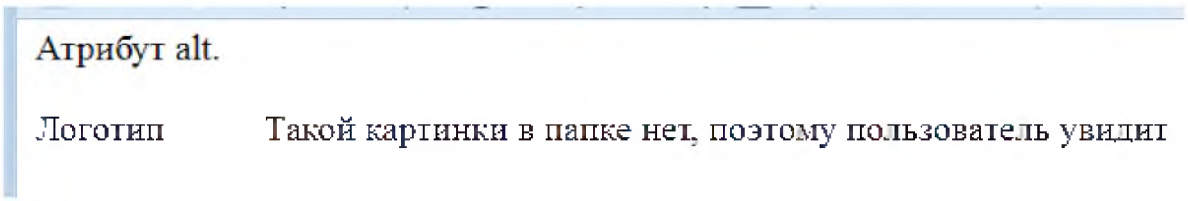
Атрибут alt используется поисковыми машинами при индексации страниц. Если отсутствует атрибут alt, то документ не будет валидным.

Несколько рекомендаций по заполнению alt:

- текст должен описывать изображение и по возможности включать ключевые слова;
- если изображение является ссылкой, то текст должен пояснять, куда будет совершен переход;
- если изображение используется только как элемент дизайна, используйте в alt пустую строку: alt = "".

Поисковые системы осуществляют поиск совпадения поисковых запросов также и с текстом атрибута alt. Это позволяет находить размещенное изображение на странице с помощью поисковых систем.

```
<p>Атрибут alt.</p>
<img src="images/html and css1111.png" width="100"
height="100" alt="Логотип">Такой картинки в папке нет,
поэтому пользователь увидит альтернативный текст.
```



Атрибуты height и width

Параметр height задает высоту области отображения в пикселях, атрибут width – ее ширину. При явном указании одного размера: width или height – и ширина, и высота картинки изменяются одновременно, при этом размер картинки изменяется пропорционально оригиналу.

Можно использовать CSS для задания размера изображений. Несмотря на то что оба атрибута: и ширины, и высоты изображения – действительны в HTML5, консорциум W3C рекомендует использовать атрибут стиля. Это предотвращает внутренние или внешние таблицы стилей от изменения оригинального размера изображений.

Если не указать размеры изображения, то загрузится изображение оригинального размера. Следует всегда указывать размеры для

изображений, даже если они не меняются. Если устанавливаются высота и ширина картинки, то пространство, необходимое для изображения, резервируется сразу при загрузке страницы. Если размеры картинки не указаны, браузер до загрузки картинки не знает ее размер и, как следствие, не сможет зарезервировать соответствующее пространство под рисунок, поэтому макет страницы может меняться во время загрузки изображения, а картинка – мерцать.

Если картинка изначально имеет большие размеры, а в макет страницы требуется маленькое изображение, то уменьшение значений атрибутов height и width не лучшее решение. В этом случае грузится большое изображение, даже если оно выглядит маленьким на странице. Чтобы избежать этого, необходимо отмасштабировать изображение с помощью специальной программы, прежде чем использовать его на странице.

В HTML 4.01 высота может быть определена в пикселях или в процентах от содержащего элемента. В HTML5 значение должно быть в пикселях.

```
<p>Атрибуты width, height</p>
<!--Плохо. Надо задавать размеры изображения-->
<img src="images/Злюка.jpg" alt="Кот">
<!--Допустимо, но стиль HTML 4.01. Надо задавать размеры
в пикселях-->
<img src="images/Злюка.jpg" style="width:50%;" alt="Кот">
<!--Хорошо. Размеры изображения заданы-->
<img  src="images/Злюка.jpg"  width="219"  height="230"
alt="Кот">
<!--Еще лучше. Размеры изображения заданы через CSS-->
<img                                  src="images/Злюка.jpg"
style="width:219px;height:230px;" alt="Кот">
```

Атрибут srcset

Атрибут ***srcset*** тесно связан с атрибутом ***src***. Он содержит список из одного или нескольких значений, разделенных запятыми, указывающих набор возможных изображений для отображения в браузере. Каждая строка из набора содержит адрес изображения и может включать дескриптор ширины (например, 600w) и дескриптор плотности пикселей (например, 4x).

Браузер выбирает наиболее подходящее изображение из списка, к примеру учитывая плотность пикселей устройства, и отображает его.

```
<img src="images/space-needle.jpg"
srcset="images/space-needle.jpg    200w,    images/space-
needle-2x.jpg 400w, images/space-needle-hd.jpg 600w">
```

Существует понятие Responsive Web Design (респонсивный веб-дизайн) – это подход, который позволяет оптимизировать веб-страницу под устройство, которым пользуется пользователь. Это касается и изображений. Изображение должно хорошо смотреться на устройствах с сильно отличающимися размерами экрана, разрешением и другими характеристиками. Нецелесообразно использовать одни и те же изображения для дисплеев с высоким разрешением и устройств с меньшим разрешением дисплея. Если просто изменить размеры подгружаемого изображения, то это будет «съедать» трафик у владельцев изображения, и наоборот, если растровое изображение отображается в размере большем, чем оригинальный, то оно становится зернистым и выглядит плохо.

Поэтому для разных типов устройств с определенным типом разрешения надо подключать свой файл изображения, подобранный таким образом, чтобы само изображение смотрелось хорошо, но при этом имело минимальное разрешение для данного типа экранов.

Существует еще адаптивный дизайн. Основная разница между адаптивным и респонсивным дизайном в том, что адаптивный дизайн состоит из нескольких отдельных страниц, созданных под определенные размеры экранов. Другими словами, дизайнеры рисуют разметку страницы под мобильные устройства, планшеты и стационарные компьютеры. По аналогии с адаптивным дизайном можно использовать адаптивные изображения. Иногда для устройств с маленьким экраном лучше использовать немного другое изображение, а не уменьшенную копию. Это так называемые адаптивные изображения, которые бывают нужны в тех случаях, когда при уменьшении размера картинки изменяется ее восприятие. Например, приведенное ниже изображение будет хорошо смотреться как полноэкранное на большом дисплее.

Однако если его уменьшить, то на маленьком экране фигура человека окажется такой маленькой, что будет теряться. Лучше использовать обрезанную версию изображения, на которой видны важные детали снимка, когда сайт отображается на узком экране, и, возможно, что-то среднее между обрезанным и оригинальным изображениями для экранов средней ширины, таких как планшеты.

Атрибут sizes

Задает размеры изображения для разных макетов страницы. Работает только при заданном атрибуте srcset.

Атрибут longdesc

Атрибуты alt и title содержат краткую информацию о картинке, но не могут содержать большой текст. Атрибут longdesc позволяет указать адрес документа, где содержится более подробная информация о картинке. Такая информация используется речевыми браузерами. В качестве значения может быть полный или относительный путь к файлу. Неречевые браузеры не выводят текст, указанный по ссылке.

```
<p>Атрибут longdesc</p>
<!-- Описание на той же странице, что и изображение -->
<img src="pes.gif" alt="Пёсик" width="100"
longdesc="#dog"><br>
<!-- Описание на другой странице сайта -->
<img src="pes.gif" alt="Песик" width="100"
longdesc="dog.txt"><br>
<!-- Описание на  странице другого сайта-->
<img src="pes.gif" alt="Песик" width="100"
longdesc="https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%
B1%D0%B0%D0%BA%D0%B0"><br>
<!-- Описание включено в тег -->
<img src="pes.gif" alt="Песик" width="100"
longdesc="data:text/html;charset=utf-
8;,%3C!DOCTYPE%20html%3E%3Chtml%3E%3Chead%3E%3Ctitle%3ED
```

```
escription%20of%20the%20Logo%3C/title%3E%3C/head%3E%3Cbo
dy%3E%3Cp%3ESome%20description%20goes%20here%3C/body%3E%
3C/html%3E"><br>
<p id="dog">Мой песик....(описание).</p><br>
```

В данном примере в первом случае подробная информация о картинке размещена на той же странице, во втором случае – в отдельном текстовом документе в той же директории, что и исходная страница, в третьем – на странице Википедии, а в четвертом описание включено в текст.

Атрибут crossorigin

Определяет, используется ли CORS при загрузке изображения. Изображения, загруженные с помощью CORS, могут использоваться в элементах, не ограничивая функциональность последних (англ. tainted canvas).

Атрибут ismap

Атрибут ismap говорит браузеру, что рисунок является картой-изображением. Карты-изображения позволяют привязывать ссылки к разным областям одного изображения. Реализуется в двух различных вариантах – серверном и клиентском. В случае применения серверного варианта браузер посылает запрос на сервер для получения адреса выбранной ссылки и ждет ответа с требуемой информацией. Такой подход требует дополнительного времени на ожидание результата и отдельных файлов для каждой карты-изображения.

Отправка данных на сервер происходит следующим образом. Необходимо поместить тег <img> в контейнер <a>, где в качестве значения атрибута href указать адрес CGI-программы (англ. Common Gateway Interface – стандарт интерфейса, используемого для связи внешней программы с веб-сервером). Программа анализирует полученные координаты нажатия мыши, которые считаются от левого верхнего угла изображения, и возвращает требуемую веб-страницу. Пример можно посмотреть по адресу https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_img_ismap.

Атрибут usemap

Атрибут usemap связывает между собой картинку и карту-изображение, задаваемую с помощью контейнера <map>.

В качестве такой связи выступает имя.

Идентификатору, который указывается в значении атрибута usemap, ставится в соответствие атрибут name тега <map>. При этом в <img> идентификатор должен начинаться с символа решетки (#).

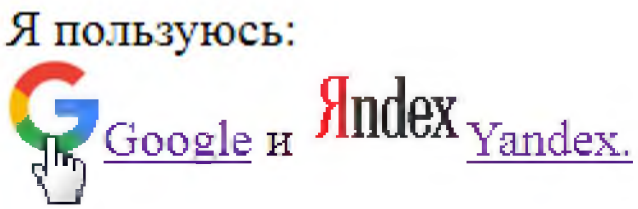
Атрибуты align, border, hspace, vspace не поддерживаются HTML5. Тег <img> поддерживает все глобальные атрибуты и атрибуты событий.

Атрибут border

Атрибут border устанавливает ширину рамки вокруг изображения, а также определяет, будет ли изображение обрамлено в рамку, в случае если оно является частью тега <a>. Если при переходе по ссылке отображение рамки нежелательно, то следует установить `border="0"`.

Атрибут border не поддерживается в HTML5. Вместо него надо пользоваться таблицами стилей CSS.

```
Я пользуюсь:<br>
<a href="https://www.google.by/"><img
src="images/googlechrome.jpg" width="30"
style="border:0;">Google</a>  и
<a href="https://www.yandex.by/"><img
src="images/Яндекс.png" width="50">Yandex.</a>
```



Упражнения

1. Поместите на страницу несколько изображений. Часть из них загрузите с диска, а часть из интернета.
2. Вставьте анимированное изображение.
3. Вставьте изображение. Уменьшите его размер в два раза.
4. Сделайте изображение ссылкой.
5. Вставьте изображение и текст. Сделайте так, чтобы текст абзаца обтекал текст картинки справа. Увеличьте размер чистого поля слева и справа от изображения до окружающего его контента.
6. В теге <img> укажите адрес документа, где содержится аннотация к картинке.
7. Вставьте альтернативный текст, который будет отображаться в случае, если картинка не загрузится.

Семантические изображения с подписью <figure> и <figcaption>

Если картинка представляет собой независимый элемент контента, например фото, иллюстрацию, диаграмму, листинг кода, то тег <img> оборачивают в семантический контейнер <figure>. <figcaption> задает описание для картинки.

```
<figure>
    <img src="images/frog.gif" alt="" />
    <figcaption>Лягушка</figcaption>
</figure>
<figure>
    <img src="images/pes.gif" alt="" />
    <figcaption>Щенок</figcaption>
</figure>
```

Тег <figure> используется не только в связке с изображениями. Он представляет собой независимый структурный элемент и может включать несколько изображений, кусок кода, аудио, видео, уравнение, таблицу и т. д.

Альтернативные изображения

Иногда желательно выводить различные изображения в зависимости от размера экрана устройства. Например, если при просмотре в браузере на настольном компьютере на странице отображается большой пейзажный снимок с человеком посередине, то при просмотре этой же страницы в мобильном браузере он будет выглядеть плохо, так как человек окажется очень мелким и его будет тяжело разглядеть. Вероятно, будет лучше показать меньшую портретную картинку в мобильной версии, на которой человек отображается в приближении. Элемент <picture> позволяет применять именно такое решение. Например:

```
<picture>
  <source media="(max-width: 799px)" srcset="ivan-at-the-sea-480w.jpg">
  <source media="(min-width: 800px)" srcset="ivan-at-the-sea-800w.jpg">
  <img src="ivan-at-the-sea-800w.jpg" alt="Ivan is resting in Yalta">
</picture>
```

Форматы изображений

Существует множество форматов изображений. Наиболее часто используются три из них: JPEG, PNG, GIF.

JPEG

Формат JPEG (Joint Photographic Experts Group – объединенный комитет экспертов по фотографии) разрабатывался первоначально для хранения и сжатия полноцветных фотографий. Это растровый графический формат, т. е. изображение представляет собой сетку цветных точек или пикселей. Поддерживает 16,8 млн цветов. Имеет алгоритм сжатия. Сжатие использует разбиение изображения на блоки 8×8 пикселей и последующую их группировку. При этом происходит потеря информации. Формат предназначен для сжатия изображений с плавными переходами яркости и цвета. Непригоден для сжатия текстовой графики, чертежей или изображений, где имеется резкий контраст между соседними пикселями. На таких контрастных границах при сжатии возможно искажение.

При работе с JPEG важно подбирать такой уровень качества, чтобы вес изображений был минимальным, но при этом качество картинки было приемлемым. Формат не поддерживает прозрачность и анимацию.

GIF

Формат GIF (Graphics Interchange Format – формат для обмена изображениями) является растровым, т. е. картинка представляет собой матрицу из цветных пикселей. Поддерживает прозрачность и анимацию. Использует только 256 цветов, поэтому малопригоден для хранения фотографий. Алгоритм сжатия позволяет хранить файлы, в которых много однородных заливок, например логотипы, надписи, схемы, без потери качества. Долгое время был одним из наиболее распространенных форматов в интернете. К недостаткам можно отнести то, что размер файла может быть достаточно большим. Используется для простых изображений с четкими краями, таких как текстовые изображения, иконки, баннеры, для анимированных изображений и изображений, где требуется прозрачность.

В настоящее время для замены GIF создан более современный бесплатный формат PNG.

PNG

PNG (Portable Network Graphics – портативная сетевая графика) является растровым форматом хранения графической информации, использует алгоритм сжатия без потерь. Создавался как бесплатная альтернатива GIF. Анимацию не поддерживает.

Горизонтальная линия. <hr>

Часто в HTML-документе возникает необходимость отделения одной части документа от другой линией. Чаще всего горизонтальной линией отмечают конец тематического раздела, например главы. Такую линию можно составить из символов подчеркивания или тире. Однако, во-первых, это может привести к тому, что при открытии документа на компьютере пользователя горизонтальная линия будет либо короткой, либо слишком длинной, в зависимости от размера открытого окна браузера, что отрицательно скажется на внешнем виде документа. А во-вторых, такая линия не будет нести семантического смысла и будет только элементом декора. В HTML5 все атрибуты макета стараются исключить из HTML и переместить в CSS. Тег <hr> определяет тематический разрыв и визуализируется как горизонтальная линия.

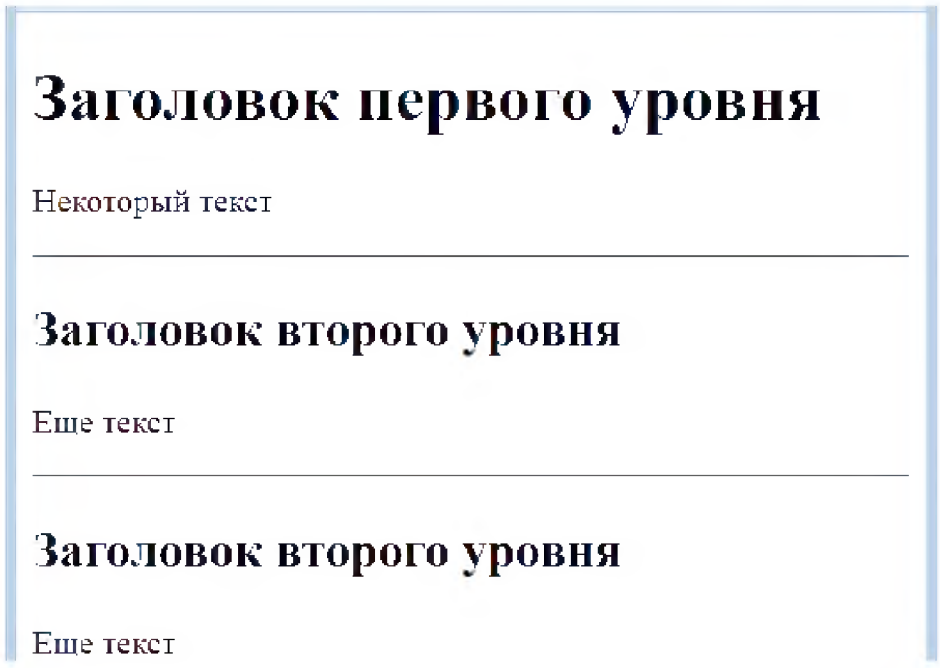
Тег <hr> относится к блочным элементам, поэтому линия всегда начинается с новой строки, а после нее все элементы отображаются на следующей строке.

Атрибуты <hr>, которые были в предыдущих версиях HTML, в HTML5 не поддерживаются.

Большинство браузеров будут отображать элемент <hr> со следующими значениями по умолчанию:

```
hr {
    display: block;
    margin-top: 0.5em;
    margin-bottom: 0.5em;
    margin-left: auto;
    margin-right: auto;
    border-style: inset;
    border-width: 1px;
}
```

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Тег hr</title>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    <p>Некоторый текст</p>
    <hr>
    <h2>Заголовок второго уровня</h2>
    <p>Еще текст</p>
    <hr>
    <h2>Заголовок второго уровня</h2>
    <p>Еще текст</p>
  </body>
</html>
```



Упражнение

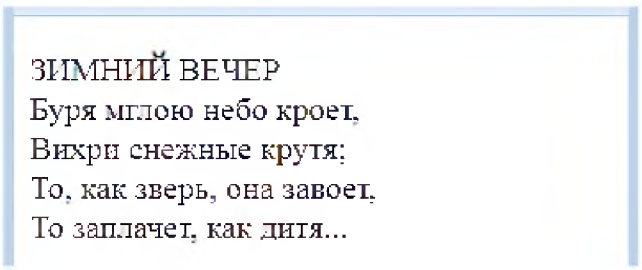
Вставьте тег горизонтальной линии, где это уместно для улучшения восприятия материала.

```
<!doctype html>
<html>
<body>
<h1>ТЕСТЫ И КОНКУРСЫ</h1>
<h2>Конкурс KP.BY: пришли рецепт с фото своего любимого блюда – и выиграй микроволновку или пылесос!</h2>
<p>Хотите порадовать себя или своих близких новенькой бытовой техникой? Участвуйте в нашем конкурсе и выигрывайте приятные призы!</p>
<h2>Конкурс на kp.by: «Тайная жизнь моего питомца»</h2>
<p>Присылайте снимки ваших любимцев – котиков, собачек, кроликов, хомячков, морских свинок… Да хоть игуан! И выигрывайте приз: упаковку корма, билеты в кино и игрушечных героев «Тайной жизни домашних животных»</p>
</body>
</html>
```

Разрыв строки

Тег
 – это тег перехода на следующую строку. Тег не парный, он не имеет закрывающего тега. В отличие от тега параграфа <p> использование тега
 не добавляет пустой отступ перед строкой. Тег также поддерживает глобальные атрибуты в HTML. Элементы
 предназначены только для разрывов строк, которые являются частью содержимого, например в строфах стихотворений или адресах:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p>ЗИМНИЙ ВЕЧЕР<br>
       Буря мглою небо кроет,<br>
       Вихри снежные крутя;<br>
       То, как зверь, она завоет,<br>
       То заплачет, как дитя...
  </body>
</html>
```

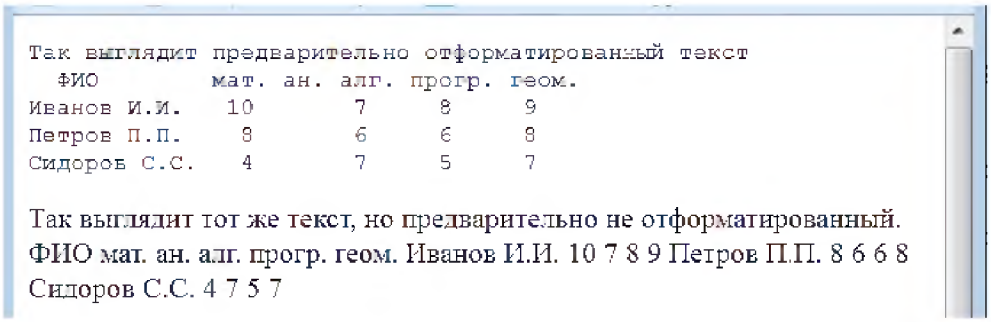


Тег предварительного форматирования <pre>

Тег <pre> определяет блок предварительно отформатированного текста. Такой текст отображается обычно шрифтом фиксированной ширины (чаще всего Courier), и он сохраняет как пробелы, так и разрывы строк. Внутри контейнера <pre> допустимо применять другие вложенные теги.

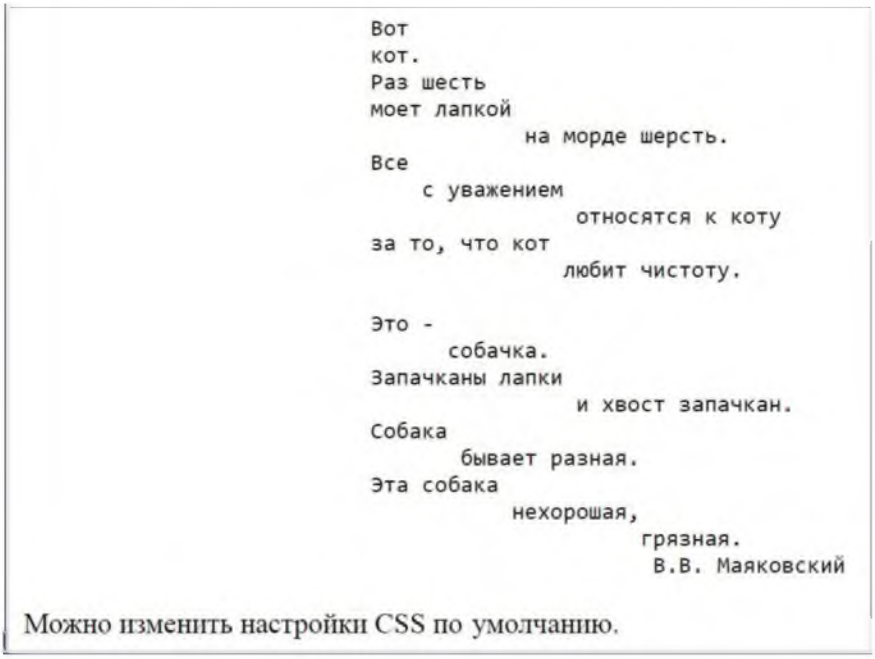
Элемент <pre> можно использовать для оформления колонок, например в таблице без использования тега <table>.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
<pre>Так выглядит предварительно отформатированный текст
  ФИО         мат. ан. алг. прогр. геом.
Иванов И.И.    10       7      8      9
Петров П.П.     8       6      6      8
Сидоров С.С.    4       7      5      7</pre>
Так выглядит тот же текст, но предварительно не
отформатированный.
ФИО         мат. ан. алг. прогр. геом.
Иванов И.И.    10      7      8      9
Петров П.П.     8      6      6      8
Сидоров С.С.    4      7      5      7
  </body>
</html>
```



Упражнение

Выведите на странице одно из стихотворений Владимира Маяковского, написанное в его фирменном «ступенчатом» стиле. Например, как на рисунке ниже.



Форматирование текста

В HTML есть теги, которые могут различным образом выделять текст. Следующие элементы являются основными тегами форматирования: <b>, , <i>, , <mark>, <small>, <del>, <ins>, <sub>, <sup>. Полный список всех тегов форматирования можно найти по адресу https://www.w3schools.com/tags/ref_byfunc.asp. Теги для форматирования текста допустимо использовать совместно и комбинировать их в любом порядке.

В HTML5 различают теги физического и логического форматирования. Теги физического форматирования изменяют внешний вид заключенного в них контента, но никаким образом не выделяют его логически. Это будет заметно, например, при прочтении текста звуковым веб-обозревателем для незрячих. Поисковые машины также не выделяют такой контент и не учитывают его особым образом.

Теги логического форматирования не просто выделяют текст зрительно, но и несут семантическую нагрузку. Поисковые машины учитывают их при сканировании и ранжировании страницы. Браузеры для незрячих и скринридеры (программы, которые считывают информацию с экрана монитора) также выделяют такой контент голосом.

Поскольку теги физического и логического форматирования зрительно не отличаются, то всегда предпочтительнее использовать теги логического форматирования, а физическое форматирование использовать только для дизайнерских целей.

В спецификации HTML5 указано, что заголовки должны быть обозначены тегами <h1> – <h6>, выделенный текст следует обозначать тегом , важный текст – тегом , а отмеченный текст – тегом <mark>.

<b>

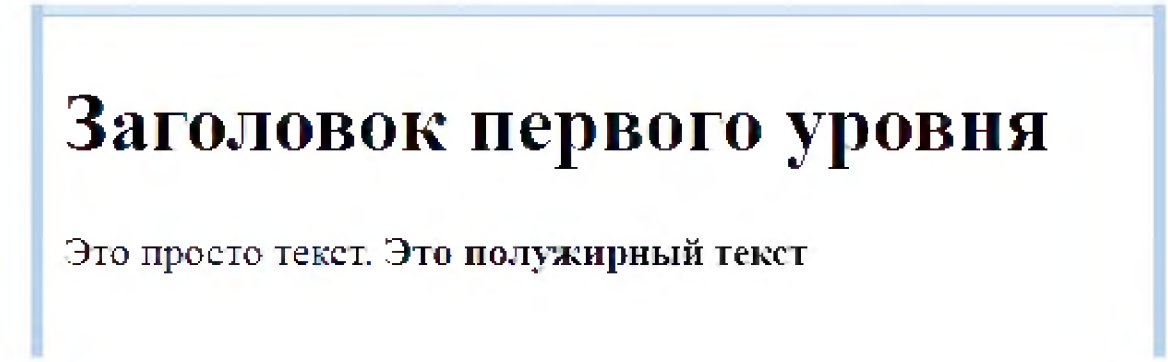
Тег <b> является тегом физического форматирования. Он устанавливает полужирное начертание шрифта, но не придает заключенному в нем контенту никакого дополнительного смысла. Использование тега <b> допускается в HTML, но применение таблиц стилей CSS предоставляет дополнительные возможности по управлению жирностью текста.

Большинство браузеров применяют по умолчанию к тегу <b> следующий стиль:

```
b {
    font-weight: bold;
}
```

Ниже приведен пример применения тега ***<b>***.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Теги форматирования</title>
  </head>
<body>
    <h1>Заголовок первого уровня</h1>
    Это просто текст.
    <b>Это полужирный текст.</b>
  </body>
</html>
```



Тег <b> используется для передачи значения, традиционно передаваемого жирным шрифтом: названия продуктов, выделения ключевых слов в инструкциях и т. д.

Упражнение

1. Выделите текст «<b>» полужирным начертанием, не придавая ему дополнительной смысловой нагрузки.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Теги форматирования</title>
  </head>
  <body>
    Тег &lt;b&gt; является тегом физического форматирования.
  </body>
</html>
```

2. Составьте инструкцию по приготовлению вашего любимого кулинарного блюда, а затем подготовьте ее для размещения на веб-странице, используя теги форматирования.

Тег относится к тегам логического форматирования и предназначен для привлечения особого внимания к тексту. Он устанавливает полужирное начертание шрифта и указывает, что заключенный в нем контент содержит важную информацию. В большинстве браузеров тег будет отображаться по умолчанию со следующим стилем:

```
strong {
    font-weight: bold;
}
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Теги форматирования</title>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    Это просто текст.
    <strong>Это важный текст.</strong>
  </body>
</html>
```

Тег выделяется голосом при воспроизведении считывающими с экрана программами.

Упражнения

1. Особо выделите в тексте, что тег <b> является тегом именно физического, а не логического форматирования.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Теги форматирования</title>
  </head>
  <body>
    Тег &lt;b&gt; является тегом физического форматирования.
  </body>
</html>
```

2. Сверстайте следующее объявление.

<i>

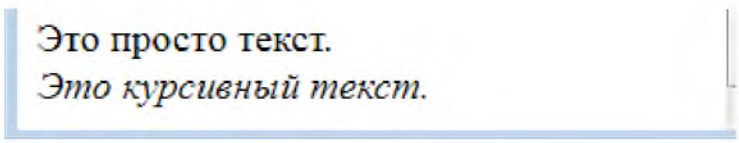
Тег <i> относится к тегам физического форматирования. Он придает заключенному в него тексту курсивное начертание, при этом не изменяя его семантическое значение.

Стиль отображения по умолчанию в браузерах:

```
i {
    font-style: italic;
}
```

...

```
Это просто текст.<br>
<i>Это курсивный текст.</i>
```



Используется для передачи значения, традиционно передаваемого курсивом: иностранные слова, таксономические обозначения, технические термины, высказывания.

Упражнение

1. Зрительно выделите курсивом название корабля, не меняя смысла текста.

```
<p>
    Флагманским кораблем Френсиса Дрейка был "Ревендж".
</p>
```

2. Сверстайте контент.

Элементарная таксономическая единица всего живого – вид. Латинское название вида – *species*. В современной таксономии принята бинарная номенклатура, введенная Карлом Линнеем в 18 веке. Основным отличием бинарной номенклатуры является двойное название вида, где существительное обозначает род, а видовой эпитет выражен прилагательным. Например, *Euphorbia splendens* – Молочай блестящий, *Salix caprea* –Ива козья.

Тег является тегом логического форматирования. Он выводит помеченный текст курсивом и слегка акцентирует внимание на нем.

Стиль отображения по умолчанию:

```
em {
    font-style: italic;
    }
```

Чаще всего используется для зрительного выделения терминов в тексте, фраз на другом языке, крылатых выражений или названий.

Это просто текст.
Акцентируем внимание на этом тексте.

Элемент распознается программами, считывающими с экрана, и произносится другим тоном.

Упражнение

1. Выделите англоязычное слово в тексте курсивом, акцентируя на нем внимание. Незначительно усильте семантическое значение терминов в тексте.

```
<p>Тег (англ. tag — ярлык, этикетка) — идентификатор для
поиска данных и задания внутренней структуры, а также
жаргонное название дескриптора, метки как ключевого слова.
</p>
```

2. Нужно ли, используя семантическую разметку, выделять слово «герой» в приведенных ниже случаях и почему?
   А. «Ну что, герой, восьмая двойка?» –
   В сердцах спросила сына мать. –
   «Вчера была головомойка,
   Придется снова наказать!»
   Б. «Вы – мой герой»: Полицейский в Хабаровском крае едва не погибла, спасая тонущего ребенка.
   В. Яркий светящийся меч «Веселая затея» – уникальная игрушка, которая, несомненно, привлечет внимание каждого любителя сражений. Возьмите светящийся меч, и вы – настоящий герой «Звездных войн» или космический пират! Аксессуар выполнен из яркой палочки, которая дает неоновое свечение голубого цвета. Меч светится в темноте в течение нескольких часов. Такой аксессуар дополнит карнавальный костюм и придаст настроение любому празднику.
   Г. Геро́й (от др.-греч. ἥρως – ‘доблестный муж, предводитель’) – человек исключительной смелости и доблести либо главное действующее лицо литературного произведения.

<mark>

Тег <mark> используется, если нужно как-то выделить, пометить участок текста. При этом помеченный текст выделяется фоном как маркером. В браузере Chrome и Firefox фоновый цвет текста внутри <mark> выделяется желтым цветом. Тег поддерживает глобальные атрибуты в HTML и атрибуты событий в HTML.

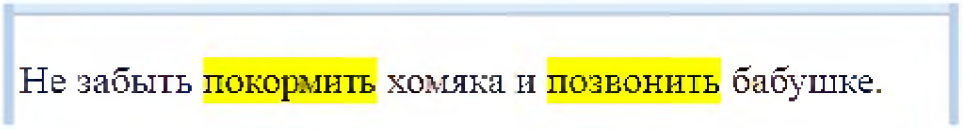
Большинство браузеров будут отображать тег <mark> со следующими значениями по умолчанию:

```
mark {
    background-color: yellow;
    color: black;
}
```

Пример верстки «напоминалки» на день:

```
<!doctype html>
<html>
  <head>
    <meta "charset=utf-8">
  </head>
  <body>
    <p>Не забыть <mark>покормить</mark> хомяка и
      <mark>позвонить
```

```
      </mark> бабушке.
    </p>
</body>
</html>
```



Упражнение

Выделите как маркером в тексте, что именно нужно купить в магазине.

После работы зайти в магазин и купить батон и сыр.

<small>

Тег <small> устанавливает относительный размер шрифта текста и уменьшает его относительно базового. Является тегом физического форматирования.

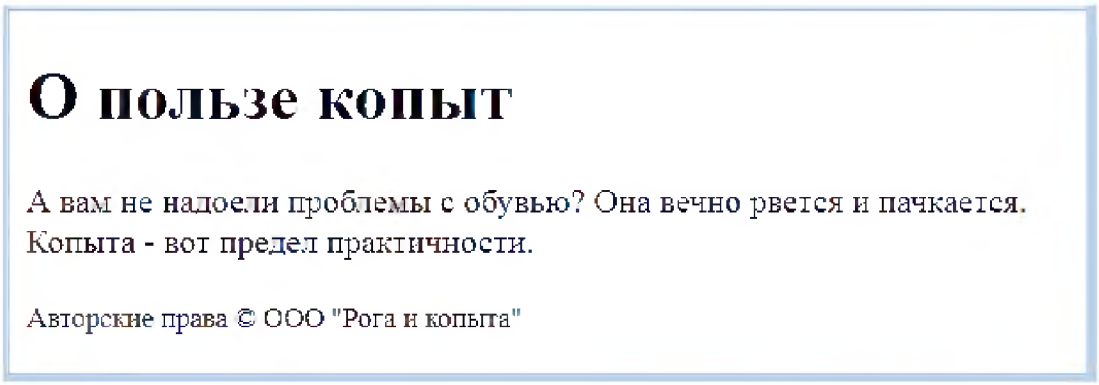
Стиль отображения в браузерах по умолчанию:

```
small {
    font-size: smaller;
}
```

Тег поддерживает глобальные атрибуты в HTML и атрибуты событий.

```
<!doctype html>
<html>
  <head>
    <style>
      small {
        font-size: smaller;
      }
    </style>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>О пользе копыт</h1>
    <p>А вам не надоели проблемы с обувью?
      Она вечно рвется и пачкается.
      Копыта — вот предел практичности.</p>
      <small>Авторские права &copy; ООО "Рога и
     копыта"</small>
     <!--<p>Если присоединить правило CSS, эффект будет
таким же</p>-->
```

```
  </body>
</html>
```



Все, что делает элемент <small>, можно сделать при помощи CSS, но тем не менее этот элемент продолжает использоваться. Теперь <small> используется для обозначения информации, которую владельцы ресурса обязаны указать согласно каким-либо требованиям, но не желают привлекать к ней внимание. Например, предупреждение об отказе от ответственности, помещенное в самом низу сайта, наподобие следующего:

```
<small> Любой желающий может разместить информационные материалы на сайте при условии, если предоставляемый контент соответствует тематике ресурса и размещается в соответствующем разделе. Администрация сайта не дает каких-либо гарантий в отношении сайта и его содержимого, в том числе не подтверждает: достоверность, точность, своевременность, актуальность и полноту размещенной на сайте информации.</small>
```

Упражнение

Независимо от размера основного шрифта текста абзаца, замечание к нему выведите более мелким шрифтом.

```
<p>Поклонники творчества студии Disney, ожидая выхода фильма «Красавица и чудовище», не теряли времени зря. Тысячи людей набили себе татуировки с изображениями главных героев – принцессы Белль, принца Адама, других сказочных персонажей.</p>
<p>Замечание: Это доказывает, что фанаты – сумасшедшие люди.</p>
```

Маркеры изменений. <ins> и <del>

Относятся к тегам логического форматирования. Тег <ins> предназначен для выделения изменений, внесенных в текст документа. Браузеры обычно помечают текст в теге <ins> как подчеркнутый.

Тег <del> используется для выделения текста, который был удален при исправлении документа. Браузеры обычно помечают текст в контейнере <del> как перечеркнутый.

Обычно теги <ins> и <del> используются вместе, что позволяет легко отследить, какие изменения в тексте документа были сделаны.

Стиль отображения в браузерах по умолчанию:

```
del {
    text-decoration: line-through;
}
ins {
    text-decoration: underline;
}
```

Теги <ins> и <del> имеют одинаковые атрибуты cite и datetime.

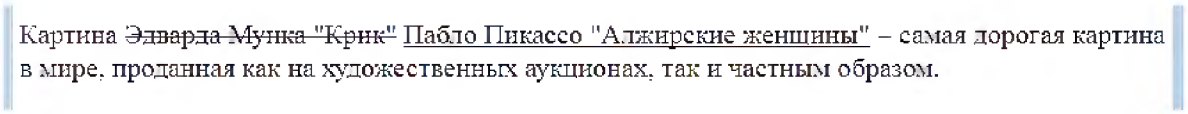
cite

Атрибут cite содержит ссылку на документ, в котором можно найти причину внесения изменений.

datetime

Атрибут datetime используется для сохранения времени внесения изменений. Такое форматирование позволяет отследить, какие изменения и когда были сделаны в тексте документа.

```
<p>Картина <del>Эдварда Мунка "Крик"</del> <ins
cite="https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D
1%81%D0%BE%D0%BA_%D1%81%D0%B0%D0%BC%D1%8B%D1%85_%D0%B4%D
0%BE%D1%80%D0%BE%D0%B3%D0%B8%D1%85_%D0%BA%D0%B0%D1%80%D1
%82%D0%B8%D0%BD">Пабло Пикассо "Алжирские женщины"</ins>
самая дорогая картина в мире, проданная как на
художественных аукционах, так и частным образом.</p>
```


Картина ~~Эдварда Мунка "Крик"~~ Пабло Пикассо "Алжирские женщины" – самая дорогая картина в мире, проданная как на художественных аукционах, так и частным образом.

Упражнение

Возьмите информацию о топ-10 богатейших людей мира из рейтинга Forbes за прошлый год и, используя теги <ins> и <del>, внесите в него изменения согласно тому же списку, но уже за текущий год.

<sub> и <sup>

Теги <sub> и <sup> относятся к тегам физического форматирования. Тег <sub> отображает шрифт в виде нижнего индекса. При этом текст располагается ниже базовой линии остальных символов строки и имеет уменьшенный размер. Может использоваться, например, для набора химических формул.

В большинстве браузеров <sub> будет отображаться со следующими настройками стиля по умолчанию:

```
sub {
    vertical-align: sub;
    font-size: smaller;
}
```

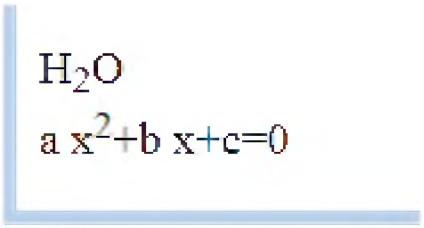
Тег <sup> отображает шрифт в виде верхнего индекса. При этом текст располагается выше базовой линии остальных символов строки и имеет уменьшенный размер. Может использоваться, например, для оформления сносок.

Стиль отображения по умолчанию:

```
sup {
    vertical-align: super;
    font-size: smaller;
}
```

Теги поддерживают глобальные атрибуты и атрибуты событий.

```
<p>
    H<sub>2</sub>O<br>
    a x<sup>2</sup>+b x+c=0
</p>
```



Упражнение

1. На странице разместите текст, содержащий неоригинальные части, и расставьте сноски для указания заимствований.
2. Разместите на веб-странице расписание занятий своей группы на неделю. При выводе времени занятий минуты выводите в виде верхнего индекса, например 8^{15}. Букву подгруппы также выводите в виде верхнего индекса.
3. Разместите на веб-странице формулу скалярного произведения векторов, не используя вставку изображений.

Теги для выделения цитат <q>, <blockquote>

Теги <q> и <blockquote> применяются для выделения цитат в тексте. Тег <q> применяется для коротких строчных цитат, не требующих прерывания абзаца. При выводе цитируемого текста он обрамляется в кавычки.

Большинство браузеров применяют по умолчанию к тегу <q> следующий стиль:

```
q {
    display: inline;
}

q:before {
    content: open-quote;
}

q:after {
    content: close-quote;
}
```

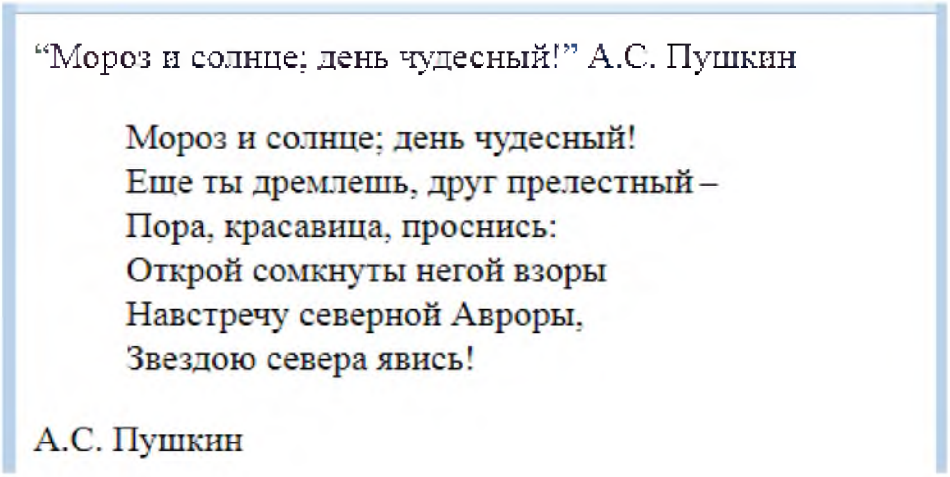
Тег <blockquote> применяется в том случае, если на страницу вставляются целые разделы из другого источника. По умолчанию в браузере выводится как выровненный блок с отступами слева и справа по 40 пикселей, а также с выделением сверху и снизу.

```
blockquote {
    display: block;
    margin-top: 1em;
    margin-bottom: 1em;
    margin-left: 40px;
    margin-right: 40px;
}
```

Оба тега имеют единственный атрибут cite, который указывает на источник цитаты. Для пользователя этот атрибут не отображается (невидим). Используется он в статистических целях при работе скриптов. Для этого тега доступны универсальные атрибуты и атрибуты событий.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>кавычки</title>
    <style>
      q {
         display: inline;
      }
      q:before {
         content: open-quote;
      }
         q:after {
    content: close-quote;
      }
```

```
      blockquote {
         display: block;
         margin-top: 1em;
         margin-bottom: 1em;
         margin-left: 40px;
         margin-right: 40px;
       }
    </style>
  </head>
  <body>
    <q    cite="http://pushkin.niv.ru/pushkin/stihi/stih-
640.htm">Мороз    и    солнце;    день    чудесный!</q>
А.С. Пушкин<br>
    <blockquote
cite="http://pushkin.niv.ru/pushkin/stihi/stih-
640.htm">Мороз и солнце; день чудесный!<br>
Еще ты дремлешь, друг прелестный -<br>
Пора, красавица, проснись:<br>
Открой сомкнуты негой взоры<br>
Навстречу северной Авроры,<br>
Звездою севера явись!
    </blockquote> А.С. Пушкин<br>
  </body>
</html>
```



<cite>

Тег <cite> является тегом логического форматирования и представляет название произведения (книги, статьи, эссе, стихотворения, партитуры, песни, сценария, фильма, телешоу, игры, скульптуры, картины, театральной постановки, спектакля, оперы, мюзикла, выставки, отчета о судебном деле, компьютерной программы и т. д.). Это может быть работа,

которая цитируется или на которую подробно ссылаются (т. е. цитата), или просто работа, которая упоминается мимоходом. Имя человека не является названием произведения – даже если люди называют этого человека произведением – и поэтому этот элемент нельзя использовать для разметки имен людей. (В некоторых случаях элемент <b> может подходить для имен: например, в статье, посвященной сплетням, имена известных людей представляют собой ключевые слова, обработанные в другом стиле, чтобы привлечь к ним внимание. В других случаях, если элемент действительно необходим, можно использовать элемент span.)

```
cite {
    font-style: italic;
}
```

Для этого тега доступны универсальные атрибуты и событийные атрибуты.

```
<p>Тег &lt;cite&gt; определяет название произведения.</p>
<p>Браузеры обычно отображают этот тег курсивом.</p>
<img src="images/12-sunflowers.jpg" height="277" alt="The
Scream">
<p><cite>Ваза с двенадцатью подсолнухами</cite> Ван Гога.
Написана в 1889.</p>
```

Упражнения

1. Разместите на странице несколько изображений картин и укажите авторство произведений.
2. Создайте HTML-документ и оформите фрагмент текста, представленный ниже, в виде веб-страницы.

Кот Матроскин лучше всех умеет отвадить назойливых соседей, выдрессировать заблудшего галчонка и обрадоваться понаехавшим гостям – потому что точно знает, как приспособить их к делу. Однажды он сказал:

Непр-равильно ты, дядя Федор, бутер-рброд ешь... Ты его колбасой кверху держишь, а надо колбасой на язык класть, м-м-м, так вкуснее получится...

Я тоже думаю, что иногда самое небольшое изменение сделает все гораздо лучше и вкуснее. Совершите что-то неожиданное. Переверните бутерброд. Найдите свой способ поедания пончиков! И поделитесь этим способом с друзьями, и тогда жизнь заиграет красками. Помните высказывание Вольтера: «Глуп тот человек, который остается всегда неизменным».

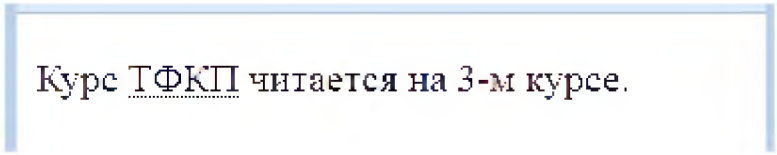
3. Преобразуйте высказывание Матроскина в блочную цитату <blockquote>, а высказывание Вольтера – в строчную цитату <q>, которые включает атрибут cite.
4. Оформите ссылки на сказку Э. Н. Успенского «Дядя Федор, пес и кот» и информацию о Вольтере на Википедии. Источники цитирования: https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%BB%D1%8C%D1%82%D0%B5%D1%80, https://vseskazki.su/eduard-uspenskij/dyadya-fjodor-pjos-i-kot.html.

Аббревиатура <abbr>

Тег <abbr> является тегом логического форматирования и определяет аббревиатуру, например ТФКП (Теория функций комплексных переменных), БГУ, «м. н. с.» (младший научный сотрудник). С помощью параметра title дается расшифровка сокращения. Расшифровка появляется на экране в подсказке при наведении курсора мыши на сокращение.

Поскольку аббревиатуры являются сокращениями некоторых понятий, то полезно, чтобы браузеры отождествляли их с полными понятиями. Разметка сокращений тегами <abbr> может дать полезную информацию для браузеров, системы перевода и поисковиков. Кроме этого, поисковые системы индексируют полнотекстовый вариант сокращения, что используется для повышения рейтинга документа.

```
<p>Курс <abbr title="Теория функций комплексной переменной">ТФКП</abbr> читается на 3-м курсе.</p>
```



Упражнение

1. Создайте страницу с расписанием своей группы на неделю. Сделайте так, чтобы все сокращения предметов воспринимались браузером как полные версии названий.

Контактная информация <address>

Тег <address> предназначен для хранения контактной информации, по которой можно связаться с автором. Информация может включать в себя любые элементы HTML вроде ссылок, текста, выделений и т. д. Поисковые системы должны анализировать содержимое этого тега для сбора информации об авторах сайтов. Если элемент <address> находится внутри элемента <body>, он представляет контактную информацию для данного документа. Если элемент <address> находится внутри элемента <article>, он представляет контактную информацию для этой статьи. По умолчанию текст внутри контейнера <address> отображается курсивным начертанием. Большинство браузеров добавят разрыв строки до и после элемента адреса.

```
address {
    display: block;
    font-style: italic;
}
...
<address>
Пишите мне <a
href="mailto:webmaster@example.com">Ковалеву И.А.</a>
<br>
Заходите на: Example.com<br>
</address>
```

Упражнения

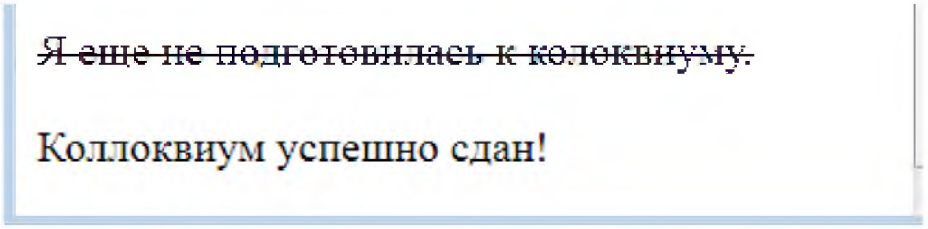
1. Укажите на странице контактную информацию.
2. Найдите и посмотрите в каком контексте используется тег <address> на странице https://www.w3.org/.

<s>

Тег используется для контента, который уже не верен. Тег <s> не должен использоваться для удаленного текста, для этого используйте <del>.

В большинстве браузеров будет отображаться как перечеркнутый текст:

```
s {
    text-decoration: line-through;
}
…
<p><s>Я еще не подготовилась к коллоквиуму.</s></p>
<p>Коллоквиум успешно сдан!</p>
```



Упражнение

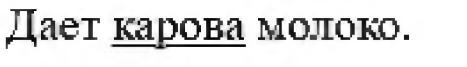
Составьте расписание дел на сегодня и перечеркните те пункты, которые уже реализованы.

<u>

Тег <u> относится к тегам физического форматирования. Он подчеркивает заключенный в него текст, при этом не изменяя его семантическое значение. Должен применяться для текста, который должен стилистически отличаться от обычного текста, например для слов с ошибками или собственных имен на иностранном языке.

Отображается как подчеркнутый текст:

```
u {
    text-decoration: underline;
}
…
Даёт <u>карова</u> молоко.
```



Тег <u> используется для передачи значения, традиционно передаваемого подчеркиванием: имя собственное, орфографическая ошибка. Подчеркнутый текст читается хуже, чем обычный. Оформление ссылок в виде подчеркнутого текста стало определенным стандартом, поэтому когда

пользователь видит подчеркнутый текст, то воспринимает его в силу привычки как ссылку. А ситуация, когда текст выглядит как ссылка, но при этом ссылкой не является, чаще всего будет раздражать пользователя.

Упражнение

Где в приведенном ниже контенте можно использовать тег подчеркивания?

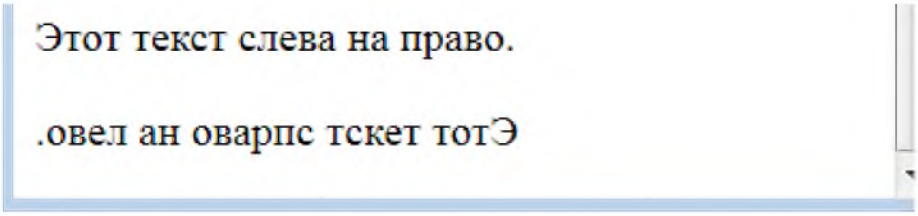
Почему-то то тут, то там предлагают купить «абонимент». Действительно, в слове слышится злосчастная «и». Проверить написание никак нельзя, потому что лексема словарная. Но запомнить можно, если подобрать пароним-рифму с буквой «е» – абонент. В обоих случаях букве «и» не место.

Изменение порядка следования слов <bdo> и <bdi>

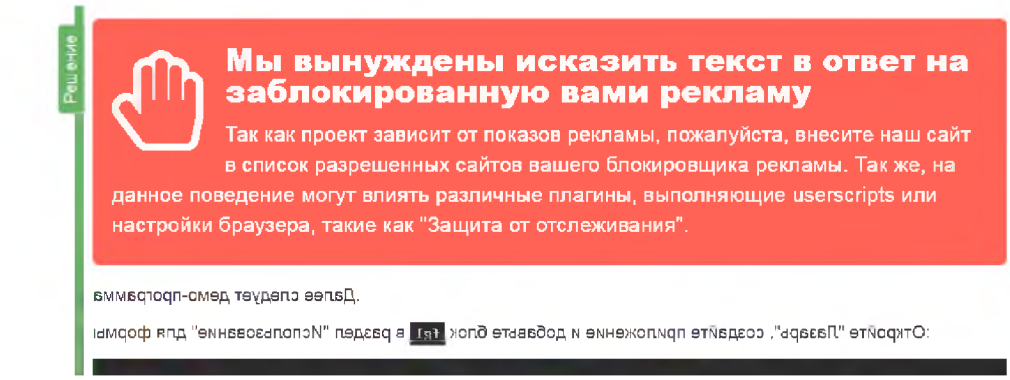
Тег <bdo> используется для переопределения текущего направления текста. Имеет только один атрибут – dir.

Атрибут dir определяет направление текста. Может принимать одно из двух значений: ltr – слева направо или rtl – справа налево.

```
<p>Этот текст слева направо.</p>
<p><bdo dir="rtl">Этот текст справа налево.</bdo></p>
```



При блокировке рекламы, вероятно, используется тег <bdo>.



Если необходимо часть контента вывести на другом языке, с другой направленностью текста, то можно воспользоваться тегом <bdi>. Элемент <bdi> (bidirectional isolation, или изоляция двунаправленности текста) изолирует от окружающего текста текстовый фрагмент, направление в кото-

ром может отличаться от направления окружающего текста (но не обязательно отличается). Этот тег можно использовать, например, когда текст на страницу встраивается из базы данных и заранее неизвестно, какое направление текста будет иметь встраиваемый блок контента.

В приведенном ниже примере показаны имена пользователей и через двоеточие выведено количество очков в конкурсе, полученное каждым пользователем. Сведения о пользователе с именем на арабском языке выведены два раза: с применением тега <bdi> и без него. Как видно, если элемент <bdi> не поддерживается в браузере (четвертая строка), то порядок вывода текста будет нарушен: двунаправленный алгоритм поместит двоеточие и число 90 рядом со словом «Пользователь», а не за именем пользователя.

```
<ul>
  <li>User <bdi>hrefs</bdi>: 60 points</li>
  <li>User <bdi>jdoe</bdi>: 80 points</li>
  <li>User <bdi>إيان</bdi>: 90 points</li>
  <li>User إيان: 90 points</li>
</ul>
```

- User hrefs: 60 points
- User jdoe: 80 points
- User إيان: 90 points
- User 90 :إيان points

Отображение фрагментов программного кода <code>, <kbd>, <samp>, <var>

Тег <code> предназначен для отображения фрагментов программного кода: имен переменных, ключевых слов, текста программ, функций и т. д.

Тег <kbd> используется для обозначения текста, который набирается на клавиатуре, или для названия клавиш.

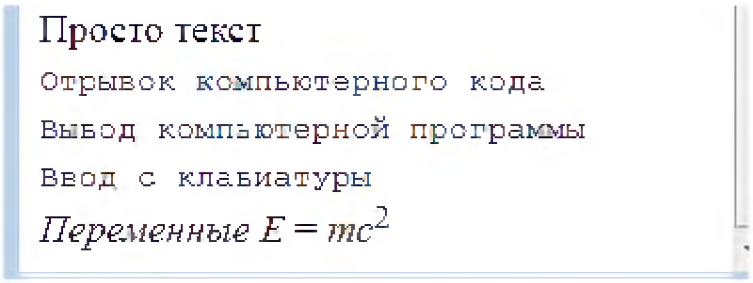
Элемент <samp> используется для отображения текста, который является результатом вывода компьютерной программы или скрипта.

Браузеры обычно отображают содержимое тегов <code>, <kbd>, <samp> как моноширинный текст.

Тег <var> используется для выделения переменных компьютерных программ. Браузеры обычно помечают текст в контейнере <var> курсивным начертанием.

```
Просто текст<br>
<code>Отрывок компьютерного кода</code><br>
<samp>Вывод компьютерной программы</samp><br>
```

```
<kbd>Ввод с клавиатуры</kbd><br>
<var>Переменные </var><var>E</var> =
<var>m</var><var>c</var><sup>2</sup>
```



Упражнение

Создайте веб-страницу в помощь школьникам, начинающим учить Pascal. В качестве контента подготовьте описание создания примера программы, находящей решение простейшего линейного уравнения $ax + b = c$. При разметке контента используйте теги для отображения фрагментов программного кода <code>, <kbd>, <samp>, <var>.

<time>

Тег <time> служит для разметки даты и времени в машиночитаемом формате.

Можно по-разному представить дату и время. Например, на веб-странице можно представить дату следующим образом:

- 20 Января 2030;
- 20-ое Января 2030;
- Янв 20 2030;
- 20/09/30;
- 09/20/30;
- 28 September 2030.

В разных странах есть традиционно принятые формы вывода, применяемые на письме. Кроме этого, существуют стандарты, принятые для представления даты и времени на компьютере. Если дата или время просто являются частью контента, то их можно не оформлять в тег <time>, но в этом случае эти даты не смогут быть легко распознаны компьютерами. Если необходимо каким-либо образом обрабатывать время/даты на странице, например автоматически захватить даты всех событий на странице и вставить их в календарь, то каждую дату или время нужно обернуть в тег <time>, который позволяет прикрепить к своему содержимому однозначное машиночитаемое время/дату.

Например:

```
<!-- Стандартная дата -->
```

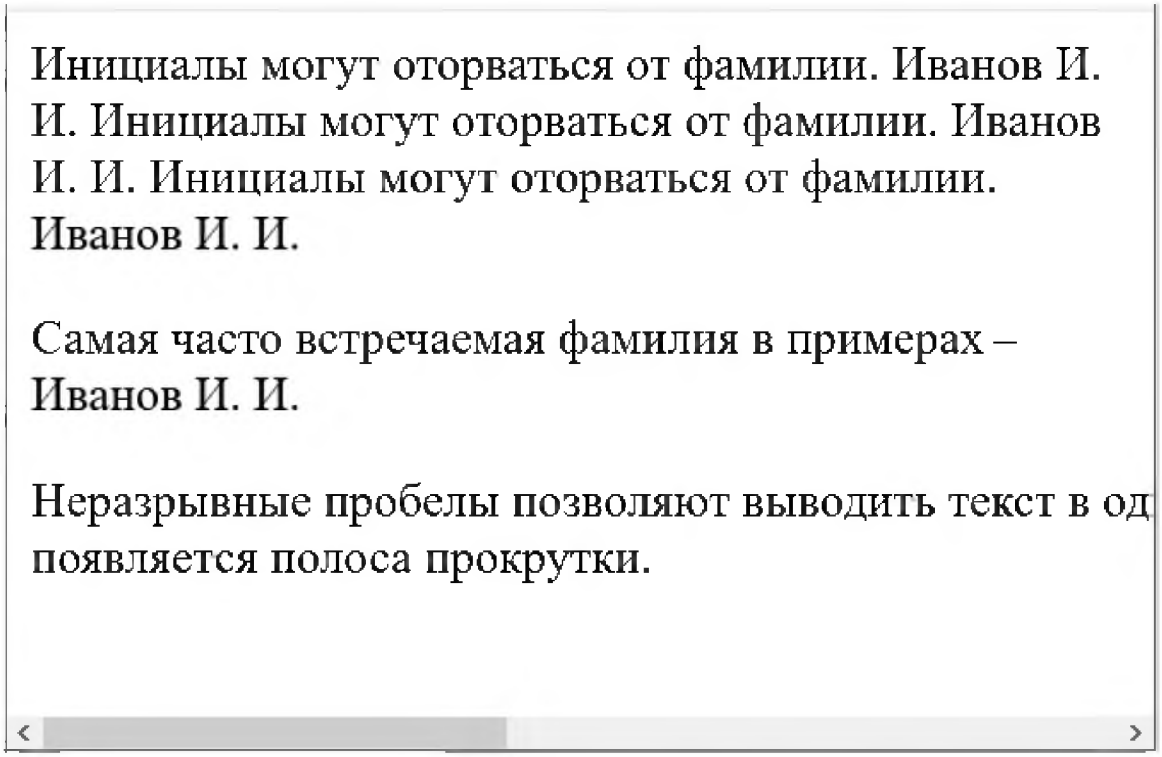
```
<time datetime="2020-01-20">20 Января 2020</time><br>
<!-- Только год и месяц -->
<time datetime="2020-01">Январь 2020</time><br>
<!-- Только месяц и день -->
<time datetime="01-20">20 Января</time><br>
<!-- Только время, часы и минуты -->
<time datetime="19:30">19:30</time><br>
<!-- Также вы можете отобразить секунды и миллисекунды! -
->
<time datetime="19:30:01.856">19:30:01.856</time><br>
<!-- Дата и время -->
<time   datetime="2020-01-20T19:30">7.30pm,   20   Января
2020</time><br>
<!-- Дата и время со смещением по часовому поясу -->
<time datetime="2020-01-20T19:30+01:00">7.30pm, 20 Января
2020, — это 8.30pm во Франции.</time><br>
<!-- Вызов номера недели -->
<time datetime="2020-W04">Четвертая неделя 2020</time>
```

20 Января 2020
Январь 2020
20 Января
19:30
19:30:01.856
7.30pm, 20 Января 2020
7.30pm, 20 Января 2020, — это 8.30pm во Франции.
Четвертая неделя 2020

Символ неразрывного пробела ( )

В языке HTML любая последовательность подряд идущих пробелов, переходов на новую строку, табуляций, пустых строк отображается в браузере как один пробел. При выводе текста, длина строки которого превышает ширину окна браузера, происходит перенос текста по словам на другую строку. Иногда это нежелательно. Например, при выводе фамилии, имени и отчества нужно, чтобы инициалы не отрывались от фамилии. Для этого используется символ неразрывного пробела *** ***. Неразрывные пробелы позволяют также выводить текст в одной строке, независимо от ширины окна браузера. Если строка не помещается в окне браузера, то появляется полоса прокрутки. Пример использования символа неразрывного пробела для того, чтобы инициалы не отрывались от фамилии, приведен ниже.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
     Инициалы могут оторваться от фамилии. Иванов И. И.
     Инициалы могут оторваться от фамилии. Иванов И. И.
     Инициалы могут оторваться от фамилии. Иванов И. И.
     <p>Самая часто встречаемая фамилия&nb
     sp;в примерах Иванов И. И.
     <!--Неразрывные пробелы позволяют выводить текст
     в одной строке. Если строка не помещается в окне
     браузера, то появляется полоса прокрутки.-->
     </p>
     <p>Неразрывные пробелы позволяют выво
     дить текст в одной строке. 
     Если строка не помещается в 
     ;окне браузера, то
     появляется полоса прокрутки.
     </p>
  </body>
</html>
```



В приведенном примере в первом абзаце текста инициалы располагаются без связи с фамилией, во второй части ни имя, ни отчество не могут быть отделены от фамилии, а третий абзац выводится одной строкой. Обратите внимание, что содержимое первого и второго абзацев подстраивается при выводе под ширину окна браузера.

Символы неразрывного пробела можно также использовать для получения эффекта «ступенчатого» текста.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
<body>
Мир<br>
     опять
<br>
         &n
bsp;цветами оброс,<br>
у мира<br>
     весенний вид.
  </body>
</html>
```

Мир
опять
цветами оброс,
у мира
весенний вид.

Специальные символы и диакритические знаки

Некоторые символы, которые иногда нужно поместить на страницу, отсутствуют на обычной клавиатуре: например, математические, технические, валютные символы или символы национальных алфавитов, которые не входят в базовую часть таблицы ASCII-кодов. Некоторые символы присутствуют на клавиатуре, но их нельзя непосредственно ввести в HTML-документ. Например, символы «меньше» и «больше» зарезервированы

для обозначения тегов. Такие символы можно вывести на странице при помощи специальных кодов, каждый из которых начинается со знака ***&***, за которым следует название символа либо его числовой код в десятичной или шестнадцатеричной системе. Завершается код символом «точка с запятой». Ниже приведены некоторые специальные символы.

Имя	**Код**	**Вид**	**Описание**
&quot;	&#34;	"	двойная кавычка
&amp;	&#38;	&	амперсанд
&lt;	&#60;	<	знак «меньше»
&gt;	&#62;	>	знак «больше»
			неразрывный пробел
&sect;	&#167;	§	параграф
&copy;	&#169;	©	знак копирайта
&laquo;	&#171;	«	левая двойная угловая скобка
&reg;	&#174;	®	знак зарегистрированной торговой марки
&deg;	&#176;	°	градус
&plusmn;	&#177;	±	плюс-минус
&acute;	&#180;	´	знак ударения
&middot;	&#183;	·	точка
&raquo;	&#187;	»	правая двойная угловая скобка
&ndash;	&#8211;	–	тире
&mdash;	&#8212;	—	длинное тире
&lsquo;	&#8216;	‘	левая одиночная кавычка
&rsquo;	&#8217;	’	правая одиночная кавычка
&sbquo;	&#8218;	‚	нижняя одиночная кавычка
&ldquo;	&#8220;	“	левая двойная кавычка
&rdquo;	&#8221;	”	правая двойная кавычка
&bdquo;	&#8222;	„	нижняя двойная кавычка
&frasl;	&#8260;	⁄	косая дробная черта
&euro;	&#8364;	€	евро
	&#8470;	№	знак номера
&trade;	&#8482;	™	знак торговой марки
&loz;	&#9674	◊	ромб
	&#9675	○	круг

Справочник по специальным символам есть, например, на сайте w3schools: https://www.w3schools.com/charsets/ref_html_entities_4.asp.

Диакритический знак является «глифом», добавляемым к письму. Диакритические знаки могут появляться как выше, так и ниже буквы, внутри письма, а также между двумя буквами. Они могут быть использованы в сочетании с буквенно-цифровыми символами, чтобы воспроизвести символ, который не присутствует в наборе символов (кодировке), используемом в странице.

Знак	Буква	Код	Результат
̀	a	à	à
́	a	á	á
̂	a	â	â
̃	a	ã	ã
̀	O	Ò	Ò
́	O	Ó	Ó
̂	O	Ô	Ô
̃	O	Õ	Õ

Упражнение

1. Сделайте веб-страницу с шуточным гороскопом на предстоящую неделю для своих одногруппников. Специальные символы для обозначения знаков зодиака можно найти на сайте https://www.w3schools.com/charsets/ref_utf_symbols.asp.
2. Продумайте, что может быть включено в футер для персонального сайта. Отобразите в футере информацию о том, кому принадлежат все права на сайт, ваш электронный адрес, телефон и т. д. Используйте специальные символы.

Карты изображений <map>, <area>

Карта изображения – это изображение с интерактивными областями. Карты изображений, или графические навигационные карты, создаются для обработки щелчка в определенных областях картинки на компьютере клиента. Для создания карт изображений предусмотрен тег <map>.

<map>

Тег <map> можно рассматривать как контейнер для тегов <area>, которые, собственно, описывают области и управляют ими. В теге <map> имеется единственный атрибут name, который является обязательным.

Атрибут usemap в теге <img> связан с атрибутом name тега <map> и создает связь между изображением и картой. Значение этого атрибута с предшествующим ему символом решетки (#) записывается в качестве значения атрибута usemap тега <img>. Внутри тега <map> перечисляются теги <area>.

<area>

Тег <area> определяет область внутри карты изображения. Имеет следующие атрибуты: alt, cords, download, href, hreflang, media, rel, shape, target, type.

href

Хранит адрес сетевого ресурса, куда переходит пользователь после щелчка по области.

alt

Содержит дополнительный текст для отображения в текстовых браузерах или браузерах с ручной загрузкой изображений. Некоторые браузеры отображают этот текст как всплывающие подсказки, которые появляются при наведении указателя мыши на контролируемую область.

shape

Определяет форму контролируемой области: rect (прямоугольник), circle (окружность) или poly (многоугольник).

coords

Содержит координаты, необходимые для описания фигуры, тип которой задан в coords. Для rect: `coords="x1,y1,x2,y2"`, где ($x1$,$y1$), ($x2$,$y2$) – координаты левого верхнего и правого нижнего угла прямоугольника соответственно. Для circle: `coords="x1,y1,R"`, где ($x1$,$y1$) – координаты центра окружности, R – радиус. Для poly: `coords="x1,y1,x2,y2,...,xn,yn"`, где ($x1$,$y1$), ($x2$,$y2$), …, (xn,yn) – координаты n точек, задающих n-угольник. Если области перекрываются, то подключается определение той области, чье определение в теге <map> стоит раньше.

download

Указывает, что целевое изображение будет загружаться, когда пользователь щелкнет по ссылке. Содержит имя загружаемого файла.

hreflang

Задает язык целевого URL.

media

Определяет, для какого устройства оптимизирован целевой URL, например для iPhone, устройств воспроизведения речи или печатных средств массовой информации.

rel

Задает связь между текущим документом и целевым URL.

target

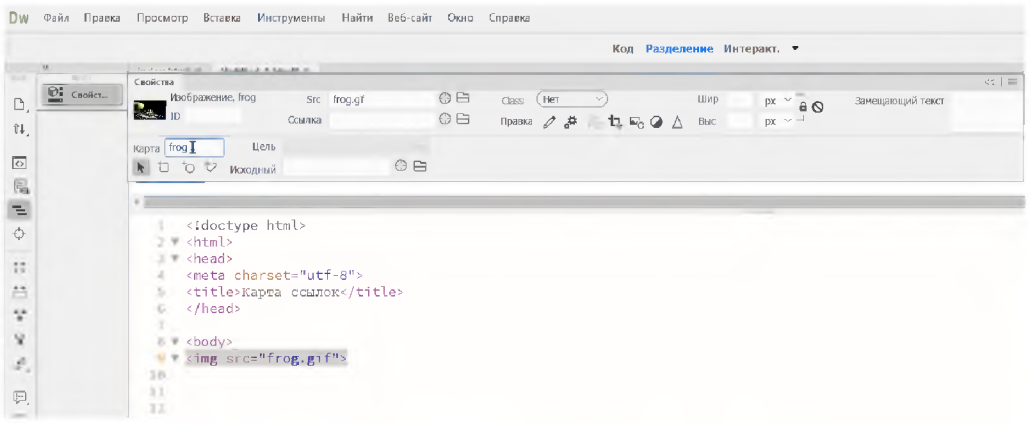
Определяет, где будет открываться целевой URL.

type

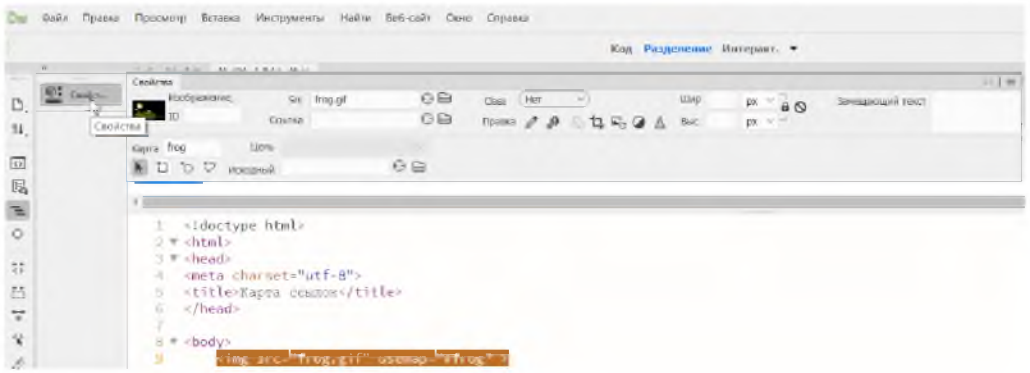
Задает медиатип целевого URL.

Рассмотрим на примере, как в Adobe Dreamweaver создать карту изображений.

Сначала вставляем картинку, используя тег <img>.



В окне документа выберите изображение. В инспекторе свойств щелкните стрелку расширения в правом нижнем углу для просмотра всех свойств. В поле «Карта» введите имя для карты ссылок. Каждая карта должна иметь уникальное имя.



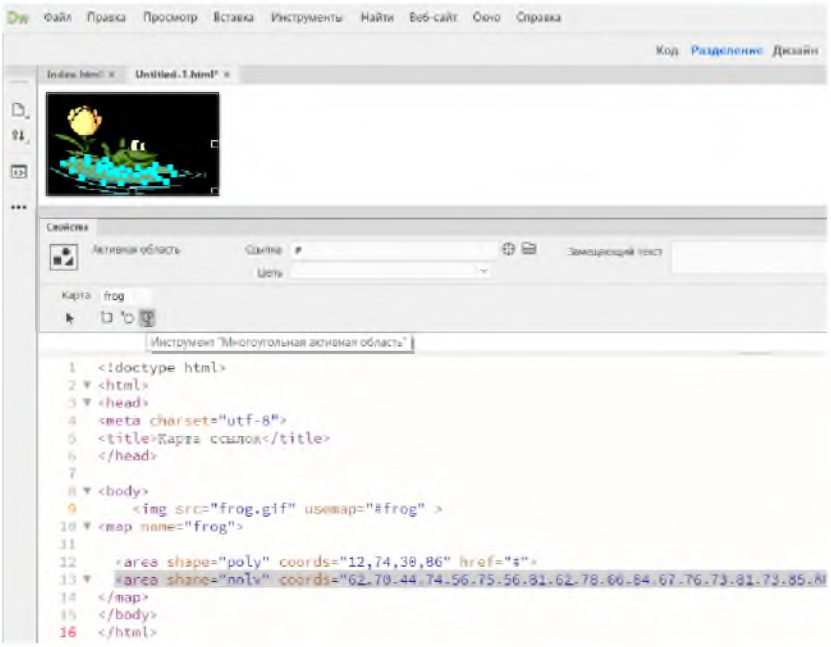
Этими действиями мы заполнили атрибут usemap.

```
<img src="frog.gif" usemap="#frog">
```

Затем указываем на наличие навигационной карты, которая будет создана при помощи тега <map>.

```
<img src="frog.gif" usemap="#frog" >
<map name="frog">
</map>
```

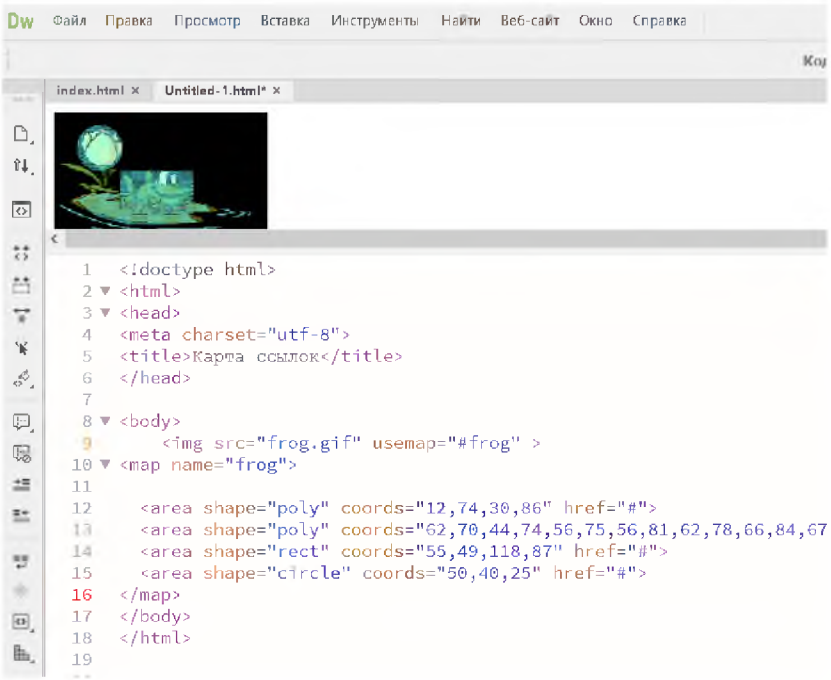
При вставке клиентской карты ссылки создается активная область, затем определяется ссылка, открывающаяся по щелчку пользователя в активной области. Выбираем инструмент «Многоугольная активная область» и точками отмечаем область.



В код добавится тег <area> с уже описанной областью и неинициализированным атрибутом href.

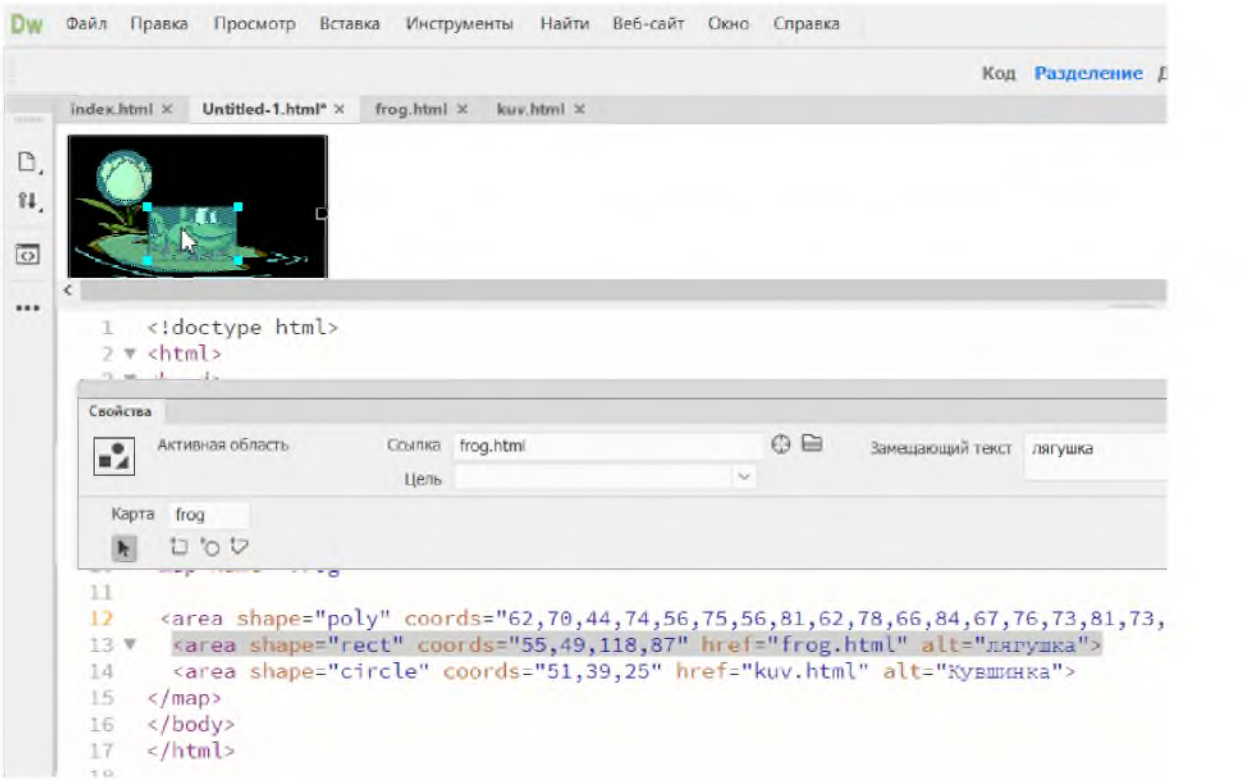
```
<area shape="poly"
coords="62,70,44,74,56,75,56,81,62,78,66,84,67,76,73,81,
73,85,80,79,82,86,87,82,94,86,96,79,102,83,110,82,108,81
,114,76,129,78,133,85,113,85,117,90,125,93,103,94,78,95,
60,94,37,89,27,82,18,74,28,70,43,68" href="#">
```

Последовательно отмечаем все области. Для демонстрации использования разных областей выделим лягушку при помощи «Прямоугольной активной области», а бутон – при помощи «Круглой активной области».



```
<img src="frog.gif" usemap="#frog" >
<map name="frog">
  <area shape="poly" coords="12,74,30,86" href="#">
  <area                                    shape="poly"
coords="62,70,44,74,56,75,56,81,62,78,66,84,67,76,73,81,
73,85,80,79,82,86,87,82,94,86,96,79,102,83,110,82,108,81
,114,76,129,78,133,85,113,85,117,90,125,93,103,94,78,95,
60,94,37,89,27,82,18,74,28,70,43,68" href="#">
  <area shape="rect" coords="55,49,118,87" href="#">
  <area shape="circle" coords="50,40,25" href="#">
</map>
```

Осталось расставить ссылки и вставить альтернативное описание. Это можно сделать в инспекторе свойств, а можно прописать в коде самостоятельно.




```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Карта ссылок</title>
</head>

<body>
     <img src="frog.gif" usemap="#frog" >
<map name="frog">
```

```
 <area shape="poly"
coords="63,70,45,74,57,75,57,81,63,78,67,84,68,76,74,81,
74,85,81,79,83,86,88,82,95,86,97,79,103,83,111,82,109,81
,115,76,130,78,134,85,114,85,118,90,126,93,104,94,79,95,
61,94,38,89,28,82,19,74,29,70,44,68" href="LIst.html"
alt="Лист кувшинки">
 <area shape="rect" coords="55,49,118,87"
href="frog.html" alt="лягушка">
 <area shape="circle" coords="51,39,25" href="kuv.html"
alt="Кувшинка">
</map>
</body>
</html>
```

Если области перекрываются, то подключается определение той области, чье определение в теге <map> стоит раньше.

Упражнение

Поместите на страницу графическую навигационную карту (на стороне клиента). Например, фото вашей группы, по щелчку на персоне должен осуществляться переход на персональную страничку товарища, в группу «ВКонтакте» или др.

Списки

В HTML списки используются для группировки связанных между собой фрагментов информации. Существует три вида списков: упорядоченные, неупорядоченные и списки определений. Каждый список представляет собой контейнер, внутри которого располагаются элементы списка или пары «термин – определение» и его расшифровка.

Элементы списка ведут себя как блочные элементы: располагаются друг под другом и занимают всю ширину родительского блока. При этом каждый элемент списка имеет отступ слева.

Неупорядоченный маркированный список <ul>,

Неупорядоченные списки используются для элементов, для которых порядок не имеет значения. Для создания маркированных или ненумерованных списков служит тег <ul>. Каждый элемент списка должен заключаться в тег <li>. Внутри <ul> могут быть только <li>, другие теги или текст недопустимы. Элементы списка по умолчанию помечаются маленькими черными кружками, или буллитами.

```
<!doctype html>
```

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Маркированные списки</title>
  </head>
  <body>
    <h1>Список героев сказки А. С. Пушкина о золотой
рыбке</h1>
    <ul>
      <li>Старик-рыбак;</li>
      <li>Ворчунья-старуха;</li>
      <li>Золотая рыбка.</li>
    </ul>
  </body>
</html>
```

Список героев сказки А. С. Пушкина о золотой рыбке

- Старик-рыбак;
- Ворчунья-старуха;
- Золотая рыбка.

В большинстве браузеров стиль для списков по умолчанию следующий:

```
ul {
    display: block;
    list-style-type: disc;
    margin-top: 1em;
    margin-bottom: 1 em;
    margin-left: 0;
    margin-right: 0;
    padding-left: 40px;
}
```

В HTML 4.01 тег <ul> имел атрибут type, который отвечал за вид маркеров списка, он мог принимать значения disc, square, circle, none. Этот атрибут уже не поддерживается в HTML5. Для определения стиля маркера элемента списка в HTML5 должно использоваться только свойство list-style-type таблицы стилей. Это свойство имеет несколько предопределенных значений: disc, square, circle, none.

```
<ul style="list-style-type:circle">
  <li>Старик-рыбак;</li>
  <li>Ворчунья-старуха;</li>
  <li>Золотая рыбка.</li>
</ul>
```

```
<ul style="list-style-type:square">
  <li>Старик-рыбак;</li>
  <li>Ворчунья-старуха;</li>
  <li>Золотая рыбка.</li>
</ul>
```

- Старик-рыбак;
- Ворчунья-старуха;
- Золотая рыбка.

- Старик-рыбак;
- Ворчунья-старуха;
- Золотая рыбка.

Если в качестве параметра list-style-image указать адрес иконки, то можно вставить свой маркер в список.

```
<ul style="list-style-image:url(images/BD21301_.GIF);" >
  <li>Старик-рыбак;</li>
  <li>Ворчунья-старуха;</li>
  <li>Золотая рыбка.</li>
</ul>
```

☑ Старик-рыбак;
☑ Ворчунья-старуха;
☑ Золотая рыбка.

Упорядоченный список <ol>,

Если в списках порядок элементов имеет значение, то используются упорядоченные списки. Тег <ol> служит для создания упорядоченных списков. Каждый элемент списка записывается в теге <li>. Браузер нумерует элементы по порядку автоматически, и если удалить один или несколько элементов такого списка, то остальные номера будут автоматически перенумерованы.

Так выглядит упорядоченный список по умолчанию.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Маркированные списки</title>
  </head>
  <body>
    <h1>Купить в магазине:</h1>
    <ol>
```

```
      <li>молоко;</li>
      <li>хлеб;</li>
      <li value="5">сырки;</li>
      <li>сосиски.</li>
    </ol>
  </body>
</html>
```

Купить в магазине:

1. молоко;
2. хлеб;
3. сырки;
4. сосиски.

У тега <ol> есть несколько атрибутов.

Атрибут type задает вид маркера для использования в списке (в виде букв или цифр). Возможные значения:

- 1 – значение по умолчанию, десятичная нумерация;
- A – нумерация списка в алфавитном порядке, заглавные буквы (A, B, C, D);
- a – нумерация списка в алфавитном порядке, строчные буквы (a, b, c, d);
- I – нумерация римскими заглавными цифрами (I, II, III, IV);
- i – нумерация римскими строчными цифрами (i, ii, iii, iv);

Например, при `<ol type="I">` получим следующий вид списка:

I. молоко;
II. хлеб;
III. сырки;
IV. сосиски.

Атрибут start задает начальное значение, от которого пойдет отсчет нумерации. Например, `<ol start="3">` первому пункту присвоит порядковый номер «3».

3. молоко;
4. хлеб;
5. сырки;
6. сосиски.

Также можно одновременно задавать тип нумерации, например `<ol type="A" start="9">`.

I. молоко;
J. хлеб;
K. сырки;
L. сосиски.

Атрибут reversed имеет булевское значение, указывает на прямой или обратный порядок списка. Так выглядит алфавитный список, в котором сменили порядок следования, тип маркера списка – `<ol type="a" reversed>`:

d. молоко;
c. хлеб;
b. сырки;
a. сосиски.

У тега <li> есть атрибут value, который позволяет изменить номер по умолчанию для выбранного элемента списка. Например, если для первого пункта списка задать <li value="3">, то остальная нумерация будет пересчитана относительно нового значения.

```
<ol>
  <li value="3">молоко;</li>
  <li>хлеб;</li>
  <li value="5">сырки;</li>
  <li>сосиски.</li>
</ol>
```

3. молоко;
4. хлеб;
5. сырки;
6. сосиски.

В большинстве браузеров стиль для списков по умолчанию следующий:

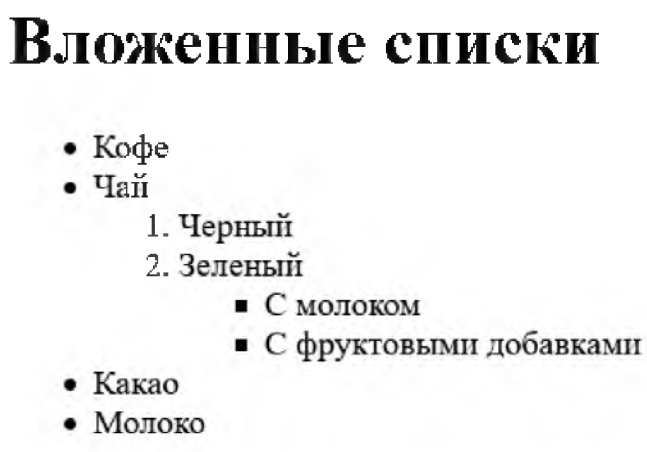
```
ol {
    display: block;
    list-style-type: decimal;
    margin-top: 1em;
    margin-bottom: 1em;
    margin-left: 0;
    margin-right: 0;
```

```
    padding-left: 40px;
}
```

Вложенные списки

Списки можно вкладывать внутрь списков, независимо от их типов. Например, маркированный список можно вложить в нумерованный и наоборот. При этом теги списков <ul> или <ol> вкладываются внутрь элемента списка <li>, но не в <ul> или <ol> непосредственно.

```
<h1>Вложенные списки</h1>
<ul>
  <li>Кофе</li>
  <li>Чай
  <ol>
     <li>Чёрный</li>
        <li>Зелёный
         <ul style="list-style-type:square">
             <li>С молоком</li>
                <li>С фруктовыми добавками</li>
            </ul>
        </li>
    </ol>
  </li>
  <li>Какао</li>
  <li>Молоко</li>
</ul>
```



Как видно, многоуровневый список отображает элементы подсписков на разных уровнях с различными отступами соответственно порядку вложенности.

Многоуровневый список со сложной составной нумерацией

Можно создать многоуровневый список со сложной составной нумерацией, но только средствами CSS. Пусть имеется контент, например оглавление, имеющее следующую структуру:

```
<ol>
  <li>Глава</li>
  <li>Глава
    <ol>
      <li>Параграф</li>
      <li>Параграф
        <ol>
          <li>Пункт</li>
          <li>Пункт</li>
        </ol>
      </li>
    </ol>
  </li>
  <li>Глава</li>
  <li>Глава</li>
</ol>
```

Такая разметка по умолчанию создаст для каждого вложенного списка новую нумерацию, начинающуюся с единицы.

1. Глава
2. Глава
    1. Параграф
    2. Параграф
        1. Пункт
        2. Пункт
3. Глава
4. Глава

Чтобы сделать вложенную нумерацию, нужно использовать следующие свойства:

counter-reset – сбрасывает один или несколько счетчиков, задавая значение для сброса;

counter-increment – задает значение приращения счетчика, т. е. с каким шагом будет нумероваться каждый последующий пункт;

content – содержит генерируемое содержимое, в данном случае отвечает за вывод номера перед каждым пунктом списка.

```
<style>
ol {
   /* убираем стандартную нумерацию */
```

```
   list-style: none;
/* Идентифицируем счетчик и даем ему имя li-counter.
Значение счетчика не указано - по умолчанию оно равно 0
*/
   counter-reset: li-counter;
}
li:before {
/* Определяем элемент, который будет нумероваться, — li.
Псевдоэлемент before указывает, что содержимое,
вставляемое при помощи свойства content, будет
располагаться перед пунктами списка. Здесь же
устанавливается значение приращения счетчика (по
умолчанию равно 1). */
   counter-increment: li-counter;
/* С помощью свойства content выводится номер пункта
списка. counters() означает, что генерируемый текст
представляет собой значения всех счетчиков с таким именем.
Точка в кавычках добавляет разделяющую точку между
цифрами, а точка с пробелом добавляется перед содержимым
каждого пункта списка */
   content: counters(li-counter, ".") ". ";
}
</style>
```

Список определений <dl>, <dt>, <dd>

Тег <dl> (от description list) используется для создания списка определений. Текст каждого определения (термин) должен быть заключен в тег <dt> (description term), а его расшифровка или описание – в тег <dd> (description definition). Разрешено давать одному элементу несколько описаний.

Синтаксис следующий:

```
<dl>
  <dt>Термин 1</dt>
  <dd>Определение термина 1</dd>
  <dt>Термин 2</dt>
  <dd>Определение термина 2</dd>
</dl>
```

Например:

```
<dl>
  <dt>HTML</dt>
  <dd>- язык гипертекстовой разметки</dd>
  <dt>CSS</dt>
  <dd>- каскадные таблицы стилей</dd>
</dl>
```

HTML
: – язык гипертекстовой разметки

CSS
: – каскадные таблицы стилей

При семантической верстке список определений может использоваться как модель «вопрос – ответ» для создания страниц часто задаваемых вопросов.

Упражнения

1. Создайте неупорядоченный список, маркированный квадратиками.
2. Создайте упорядоченный список, нумерованный от 20 до 15.
3. Создайте список. В качестве маркеров вставьте свою иконку.
4. Закон Мерфи – шутливый философский принцип, иностранный аналог русского «закона подлости». Следствия из закона Мерфи были впервые опубликованы в книге Артура Блоха «Закон Мерфи». Создайте новый HTML-документ и разместите в нем фрагмент текста, представленный ниже. При верстке самостоятельно продумайте и организуйте список определений с вложенными в него маркированным и нумерованным списками.

Мерфология

Закон Мерфи

Если какая-нибудь неприятность может произойти, она случится.

Следствия

1. Все не так легко, как кажется.
2. Всякая работа требует больше времени, чем вы думаете.
3. Из всех неприятностей произойдет именно та, ущерб от которой больше.
4. Если четыре причины возможных неприятностей заранее устранимы, то всегда найдется пятая.
5. Предоставленные самим себе события имеют тенденцию развиваться от плохого к худшему.
6. Как только вы принимаетесь делать какую-то работу, находится другая, которую надо сделать еще раньше.
7. Всякое решение плодит новые проблемы.

Комментарий Каллагана к закону Мерфи

Мерфи был оптимистом!

Наблюдение Ренара

Есть моменты, когда все удается. Не ужасайтесь, это пройдет.

Законы Чизхолма

1. Все, что может испортиться, – портится.

   Следствие:

   - Все, что не может испортиться, – портится тоже.
2. Когда дела идут хорошо, что-то должно случиться в самом ближайшем будущем.

   Следствия:

   - Когда дела идут хуже некуда, в самом ближайшем будущем они пойдут еще хуже.
   - Если вам кажется, что ситуация улучшается, значит вы чего-то не заметили.
3. Любые предложения люди понимают иначе, чем тот, кто их вносит.

   Следствия:

   - Даже если ваше объяснение настолько ясно, что исключает всякое ложное толкование, все равно найдется человек, который поймет вас неправильно.
   - Если вы уверены, что ваш поступок встретит всеобщее одобрение, кому-то он обязательно не понравится.

Закон Скотта

Неважно, что что-то идет неправильно. Возможно, это хорошо выглядит.

Законы Финэйгла

1. Если эксперимент удался, что-то здесь не так...
2. В любом наборе исходных данных самая надежная величина, не требующая никакой проверки, является ошибочной.
3. Если уж работа проваливается, то всякая попытка ее спасти только ухудшит дело.

Теорема Гинзберга

Выиграть нельзя. Остаться при своих нельзя. Нельзя даже выйти из игры.

Комментарии Эрмана

- Перед тем, как улучшиться, ситуация ухудшается.
- Кто сказал, что она улучшится?

Закон термодинамики Мерфи

Под давлением все ухудшается.

Второй закон термодинамики Эверитта

Неразбериха в обществе постоянно возрастает. Только очень упорным трудом можно ее несколько уменьшить. Однако сама эта попытка приведет к росту совокупной неразберихи.

Законы Паддера

1. Все, что начинается хорошо, кончается плохо.
2. Все, что начинается плохо, кончается еще хуже.

Теорема Стокмайера

Если кажется, что работу сделать легко, то будет непременно трудно. Если на вид она трудна, то выполнить ее абсолютно невозможно.

Закон создания динамики систем Зимерги

Если вы уже открыли банку с червями, то единственный способ их снова запечатать – это воспользоваться банкой большего размера.

Закон бесконечного падения Эмерсона

Под всякой бездной раскрывается другая, еще более глубокая.

Закон звездного часа Марка Твена

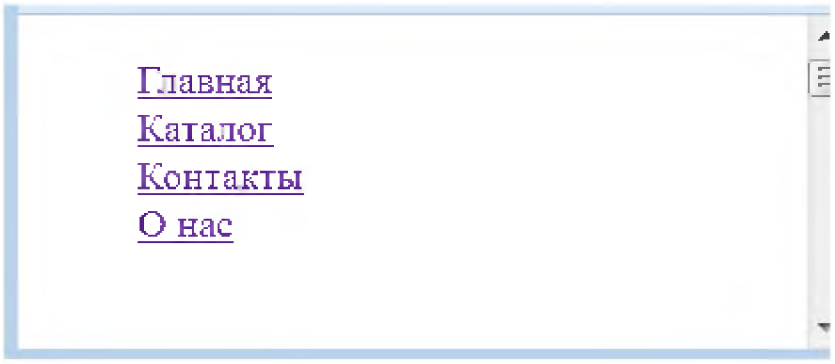
Один раз в жизни фортуна стучится в дверь каждого человека, но во многих случаях человек в это время сидит в соседнем кабачке и не слышит ее стука.

Использование списков для создания меню и навигационных панелей

Есть несколько способов создания различных навигационных панелей и меню на сайте. Самый простейший из них – это когда группу ссылок оборачивают в маркированный список. При этом значок маркера,

как правило, убирают, определяя для тега <ul> свойство стиля list-style-type:none.

```
<ul style="list-style-type:none">
  <li><a href="#home">Главная</a></li>
  <li><a href="#catalog">Каталог</a></li>
  <li><a href="#contact">Контакты</a></li>
  <li><a href="#about">О нас</a></li>
</ul>
```



К маркированному списку

```
<ul>
  <li><a href="#home">Главная</a></li>
  <li><a href="#catalog">Каталог</a></li>
  <li><a href="#contact">Контакты</a></li>
  <li><a href="#about">О нас</a></li>
</ul>
```

применим, например, следующие стили:

- для неупорядоченного списка – не выводить маркеры;
- для элементов списка – отображать каждый элемент как блочный, обтекание блоков по правому краю, внешний отступ слева от элемента – 5 пикселей;
- все ссылки, которые являются дочерними для элементов списка, – не подчеркивать.

```
<style>
ul
{
    list-style-type:none;
}
 li {
    display: block;
    float: left;
    margin-left: 5px;
}
li a {
```

```
    text-decoration: none;
}
</style>
```

Таким образом можно получить горизонтальное меню. Оно выглядит очень непритязательно.

Используя стили CSS, можно сделать горизонтальное меню зрительно более привлекательным. Для этого надо добавить следующие стилевые свойства:

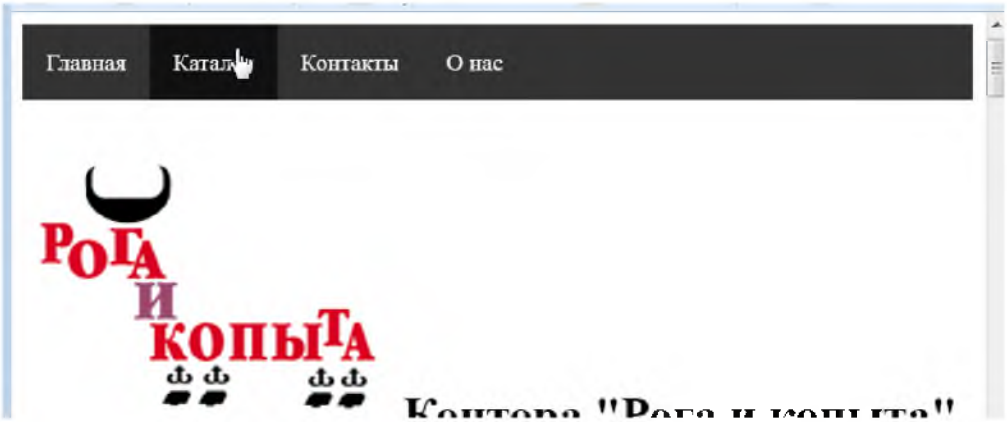
- для неупорядоченного списка – не выводить маркеры, установить внешний и внутренний отступы по 0 пикселей, если список не помещается и выходит за область заданных размеров, то отображать только часть элемента, которая попала в область, остальное скрывать, цвет фона темно-серый;
- для элементов списка – оставить только обтекание блоков по правому краю;
- для всех ссылок, которые являются дочерними для элементов списка, – отображать каждую ссылку как блочный элемент, цвет текста белый, выравнивание текста по ширине блока, внутренние отступы – 16 пикселей;
- при наведении курсора мыши на ссылку цвет фона изменить на более темный.

```
<style>
ul
{
    list-style-type:none;
    padding: 0;
    margin: 0;
    overflow: hidden;
    background-color: #333333;
}
 li {
    float: left;
}
li a {
    display: block;
    color: white;
    text-align: center;
```

```
    padding: 16px;
    text-decoration: none;
}
li a:hover {
    background-color: #111111;
}
</style>
```

Применим данные стили к HTML-коду:

```
<ul class="mnu">
  <li><a href="#home">Главная</a></li>
  <li><a href="#catalog">Каталог</a></li>
  <li><a href="#contact">Контакты</a></li>
  <li><a href="#about">О нас</a></li>
</ul>
```



Упражнение

Добавьте на страницу горизонтальное меню.

Таблицы. <table>, <tr>, <th>, <td>

Для вставки таблиц в HTML-документ предназначен тег <table>. Перед таблицей и после нее происходит переход на следующую строку. Тег <table> включает в себя один или несколько тегов <tr>. Тег <tr> (table row) определяет строку таблицы и служит контейнером для тегов <td> или <th>. Тег <td> (table data) определяет отдельную ячейку в таблице. Тег <th> (table header) определяет ячейку заголовка таблицы и визуально отличается от тега <td> только форматированием. Обычно содержимое заголовка выделяется полужирным шрифтом и выравнивается по центру ячейки. Ячейки таблиц могут содержать любые элементы, такие как заголовки, списки, текст, изображения, элементы форм, а также другие таблицы. Более сложная таблица HTML может также включать элементы

<caption>, <col>, <colgroup>, <thead>, <tfoot> и <tbody>. Каждой таблице можно добавить связанный с ней заголовок, расположив его перед таблицей или после нее.

Размеры ячеек таблицы автоматически масштабируются в соответствии с хранящейся с ней информацией.

```
<table>
  <tr>
    <th> ФИО </th>
    <th> алгебра </th>
    <th> геометрия </th>
    <th> мат. анализ </th>
    <th> программирование </th>
  </tr>
  <tr>
    <td> Иванов И. И.</td>
    <td> 8 </td>
    <td> 7 </td>
    <td> 9 </td>
    <td> 8 </td>
  </tr>
  <tr>
    <td> Петров П. П.</td>
    <td> 5 </td>
    <td> 9 </td>
    <td> 6 </td>
    <td> 5 </td>
  </tr>
  <tr>
    <td> Сидоров С. С.</td>
    <td> не явился </td>
    <td> 4 </td>
    <td> 4 </td>
    <td> 7 </td>
  </tr>
</table>
```

ФИО	алгебра	геометрия	мат. анализ	программирование
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Как видно из примера, рамка таблицы автоматически не выводится, ширина столбца определяется по максимальной ширине его ячеек, текст в ячейках выравнивается по левому краю, а в заголовках столбцов – центрируется. Ширина таблицы определяется исходя из ее содержимого. Сама таблица прижимается к левому краю окна.

Чтобы отобразить рамку таблицы, следует воспользоваться свойствами CSS. В свойстве border необходимо указать тип границы, ее цвет и толщину. Граница таблицы представляет собой рамку вокруг всей таблицы. Кроме этого, граница есть у каждой ячейки таблицы, и по умолчанию ячейки находятся на некотором расстоянии друг от друга. Также можно определить цвет фона как для всей таблицы, так и для ее ячеек.

```
<style>
  table{
        border: 2px solid black;
        background: cyan;
        width: 90%;
  }
  th, td{
        border: 1px solid black;
        background: yellow;
  }
</style>
```

ФИО	алгебра	геометрия	мат. анализ	программирование
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Более привычно, когда между ячейками таблицы нет расстояния. Если необходимо объединить границы ячеек в одну границу, то можно воспользоваться свойством CSS border-collapse.

```
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
```

ФИО	алгебра	геометрия	мат. анализ	программирование
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Чтобы увеличить пространство между ячейками таблицы, можно использовать свойство border-spacing CSS, например:

```
table {
    border-spacing: 5px;
}
```

Элементы форматирования могут быть применены не только ко всей таблице целиком, но и к отдельным ее строкам. Управлять форматированием строки можно, изменяя свойства стиля для тега <tr>. На примере все той же таблицы выделим цветом каждую четную строку (строки нумеруются от нуля). Для выбора четных строк воспользуемся селектором tr:nth-child(even). Саму таблицу сделаем «резиновой», т. е. выровняем ее ширину по ширине окна: width: 100 %, «схлопнем» расстояние между ячейками и для текста в ячейках установим отступ в 8 пикселей.

```
<style>
table {
    font-family: arial, sans-serif;
    border-collapse: collapse;
    width: 100%;
}

td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
}

tr:nth-child(even) {
    background-color: #dddddd;
}
</style>
```

ФИО	алгебра	геометрия	мат. анализ	программирование
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Атрибуты colspan и rowspan

Теги <tr> и <th> имеют атрибуты colspan и rowspan.

Атрибут colspan определяет количество столбцов, охваченных текущей ячейкой. Значение по умолчанию равно единице. Значение, равное

нулю, означает, что ячейка охватывает все столбцы, начиная с текущего и заканчивая последним столбцом группы, в которой ячейка определена. Только Firefox поддерживает значение 0 для colspan.

Атрибут rowspan определяет количество строк, охваченных текущей ячейкой. Значение по умолчанию равно единице. Значение, равное нулю, означает, что ячейка охватывает все ряды раздела таблицы, начиная с текущего и заканчивая последним рядом раздела (<thead>, <tbody> или <tfoot>), в котором ячейка определена. Значение rowspan, равное 0, поддерживает только Firefox.

Для таблицы:

```
<style>
table {
    border: 5px solid #dddddd;
    font-family: arial, sans-serif;
    border-collapse: collapse;
    width: 100%;
}
td, th {
    border: 1px solid #dddddd;
    text-align: center;
    padding: 8px;
}
.mohth{
    background-color: #ffff33
}
#sum {
    background-color: #ff99ff
}
td:last-child {
    background-color: #33ffff
}
tr:first-child {
    background-color: #cc9933;
}
</style>
```

№ п/п	название товара	квартал			итого
		январь	февраль	март	
1	молоко	1000	1206	500	2706
2	творог	0	5000	0	5000
3	кефир	200	200	400	800

Атрибут scope тега <th>

Атрибут scope для ячейки заголовка содержит информацию о том, является ли ячейка заголовком для столбца, строки или группы столбцов или строк. Может принимать значения col, row, colgroup или rowgroup. Атрибут scope не имеет визуального эффекта в обычных веб-браузерах, но может использоваться программами чтения с экрана.

Пример применения scope:

```
<table>
  <caption>Продано на сумму в сентябре 2020</caption>
  <thead>
    <tr>
      <td> </td>
      <td> </td>
      <th   colspan="3"   scope="colgroup">Кисломолочные
продукты</th>
      <th colspan="2" scope="colgroup">Сыры</th>
    </tr>
    <tr>
      <td> </td>
      <td> </td>
      <th scope="col">Творог</th>
      <th scope="col">Сметана</th>
      <th scope="col">Кефир</th>
      <th scope="col">Твердые</th>
      <th scope="col">Плавленые</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th      rowspan="3"      scope="rowgroup">Минская
область</th>
      <th scope="row">«Агрокомбинат Снов» СПК</th>
      <td>5670</td>
      <td>2200</td>
      <td>4380</td>
      <td>0</td>
      <td>0</td>
     </tr>
    <tr>
      <th scope="row">«Здравушка-милк»  ОАО </th>
      <td>4608</td>
```

```
      <td>1890</td>
      <td>5000</td>
      <td>61000</td>
      <td>1500</td>
      </tr>
    <tr>
      <th scope="row">«Молодечненский  молочный комбинат»
ОАО</th>
      <td>5100</td>
      <td>2700</td>
      <td>3800</td>
      <td>6900</td>
      <td>2800</td>
    </tr>
    <tr>
      <th    rowspan="2"    scope="rowgroup">Гродненская
область</th>
      <th scope="row">«Молочный Мир»  ОАО</th>
      <td>8900</td>
      <td>14220</td>
      <td>6900</td>
      <td>85907</td>
      <td>3800</td>
      </tr>
    <tr>
      <th     scope="row">«Лидский     молочно-консервный
комбинат»  ОАО</th>
      <td>8000</td>
      <td>1200</td>
      <td>4300</td>
      <td>0</td>
      <td>1900</td>
    </tr>
  </tbody>
</table>
```

Продано на сумму в сентябре 2020

		Кисломолочные продукты			Сыры	
		Творог	Сметана	Кефир	Твердые	Плавленые
Минская область	«Агрокомбинат Снов» СПК	5670	2200	4380	0	0
	«Здравушка-милк» ОАО	4608	1890	5000	61000	1500
	«Молодечненский молочный комбинат» ОАО	5100	2700	3800	6900	2800
Гродненская область	«Молочный Мир» ОАО	8900	14220	6900	85907	3800
	«Лидский молочно-консервный комбинат» ОАО	8000	1200	4300	0	1900

Атрибут headers тегов <th> и <td>

Атрибут headers указывает одну заголовочную ячейку или определяет список заголовочных ячеек, предоставляющих заголовочную информацию для текущей ячейки данных. Значением этого атрибута является список разделенных пробелами имен ячеек. Эти ячейки должны быть именованы путем установки их атрибутов id. Атрибут не имеет визуального эффекта в обычных веб-браузерах, но используется считывателями экрана и может быть задействован в JavaScript.

Применение атрибута headers является альтернативой применению атрибута scope. Используя id и headers, можно задать соответствие между заголовками и ячейками: нужно установить уникальный id для каждого <th>-элемента и атрибут headers для каждого <td>-элемента. Атрибут headers должен содержать список всех id, разделенных пробелами, ко всем тегам <th>, которые являются заголовками для этой ячейки.

Пример применения headers (таблица взята из примера применения scope):

```
<table>
  <caption>Продано на сумму в сентябре 2020</caption>
  <thead>
    <tr>
      <td> </td>
      <td> </td>
      <th  colspan="3"  id="fermentedmilk">Кисломолочные
продукты</th>
      <th colspan="2" id="cheese">Сыры</th>
    </tr>
    <tr>
      <td> </td>
      <td> </td>
      <th id="tworog">Творог</th>
      <th id="smetana">Сметана</th>
      <th id="kefir">Кефир</th>
      <th id="twerd">Твердые</th>
      <th id="plaw">Плавленые</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th rowspan="3" id="minscobl">Минская область</th>
      <th id="snow">«Агрокомбинат Снов» СПК</th>
      <td   headers="fermentedmilk   tworog   minscobl
snow">5670</td>
```

```
      <td   headers="fermentedmilk   smetana   minscobl
snow">2200</td>
      <td    headers="fermentedmilk    kefir    minscobl
snow">4380</td>
      <td headers="cheese twerd minscobl snow">0</td>
      <td headers="cheese plaw minscobl snow">0</td>
    </tr>
    <tr>
      <th id="zdraw">«Здравушка-милк»  ОАО </th>
      <td   headers="fermentedmilk   tworog   minscobl
zdraw">4608</td>
      <td   headers="fermentedmilk   smetana   minscobl
zdraw">1890</td>
      <td    headers="fermentedmilk    kefir    minscobl
zdraw">5000</td>
      <td      headers="cheese      twerd      minscobl
zdraw">61000</td>
      <td headers="cheese plaw minscobl zdraw">1500</td>
    </tr>
    <tr>
      <th ii="molod">«Молодечненский  молочный комбинат»
ОАО</th>
      <td   headers="fermentedmilk   tworog   minscobl
molod">5100</td>
      <td   headers="fermentedmilk   smetana   minscobl
molod">2700</td>
      <td    headers="fermentedmilk    kefir    minscobl
molod">3800</td>
      <td headers="cheese twerd minscobl molod">6900</td>
      <td headers="cheese plaw minscobl molod">2800</td>
    </tr>
    <tr>
      <th     rowspan="2"     id="grodnoobl">Гродненская
область</th>
      <th id="milkmir">«Молочный Мир»  ОАО</th>
      <td   headers="fermentedmilk   tworog   grodnoobl
milkmir">8900</td>
      <td   headers="fermentedmilk   smetana   grodnoobl
milkmir">14220</td>
      <td   headers="fermentedmilk   kefir   grodnoobl
milkmir">6900</td>
      <td      headers="cheese      twerd      grodnoobl
milkmir">85907</td>
```

```
      <td      headers="cheese      plaw      grodnoobl
milkmir">3800</td>
    </tr>
    <tr>
      <th     id="milida">«Лидский     молочно-консервный
комбинат»  ОАО</th>
      <td    headers="fermentedmilk    tworog    grodnoobl
milida">8000</td>
      <td   headers="fermentedmilk    smetana    grodnoobl
milida">1200</td>
      <td    headers="fermentedmilk    kefir    grodnoobl
milida">4300</td>
      <td headers="cheese twerd grodnoobl milida">0</td>
      <td      headers="cheese      plaw      grodnoobl
milida">1900</td>
    </tr>
 </tbody>
</table>
```

Продано на сумму в сентябре 2020

		Кисломолочные продукты			Сыры	
		Творог	Сметана	Кефир	Твердые	Плавленые
Минская область	«Агрокомбинат Снов» СПК	5670	2200	4380	0	0
	«Здравушка-милк» ОАО	4608	1890	5000	61000	1500
	«Молодечненский молочный комбинат» ОАО	5100	2700	3800	6900	2800
Гродненская область	«Молочный Мир» ОАО	8900	14220	6900	85907	3800
	«Лидский молочно-консервный комбинат» ОАО	8000	1200	4300	0	1900

Оформление заголовка таблицы <caption>

Тег <caption> предназначен для создания заголовка таблицы, который представляет собой текст, по умолчанию отображаемый перед таблицей и описывающий ее содержание. Тег <caption> может размещаться только внутри контейнера <table>, сразу за тегом <table>. В таблице может быть только один заголовок <caption>. По умолчанию заголовок таблицы будет размещен над таблицей и выровнен по центру. Если необходимо переместить заголовок снизу таблицы или выровнять его иным образом, то можно воспользоваться свойствами CSS. Для таблицы:

```
<table>
  <caption>Таблица 1. Итоги сессии</caption>
  <tr>
    <th> ФИО </th>
    <th> алгебра </th>
    <th> геометрия </th>
    <th> мат. анализ </th>
    <th> программирование </th>
```

```
  </tr>
  <tr>
    <td> Иванов И. И.</td>
    <td> 8 </td>

    <td> 9 </td>
    <td> 8 </td>
  </tr>
  <tr>
    <td> Петров П. П.</td>
    <td> 5 </td>
    <td> 9 </td>
    <td> 6 </td>
    <td> 5 </td>
  </tr>
  <tr>
    <td> Сидоров С. С.</td>
    <td> не явился </td>
    <td> 4 </td>
    <td> 4 </td>
    <td> 7 </td>
  </tr>
</table>
К заголовку применим следующие стили.
<style>
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
caption {
    caption-side: bottom;
    text-align: right;
}
</style>
```

ФИО	алгебра	геометрия	мат. анализ	программирование
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Таблица 1. Итоги сессии

Создание групп строк в таблице <thead>, <tfoot>, <tbody>.

Иногда для лучшего восприятия информации желательно выделять группы строк. Строки таблицы могут быть сгруппированы в «шапку», «итоги» и «тело» с помощью элементов <thead>, <tfoot>, <tbody> соответственно.

Браузеры могут использовать эти элементы для включения прокрутки тела таблицы независимо от верхнего и нижнего колонтитула. Кроме того, при печати большой таблицы, которая охватывает несколько страниц, эти элементы могут включать верхний и нижний колонтитулы таблицы вверху и внизу каждой страницы.

Теги <thead>, <tfoot> и <tbody> не влияют на макет таблицы по умолчанию. Можно использовать стили CSS для форматирования этих тегов.

Допустимо использовать не более чем по одному элементу <thead> и <tfoot> в пределах одной таблицы. Допускается применять несколько тегов <tbody> внутри контейнера <table>. Секции могут располагаться в любом порядке в контейнере <table>.

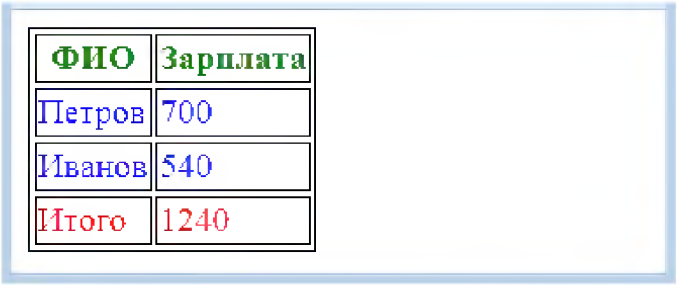
Секции <thead>, <tfoot>, <tbody> необязательны, т. е. <tr> могут быть непосредственно вложены в <table>. Однако присутствие этих тегов желательно, если в таблице очевидно присутствуют шапка (header) и итоги (footer).

```
<style>
     thead {color:green;}
     tbody {color:blue;}
     tfoot {color:red;}

     table, th, td {
         border: 1px solid black;
     }
</style>
```

Применим эти стили к следующей таблице:

```
<table>
  <thead>
    <tr>
      <th>ФИО</th>
      <th>Зарплата</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Итого</td>
      <td>1240</td>
    </tr>
```

```
  </tfoot>
  <tbody>
    <tr>
      <td>Петров</td>
      <td>700</td>
    </tr>
    <tr>
      <td>Иванов</td>
      <td>540</td>
    </tr>
  </tbody>
</table>
```



ФИО	Зарплата
Петров	700
Иванов	540
Итого	1240

Создание групп столбцов. <colgroup>, <col>

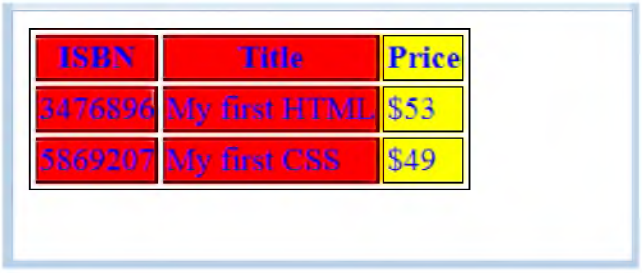
Теги <colgroup> и <col> предназначены для формирования вертикальной структуры таблицы. Тег <colgroup> является контейнером для тегов <col>. Тег <col> указывает свойства для каждого столбца в элементе <colgroup>. Тег <col> имеет единственный атрибут span, который определяет число столбцов, определенных в группу тегом <col>. Тег <colgroup> должен быть дочерним по отношению к элементу <table> после любых элементов <caption> и перед любыми элементами <thead>, <tbody>, <tfoot> и <tr>. Добавим обрамление к таблице:

```
<style>
  table, th, td {
        border: 1px solid black;
}
</style>
```

Применим этот стиль к следующей таблице:

```
<table>
  <colgroup>
    <col span="2" style="background-color:red">
    <col style="background-color:yellow">
  </colgroup>
  <tr>
    <th>ISBN</th>
```

```
    <th>Title</th>
    <th>Price</th>
  </tr>
  <tr>
    <td>3476896</td>
    <td>My first HTML</td>
    <td>$53</td>
  </tr>
  <tr>
    <td>5869207</td>
    <td>My first CSS</td>
    <td>$49</td>
  </tr>
</table>
```



ISBN	Title	Price
3476896	My first HTML	$53
5869207	My first CSS	$49

Первоначально, когда HTML-страницы были достаточно просты и незамысловаты, таблицы использовались при верстке для создания макетов страниц. В настоящее время этот способ устарел. Существует множество альтернативных способов создания макетов, в первую очередь с использованием блочной верстки и CSS.

Использование таблиц для создания макета

Таблицы больше не используются для верстки веб-страниц и компоновки отдельных элементов. Использование таблиц для создания макета страницы не рекомендуется по следующим причинам:

- табличная верстка плохо воспринимается программами чтения с экрана, что уменьшает доступность веб-страниц для людей, имеющих проблемы со зрением;
- страницы могут иметь достаточно сложную структуру, а значит, и таблицы, которые использовались для оформления страниц также достаточно сложны. Соответственно, такой код труднее писать. При необходимости изменения дизайна вносить изменения в сложные таблицы тяжело, поддерживать такой код трудно;
- таблицы не реагируют автоматически на тип устройства: у семантических контейнеров, таких как, например, <header>, <section>, <article>,

или , ширина по умолчанию равна 100 % от их родительского элемента. У таблиц размер по умолчанию подстраивается под их содержимое, поэтому, чтобы они одинаково хорошо работали на разных типах устройств, необходимо принимать дополнительные меры;

- Скорость загрузки страниц с макетом, основанном на «резиновых таблицах», более медленная.

Пример таблицы, в которой изменяется цвет строки при наведении курсора мыши

За стиль элемента при наведении на него курсора мыши отвечает псевдокласс :hover. Для изменения стиля строки таблицы псевдокласс :hover следует добавить к тегу <tr> и задать в нем цвет фона через свойство background.

Можно сделать, чтобы таким образом при наведении на строку подсвечивались не все строки, например чтобы не подсвечивался заголовок или строка подведения итогов. Для этого можно выделить группы строк и к строкам тела таблицы <tbody> применить стиль:

```
tr:hover {
    background-color: #dddddd;
}
```

Упражнение

Поместите на страницу следующую таблицу. Используйте теги <table>, <tr>, <th>, <td>, <caption>, <thead>, <tbody>, <colgroup> и <col>. Разумно примените атрибуты scope для ячеек заголовка. При помещении курсора мыши над строкой строка должна подсвечиваться.

Численные методы. 4 курс 5 группа.

Фамилия студента	Л. р. № 1. Метод деления отрезка пополам				Л. р. № 2. Метод простой итерации			
	Расчетные задания		Теоретическая задача	Дата	Расчетные задания		Теоретическая задача	Дата
	1	2			1	2		
Иванов И.								
Петров П.								
Сидоров С.								

Блочные и встроенные теги

Каждый HTML-тег по умолчанию отображается определенным образом. По способу отображения по умолчанию теги условно можно разделить на блочные и встроенные. Правда, такое однозначное разделение эле-

ментов существовало до появления HTML5. В спецификации HTML5 модель содержимого была расширена, теперь каждый элемент может принадлежать нулю, одной или более категориям. Однако разделение на блочные и строчные теги наглядно и понятно.

Блочные теги всегда зрительно выделены. Применение блочного тега приводит к переводу на новую строку до и после него. Кроме этого, блок отделяется вертикальными отступами до и после него. Блочные теги растягиваются на всю доступную ширину родительского элемента. Типичный блочный тег – .

К блочным элементам относятся: <address>, <article>, <aside>, <blockquote>, <canvas>, <dd>, , <dl>, <dt>, <fieldset>, <figcaption>, <figure>, <footer>, <form>, <h1> – <h6>, <header>, <hr>, <li>, <main>, <nav>, <noscript>, <ol>, <output>, <p>, <pre>, <section>, <table>, <tfoot>, <ul>, <video>.

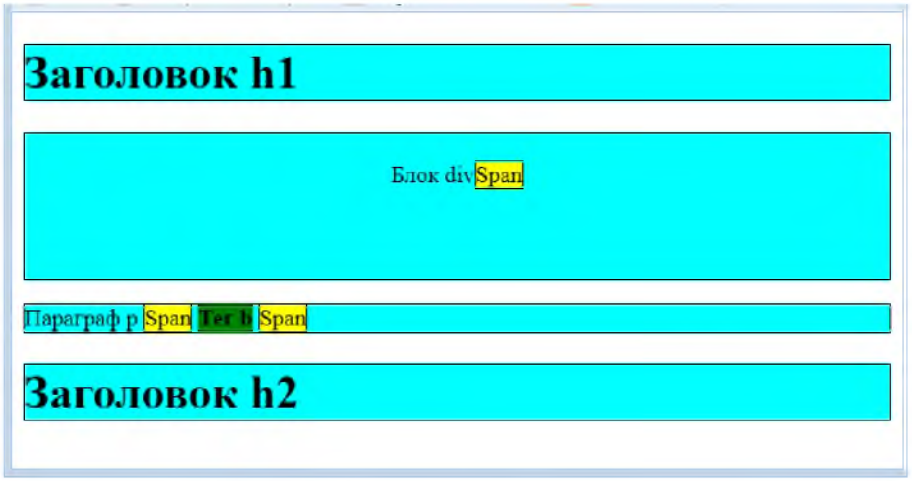
Встроенные теги еще называют строчными. Типичный строчный тег – . Встроенные теги используются для выделения куска текста и последующего его форматирования при помощи CSS.

К встроенным тегам относятся: <a>, <abbr>, <acronym>, <b>, <bdo>, <big>, <br>, <button>, <cite>, <code>, <dfn>, , <i>, <img>, <input>, <kbd>, <label>, <map>, <object>, <q>, <samp>, <script>, <select>, <small>, , , <sub>, <sup>, <textarea>, <time>, <tt>, <var>.

Блочные теги	**Встроенные теги**
Ширина равна ширине родительского блока (окна)	Ширина блока определяется размером содержимого
Высота определяется содержимым	Высота определяется содержимым
Можно задать ширину width и высоту height	Ширина и высота определяются автоматически, и задать их самостоятельно нельзя
Блок всегда начинается с новой строки. Следующий тег также начинается с новой строки	Встроенные теги ведут себя как текст: выстраиваются в ряд по горизонтали и переносятся на следующую строчку, если не хватает места
Могут содержать блочные, встроенные теги или текст	Могут содержать только встроенные теги или текст
Свойство vertical-align не действует	Высота элемента определяется автоматически по высоте контента
На теги внутри и текст действует text-align	Ширина элемента определяется автоматически по ширине контента

Как следует из свойств, блочные теги, следующие друг за другом, выстраиваются по вертикали и без применения CSS не могут расположиться рядом по горизонтали. Строчные элементы, наоборот, располагаются по горизонтали.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Блоки</title>
  <style>
    h1,h2,div,p{
        background: cyan;
        border: solid black 1px;
       }
    div{
        height: 100px;
        text-align: center;
       }
    span{
        border: solid black 1px;
        background: yellow;
        }
    b{
        background: green;
     }
    </style>
  </head>
  <body>
    <h1>Заголовок h1</h1>
    <div><br>Блок div<span>Span</span></div>
    <p>Параграф p
    <span>Span</span>
    <b>Тег b</b>
    <span>Span</span></p>
    <h1>Заголовок h2</h1>
  </body>
</html>
```



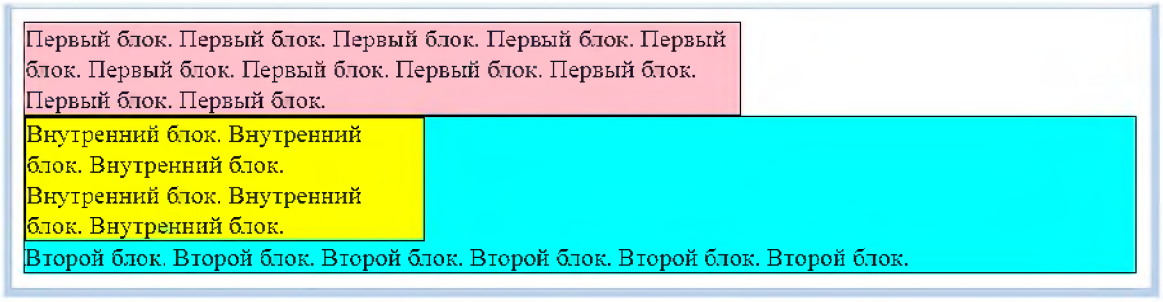
В данном примере видно, что для блочного тега , тегов заголовка и параграфа можно определить горизонтальное выравнивание контента внутри блока, сами блоки по умолчанию растянуты на всю штрину экрана. Высоту блока можно задать, а по умолчанию она определяется контентом. Блоки всегда начинаются с новой строки и оканчиваются новой строкой. Строчные блоки <b> и вкладываются в блочные элементы по горизонтали и занимают ровно столько места, сколько занимает их контент.

Блок

Тег определяет раздел или подраздел в документе и используется для группировки контента для последующего его форматирования при помощи CSS. Это блочный тег, применение которого приводит к переводу на новую строку до и после него. Кроме этого, блок отделяется вертикальными отступами до и после него. Тег активно используется при верстке макетов страниц.

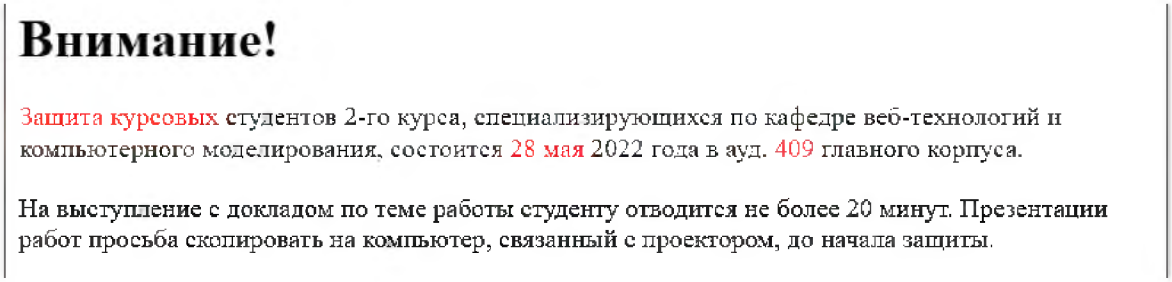
```
<style>
  div {
        border: solid 1px black;
      }
  #one {
        width: 450px;
        background: pink;
        border: solid 1px black;
       }
  #two {
        width: 700px;
        background: cyan;
       }
  #inner {
        width: 250px;
        background: yellow;
         }
</style>
…
<div id="one">Первый блок. Первый блок. Первый блок. Первый блок. Первый блок. Первый блок. Первый блок. Первый блок. Первый блок. Первый блок.</div>
  <div id="two">
  <div id="inner">Внутренний блок. Внутренний блок. Внутренний блок. Внутренний блок. Внутренний блок. Внутренний блок. </div>
```

```
Второй блок. Второй блок. Второй блок. Второй блок. Второй
блок. Второй блок. </div>
```



 – это встраиваемый блок. Используется для выделения части текста для последующего форматирования при помощи CSS или придания общих значений атрибутам, например lang. Без применения стилей никак не влияет на отображение контента страницы.

```
span {
      color: red;
     }

<body>
  <h1>Внимание!</h1>
  <p>
    <span>Защита курсовых</span> студентов 2-го курса,
    специализирующихся по кафедре веб-технологий и
    компьютерного моделирования, состоится <span>28
    мая</span> 2022 года в ауд. <span>409</span>
    главного корпуса.
  </p>
  <p>
    На выступление с докладом по теме работы студенту
    отводится не более 20 минут. Презентации работ просьба
    скопировать на компьютер, связанный с проектором, до
    начала защиты.
  </p>
</body>
```



Блочные теги для структурирования контента

Форматирование страниц с применением элемента и таблиц стилей – метод для разработчика прямолинейный, мощный и гибкий, но совершенно непрозрачный для других разработчиков, которые захотят разобраться в дизайне страницы. Требуются немалые усилия для того, чтобы разобраться в назначении каждого элемента на странице.

Можно заметить, что веб-страницы при всем своем многообразии по большей части состоят из стандартных компонентов, если только страница не отображает полноэкранное видео или игру, не является частью какого-либо художественного проекта или просто неплохо структурирована.

К стандартным компонентам относятся: заголовок, навигационное меню, основной контент, боковая панель.

Заголовок – это обычно большая полоса вверху страницы с крупным заголовком и логотипом. В заголовке указывается общая информация о веб-сайте, не меняющаяся от страницы к странице, нижний колонтитул.

Навигационное меню – это совокупность ссылок на основные разделы сайта. Обычно меню представляется в виде кнопок, ссылок или вкладок. Так же как и заголовок, меню остается неизменным на всех страницах сайта.

Основной контент обычно занимает большую область в центре страницы. Эта часть сайта уникальна на каждой странице.

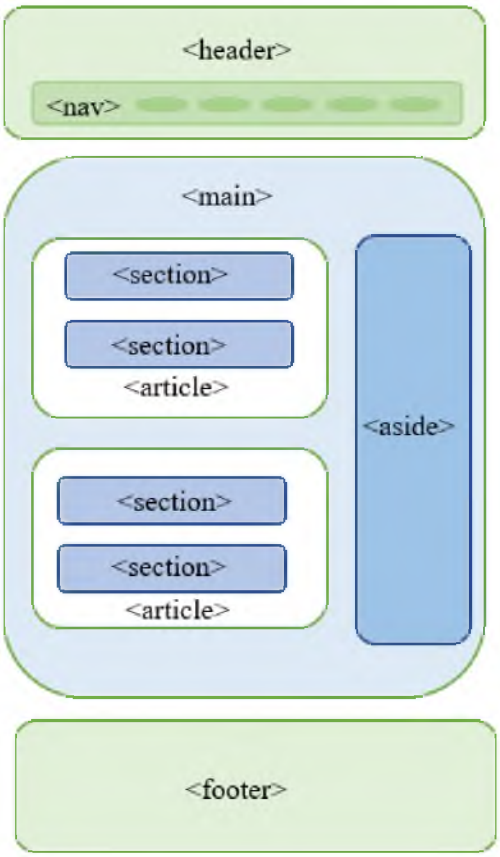
Боковая панель обычно содержит второстепенную информацию, например ссылки, цитаты, рекламу и т. д. Обычно она относится к содержимому в основном контенте, например, на странице со статьей боковая панель может содержать биографию автора или ссылки на связанные статьи. Однако она может содержать и несвязанную информацию, например дополнительное меню.

Нижний колонтитул, или **футер,** представляет собой полосу в нижней части страницы, которая обычно содержит уведомления об авторских правах или контактную информацию. Нижний колонтитул активно используется для SEO-целей, предоставляя ссылки для быстрого доступа к популярному контенту.

Стандарт HTML5 позволяет заменить элемент другими элементами, которые работают так же, как и , но являются более осмысленными и позволяют отделить боковые панели от заголовков, а рекламные панели – от меню. В HTML5 для структурирования кода введено несколько новых тегов: <article>, <aside>, <footer>, <header>, <nav>, <section>, <details> <summary>, <main>. Пользователю, который не видит код страницы, кажется, что особой разницы между тегами <div

class="header"> и <header> нет. Однако семантические теги ориентированы не на людей, которым нет смысла заглядывать в исходный код страницы, а на машины, интерпретирующие код. Машины или поисковые роботы не отличают от любого другого тега . Для них это просто типовой тег разметки, и от того, что у тега появился id, смысл тега не поменялся. Другое дело тег <header>: робот, обнаружив этот тег, воспринимает его именно как шапку сайта или раздела.

В итоге поисковые системы начинают лучше индексировать сайт, потому что четко отделяют контент страницы от вспомогательных элементов. Речевые браузеры, предназначенные для слабовидящих людей, пропускают заголовок <header> и переходят непосредственно к содержимому. Сайты могут автоматически обмениваться контентом и другой информацией между собой. Все эти возможности называются семантикой и позволяют представить данные в удобном для поисковых роботов виде.



В большинстве браузеров <article>, <aside>, <footer>, <header>, <nav>, <section>, <summary>, <main> будут отображаться со стилем:

```
article, aside, footer, header, nav, section, summary,
main {
    display: block;
}
```

<header>

Тег <header> задает «шапку» сайта или раздела, в которой обычно располагается заголовок или набор ссылок.

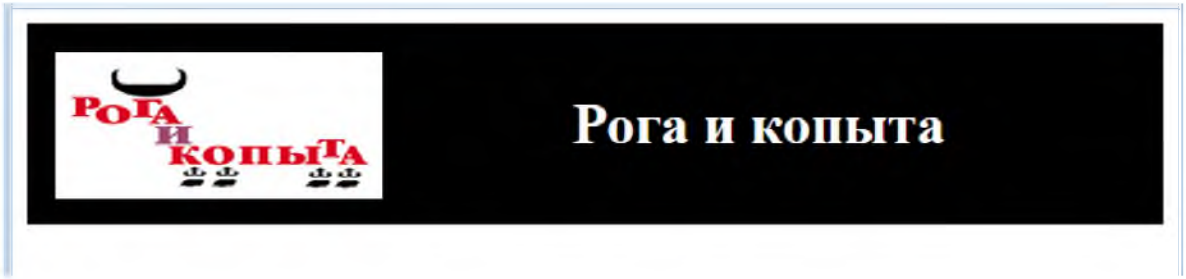
В <header>, как правило, помещают один или несколько тегов заголовка, логотип или картинку и информацию о владельце. В одном документе может быть несколько тегов <header>.

Если <header> – дочерний элемент <body>, то он определяет глобальный заголовок веб-страницы, но если он дочерний элемент <article> или <section>, то определяет конкретный заголовок для этого конкретного раздела. Определим стиль

```
<style>
  header {
    color: white;
    background-color: black;
    text-align: center;
    padding: 1em;
    }
</style>
```

для тега <header>:

```
<header>
  <img src="images/рога и копыта.png" alt="логотип"
width="184" height="82" style="float:left;">
  <h1>Рога и копыта</h1>
</header>
```



<article>

Тег <article> должен содержать независимую, автономную часть контента. Это может быть новость, статья, запись в блоге или на форуме, комментарий, т. е. независимый документ в рамках текста, который имеет смысл сам по себе и который можно распространять независимо от остальной части сайта.

<aside>

Определяет блок для информации, которая не является основной на странице, но имеет к ней некоторое отношение. Например, это может быть

врезка к печатной работе, чтобы ввести связанную тему или развить вопрос, исследуемый в основном документе, или просто несколько ссылок на связанное содержимое или архивы. Логично применять элемент <aside> также для блока объявлений. Как правило, элемент <aside> размещают сбоку от контента. Такой блок называется «сайдбар» или «боковая панель». Тем не менее <aside> может быть размещен, например, внизу страницы.

<footer>

Определяет нижний колонтитул документа или раздела. Обычно содержит информацию об авторе документа, авторских правах, ссылки на условия использования, контактную информацию и т. д. Может определяться как для всей страницы в целом, так и для каждого раздела в частности, поэтому может быть несколько тегов <footer> в одном документе.

<nav>

Тег <nav> является контейнером для навигационного блока. Если на странице несколько блоков ссылок, то в <nav> обычно помещают главное меню. Допустимо использовать несколько тегов <nav> в документе.

<section>

Задает раздел документа, как правило с заголовком. Тег <section> похож на <article>. Несмотря на то что обычно содержит заголовок, не является независимым документом в рамках общего текста. Может применяться для тематической группировки контента, блока новостей, контактной информации, глав текста, вкладок в диалоговом окне и др.

В зависимости от контекста можно разбить <article> на несколько <section> или, наоборот, <section> на несколько <article>. Допускается вкладывать один тег <section> внутрь другого.

<details> и <summary>

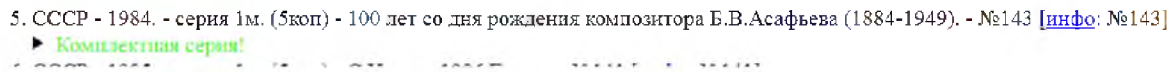
Служат для создания разворачиваемых блоков, которые можно показать или скрыть, щелкнув на заголовке. Разворачиваемый блок вставляется в элемент <details>, а заголовок блока – в элемент <summary>. Используется для придания интерактивности странице. Пользователь по щелчку может открывать или сворачивать контент, хранящийся в <details>. В тег <details> можно поместить любой контент.

Тег <summary> определяет видимый заголовок для элемента <details>. Перед заголовком выводится треугольник. Щелчок по заголовку или по треугольнику приводит к развороту или скрытию данных. Тег

<summary> должен идти первым внутри <details>. Ниже приведен стиль и код.

```
.completeSeries{
    color:#45F123;
}
...
<li>СССР - 1984. - серия 1м. (5коп) - 100 лет со дня рождения композитора Б.В.Асафьева (1884-1949). - №143 [<a href="http://nacekomie.ru/forum/viewtopic.php?p=1283748#p1283748" target="_blank">инфо</a>: №143]
  <br>
  <details><summary><span class="completeSeries">Комплектная серия!</span></summary>
  <img src="Images/SU/СССР 1984 1м 100 лет со дня рождения композитора Б.В.Асафьева.jpg">
  </details>
</li>
```

В свернутом состоянии элемент имеет вид:

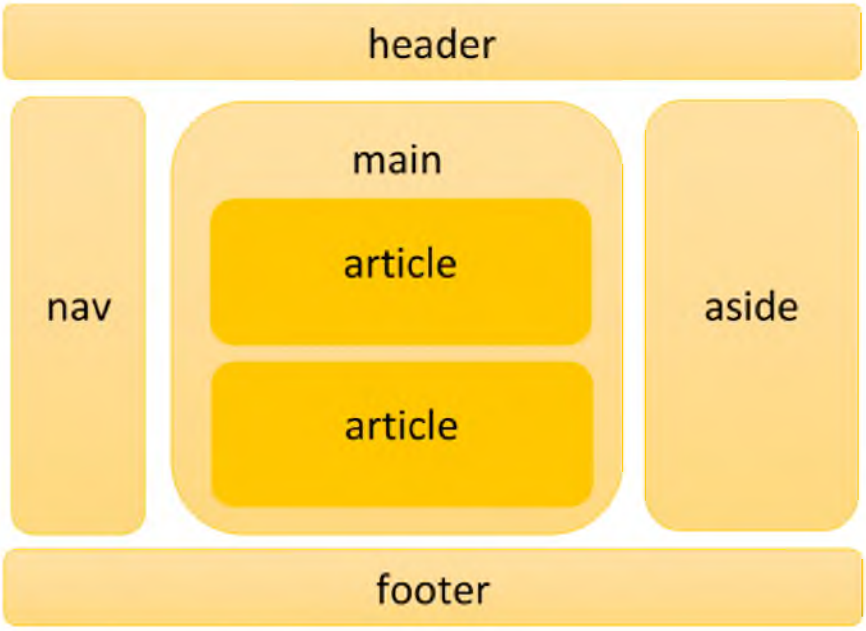


В развернутом состоянии:

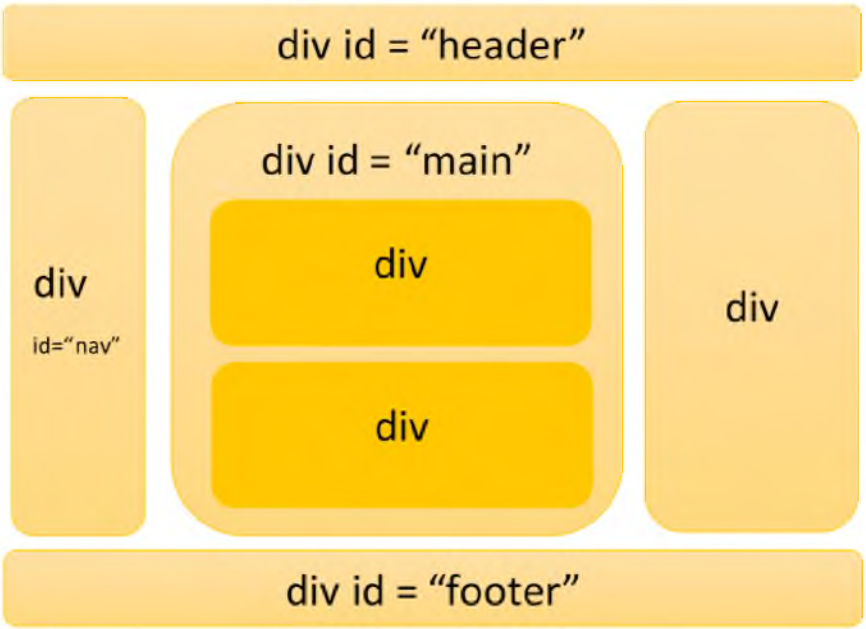
<main>

Тег <main> указывает на основное содержимое документа. Контент внутри элемента <main> должен быть уникальным для документа, т. е. его содержимое не должно повторяться ни в боковых панелях, ни в информации об авторских правах, ни в «шапке» страницы, ни в каком-либо другом разделе. Поэтому <main> не может содержаться в тегах <article>, <aside>, <footer>, <header> или <nav>. В документе может быть не более одного тега <main>.

С помощью этих тегов можно создавать, например, следующий макет страницы:



Ранее макеты создавались с помощью блочной или, как ее еще называют, дивной верстки, в которой для всех блоков контента использовались теги . Для пользователей зрительно оба макета не будут отличаться, но документ, сверстанный на макете из семантических тегов, будет лучше продвигаться в поисковых машинах, и, как следствие, пользователи будут его лучше находить.



Еще раньше применялась верстка на основе таблиц, или табличная верстка. Применять ее сегодня категорически не рекомендуется в силу ее несемантичности, усложнения структуры документа и медлительности при загрузке.

<table>
<tr><td colspan="3">header
<thead><tr><th></td></tr>
<tr><td rowspan="2">navigation
<tr>, <td></td><td rowspan="2">main content
<tr>, <td></td><td>sidebar
<tr>, <td></td></tr>
<tr><td>sidebar
<tr>, <td></td></tr>
<tr><td colspan="3">footer
<tfoot><tr><td></td></tr>
</table>

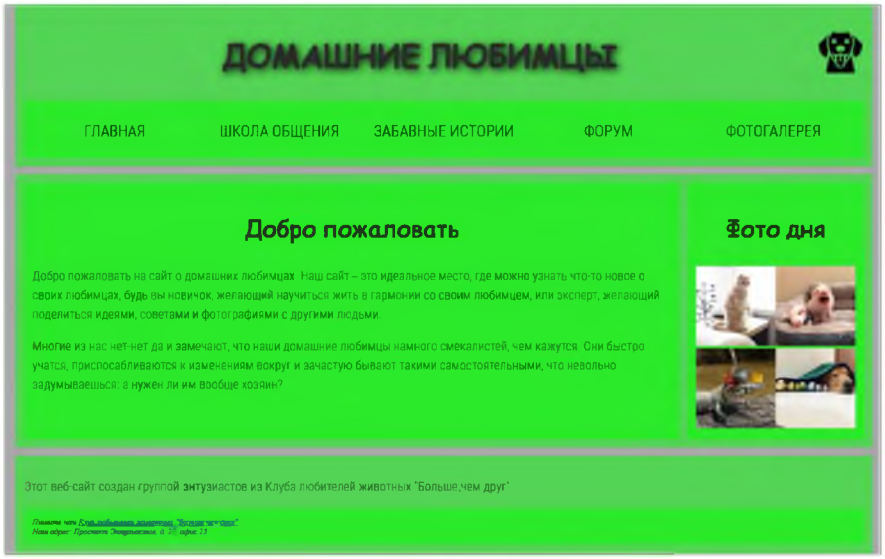
Упражнение

Разметить страницу так, чтобы к ней было просто применить CSS, – первое, чему должен научиться будущий веб-разработчик. В этом задании даны заготовки для веб-страницы:

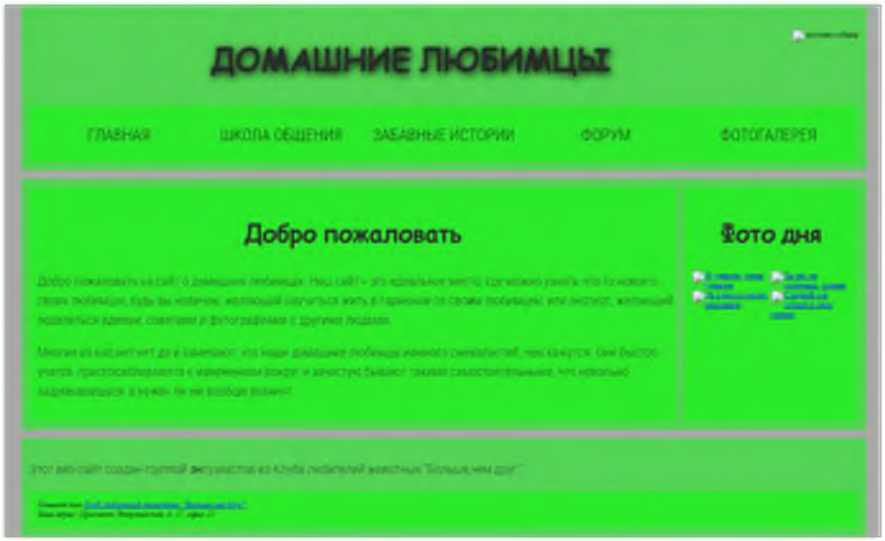
- файл index.html с набранным контентом;
- файл CSS style.css с уже определенными стилями;
- изображения, которые используются на странице.

При разметке index.html не использовались семантические блочные теги, а style.css предназначен для страниц, имеющих семантическую разметку, в частности использующих элементы main, header, nav, article, aside, footer, section, address.

Разметьте страницу с применением семантических тегов. Если вы правильно используете семантические теги, то страница в браузере примет следующий вид:



Если нет картинок:



Подсказки. Заголовок охватывает всю ширину сайта, содержит основное название страницы, логотип сайта и меню навигации. Заголовок и логотип появляются рядом друг с другом, когда применяется стилизация, и навигация появляется ниже этих двух элементов.

Основная область содержимого содержит два столбца: основной блок, содержащий текст приветствия, и боковую панель для размещения миниатюр изображений.

Нижний колонтитул содержит информацию о разработчиках и их адрес.

Вам необходимо добавить подходящую обертку:

- для заголовка;
- меню навигации;
- основного содержимого;
- приветственного текста;
- боковой панели изображений;
- нижнего колонтитула;
- адреса.

Чтобы сделать это, не нужно знать CSS и вносить правки в файл style.css. Нужно просто вставить нужные теги в HTML-документ index.html.

CSS для документа сделан таким образом, что при добавлении правильных структурных элементов в разметку они будут отображаться зелеными на отображаемой странице.

Файл index.html:

```
<!doctype html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <title>Домашние любимцы</title>
    <link
href="https://fonts.googleapis.com/css?family=Roboto+Con
densed:300|Cinzel+Decorative:700" rel="stylesheet">
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
      <h1>Домашние любимцы</h1>
      <img src="labrador-head.png" alt="логотип собаки">
        <ul>
          <li><span>Главная</span></li>
          <li><a href="#">Школа общения</a></li>
          <li><a href="#">Забавные истории</a></li>
          <li><a href="#">Форум</a></li>
          <li><a href="#">фотогалерея</a></li>
        </ul>
        <h2>Добро пожаловать</h2>
        <p>Добро пожаловать на сайт о домашних любимцах.
Наш сайт – это идеальное место, где можно узнать что-то
новое о своих любимцах, будь вы новичок, желающий
научиться жить в гармонии со своим любимцем, или эксперт,
желающий поделиться идеями, советами и фотографиями
с другими людьми. </p>
        <p>Многие из нас нет-нет да и замечают, что наши
домашние любимцы намного смекалистей, чем кажутся. Они
быстро учатся, приспосабливаются к изменениям вокруг
и зачастую бывают такими самостоятельными, что невольно
задумываешься: а нужен ли им вообще хозяин?</p>
        <h2>Фото дня</h2>
        <a href=" Я удивлен, очень удивлен"></a>
        <a   href="favorite-2.jpg"><img   src="favorite-
2_th.jpg" alt="Ты нас не отличишь, хозяин"></a>
        <a   href="favorite-3.jpg"><img   src="favorite-
3_th.jpg" alt=" Да я просто кроко-красавица"></a>
        <a   href="favorite-4.jpg"><img   src="favorite-
4_th.jpg" alt=" Сладкий сон собаки и семи птичек"></a>
      <p>Этот веб-сайт создан группой энтузиастов из Клуба
любителей животных "Больше чем друг".</p>
      <p>Пишите нам <a
href="mailto:petfriend@example.com">Клуб любителей
животных "Больше чем друг"</a><br>
Наш адрес: Проспект Энтузиастов, д. 27, офис 25
      </p>
  </body>
</html>
```

Файл style.css:

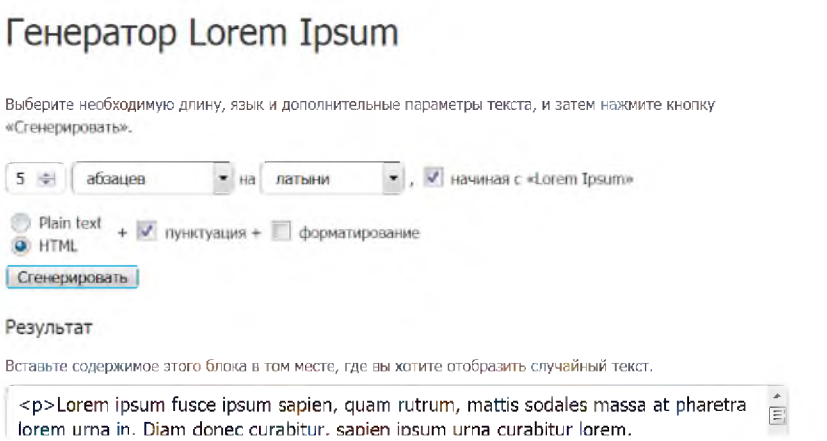
```
html, body {
  margin: 0;
  padding: 0;
}
html {
  font-size: 10px;
  background-color: #a9a9a9;
}
body {
  width: 70%;
  min-width: 800px;
  margin: 0 auto;
}
/* || typography */

h1, h2, h3 {
  font-family: 'Cinzel Decorative', cursive;
  color: #2a2a2a;
}
p, input, li {
  font-family: 'Roboto Condensed', sans-serif;
  color: #2a2a2a;
}
h1 {
  font-size: 4rem;
  text-align: center;
  text-shadow: 2px 10px black;
}
h2 {
  font-size: 3rem;
  text-align: center;
}
h3 {
  font-size: 2.2rem;
}
p, li {
  font-size: 1.6rem;
  line-height: 1.5;
}
/* || header layout */
header {
  margin-bottom: 10px;
  display: flex;
  flex-flow: row wrap;
}
```

```
body > * {
  background-color: red;
  padding: 1%;
}
main, header, nav, article, aside, footer, section,
address {
  background-color: rgba(0,255,0,0.5);
  padding: 1%;
}
h1 {
  flex: 5;
  text-transform: uppercase;
}
header img {
  display: block;
  height: 60px;
  padding-top: 20.15px;
}
nav {
  height: 50px;
  flex: 100%;
  display: flex;
}
nav ul {
  padding: 0;
  list-style-type: none;
  flex: 2;
  display: flex;
}
nav li {
  display: inline;
  text-align: center;
  flex: 1;
}
nav a, nav span {
  display: inline-block;
  font-size: 2rem;
  height: 3rem;
  line-height: 1.7;
  text-transform: uppercase;
  text-decoration: none;
  color: black;
}
/* || main layout */
main {
  display: flex;
```

```
}
article,section {
  flex: 4;
}
aside {
  flex: 1;
  margin-left: 10px;
  padding: 1%;
}
aside a {
  display: block;
  float: left;
  width: 50%;
}
aside img {
  max-width: 100%;
}
footer {
  margin-top: 10px;
}
```

Lorem ipsum

При верстке страниц иногда возникает необходимость заполнять какие-то фрагменты произвольным текстом, чтобы проверить, как будет выглядеть страница, поскольку реального контента в распоряжении разработчика на данный момент нет. Для этого используют генераторы шаблонного текста Lorem Ipsum. Как правило, в таких генераторах можно указать количество абзацев текста, число слов в абзаце. Пример такого генератора: http://exlab.net/tools/lorem-ipsum.html.

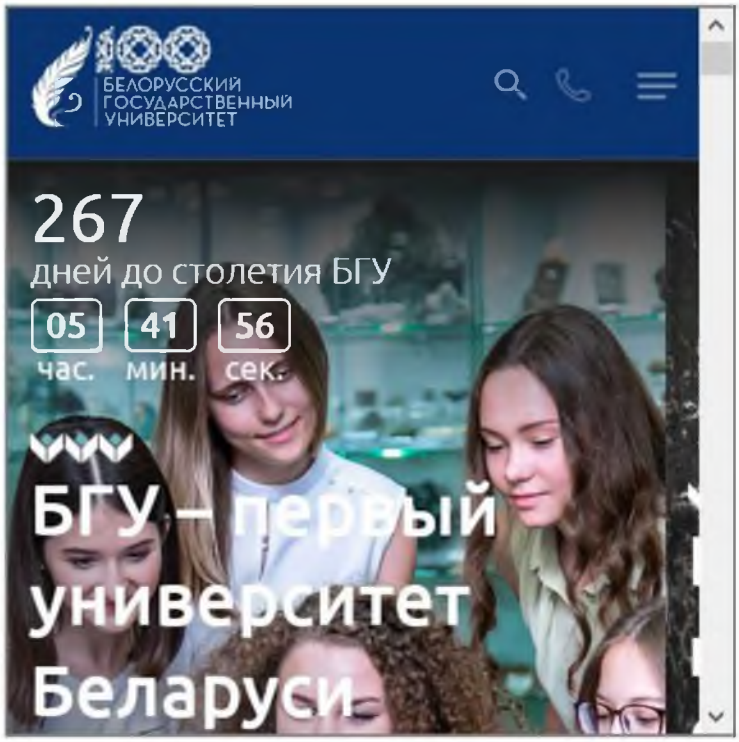


Есть также онлайн-генераторы текста на русском языке, например https://www.blindtextgenerator.com/ru.

<iframe>. Внедрение «Google Карт»

Тег <iframe> создает плавающий фрейм, который находится внутри обычного документа и позволяет загружать в область заданных размеров любые другие независимые документы. По умолчанию <iframe> имеет рамку. Чтобы удалить границу, можно добавить атрибут стиля и использовать свойство CSS.

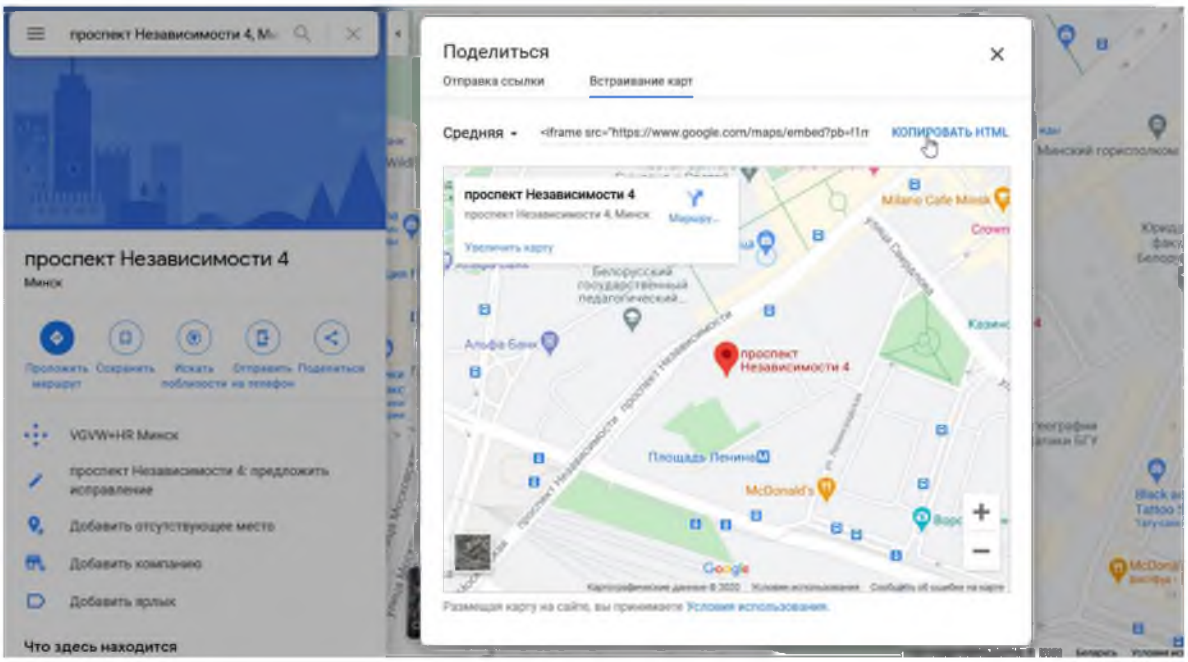
```
<iframe src="https://www.bsu.by/" height="400"
width="400" style="border:2px solid grey;"
name="iframe_a"></iframe>
<p><a href="https://www.bsu.by/" target="iframe_a">сайт
БГУ</a></p>
```



сайт БГУ

Многие сайты внедряют на свои страницы «Google Карты» для более наглядного представления местоположения своей компании.

Это можно сделать следующим образом: перейдите в «Google Карты» и выберите нужное вам место. Затем нажмите «Меню» (три горизонтальные линии) в верхнем левом углу пользовательского интерфейса и выберите параметр «Ссылка/код». Далее перейдите на вкладку «Встраивание карт», на которой будет код <iframe>. Этот код можно вставлять в свой HTML-документ.



<iframe> имеет следующие атрибуты:

- allowfullscreen – указывает, что <iframe> может быть помещен в полноэкранный режим с использованием полноэкранного API;
- frameborder – значение 1 (по умолчанию) указывает браузеру нарисовать границу между этим фреймом и другими фреймами, 0 удаляет границу;
- src – содержит адрес внедряемого документа;
- width and height – определяют ширину и высоту фрейма;
- sandbox – запрашивает повышенные настройки безопасности.

Следует понимать, что <iframe> может стать целью хакеров при попытке изменить веб-страницы или обманом заставить посетителей сделать то, чего они не хотят, например раскрыть конфиденциальную информацию вроде имени пользователя и пароля. Поэтому пользоваться <iframe> надо с осторожностью, используя различные механизмы безопасности для обеспечения защиты, разработанные создателями браузеров.

Упражнение

На странице сайта механико-математического факультета БГУ, посвященной проводимым факультетом дням открытых дверей (https://mmf.bsu.by/ru/dni-otkrytyh-dverej/), информация, как добраться до факультета, не бросается в глаза. Она расположена по ссылке в подвальной части сайта. Что, с вашей точки зрения, должно быть размещено на этой странице, чтобы было удобно абитуриенту? Сделайте страницу (центральную часть без подвальной, заголовочной частей и меню) так, как вы это видите.

Формы

Практически на каждом сайте есть поля для ввода текста, кнопки, переключатели, выпадающие списки или флажки. Эти элементы помогают организовывать интерфейсы поиска по ключевым словам, добавления комментариев, опросов, прикрепления фотографии и т. д. Интерактивный интерфейс на странице может быть организован с помощью средств языка HTML. Для этого предусмотрен тег <form>.

Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер: с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна из них. По этой причине данные форм должны быть независимы и не могут вкладываться друг в друга.

Тег <form> определяет форму на веб-странице и служит контейнером для остальных элементов интерфейса. Внутри тега <form> помещаются теги, которые определяют элементы для ввода информации: текстовые поля, радиокнопки, флажки и т. д. Сама форма никак не отображается на веб-странице, видны только ее элементы и результаты вложенных тегов.

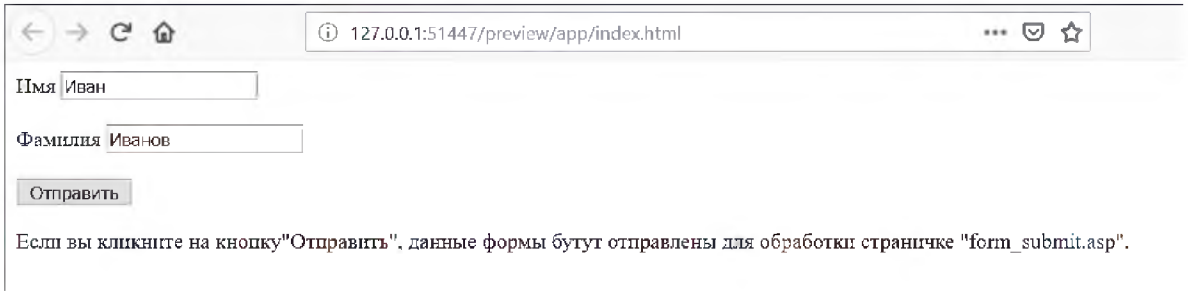
Тег <form> может содержать следующие элементы формы:

- <input> – наиболее часто используемый элемент, может отображаться несколькими способами, в зависимости от атрибута type;
- <label> – метка для элементов формы;
- <select> – выпадающий список;
- <textarea> – многострочная текстовая область для ввода;
- <button> – кнопка;
- <fieldset> – используется для выделения группы элементов;
- <legend> – определяет заголовок для элемента<fieldset>;
- <datalist> – список предопределенных вариантов;
- <output> – результат вычисления;
- <option> – раскрывающий список;
- <optgroup> – определяет группу в выпадающем списке.

Приведем пример простейшей формы.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Простейшая форма</title>
  </head>
```

```
  <body>
    <form name="nameForm" action="form_submit.asp">
    Имя <input type="text" name="fname"
value="Иван"><br><br>
    Фамилия <input type="text" name="lname"
value="Иванов"><br><br>
    <input type="submit" value="Отправить">
    </form>
    <p>Если вы кликнете на кнопку "Отправить", данные
формы будут отправлены для обработки страничке
"form_submit.asp".</p>
  </body>
</html>
```



В данном случае на форме есть два текстовых поля для ввода. Тег <input> с атрибутом `type = "text"` определяет однострочное поле для ввода текста, тег <input> с атрибутом `type = "submit"` – кнопку для отправки данных на сервер. Данные формы отправляются на веб-страницу на сервере form_submit.asp.

Перед отправкой данных браузер представляет информацию в виде пар «имя=значение», где имя определяется атрибутом name элемента формы, а значение – введенными пользователем данными. Стандарт URL позволяет использовать в ссылках только ограниченный набор символов: латинские буквы, цифры и некоторые знаки препинания. Если нужно использовать в URL символы кириллицы, или иероглифы, или, скажем, специфические символы французского языка, то не входящие в стандарт символы должны быть перекодированы особым образом. В интернете достаточно много сервисов для кодировки и раскодировки URL-строк, например http://www.codenet.ru/services/urlencode-urldecode/.

Если из формы методом GET передается строковая переменная, содержащая текст на русском языке, то при передаче из формы переменная приходит в кодировке UTF-8. Например, если в примере, описанном выше, передаются значения по умолчанию «Иван Иванов», то адресная строка будет иметь вид file:///C:/Lena/html5/2016/form/form_submit.asp?fname=%D0%98%D0%B2%D0%B0%D0%BD&lname=%D0%98%D0%B2%D0%B0%D0%BD%D0%BE%D0%B2,

где %D0%98%D0%B2%D0%B0%D0%BD соответствует значению «Иван», а %D0%98%D0%B2%D0%B0%D0%BD%D0%BE%D0%B2 – «Иванов».

После нажатия пользователем кнопки submit происходит вызов обработчика формы, который задается атрибутом action тега <form>. Обработчиком формы обычно выступает программа, написанная на любом серверном языке, например PHP, Python, C# и т. д. Если на форме нет кнопки Submit, то данные на сервер можно отправить, нажав клавишу Enter.

Если данные не отправляются на сервер, а обрабатываются, например, JavaScript, то вместо кнопки типа "submit" можно воспользоваться элементом <input> с атрибутом `type = "button"`.

Нельзя размещать форму внутри другой формы. Это может привести к непредсказуемому поведению форм в зависимости от браузера.

Атрибуты тега <form>

Рассмотрим основные атрибуты тега <form>. Все атрибуты являются опциональными, но принято указывать action и method.

Атрибут action определяет URL программы – обработчика формы, т. е. куда адресуется HTTP-запрос, по умолчанию это адрес самой HTML-страницы, где расположена форма. Если значение атрибута не будет указано, то после перезагрузки страницы элементы формы примут значения по умолчанию. В случае если вся работа будет выполняться на стороне клиента сценариями JavaScript, то для атрибута action можно указать значение #. Также можно сделать так, чтобы заполненная посетителем форма приходила на электронную почту:

```
<form action="mailto:адрес электронной почты"
enctype="text/plain"></form>
```

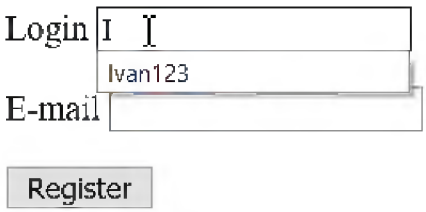
Атрибут method – метод HTTP-запроса, по умолчанию GET, может быть POST. Если для отправки данных используется метод GET, то в адресной строке после имени страницы может стоять вопросительный знак, после которого перечисляются параметры. Параметры разделяются между собой символом амперсанда (&). Нелатинские символы преобразуются в шестнадцатеричное представление (в форме %HH, где HH – шестнадцатеричный код для значения ASCII-символа), пробел заменяется на плюс (+). Например, http://mysite.com/example/handler.php?login=Anna&pass=123456. Как видно, метод get не является безопасным для пересылки конфиденциальных данных. Кроме GET и POST, существуют и другие методы, например PUT, PATCH и DELETE. Разработчики выбирают тот или иной метод, исходя из смысла действия.

Атрибут name содержит имя формы, используемое для ее идентификации, например в DOM. Он необходим, если на странице расположено

несколько форм. Этот атрибут был введен для обеспечения обратной совместимости, теперь активно используется атрибут id для идентификации элементов. В целях совместимости лучше инициализировать оба атрибута – и name, и id – одинаковым значением.

Атрибут accept-charset содержит список кодировок, необходимых серверу для обработки данных, пересылаемых формой. Значением является список кодировок, разделенных пробелами или запятыми, по умолчанию кодировка совпадает с кодировкой страницы.

Атрибут autocomplete выводит в браузере подсказки для заполнения на основе значений, которые пользователь ввел ранее, если автозаполнение включено.



Атрибут novalidate используется, если браузер не должен проверять форму.

Атрибут target задает имя окна или фрейма, куда обработчик будет загружать возвращаемый результат.

Атрибут enctype указывает, каким образом будут закодированы данные формы при отправке на сервер. Применяется только для метода POST. Может принимать значения application/x-www-form-urlencoded, multipart/form-data, text/plain. По умолчанию используется значение application/x-www-form-urlencoded. Все символы кодируются перед отправкой: пробелы преобразуются в символы «+», а специальные символы – в значения ASCII HEX. При значении multipart/form-data никакие символы не кодируются. Это значение используется в формах, у которых есть контроль загрузки файлов. При значении text/plain пробелы преобразуются в символы «+», но специальные символы не кодируются.

Атрибут rel определяет связь между связанным ресурсом и текущим документом.

Упражнение

```
<form>
  Имя <input type="text" name="fname"><br><br>
  Фамилия <input type="text" name="lname"><br><br>
  <input       type="button"       onclick="formSubmit()"
value="Отправить">
</form>
```

Для вышеприведенной формы:

- задайте имя «FIO»;
- укажите, что форму будет обрабатывать программа form_handler.php;
- данные будут отправляться по методу POST;
- данные будут отправляться в кодировке UTF-8;
- перед отправкой правильность заполнения формы не нужно проверять;
- определите кодировку символов ISO-8859-1 для символов, которые должны использоваться для отправки формы (латинский алфавит);
- включите опцию автозаполнения для полей формы.

Подпись к элементу пользовательского интерфейса <label>

Тег <label> используется для создания поясняющих текстов или меток к элементам управления HTML-форм. Он устанавливает связь между определенной текстовой меткой и элементом формы <input>. При установлении связи в случае нажатия курсором мыши на текст подсказки связанный с ней элемент формы получает фокус. Это полезно, когда элемент занимает достаточно маленькую область экрана и пользователям трудно нажимать на эти небольшие области, например флажок. При использовании меток, когда пользователь щелкает текст внутри элемента <label>, он переключает ввод на нужный элемент и тем самым увеличивает область нажатия. Программы чтения с экрана (скринридеры) будут проговаривать вслух содержимое метки, когда связанный с ней элемент будет находиться в фокусе.

Для элемента может быть создано несколько меток. Кроме того, с помощью <label> можно устанавливать горячие клавиши на клавиатуре и переходить на активный элемент, как по ссылкам.

Метки могут использоваться не только для элементов форм.

Существует два способа связывания объекта и метки. Первый заключается в использовании идентификатора id внутри тега <input> и указании его имени в качестве параметра for тега <label>. При втором способе тег <input> помещается внутрь контейнера <label>. В браузерах применение <label> ничем не отличается от надписей, сформированным обычным текстом.

Атрибуты тега <label>

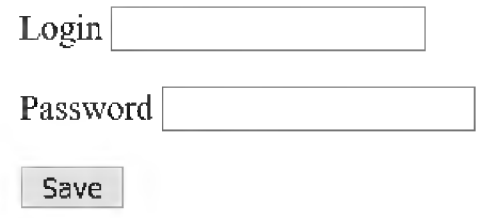
Тег <label> имеет два атрибута: for и form. Атрибут for содержит идентификатор элемента HTML-формы, с которым связывается текущая

метка. Атрибут form содержит идентификатор формы, с которой связывается метка. Кроме этого, удобно использовать глобальный атрибут accesskey для задания клавиш быстрого доступа к форме.

Приведем пример использования меток к элементам формы.

Каждый элемент формы обернут в блочный тег, чтобы разделить элементы по вертикали. К текстовым полям добавлены метки.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Использование надписи label в форме</title>
</head>

<body>
 <form id="frm1" name="frm1" action="hand.php"
method="post" enctype="application/x-www-form-
urlencoded" >
  <p>
   <label for="login" accesskey="l">Login</label>
   <input id="login" name="login" value="">
  </p>
  <p>
   <label for="psw">Password</label>
   <input type="password" id="psw" name="psw" value="">
  </p>
  <p>
   <input type="submit" value="Save">
  </p>
 </form>
</body>
</html>
```



Упражнение

```
<form>
Имя <input type="text" name="fname"><br><br>
Фамилия <input type="text" name="lname"><br><br>
```

```
<input type="button" onclick="formSubmit()"
value="Отправить">
</form>
```

Для вышеприведенной формы:

- для подписи полей используйте метки;
- оберните каждое поле и метку к ней в блочный тег, чтобы между полями были вертикальные отступы;
- оберните кнопку в блочный тег, чтобы между последним полем и кнопкой были вертикальные отступы.

Атрибуты, общие для элементов формы

Многие теги, используемые для определения элементов формы, имеют собственные атрибуты, но существует набор общих атрибутов, присущих всем элементам формы.

autofocus

Атрибут autofocus логического типа указывает, что поле ввода должно автоматически получать фокус при загрузке страницы. Атрибут явно определяется только для одного элемента в форме.

disabled

Атрибут логического типа определяет, может ли пользователь взаимодействовать с элементом. Если атрибут не определен, то по умолчанию это возможно.

form

Не все элементы формы должны размещаться в форме. Теоретически возможно помещать элементы формы за рамками элемента <form>, при этом атрибут содержит id формы в том же документе, с которой он ассоциирован.

name

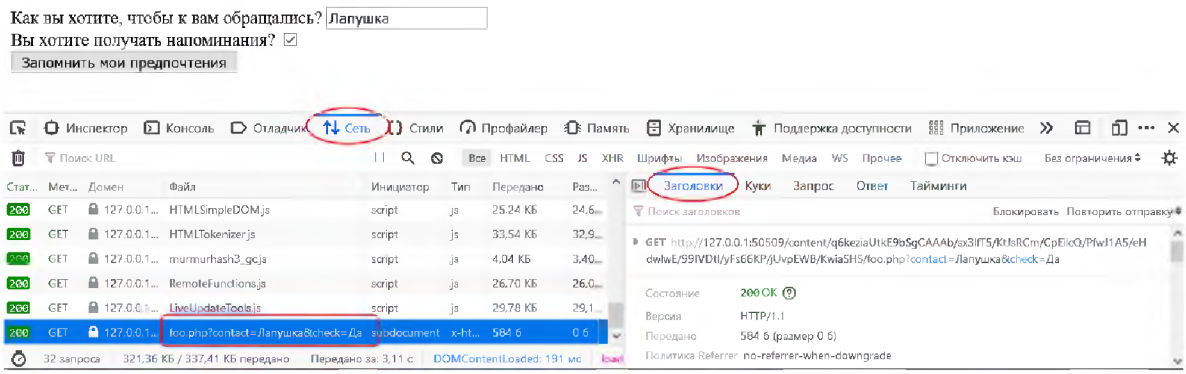
Содержит имя элемента. Параметр name элементов формы использует как браузер, так и сервер: браузер узнает, какие имена дать каждой части данных, а сервер может получить эти данные, обратившись к ним по заданному имени. Данные формы отправляются на сервер в виде пары «имя/значение». Для полей формы важно определить атрибут name, так как в HTTP-запрос попадают только те значения, для которых в поле задан

name. Значение для атрибута name обычно уникально в пределах формы. Например, дана следующая форма.

```
<form action="foo.php" method="get">
  <div>
    <label for="contact">Как вы хотите, чтобы к вам
обращались?</label>
    <input name="contact" id="contact" value="Лапушка">
  </div>
  <div>
    <label for="check">Вы хотите получать
напоминания?</label>
    <input type="checkbox" name="check" id="check"
value="Да">
  </div>
  <div>
    <button>Запомнить мои предпочтения</button>
  </div>
</form>
```

Просмотреть HTTP-запрос можно, например, в «Инструментах разработчика» Firefox. Они открываются при нажатии клавиши F12 в браузере Firefox. Надо перейти на вкладку «Сеть», затем на вкладку «Заголовки» и выбрать файл, указанный в атрибуте action формы. В данном случае это foo.php.

```
contact=Лапушка&check=Да
Реально параметры равны
contact=%D0%9B%D0%B0%D0%BF%D1%83%D1%88%D0%BA%D0%B0
check=%D0%94%D0%B0
ниже.
```



value

Содержит значение параметра, которое будет передано на сервер. Имя этого параметра хранится в name.

Элементы формы. Тег <input>

Для построения различных элементов формы можно использовать тег <input>. С помощью этого тега можно создавать текстовые поля, кнопки, переключатели, флажки и т. д. Тег <input> необязательно размещать внутри контейнера <form>, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер или обработаны с помощью скриптов на языке JavaScript, то <input> обязательно помещать внутрь контейнера <form>. Тег <input> может отображаться несколькими способами, в зависимости от атрибута type.

Атрибут type

Атрибут type определяет внешний вид и поведение элемента <input>. Он может принимать следующие значения:

- button – кнопка;
- checkbox – флажок;
- color – определяет интерфейс для выбора цвета;
- date – интерфейс для выбора календарной даты;
- datetime-local – указание местной даты и времени;
- email – текстовое поле для адреса электронной почты, проверка ошибок на стороне клиента, выполняемая браузером;
- file – интерфейс для выбора файла на клиентском компьютере для последующей отправки этого файла;
- hidden – скрытое от пользователя поле формы, используемое для передачи дополнительных параметров веб-приложению. Эти параметры формируются на этапе создания HTML-документа, например с помощью JavaScript;
- image – серверное изображение на стороне клиента, т. е. поле с изображением. При нажатии на рисунок данные формы отправляются на сервер;
- month – выбор месяца;
- number – поле для ввода чисел;
- password – строка для ввода пароля, обычное текстовое поле, но при вводе символы заменяются точками или звездочками;
- radio – радиокнопка, разрешается выбрать только один вариант из набора;
- range – ползунок для выбора чисел в указанном диапазоне;
- reset – кнопка для очистки формы;
- search – поле для поиска. Основное различие между текстовым полем и полем поиска заключается в том, как браузер их стилизует.

Чаще всего поля поиска отображаются с закругленными углами и имеют крестик, который нужно нажать, чтобы очистить введенное значение;

- submit – кнопка для отправки данных формы веб-приложению;
- tel – поле для ввода телефонных номеров. Можно вводить и буквы, и цифры. Отличие в семантическом наполнении;
- text – строка редактирования (значение по умолчанию). Если в текстовое поле вводится текст с разрывами строки, браузер удаляет эти разрывы строк перед отправкой данных;
- time – для времени;
- url – для веб-адресов;
- week – выбор недели.

Если атрибут type не задан или задан, но ему присвоено значение, не указанное в спецификации, то используется значение по умолчанию text.

Приведем пример формы с текстовыми полями.

```
<body>
  form     id="form1"     name="form1"     method="post"
action="save.asp"        enctype="application/x-www-form-
urlencoded">
    <p>
     <label for="fio">Ваше имя</label>
     <input   name="fio"  id="fio"  type="text"  value=""
size="50" accesskey="f">
    </p>
    <p><label for="god">Год рождения</label>
       <input     id="god"     name="god"     type="text"
value="2005" size="4" >
    </p>
    <p>
      <label for="prof">Род занятий</label>
      <input  id="prof"      name="prof"      type="text"
value="студент">
    </p>
    <p><input type="submit" value="Отправить"></p>
  </form>
  </body>
</html>
```

<input type="text"> определяет поле ввода текста в одну строку. Ширина текстового поля по умолчанию составляет 20 символов, value – значение по умолчанию. Атрибут enctype определяется только для метода post, значение по умолчанию – "application/x-www-form-urlencoded". Это значит, что все символы кодируются перед отправкой (пробелы преобразуются в символы «+», а специальные символы – в значения ASCII HEX).

По умолчанию при переходе по клавише Tab по полям формы перемещение происходит от первого элемента формы до последнего. Измените код предыдущего примера так, чтобы переходы по полям формы происходили в обратном порядке.

Приведем пример формы с радиокнопками.

```
<form>
<h3>Самая высокая гора Африки</h3>
  <input   type="radio"   id="stenli"   name="mountain"
value="stenli">
  <label for="mountain">Стэнли</label><br>
  <input   type="radio"   id="kilim"   name="mountain"
value="kilim">
  <label for="kilim">Килиманджаро</label><br>
  <input   type="radio"   id="spic"   name="mountain"
value="spic">
  <label for="spic">Спик</label><br>
    <input   type="radio"   id="emin"   name="mountain"
value="emin">
  <label for="emin">Эмин</label><br>
    <input   type="radio"   id="lui"   name="mountain"
value="lui">
  <label for="lui">Луиджи ди Савойя</label>
</form>
```

Самая высокая гора Африки

○ Стэнли
◉ Килиманджаро
○ Спик
○ Эмин
○ Луиджи ди Савойя

Приведем пример формы с переключателями.

```
<form>
Что положить в классический салат "селедка под шубой"?<br>
```

```
  <input    type="checkbox"    id="part1"    name="part1"
value="Herring">
  <label for="part1">Сельдь</label><br>
  <input    type="checkbox"    id="part2"    name="part2"
value="Garlic">
     <label for="part2">Чеснок</label><br>
     <input   type="checkbox"   id="part3"   name="part3"
value="Beet">
  <label for="part3">Свекла</label><br>
  <input    type="checkbox"    id="part4"    name="part4"
value="Potato">
  <label for="part4">Картошка</label><br>
     <input   type="checkbox"   id="part5"   name="part5"
value="Onion">
  <label for="part5">Лук</label><br>
     <input   type="checkbox"   id="part6"   name="part6"
value="Boat">
  <label for="part6">Орехи</label><br>
     <input   type="checkbox"   id="part7"   name="part7"
value="Nuts">
  <label for="part7">Морковь</label><br>
     <input   type="checkbox"   id="part8"   name="part8"
value="Carrot">
  <label for="part8">Оливки</label><br>
     <input   type="checkbox"   id="part9"   name="part9"
value="Olives">
  <label for="part9">Куриное яйцо </label><br>
     <input  type="checkbox"  id="part10"  name="part10"
value="Egg">
  <label for="part10">Майонез</label>
</form>
```

Что положить в классический салат "селедка под шубой"?

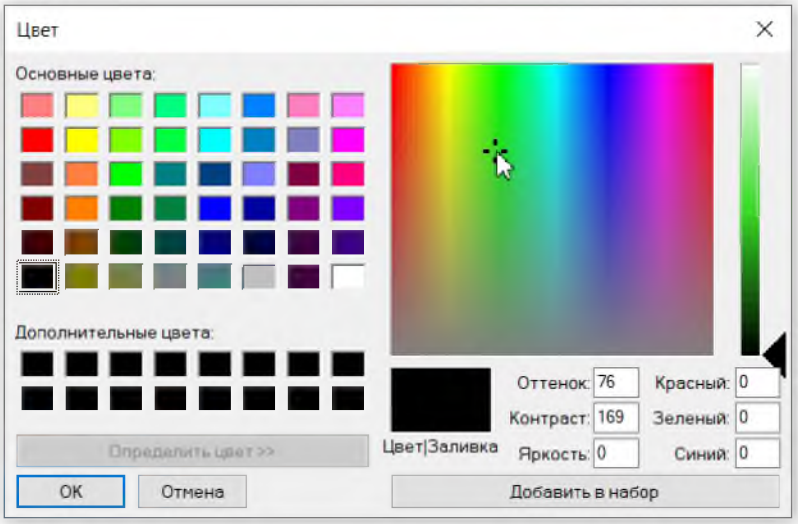
- ☐ Сельдь
- ☐ Чеснок
- ☑ Свекла
- ☐ Картошка
- ☐ Лук
- ☐ Орехи
- ☐ Морковь
- ☐ Оливки
- ☐ Куриное яйцо
- ☐ Майонез

Приведем пример формы с виджетом выбора цвета.

```
<form>
  <p>Сделаем колеровку краски компьютерным способом</p>
  <label for="favcolor">Выберите цвет краски:</label>
  <input type="color" id="favcolor" name="favcolor">
</form>
```

Сделаем колеровку краски компьютерным способом

Выберите цвет краски:



Упражнение

Дан код:

```
<form id="form1"
  name="form1"
  method="post"
  action="save.asp"
  enctype="application/x-www-form-urlencoded">
  <p>
    <b>Ваше имя</b> 
      <input name="fio" type="text" value="" size="50"
      accesskey="f">
  </p>
  <p>
   <b>Год рождения</b> 
     <input id="god" name="god" type="text" value="1999"
     size="4" >
  </p>
  <p><b>Род занятий</b> 
     <input   id="prof" name="prof" type="text"
```

```
      value="студент">
  </p>
  </p>
  <input type="submit" value="Отправить">
</form>
```

Из кода примера уберите атрибуты тегов, которые являются значениями по умолчанию.

Доступ к элементам формы

Можно получить доступ к полям формы через JavaScript.

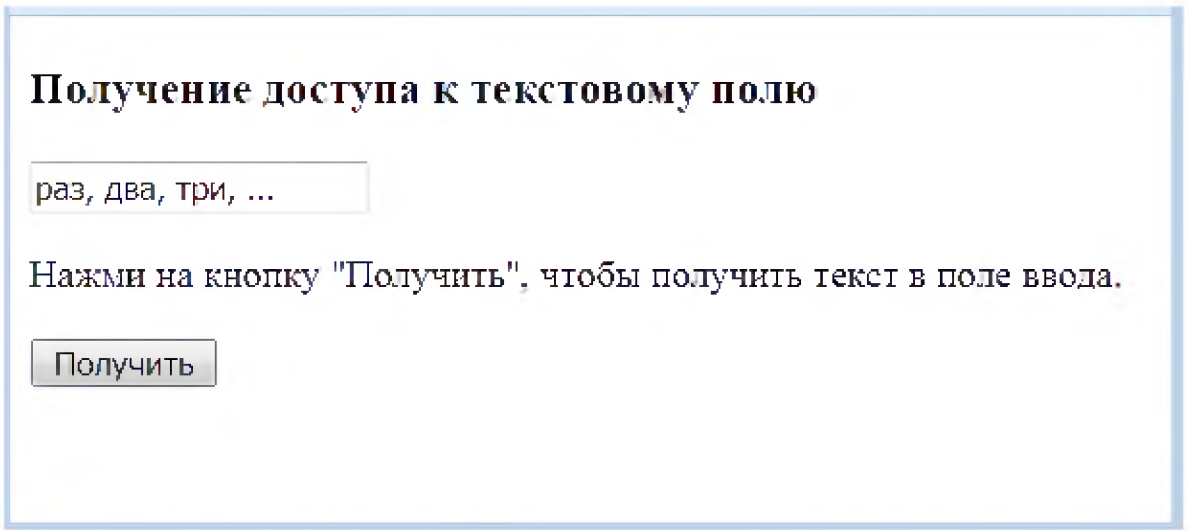
```
<h3>Получение доступа к текстовому полю</h3>
<input type="text" id="textField" value="раз, два, три, ...">
<p>Нажми на кнопку "Получить", чтобы получить текст в поле ввода.</p>
<button onclick="getText()">Получить</button>
<p id="demo"></p>
<script>
 function getText() {
    var x = document.getElementById("textField").value;
    document.getElementById("demo").innerHTML = x;
}
</script>
```

В JavaScript воспользуемся функцией getElementById, которая возвращает элемент по его id: `document.getElementById("textField")`. Затем у этого элемента извлечем свойство value

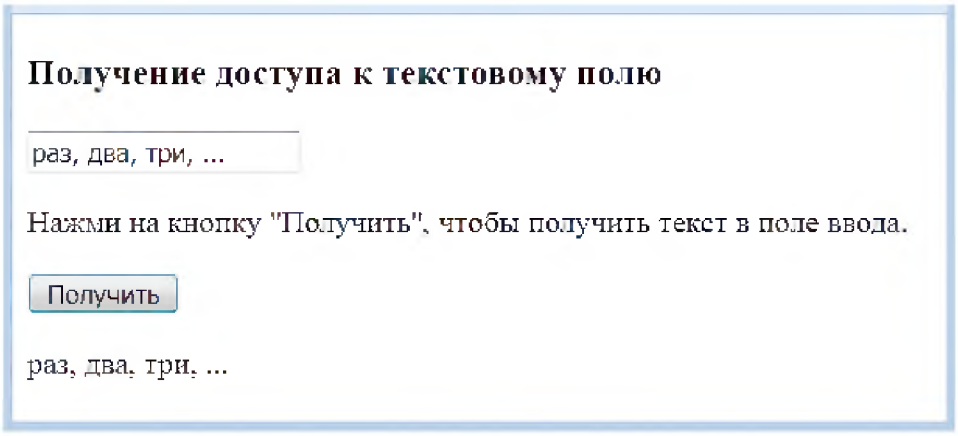
```
var x = document.getElementById("textField").value;
```

и занесем его значение в параграф с id="demo":

```
document.getElementById("demo").innerHTML = x;
```

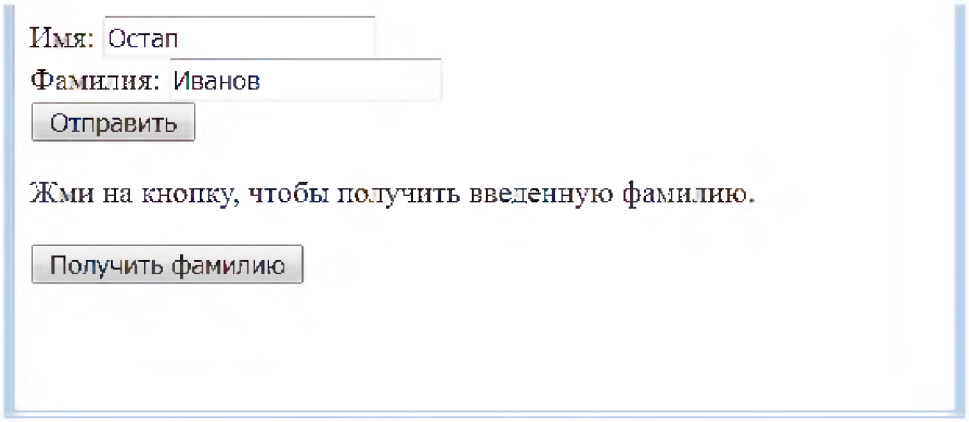


Вид после нажатия кнопки.

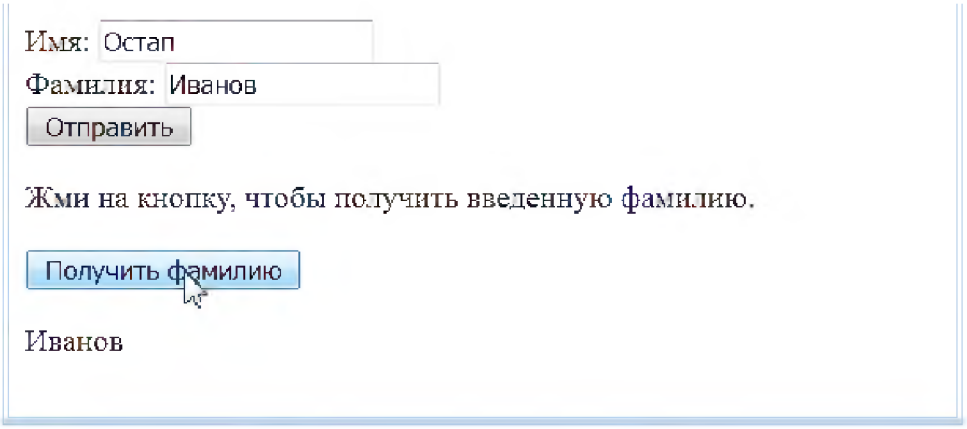


Если поле ввода находится на форме, то для доступа через JavaScript можно использовать коллекцию объектов, которые включает форма document.getElementById("nameForm").elements[].

```
<form id="nameForm" action="/action_page.php">
  Имя: <input type="text" name="fname" value="Остап"><br>
  Фамилия: <input type="text" name="lname"
value="Иванов"><br>
  <input type="submit" value="Отправить">
</form>
<p>Жми на кнопку, чтобы получить введенную фамилию.</p>
<button onclick="nameFunction()">Получить
фамилию</button>
<p id="demo2"></p>
<script>
function nameFunction() {
    var x =
document.getElementById("nameForm").elements[1].value;
    document.getElementById("demo2").innerHTML = x;
}
</script>
```



Вид после нажатия кнопки:



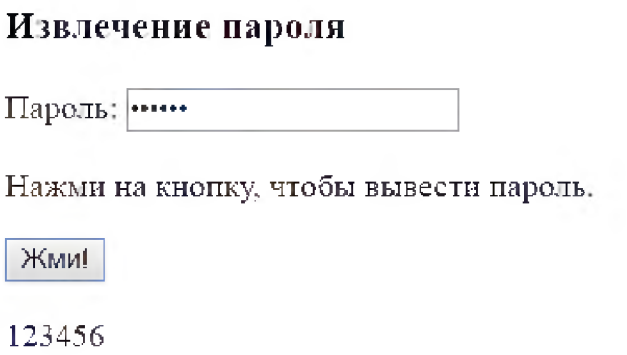
Приведем пример формы для ввода пароля.

Символы, набираемые в поле ввода пароля, заменяются звездочкой или жирной точкой. Пароль нельзя скопировать при помощи выделения и нажатия комбинации клавиш Ctrl + C. Значение пароля доступно для JavaScript и для web-приложения на сервере.

```
<form name="form2" action="save.asp">
  <p>
  <b>Пароль</b> 
  <input   id="psw" name="psw" type="password">
  </p>
  <input type="submit" value="Отправить">
</form>
```

Извлечение пароля

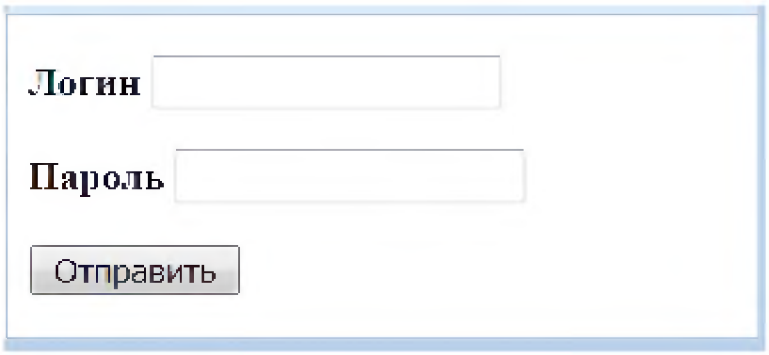
```
<h3>Извлечение пароля</h3>
Пароль:
<input type="password" id="myPsw" value="123456">
<p>Нажми на кнопку, чтобы вывести пароль.</p>
<button onclick="getPsw()">Жми!</button>
<p id="demo"></p>
<script>
function getPsw() {
    var x = document.getElementById("myPsw").value;
    document.getElementById("demo").innerHTML = x;
}
</script>
```



Упражнение

Имеется форма для ввода логина и пароля.

```
<form name="passwordForm" action="save.asp">
  <p>
    <strong>Логин</strong>
    <input id="log" name="log">
  </p>
  <p>
    <strong>Пароль</strong>
    <input   id="psw" name="psw" type="password">
  </p>
  <input type="submit" value="Отправить">
</form>
```



Если логин совпадает с паролем, то выведите на страницу предупреждение.

Создание поля для ввода пароля

Динамически создадим поле для ввода пароля.

```
<button onclick="createFieldPas()">Создать поле для ввода
пароля</button>
<script>
function createFieldPas() {
    var x = document.createElement("input");
```

```
    x.setAttribute("type", "password");
    x.setAttribute("value", "123456");
    document.body.appendChild(x);
}
```

Упражнение

Спросите у пользователя, желает ли он зарегистрироваться. Если да, то динамически создайте регистрационную форму.

Атрибут checked

Флажок, принимающий значение true или false и сигнализирующий, выбран ли элемент интерфейса. Используется только для флажков и радиокнопок. Значение по умолчанию – false.

Приведем пример формы с радиокнопками и переключателями. К форме из предыдущего раздела

```
<form id="form1" name="form1" method="post"
action="save.asp">
  <p>
    <b>Ваше имя</b> 
    <input name="fio" type="text" value="" size="50"
    accesskey="f">
  </p>
  <p>
    <b>Год рождения</b> 
    <input id="god" name="god" type="text"
    value="1999" size="4" >
  </p>
  <p>
    <b>Род занятий</b> 
    <input    id="prof" name="prof" type="text"
    value="студент">
  </p>
  <p>
    <input type="submit" value="Отправить">
  </p>
</form>
```

добавим радиокнопки. Для этого атрибуту type тега <input> установим значение radio. Радиокнопки позволяют пользователю выбрать только

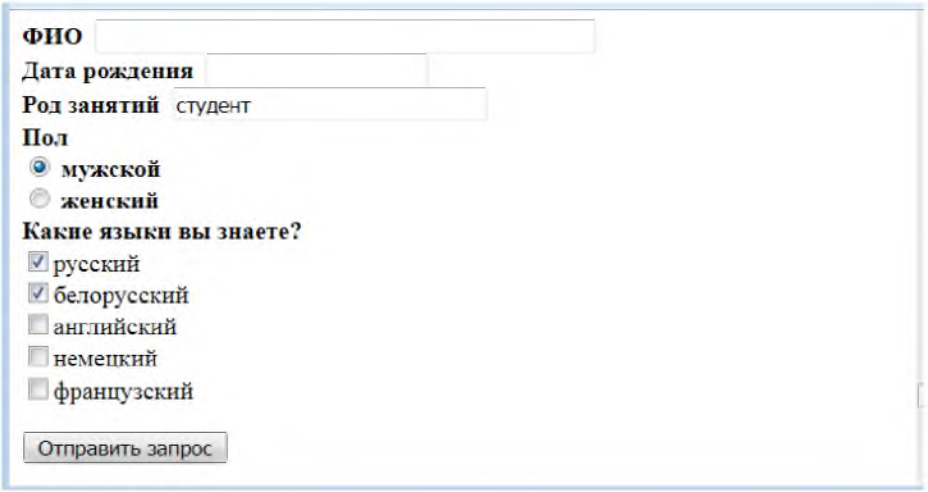
один из ограниченного числа вариантов. Радиогруппа должна иметь одно и то же имя (значение атрибута name), чтобы считаться группой.

```
<b>Пол</b><br>
<input   id="pol" name="pol" type="radio"
value="мужской"> <b>мужской</b><br>
<input id="pol" name="pol" type="radio"
value="женский"> <b>женский</b><br>
```

Затем добавим группу переключателей. Для этого атрибуту type тега <input> установим значение checkbox. Группа переключателей позволяют пользователю выбрать сразу несколько вариантов.

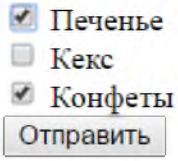
```
<b>Какие языки вы знаете?</b><br>
<input type="checkbox" name="option1" value="a1"
checked>русский<br>
<input type="checkbox" name="option2" value="a2"
checked>белорусский<br>
<input type="checkbox" name="option3"
value="a3">английский<br>
<input type="checkbox" name="option4"
value="a4">немецкий<br>
<input type="checkbox" name="option5"
value="a5">французский<br>
```

В результате получим:



Упражнение

Создайте форму для выбора продуктов к чаю. Переключатели оформите в виде списка.



Программный выбор флажков

Используя JavaScript, можно изменять значения атрибута checked программно. Для этого достаточно получить элемент, например, через id функцией getElementById:

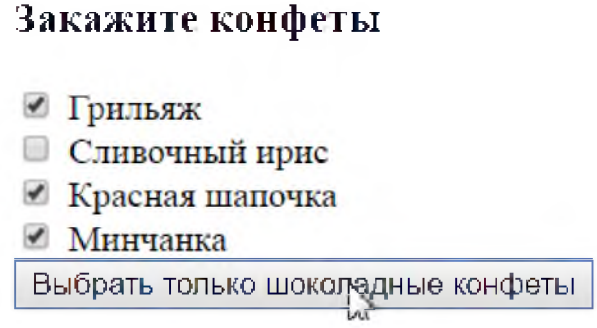
```
var x = document.getElementById("product1");
```

А затем изменить его свойство checked:

```
x.checked = true;
```

Допустим, у нас есть набор флажков для выбора конфет. Выберем только шоколадные конфеты. Выбор будем осуществлять через нажатие на кнопку.

```
<h3>Закажите конфеты</h3>
  <input type="checkbox" id="product1" value="gril">
Грильяж<br>
  <input type="checkbox" id="product2" value="iris">
Сливочный ирис<br>
  <input type="checkbox" id="product3" value="red">
Красная шапочка<br>
  <input type="checkbox" id="product4" value="min">
Минчанка<br>
   <button   onclick="selectChocolate()">Выбрать   только
шоколадные конфеты</button><br>
  <script>
function selectChocolate() {
    var x = document.getElementById("product1");
    x.checked = true;
    x = document.getElementById("product3");
    x.checked = true;
    x = document.getElementById("product4");
    x.checked = true;
}
</script>
```

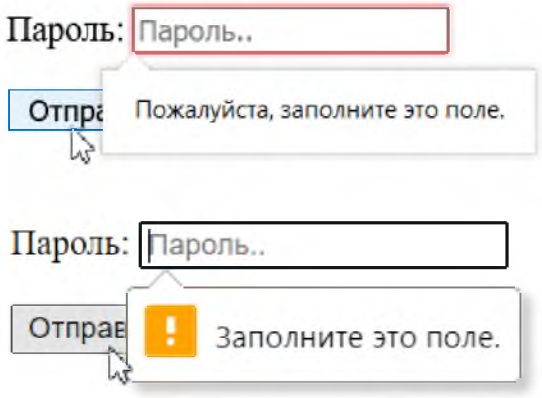


Атрибуты pattern, placeholder, required, title

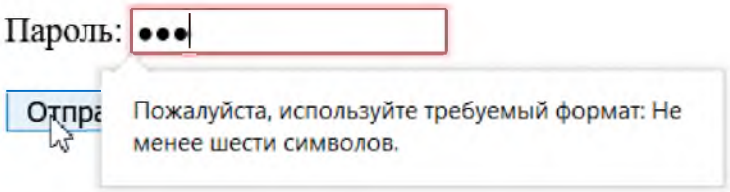
В HTML5 были добавлены возможности для валидации полей формы. Рассмотрим валидацию на примере тега <input> с типом type="password".

```
<form name="form2" action="autorize.asp">
  <p>
    <label for="psw">Пароль:</label>
    <input type="password" id="psw"
placeholder="Пароль.." pattern=".{6,}" title="Не менее
шести символов" required>
  </p>
  <p>
    <input type="submit" value="Отправить">
  </p>
</form>
```

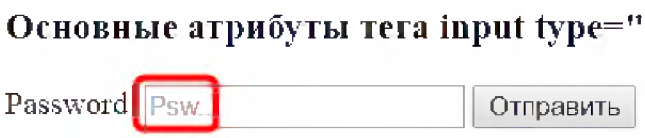
Атрибут required тега <input> указывает, что поле ввода должно быть заполнено перед отправкой формы. Вид всплывающей подсказки и ее содержание могут разниться в различных браузерах. Ниже показано, как выглядят подсказки в Firefox и Google Chrome.

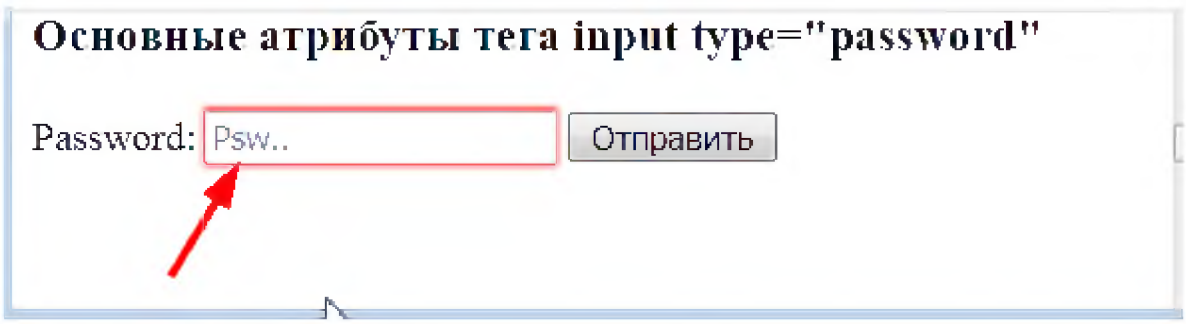


Глобальный атрибут title для тега <input> работает как всплывающая подсказка для формата ввода.



Атрибут placeholder выводит короткую подсказку, которая описывает ожидаемое значение элемента <input>.





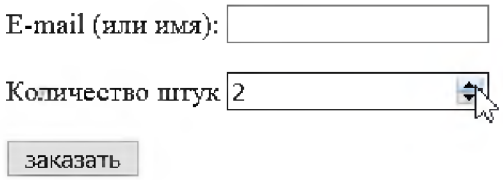
Атрибут pattern задает регулярное выражение, при помощи которого будет проверяться введенное значение элемента <input>.

Скрытые поля

Можно определить на форме скрытое поле. Для этого в теге <input> тип поля устанавливается в hidden. Часто этот атрибут скрывает элемент управления, который не отображается браузером и не дает пользователю изменять значения по умолчанию.

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>iframe</title>
 </head>
 <body>
  <form id="frm1" name="frm1" action="save.asp">
    <p>
      <label for="email" accesskey="1">
       E-mail (или имя):
      </label>
      <input type="email" id="email" name="email"
      required>
    </p>
      <input type="hidden" id="bank-account"
      name="bank-account">
    <p>
      <label for="count" accesskey="c">
        Количество штук
      </label>
      <input type="number" id="count" name="count"
      min="1" max="5" step="1">
    </p>
    <button accesskey="s" name="btn" id="btn"
    type="submit" value="save">
      заказать
```

```
    </button>
  </form>
 </body>
</html>
```



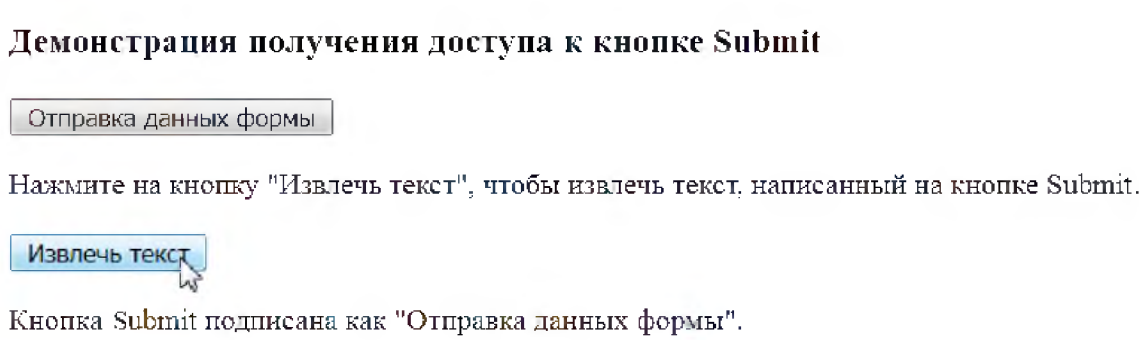
Кнопки

При конструировании формы создать кнопки можно различными способами.

Тип type="submit" для <input> определяет кнопку для отправки данных формы на сервер, value задает надпись на кнопке. По умолчанию это Submit или «Отправить» в зависимости от языковых настроек.

Покажем, как получить доступ к кнопке submit.

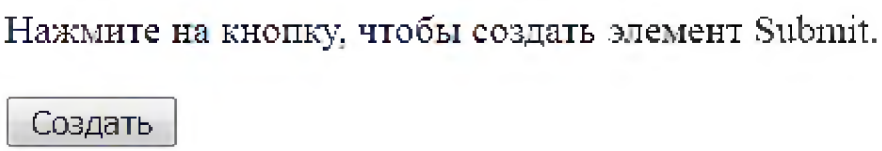
```
<h3>Демонстрация получения доступа к кнопке Submit</h3>
<input type="submit" id="mySubmit" value="Отправка данных
формы">
<p>Нажмите на кнопку "Извлечь текст", чтобы извлечь текст,
написанный на кнопке Submit.</p>
<button onclick="myFunction()">Извлечь текст</button>
<p id="submitDemo"></p>
<script>
 function myFunction() {
    var x = document.getElementById("mySubmit").value;
    document.getElementById("submitDemo").innerHTML =
"Кнопка submit подписана как \""+x+"\".";
}
</script>
```



Тип type="button" для <input> просто определяет кнопку без заданной функциональности. В атрибуте value определяет текст на кнопке.

Пример создания кнопки submit в DHTML. При щелчке по кнопке вызывается скрипт createSubmit:

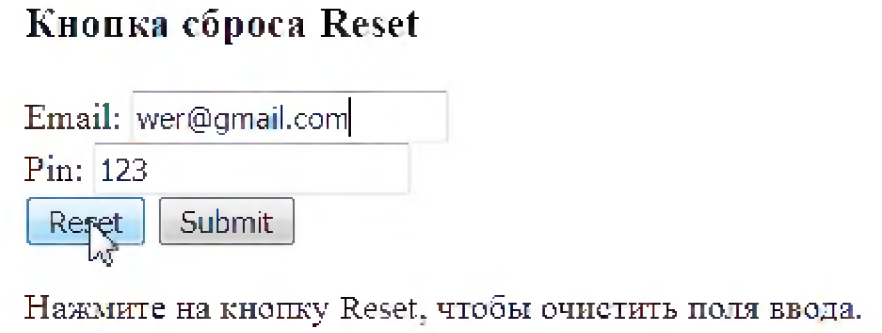
```
<p>Нажмите на кнопку, чтобы создать элемент Submit.</p>
<button onclick="createSubmit()">Создать</button>
<script>
 function createSubmit() {
    var x = document.createElement("INPUT");
    x.setAttribute("type", "submit");
    document.body.appendChild(x);
}
</script>
```



После выполнения этого кода кнопка Submit добавится в конец страницы.

Тип type="reset" для <input> задает кнопку для очистки формы. С точки зрения UX (user experience) это считается плохой практикой.

```
<h3>Кнопка сброса "reset"</h3>
<form action="demo_form.asp">
  Email: <input type="text" name="email"><br>
  Pin: <input type="text" name="pin" maxlength="4"><br>
  <input type="reset" value="Reset">
  <input type="submit" value="Submit">
</form>
<p>Нажмите на кнопку Reset, чтобы очистить поля ввода.</p>
```



Тип type="image" для <input> – это серверное изображение на стороне клиента, т. е. поле с изображением. При нажатии на рисунок данные формы отправляются на сервер, т. е. кнопка с изображением ведет себя

как кнопка Submit. Однако отправляются не значения полей, а координаты X и Y щелчка по изображению. Например, когда Вы щелкаете изображение этого виджета, то отправляетесь по URL-адресу http://foo.com?pos.x=10&pos.y=35.

Благодаря этому удобно использовать изображения как кнопки для карты кликов.

```
<form action="demo_form.asp">
  <h1>Какая собака вам подходит по характеру?</h1>
  <input type="image" src="images/dogs.jpg" alt="Submit">
</form>
```

Какая собака Вам подходит по характеру?

Многострочное текстовое поле <textarea>

Иногда в форме требуется ввести не одну, а несколько строк информации, например при сборе пользовательских данных, таких как комментарии или обзоры. В этом случае используется элемент интерфейса для многострочного текста, называемый memo. Для его создания используется тег <textarea>.

Основное различие между текстовым полем textarea и обычным однострочным текстовым полем input type="text" заключается в том, что пользователям разрешено вводить текст, который включает символы перехода на следующую строку. По умолчанию тег создает пустое поле шириной в 20 символов, состоящее из двух строк. Выводимый текст находится между тегами. Требуется закрывающий тег. Если нужно определить

значение по умолчанию для <textarea>, то можно поместить его между открывающим и закрывающим тегами:

```
<textarea>
  по умолчанию в этом элементе находится этот текст
</textarea>
```

Атрибуты тега <textarea>

Атрибут rows задает количество видимых строк. Если содержимое элемента выходит за пределы зоны видимости, возникает полоса прокрутки.

Атрибут cols задает количество столбцов символов в символах средней ширины. Вводимый текст автоматически переносится на следующую строку, для того чтобы были видимыми длинные строки без необходимости прокрутки. Если содержимое элемента выходит за пределы зоны видимости возникает полоса прокрутки.

Атрибут-флаг readonly указывает, что текстовое поле не может изменяться пользователем, в том числе нельзя вводить новый текст или модифицировать существующий. Кроме того, такое поле не может получить фокус путем нажатия на клавишу Tab, мышью или другим способом. Тем не менее состояние и содержимое поля можно менять с помощью скриптов.

Атрибут wrap устанавливает правила переноса текста в поле <textarea> и вид, в котором данные будут отправлены на сервер. Если этот параметр отсутствует, текст в поле набирается одной строкой, а когда число введенных символов превышает ширину области, появляется горизонтальная полоса прокрутки. Нажатие клавиши Enter переносит текст на новую строку, и курсор устанавливается у левого края поля. Может принимать следующие значения:

- soft – длинный текст, который самостоятельно не помещается в поле по ширине, будет автоматически перенесен на новую строку, однако передаваться на сервер будет как одна строка. Нажатие клавиши Enter на клавиатуре устанавливает перенос текста, который сохраняется при отправке формы.
- hard – слова в поле переносятся механически, чтобы они поместились в размер области, и при отправке на сервер точки автоматического переноса сохраняются.

Атрибут required указывает, что текстовая область обязательно должна быть заполнена.

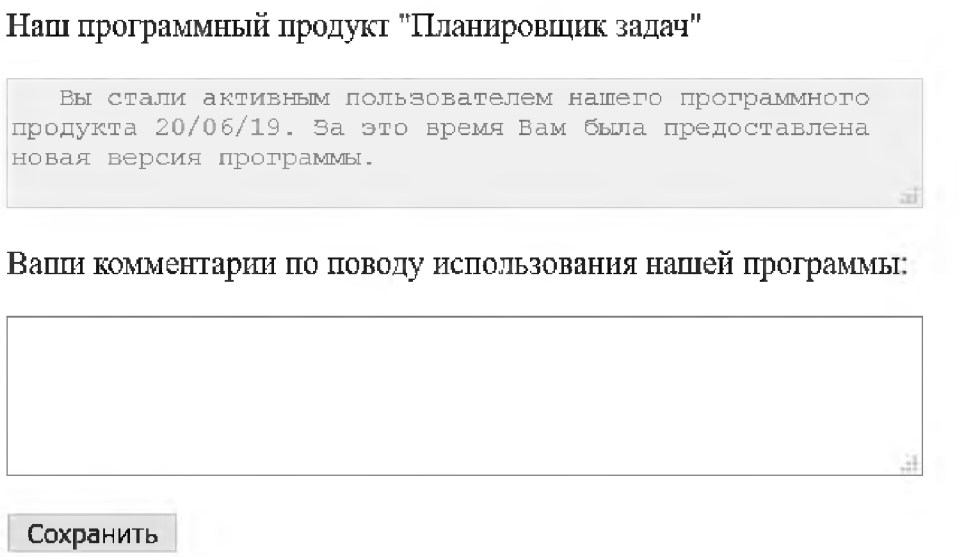
Атрибут placeholder определяет текст-заполнитель, представляющий собой краткую подсказку, которая описывает ожидаемое значение текстовой области.

Атрибут maxlength определяет максимальное количество символов, разрешенное в текстовой области.

Атрибут dirname указывает, что направление текста текстового поля будет отправлено.

Приведем пример использования текстовой области.

```
<form id="opros" name="opros" action="save.asp"
method="get">
  <p>Наш программный продукт "Планировщик задач"</p>
  <p>
    <textarea id="comment" name="comment" rows="3"
    cols="55" disabled="disabled">
     Вы стали активным пользователем нашего программного
     продукта 20/06/19. За это время Вам была
     предоставлена новая версия программы.
    </textarea>
  </p>
  <p>Ваши комментарии по поводу использования нашей
программы:</p>
  <p>
    <textarea id="comment" name="comment" rows="4"
    cols="55"></textarea>
  </p>
  <input id="btnSave" name="btnSave" type="submit"
  value="Сохранить">
</form>
```



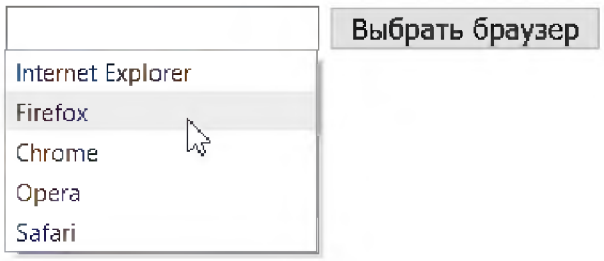
Списки. <select>, <datalist>, <optgroup> и <option>

С помощью тега <select> можно создать элемент интерфейса в виде выпадающего списка, а также список с одним или множественным выбором. Конечный вид списка зависит от использования параметра size тега <select>,

который устанавливает высоту списка. Ширина списка определяется самым широким текстом, указанным в теге <option>, а также может изменяться с помощью стилей. Каждый пункт создается с помощью тега <option>, который должен быть вложен в контейнер <select>. Если планируется отправлять данные списка на сервер, то требуется поместить элемент <select> внутрь формы. Это также необходимо, когда к данным списка идет обращение через скрипты. Тег <option> используется для задания отдельных пунктов списка, создаваемого с помощью контейнеров <select>, <optgroup> или <datalist>.

Тег <datalist>, так же как и <select>, определяет список, но используется внутри тега <input>. Пользователи будут видеть раскрывающийся список предопределенных параметров при вводе данных.

```
<form action="/action_page.php">
  <input list="browsers" name="browser">
  <datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
    <option value="Opera">
    <option value="Safari">
  </datalist>
  <input type="submit" value="Выбрать браузер">
</form>
```



При этом атрибут list тега <input> должен содержать id тега <datalist>.

Атрибуты тега <select>

Кроме атрибутов, присущих всем элементам формы, у тега <select> есть дополнительно атрибуты size, multiple и required.

Атрибут size устанавливает число видимых элементов списка и тем самым вид списка: если `size="1"`, то создается раскрывающийся список, во всех остальных случаях – развернутый список.

Атрибут-флаг multiple определяет, разрешен ли множественный выбор элементов списка.

Атрибуты тега <option>

Атрибут-флаг selected определяет, что данный пункт списка выделен по умолчанию.

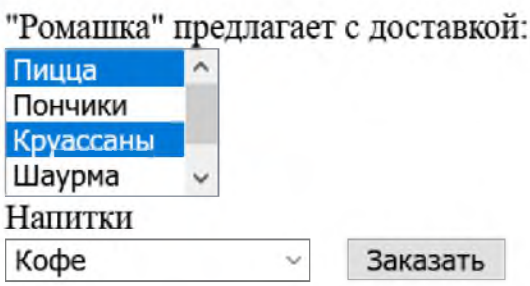
Атрибут label задает метку, которая используется для формирования иерархии списка.

Атрибут value определяет значение, которое будет отправлено на сервер в случае выбора текущего пункта списка. На сервер отправляется пара «имя=значение», где имя задается параметром name тега <select>, а значение – параметром value выделенных пунктов списка. Значение может как совпадать с текстом пункта, так и быть самостоятельным. Параметр value применяется также для получения значений данных через скрипты.

Атрибут disabled определяет, будет ли элемент доступен для пользователя.

Приведем пример создания списков.

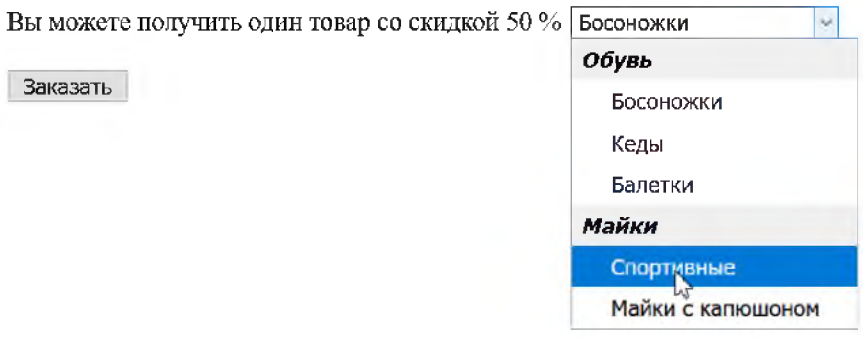
```
<form id="form1" name="form1" action="save.asp"
 method="get" enctype="application/x-www-form-
urlencoded" >
  "Ромашка" предлагает с доставкой:<br>
   <select id="menu" name="menu" size="4"
multiple="multiple">
    <option value="1" selected="selected">Пицца</option>
    <option value="2">Пончики</option>
    <option value="3"
selected="selected">Круассаны</option>
    <option value="4">Шаурма</option>
    <option value="5">Бутерброды</option>
  </select><br>
  Напитки<br>
   <select id="toys" name="toys" size="1">
    <option value="1">Минеральная вода</option>
    <option value="2" selected="selected">Кофе</option>
    <option value="3">Кола</option>
  </select>   
  <input type="submit" value="Заказать" />
</form>
```



Тег <optgroup> предназначен для создания многоуровневого списка и представляет собой контейнер, внутри которого располагаются теги <option>, объединенные в одну группу. Особенностью тега <optgroup> является то, что он выделяется с помощью полужирного начертания, а все элементы, входящие в этот контейнер, смещаются вправо от своего исходного положения. Иерархия в списке создается при помощи атрибутов label тегов <optgroup> и <option>. В атрибуте label тега <optgroup> записывается текст, который будет входить в список в виде заголовка группы, а в атрибуте label тега <option> – элемента группы.

Приведем пример двухуровневого списка.

```
<form action="/action_page.php">
  <label for="menu">Вы можете получить один товар со скидкой 50 %</label>
  <select name="menu" id="menu">
    <optgroup label="Обувь">
      <option value="b11">Босоножки</option>
      <option value="b12">Кеды</option>
      <option value="b13">Балетки</option>
    </optgroup>
    <optgroup label="Майки">
      <option value="b21">Спортивные</option>
      <option value="b22">Майки с капюшоном</option>
    </optgroup>
  </select>
  <br><br>
  <input type="submit" value="Заказать">
</form>
```



Группы. <fieldset>, <legend>

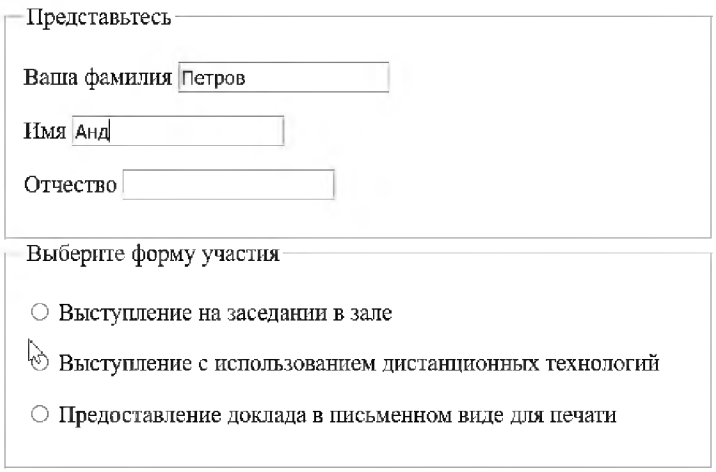
Тег <fieldset> предназначен для стилистической и семантической группировки элементов формы. Такая группировка облегчает работу с формами, содержащими большое число данных. Например, один блок может быть предназначен для идентификации пользователя, а другой –

для группы радиокнопок. Для наглядности браузеры отображают результат использования тега <fieldset> в виде рамки. Ее вид зависит от операционной системы и браузера. Тег <fieldset> не имеет атрибутов.

Тег <legend> формирует надпись на рамке вокруг группы. Он должен идти первым в теге <fieldset>.

Часть программ чтения с экрана произносят заголовок формы <legend> перед произношением названия меток элементов.

```
<form>
  <fieldset>
    <legend>Представьтесь</legend>
    <p>
      <label for="name3">Ваша фамилия</label>
      <input name="name3" id="name3">
    </p>
    <p>
      <label for="name1">Имя</label>
      <input name="name1" id="name1">
    </p>
    <p>
      <label for="name2">Отчество</label>
      <input name="name2" id="name2">
    </p>
  </fieldset>
  <fieldset>
    <legend>Выберите форму участия</legend>
    <p>
      <input type="radio" name="size" id="size_1"
value="room">
      <label for="room">Выступление на заседании
в зале</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_2"
value="dist">
      <label for="dist">Выступление с использованием
дистанционных технологий</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_3"
value="print">
      <label for="print">Предоставление доклада
в письменном виде для печати</label>
    </p>
  </fieldset>
</form>
```



Структурирование форм

Общепринятой практикой является оборачивание элемента формы и соответствующей ему метки в блочный тег или <p>. Можно также использовать для структурирования элементов формы списки. Так чаще поступают с флажками или радиокнопками.

Если форма достаточно большая, то каждый отдельный раздел функциональности можно помещать в теги <section> с элементами <fieldset>.

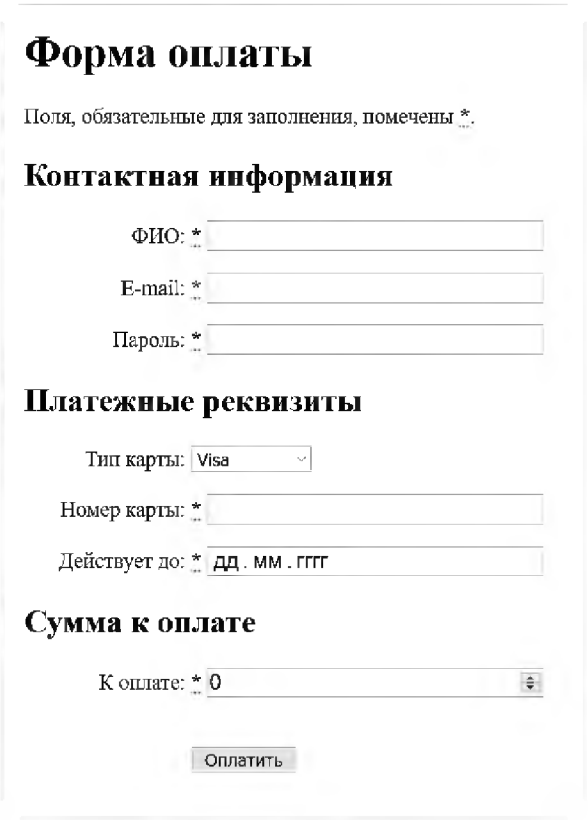
Ниже приведен пример формы оплаты с разделением на секции. Внутри тега <form> есть заголовок первого уровня и параграф, информирующий пользователей о том, как помечены поля, обязательные для заполнения. Поля с контактной информацией обернуты в отдельный элемент <section>. Заголовок второго уровня содержит информацию о содержании секции. В раздел входит три текстовых поля: для ввода ФИО, e-mail и пароля. С каждым текстовым полем связан соответствующий тег <label>. Каждое текстовое поле и связанная с ним метка обернуты в тег параграфа. Все поля, связанные с реквизитами оплаты, выделены еще в одну секцию. Сумма к оплате и кнопка также выделены в отдельные секции. Таким образом, вся форма поделена на четыре семантических части.

```
<!doctype html>
 <html>
  <head>
   <meta charset="utf-8">
   <title>форма</title>
   <style>
    h1 {
        margin-top: 0;
       }
    form {
          margin: 0 auto;
```

```
          width: 400px;
          padding: 1em;
          border: 1px solid #CCC;
          border-radius: 1em;
         }
label span {
            display: inline-block;
            width: 120px;
            text-align: right;
           }
input {
       font: 1em sans-serif;
       width: 250px;
       box-sizing: border-box;
       border: 1px solid #999;
      }
input:focus {
             border-color: #000;
            }
button {
        margin: 20px 0 0 124px;
       }
label {
       position: relative;
      }
label em {
          position: absolute;
          right: 5px;
          top: 20px;
         }
   </style>
  </head>
 <body>
 <form method="post">
  <h1>Форма оплаты</h1>
  <p>Поля, обязательные для заполнения, помечены
<strong><abbr title="Обязательно к
заполнению">*</abbr></strong>.</p>
  <section>
   <h2>Контактная информация</h2>
   <p>
    <label for="fio">
     <span>ФИО: </span>
     <strong>
      <abbr title="Обязательно к заполнению">*</abbr>
     </strong>
```

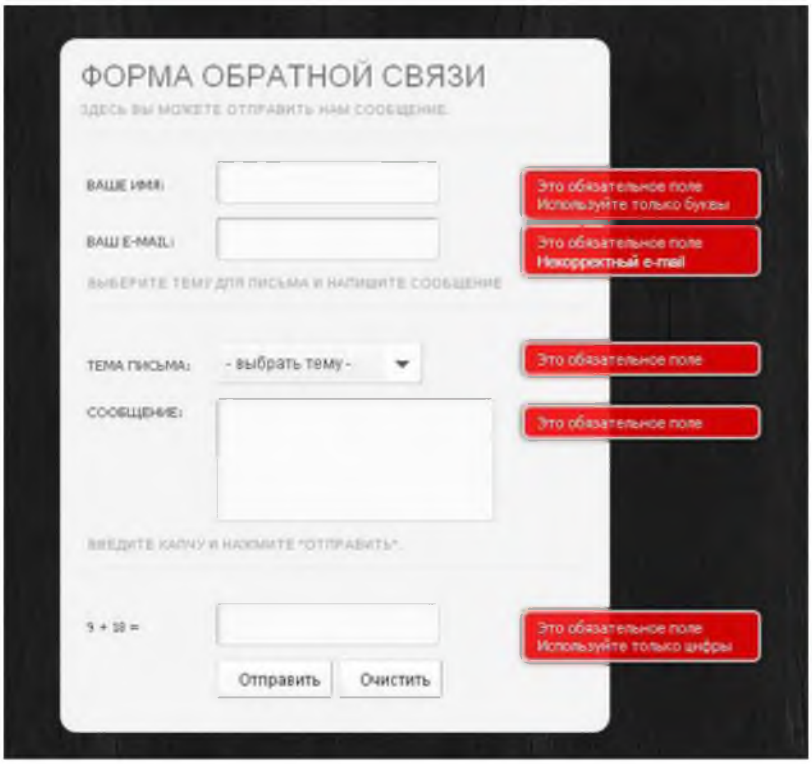
```
  </label>
  <input type="text" id="fio" name="fio">
 </p>
 <p>
  <label for="mail">
   <span>E-mail: </span>
   <strong>
    <abbr title="Обязательно к заполнению">*</abbr>
   </strong>
  </label>
  <input type="email" id="mail" name="usermail">
 </p>
 <p>
  <label for="pwd">
   <span>Пароль: </span>
   <strong>
    <abbr title="Обязательно к заполнению">*</abbr>
   </strong>
  </label>
  <input type="password" id="pwd" name="password">
 </p>
</section>
<section>
 <h2>Платежные реквизиты</h2>
  <p>
   <label for="card"><span>Тип карты:</span></label>
   <select id="card" name="usercard">
    <option value="visa">Visa</option>
    <option value="mc">Mastercard</option>
    <option value="belcart">БЕЛКАРТ</option>
   </select>
 </p>
 <p>
  <label for="number">
   <span>Номер карты:</span>
   <strong>
    <abbr title="Обязательно к заполнению">*</abbr>
   </strong>
  </label>
  <input type="tel" id="number" name="cardnumber">
 </p>
 <p>
  <label for="date">
   <span>Действует до:</span>
   <strong>
    <abbr title="Обязательно к заполнению">*</abbr>
```

```
      </strong>
     </label>
     <input type="date" id="date" name="date">
    </p>
   </section>
    <section>
    <h2>Сумма к оплате</h2>
    <p>
     <label for="sum">
      <span>К оплате: </span>
      <strong>
       <abbr title="Обязательно к заполнению">*</abbr>
      </strong>
      <input type="number" id="sum" name="sum" value="0"
      min="0" max="10000" step="0.01">
     </label>
    </p>
    </section>
    <section>
     <p><button type="submit">Оплатить</button> </p>
    </section>
   </form>
  </body>
</html>
```



Упражнения

1. Ниже представлены формы. Сверстайте одну из них на выбор.



Анкета посетителя ресторана

Контактная информация

Имя

Телефон

Email

Дата посещения

Персональная информация

Возраст

Любимая кухня — Русская

Какие блюда Вы хотели бы увидеть в меню?

Оценка нашего заведения

Почему Вы выбрали наше заведение?

- ○ Недалеко от дома/работы
- ○ Увидел рекламу
- ○ Посоветовали
- ○ Оптимальное соотношение цены и качества

Вы будете рекомендовать наше заведение своим знакомым?

- ○ Да
- ○ Нет

Отправить

2. Подготовьте несколько веб-страниц с формами для проведения теста. При построении теста используйте максимальное количество различных элементов формы.

CSS. Спецификация, версии языка

Каскадные таблицы стилей, или ***CSS*** (*Cascading Style Sheets*), – это язык описания таблиц стилей, позволяющий применять стили в структурированных документах, например в HTML- и XML-документах, а также задавать свойства элементов разметки (тегов), которые не могут быть заданы с помощью стандартных атрибутов HTML-тегов. CSS позволяет приводить к единому внешнему виду целые группы HTML-документов, даже если это и не было предусмотрено при их создании. Это делает стиль представления документов независимым от их содержания, что существенно упрощает разработку веб-страниц и поддержку сайтов.

На сайте w3schools.com есть прекрасная демонстрация применения различных стилей к простейшей странице: https://w3schools.com/css/css_intro.asp.

В 1994 г. Хокон Виум Ли из Норвегии предложил концепцию каскадных таблиц стилей. Язык CSS был разработан и дальше разрабатывается рабочей группой W3C, которая называется CSS Working Group. Эта группа состоит из представителей производителей браузеров и других компаний, которые заинтересованы в развитии CSS, и приглашенных экспертов.

Новые возможности CSS разрабатываются по различным причинам: это может быть запрос какого-либо конкретного браузера, который хочет иметь определенные возможности; запрос веб-дизайнеров и разработчиков; обобщение уже реализованных возможностей в браузерах; решение самой рабочей группы. CSS постоянно развивается, появляются новые функции. Основное требование к обновлениям заключается в том, что все старые сайты, созданные с использованием старых версий CSS, доступных на время создания сайта, должны работать без изменения дизайна.

CSS стандартизован спецификацией W3C: https://www.w3.org/Style/CSS/#specs. Стандарт CSS делится на уровни: CSS1, CSS2.1 и CSS3.

Спецификация CSS все время расширяется и, как следствие, увеличивается в объеме. Описание версии CSS2.1 было в пять раз больше по сравнению с описанием предыдущей версии, поэтому в стандарте CSS3 все функции распределили по наборам индивидуальных стандартов, называющихся модулями. Всем этим новым модулям CSS было присвоено коллективное название CSS3. Данный стандарт действует в настоящее время.

Спецификация CSS3 содержит около 50 модулей разной степени завершенности. Эти модули содержат как возможности, поддерживаемые самыми последними версиями всех современных браузеров, так и экспериментальные возможности.

Очень редко бывает так, что новое свойство, определенное в CSS, оказывается сразу реализованным во всех браузерах, поэтому для каждого нового свойства нужно проверять, какие браузеры его поддерживают.

Полезные ссылки

Стандарт CSS – спецификация W3C:
https://www.w3.org/Style/CSS/#specs.
Справочник по поддержке браузерами свойств CSS:
https://w3schools.com/cssref/css3_browsersupport.asp.
Учебники:
http://w3schools.com/css/default.asp/;
http://http://htmlbook.ru/samcss;
https://webref.ru/course/css-basics;
https://developer.mozilla.org/ru/docs/Learn/CSS;
https://professorweb.ru/my/css/css_theory/level1/css_index.php;
https://metanit.com/web/html5/5.1.php.
Справочники:
http://www.w3schools.com/cssref/default.asp;
http://htmlbook.ru/css;
https://developer.mozilla.org/ru/docs/Web/CSS/Reference.
Практикумы:
https://htmlacademy.ru/;
https://codecademy.com/learn.

Типы стилевых таблиц

Основная особенность CSS заключается в том, что таблицы стилей объединяются и именно результирующее правило в итоге определяет окончательное представление документа.

Когда браузер загружает HTML-документ, он читает таблицу стилей и форматирует страницу в соответствии с информацией в таблице. Существуют три способа добавления таблицы стилей к документу:

- внешние CSS-таблицы;
- внутренние CSS-таблицы;
- встроенный CSS-стили.

Как видно, таблицы стилей делятся на группы по расположению.

Первая группа – ***внешние***, или связанные, таблицы стилей. Внешние таблицы стилей расположены в отдельном файле с расширением ****.css***. Они могут применяться к любому HTML-документу, к которому подключены.

Вторая группа – ***внедренные*** таблицы стилей. Внедренные таблицы стилей расположены внутри HTML-документа, в его головной части, в теге <style>. Правила, описанные в такой таблице, распространяются на элементы разметки всего документа.

Третья группа – ***встроенные*** таблицы стилей. Встроенные таблицы стилей вставляются непосредственно в элемент разметки посредством атрибута style. Область применения таких стилей ограничена тем тегом, в котором они определены.

Внешние таблицы стилей

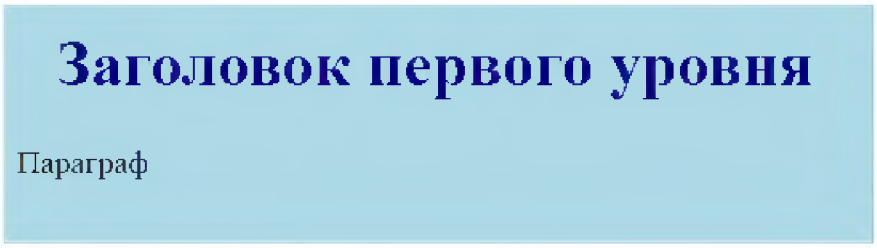
Внешние таблицы стилей являются наиболее распространенным и одновременно имеющим наибольшую общность типом. Связанная таблица стилей располагается в отдельном файле, имеющем расширение *.css. Она может быть одновременно подключена к целому ряду HTML-документов. Изменение стилевого правила в такой таблице повлечет изменение дизайна всех веб-страниц, к которому та подключена. К одному HTML-документу могут быть подключены несколько внешних стилевых таблиц. В одном html-документе могут одновременно работать стилевые правила, определенные и во внешней, и во внутренней, и во встроенных таблицах стилей. Внешние таблицы стилей могут создаваться в любом текстовом редакторе. Они подключаются к HTML-документу в теге <link>. Для этого в теге <link> значение атрибута rel должно быть установлено в stylesheets, а в атрибуте href должен быть указан сетевой адрес таблицы стилей. Внешний файл .css не должен содержать никаких HTML-тегов. Ниже приведен пример подключения внешнего стилевого файла style1.css, который расположен на диске в той же директории, что и веб-страница. Первой строкой в стилевой файл всегда добавляется @-правило. Каждое из правил имеет свой синтаксис. Правило @charset определяет кодировку символов, используемую в таблице стилей.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="style1.css">
    <title>Типы таблиц стилей</title>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
```

```
    <p>Параграф</p>
  </body>
</html>
```

Текст файла style1.css:

```
@charset "utf-8";
/* CSS Document */
body {
    background-color: lightblue;
}
h1 {
    color: navy;
    margin-left: 20px;
}
```



Внешние таблицы стилей позволяют веб-страницам загружаться быстрее. Когда они используются, веб-страницы содержат только HTML-код без громоздкого кода внутренних таблиц стилей и без встроенных стилей. Когда браузер загрузит внешнюю таблицу стилей при загрузке первой страницы сайта, он сохранит этот файл на клиентском компьютере посетителя веб-страницы в специальной системной папке, называемой кэшем, для быстрого доступа к нему. Когда посетитель веб-страницы переходит к другим страницам сайта, которые используют ту же внешнюю таблицу стилей, браузеру уже нет необходимости снова загружать таблицу стилей. Он попросту загружает запрашиваемый HTML-файл и использует внешнюю таблицу стилей из своего кэша, что дает существенный выигрыш во времени загрузки страниц.

Внутренние таблицы стилей

Внутренние, или внедренные, таблицы стилей располагаются прямо в HTML-документе, а правила, описанные в них, распространяются на весь документ. В такой таблице размещают уникальные стили, предназначенные только для этой страницы.

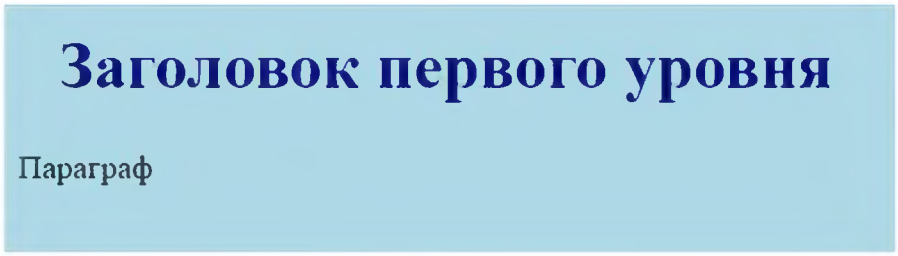
Внутренняя таблица стилей помещается внутри тега <style>. Тег <style> помещается в заголовочную часть HTML-документа.

Применение внутренних таблиц стилей не так эффективно, как использование внешних таблиц: если потребуется внести изменения, то будет необходимо прописывать стилевые правила отдельно для каждой страницы.

Можно поместить элемент style и все его стили после элемента title, но веб-дизайнеры обычно размещают их прямо перед закрывающим тегом </head>. Однако, если на странице используется JavaScript-код, он должен располагаться после таблиц стилей. Часто JavaScript-сценарии используют CSS, поэтому, добавляя таблицы стилей первыми, можно гарантировать, что JavaScript-код будет иметь все необходимые для своего выполнения данные.

Старые версии браузеров не всегда поддерживают внутренние таблицы стилей. Для того чтобы скрыть от них стили, необходимо взять тело таблицы стилей в HTML-комментарий.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
    <!--
    body {
      background-color: lightblue;
      }
    h1 {
      color: navy;
      margin-left: 20px;
    }
    -->
    </style>
    <title>Типы таблиц стилей</title>
  </head>
  <body>
    <h1>Заголовок первого уровня</h1>
    <p>Параграф</p>
  </body>
</html>
```



Встроенные стили

Встроенные стили имеют наименьшую область применения. Их область действия распространяется только на тот тег, в котором они определены. Для определения стиля используется атрибут style. Встроенные

стили активно используются в генераторах HTML-кода. Если на странице много встроенных таблиц стилей, то применение связанных таблиц стилей к такой странице может не принести должного эффекта. Встроенные стили категорически не рекомендуется использовать. Их применение делает код трудным для технического обслуживания при внесении изменений и, кроме того, тяжелым для чтения и понимания. Применение встроенных стилей оправданно только в том случае, если нужно применить такое правило к одному тегу одного документа и перекрыть CSS-правила из других способов подключения или в тестовых целях. Причем для элемента определено много противоречивых правил во внешних и внутренних стилевых таблицах, и достаточно долго просчитывать приоритет стилей по правилам каскадности. В данном случае для всего документа определен темно-синий цвет фона и белый цвет шрифта текста в элементе <body>. Во вложенном теге эти свойства переопределены на желтый и голубой соответственно. Поскольку при сложении стилей действует принцип «своя рубашка ближе к телу», очевидно, что для блока действуют правила, определенные непосредственно в нем.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="style.css">
  </head>
  <body style="background-color:darkblue; color:white">
    Мне нравится белый текст на темном фоне.
    <div style="background-color:yellow; color:blue;
    border-style:dashed;          border:4;     border-
color:crimson">
    Но иногда удобнее читать на светлом фоне
  </div>
</body>
</html>
```

Упражнение

Дан код страницы.

```
<!doctype html>
<html>
  <head>
```

```
    <meta charset= "utf-8">
  </head>
  <body>
    <ul>
      <li><a href="#">Ссылка 1</a></li>
      <li><a href="#">Ссылка 2</a></li>
      <li><a href="#">Ссылка 3</a></li>
      <li><a href="#">Ссылка 4</a></li>
    </ul>
    <p>В приведенном выше примере создан немаркированный
список, каждый элемент которого содержит ссылку.
Горизонтальное выравнивание самого списка и ссылок по
левому краю. Элементы li будут отображаться в виде
строчных inline-элементов. Это заставляет список
находиться на одной линии. Элемент ul имеет ширину 100 %,
и шрифт каждой гиперссылки в списке в 1,5 раза превышает
размер текущего шрифта.
    </p>
  </body>
</html>
```

Который при просмотре в браузере имеет следующий вид:

- Ссылка 1
- Ссылка 2
- Ссылка 3
- Ссылка 4

В приведенном выше примере создан немаркированный список, каждый элемент которого содержит ссылку. Горизонтальное выравнивание самого списка и ссылок по левому краю. Элементы li будут отображаться в виде строчных inline-элементов. Это заставляет список находиться на одной линии. Элемент ul имеет ширину 100 %, и шрифт каждой гиперссылки в списке в 1,5 раза превышает размер текущего шрифта.

Применим к нему стили, описанные ниже:

```
ul {
    float: left;
    width: 100%;
    padding: 0;
    margin :0;
    list-style-type: none;
  }
a {
    float: left;
```

```
    font-size: 1.5em;
    text-decoration: none;
    color: white;
    background-color: coral;
    padding: 0.2em 0.6em;
    border-right: 1px solid white;
    }
a:hover {
    background-color: #ff3300;
    }
li {
    display: inline;
   }
```

Получим следующий вид:

В приведенном выше примере создан немаркированный список, каждый элемент которого содержит ссылку. Горизонтальное выравнивание самого списка и ссылок по левому краю. Элементы li будут отображаться в виде строчных inline-элементов. Это заставляет список находиться на одной линии. Элемент ul имеет ширину 100 %, и шрифт каждой гиперссылки в списке в 1,5 раза превышает размер текущего шрифта.

Исправьте код, присоединив стили и используя:

1) внешние таблицы стилей;
2) внутренние таблицы стилей.

Каскадность

Каскадность – это набор правил, который определяет, каким образом браузер должен отображать многократно определенные стили, относящиеся к одному и тому тегу.

Если некоторые свойства были определены для одного и того же селектора (элемента) в разных таблицах стилей, будет использоваться значение из последней прочитанной таблицы стилей.

Например, есть внешняя таблица стилей style.css, в которой определен стиль для параграфа «темно-синий фон»:

```
h1 {
    background-color: darkblue;
}
```

А во внутренней таблице стилей для параграфа задан оранжевый фон:

```
<style>
h1 {
```

```
    color: orange;
}
</style>
```

Если в HTML-документе сначала подключен внешний файл, а потом определена внутренняя таблица стилей, то фон будет как во внутренней таблице – оранжевый.

```
<link rel="stylesheet" href="style.css">
<style>
  h1 {
      color: orange;
}
</style>
```

Если в HTML-документе сначала определена внутренняя таблица стилей, а потом подключен внешний файл, то фон будет как во внешней таблице – темно-синий.

Если для HTML-элемента задано более одного стиля, то при обработке стилей браузер следует двум основным правилам:

- сначала все правила стилей, определенные для элементов страницы, складываются (они могут быть описаны и в различных таблицах), и только после этого применяется итоговый стиль;
- при сложении стилей приоритет имеют атрибуты, определенные в стиле с более высоким приоритетом.

По способу подключения по приоритетности стили можно разделить на три группы:

1) высший приоритет имеют встроенные стили, определенные в теге через атрибут style;

2) потом идут стили из внешних и внутренних таблиц стилей с учетом порядка их подключения в разделе заголовка;

3) самый низкий приоритет имеют стили браузера.

Таким образом, встроенный стиль имеет наивысший приоритет и переопределяет внешние и внутренние стили и настройки браузера по умолчанию.

Модификатор !important

Вес правила можно задать с помощью ключевого слова !important, которое добавляется сразу после значения свойства, например:

```
p {
   font-weight: bold!important;
}
```

Правило необходимо размещать в конце объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Модификатор !important усложняет отладку и нарушает естественное каскадирование стилей, поэтому его использование не рекомендуется. Однако этот способ позволяет отменить значение свойства и установить новое для элемента из группы элементов, в случае когда нет прямого доступа к файлу со стилями.

Специфичность

В рамках одного способа подключения приоритет CSS-стилей определяется расчетом специфичности: https://www.w3.org/TR/selectors/#specificity. Специфичность – это способ, с помощью которого браузеры определяют, какие значения свойств CSS наиболее соответствуют элементу и, следовательно, будут применены. Специфичность выражается числом, которое рассчитывается для каждого селектора. Это число представляет собой вес, придаваемый конкретному стилевому правилу. Чем показатель специфичности больше, тем приоритет выше. В теории чем выше специфичность, тем более ограниченное количество тегов выбирает селектор.

При расчете специфичности третий разряд числа представляет собой количество id в селекторе (#example).

Второй – количество селекторов классов (например, .example), селекторов атрибутов (например, [type="email"]) и псевдоклассов (например, :hover).

Первый разряд числа – это количество HTML-элементов (p, div, a…) или псевдоэлементов (::after).

Разряд	3	2	1
Коэффициент			

Универсальный селектор (*), комбинаторы (+, >, ~, ' ') и отрицающий псевдокласс (:not()) не влияют на специфичность.

Приведем примеры расчета специфичности:

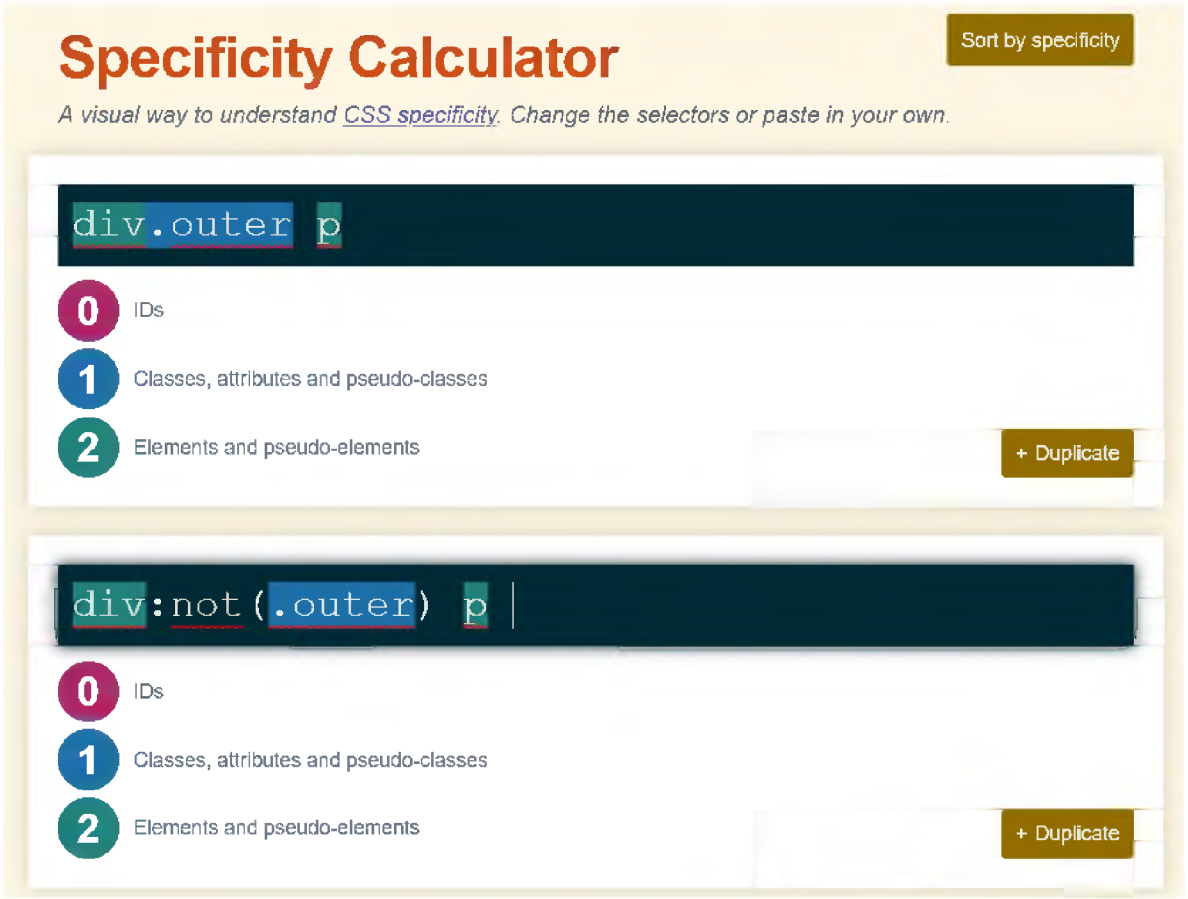
Спецификатор	**Коэффициент специфичности**
aside h1	0-0-2
*#price	1-0-0
#price	1-0-0
ol li a.book	0-1-3

Если при объявлении стиля используется модификатор !important, то этот стиль получает наивысший приоритет среди всех прочих объявлений. Хотя технически модификатор !important не имеет со специфичностью ничего общего, он непосредственно на нее влияет.

Отрицающий псевдокласс :not не учитывается как псевдокласс при расчете специфичности. Однако селекторы, расположенные внутри :not, при подсчете количества по типам селекторов рассматриваются как обычные и учитываются. Так, например, оба стиля, приведенные ниже, имеют одинаковую специфичность 012.

```
div.outer p {color: orange;}
div:not(.outer) p {color: lime;}
```

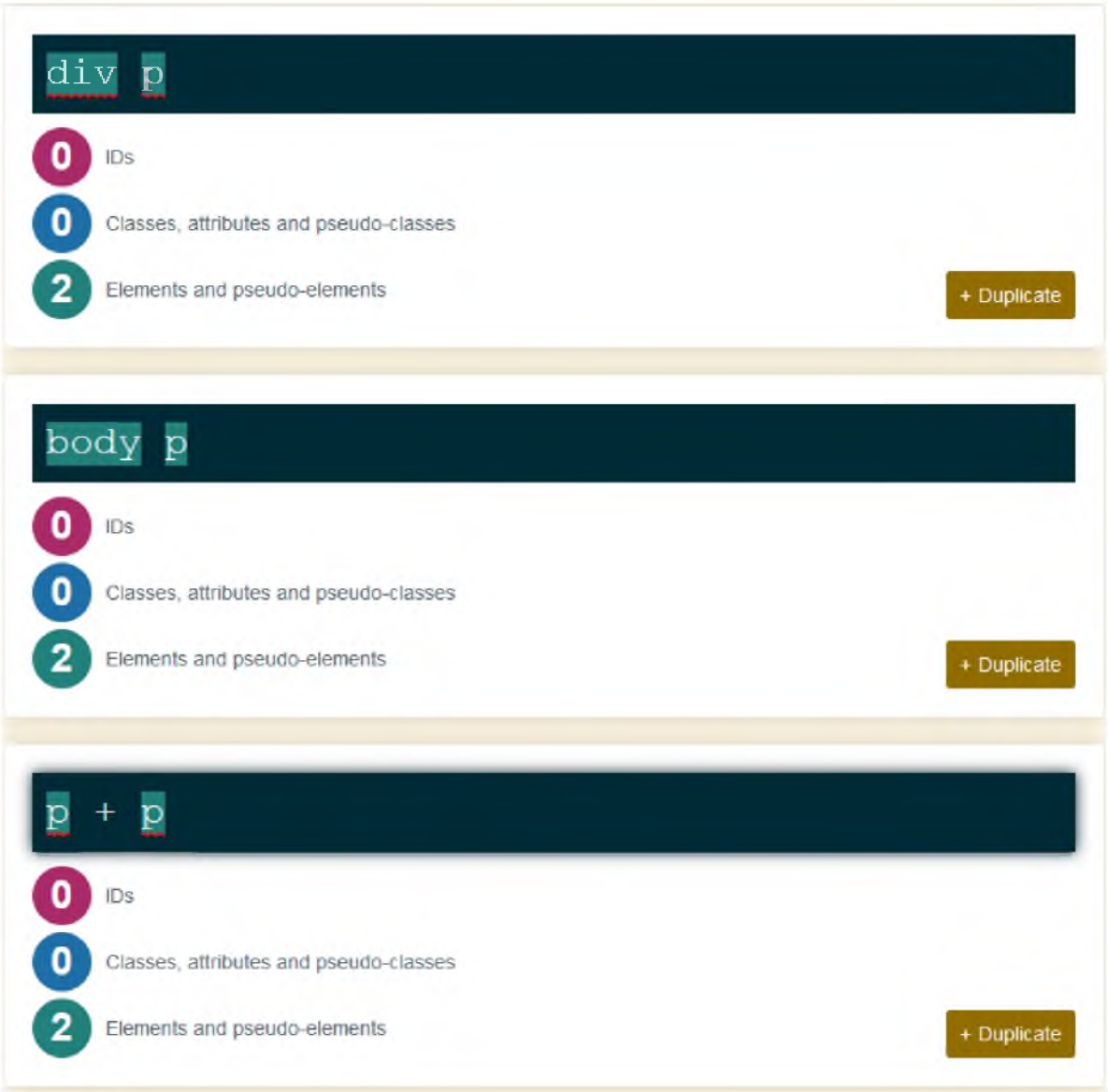
Для подсчета специфичности можно пользоваться онлайн-калькуляторами специфичности, например https://specificity.keegan.st/.



При наличии разных правил с одинаковой специфичностью действует объявленное последним.

```
div p {color: red; border: 1px solid black;}
body p {text-align: center; color: yellow;}
p + p {background-color: blue; color: pink;}
```

Для всех трех селекторов коэффициент специфичности – 0-0-2.



Для тега <p>, который захватится всеми этими селекторами, сформируется следующий список свойств:

```
border: 1px solid black;
text-align: center;
background-color: blue;
color: pink;
```

Наследование

Наследование – это механизм, с помощью которого определенные свойства передаются от предка к его потомкам.

Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как color, font, letter-

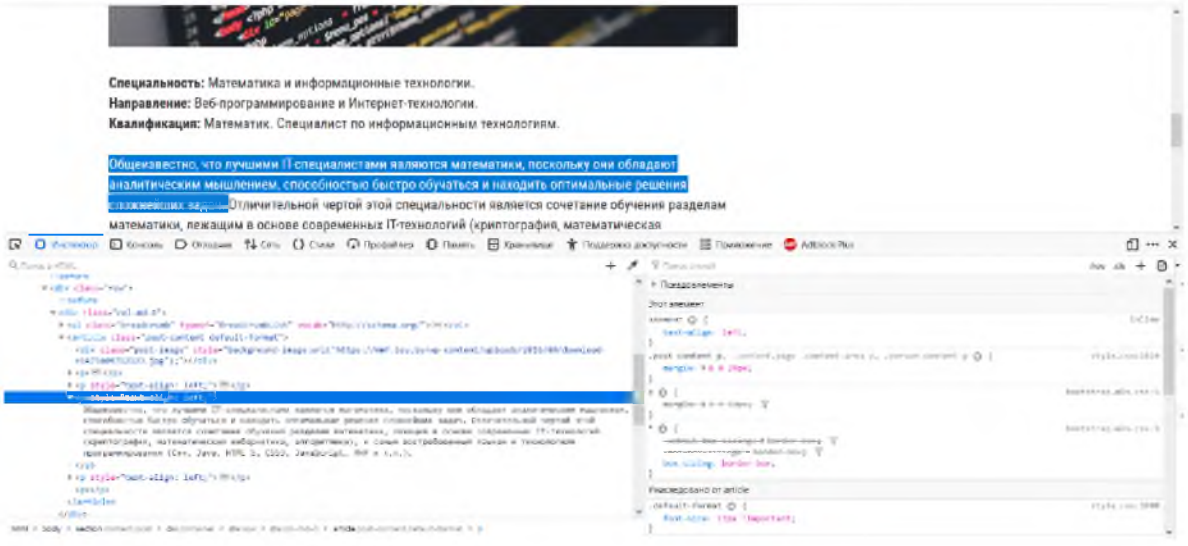
spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space и word-spacing. Во многих случаях это удобно, так как не нужно многократно задавать повторяющиеся стили для вложенных элементов, например размер шрифта и семейство шрифтов для каждого элемента веб-страницы.

Однако наследуются не все свойства. Например, свойства, относящиеся к форматированию блоков, не наследуются. Это background, border, display, float и clear, height и width, margin, min-max-height и -width, outline, overflow, padding, position, text-decoration, vertical-align и z-index.

Можно принудить элемент наследовать любое значение свойства родительского элемента. Для этого используется ключевое слово inherit. Принудительное наследование свойства родителя при помощи ключевого слова inherit работает даже для тех свойств, которые не наследуются по умолчанию.

Просмотр стиля в «Инструментах разработчика» браузера

К одному элементу могут применяться стили из разных источников. Проверить, какие стили применяются, можно в режиме разработчика браузера. Для этого над элементом нужно щелкнуть правой кнопкой мыши и выбрать пункт «Исследовать элемент», «Посмотреть код элемента» или что-то аналогичное в зависимости от браузера. Откроются «Инструменты разработчика». В правом столбце будут перечислены все свойства, которые заданы для этого элемента или наследуются от родительского элемента.



Синтаксические правила CSS

CSS – это язык на основе правил: с помощью CSS задаются правила, определяющие группы стилей, которые должны применяться к определенным элементам или группам элементов на веб-странице. Файлы, содержащие каскадные таблицы стилей, – это текстовые файлы, имеющие расширение *.css. Каждая таблица стилей состоит из одного или нескольких правил. Правило состоит из двух частей: селектора и описания. ***Селектор*** – это название элемента разметки, к которому будет применяться правило, а ***описание*** – это задание значений одному или более свойствам выбранного элемента разметки. Описание заключается в фигурные скобки. Каждое определение, входящее в описание, в свою очередь, также состоит из двух частей: *свойства* и *значения*. Свойство отделяется от значения двоеточием. CSS-свойства имеют разные допустимые значения в зависимости от того, какое свойство указывается. Если описание содержит определение более чем одного свойства, то определения различных свойств отделяются друг от друга точкой с запятой. После последнего определения точку с запятой ставить необязательно, но это не является ошибкой.

Формат правила имеет следующий вид:

```
селектор { свойство : значение}
```

Для нескольких свойств:

```
селектор
        {
          свойство1 : значение1;
          свойство2 : значение2;
                    …
          свойствоN : значениеN
         }
```

Например, для тега параграфа <p> устанавливается желтый цвет фона, синий цвет текста и размер шрифта 18 пикселей.

```
p{
  background:yellow;
  font-size:18px;
  color:blue}
```

Таблицы стилей, так же как и HTML, нечувствительны к регистру, т. е. строчные и прописные символы не различаются, за исключением тех частей стилевых правил, которые не являются объектами языка CSS.

Комментарии начинаются с символов /* и заканчиваются символами */.

Значения свойств, являющихся адресами, устанавливаются следующим образом:

```
url(адрес)
```

Например, при установке изображения bg.gif в качестве фона для блока тега правило имеет вид

```
div { background:url(bg.gif) }
```

При установке различных свойств одному объекту свойства могут группироваться, при этом значения свойств в группе отделяются друг от друга пробелом. Например, следующие правила для установки свойств шрифта в теле HTML-документа эквивалентны:

```
body
 {
  font-family:Arial, Helvetica, sans-serif;
  font-size:10px;
  font-style:italic; }
```

и

```
body { font:Arial, Helvetica, sans-serif 10px italic }
```

Первое правило использовать предпочтительнее, так как если вы сделали ошибку в назначении свойства, то строка, в которой есть незнакомое для браузера значение свойства, проигнорируется. Таким образом, в первом случае будет проигнорировано одно свойство из трех, а во втором – все три свойства.

Селекторы

Селектор – это формальное описание элемента HTML, группы элементов или части элемента. Селектор определяет, к каким элементам DOM будет применяться стилевое правило.

Селекторы по сущностям, участвующим в описании, условно можно разбить на группы:

- селекторы элементов;
- универсальный селектор;
- селекторы классов и id;
- селекторы на основе их расположения;
- селекторы атрибутов;
- селекторы псевдоклассов;
- селекторы псевдоэлементов.

При определении стиля можно использовать любую комбинацию селекторов.

В качестве селектора могут быть указаны теги. Например, ниже для тега параграфа <p> устанавливается желтый цвет фона, синий цвет текста и размер шрифта 18 пикселей.

```
p{
  background:yellow;
```

```
  font-size:18px;
  color:blue}
```

Атрибут id предназначен для выбора конкретного элемента, уникального в пределах страницы, поэтому селектор идентификатора используется, чтобы выбрать один уникальный элемент. Чтобы выбрать элемент с определенным идентификатором, надо перед идентификатором элемента указать хеш-символ (#). id не может начинаться с цифры. В примере ниже параграф имеет id="par1". Для него определено правило: текст выравнивается по центру, цвет шрифта – красный.

```
#par1 {
    text-align: center;
    color: red;}
<p id="par1">Параграф определяется по id</p>
```

CSS-классы – это предопределенные стили на уровне тега. Класс стиля можно представлять как обычный стиль, имеющий имя. Это означает, что для каждого тега можно создать несколько стилей, каждый из которых будет иметь свое уникальное имя. При указании класса в селекторе после названия тега ставят точку, а затем указывают имя класса. Для применения стиля в атрибуте class соответствующего тега необходимо указать название класса.

```
.par2 {
    text-align: center;
    color: red;}
<p class="par2">Параграф определяется через class</p>
```

Можно также указать, что стиль относится не ко всем элементам, имеющим атрибут class с определенным значением, а только к конкретным тегам. Для этого начните с имени элемента, затем введите символ точки (.), а затем имя класса: например, в этом примере только те теги параграфа <p>, которые помечены как class="center", будут выровнены по центру. Комментарии используются для пояснения кода и могут помочь при его редактировании спустя какое-то время. Комментарии игнорируются браузерами. Комментарии си подобные /**/.

```
div.par2 {
/* Меняем цвет шрифта*/
    color:aqua;
/* Разместим два блока,
К одному применим class="par2"*/}
…
<div class="par2">Блок определяется через class</div>
<div>Определяется блок</div>
```

Блок определяется через class

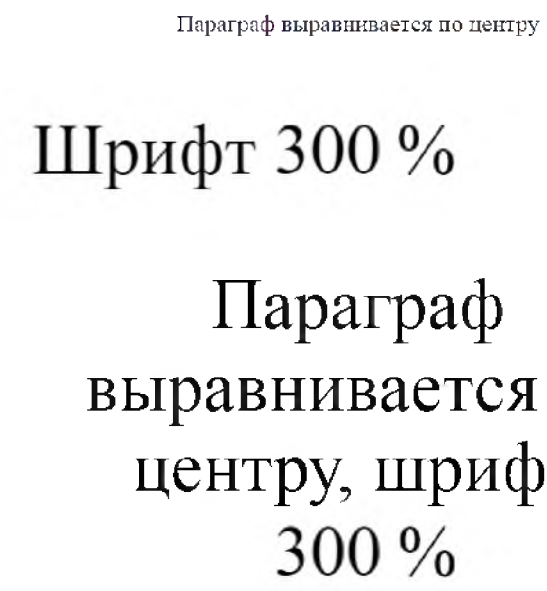
Определяется блок

В селекторах можно ссылаться на несколько классов. Применим стили:

```
.center {text-align: center;};
.large {font-size: 300%;}.
```

Для кода:

```
<p class="center">Параграф выравнивается по центру</p>
<p class="large">Шрифт 300%</p>
<p class="center large">Параграф выравнивается по центру,
шрифт 300 %</p>
```



Универсальный селектор * позволяет выбрать все элементы веб-страницы вместо указания каждого тега по отдельности. Например, если нужно, чтобы все отображалось полужирным шрифтом, можно добавить следующий код:

```
a, p, img, h1, h2, h3, h4, h5, ... и т. д. (тут надо
перечислить все теги документа) { font-weight: bold; }
```

Использование символа * – более быстрый способ сообщить CSS о выборке всех HTML-элементов веб-страницы:

```
* {font-weight: bold;}
```

Довольно часто веб-дизайнеры используют универсальный селектор * как способ отмены отступов вокруг блочных элементов. С помощью свойства margin можно добавить отступы вокруг элемента, а с помощью свойства padding – пространство между рамкой элемента и его содержимым. Для разных элементов браузеры автоматически формируют отступы различного размера, поэтому один из способов начать с чистого листа заключается в том, чтобы удалить все пространство вокруг элементов с определенным стилем:

```
* {
padding: 0;
margin: 0;
}
```

Кроме того, можно использовать универсальный селектор в составе селектора потомков, также называемого вложенным селектором. В этом случае стиль применяется ко всем элементам-потомкам, подчиненным определенному элементу веб-страницы. Например:

```
.banner * { font-weight: bold; }
```

выбирает все элементы внутри элемента, имеющего атрибут class="banner".

Упражнения

1. Напишите селектор, предназначенный для элементов списка на странице.
2. Напишите селектор для элемента с идентификатором main.
3. Какая ошибка содержится в следующем коде?

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<style>
#_nav{
 font-weight: 500;
 font-size: 24px;
 line-height: 28px;
 text-decoration: none; }
</style>
</head>
<body>
...
<nav class="asidemnu">
 <div id="_nav">
  <a href="Introduction.html">Введение   в CSS</a>
 </div>
 <div id="_nav">
  <a href="Types.html">Типы таблиц стилей</a>
 </div>
 <div id="_nav">
  <a href="Rules.html">Синтаксические правила CSS</a>
 </div>
</nav >
...
</body>
```

```
</html>
```

4. На веб-странице есть элемент section, в котором размещены только изображения с подписями. Каждое изображение и подпись к нему обернуты в тег figure. Составьте селекторы (не менее трех) для выбора картинки и подписи.
5. Составьте селектор для всех элементов внутри элемента с идентификатором test.
6. * p.p * {color: blue;} определяет для всех элементов внутри абзацев с классом p синий цвет. Объясните селектор. Замените селектор * p.p * равнозначным.
7. Напишите селектор для элементов с классом important.
8. Напишите селектор для элементов с классами important и urgently.

Группировка селекторов

Если одинаковый стиль назначается для нескольких элементов разметки, то их можно сгруппировать. Для этого эти теги должны быть перечислены через запятую в селекторе. Например, есть три одинаковых стилевых правила для заголовка первого уровня, заголовка второго уровня и параграфа:

```
h1 {
    text-align: center;
    color: red;
}
h2 {
    text-align: center;
    color: red;
}
p {
    text-align: center;
    color: red;
}
```

Их можно заменить одним правилом

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

Упражнение

Составьте селектор для всех нумерованных и всех немаркированных списков.

Целые и вещественные числа

Некоторые типы значений могут быть выражены в целых или вещественных числах. Вещественные и целые числа представляются только в десятичной системе счисления. Значения типа «целое число» состоят из одной или нескольких цифр от 0 до 9. Значения типа «вещественное число» могут быть целым числом и состоять из группы цифр, разделенных точкой. Вещественные и целые числа могут быть как положительными, так и отрицательными.

Единицы измерения длины

В CSS есть несколько различных единиц измерения длины. Длина представляет собой число с последующим указанием единицы длины, например 15px, 2em, 3 % и т. д. Пробел между значением длины и единицей измерения не ставится. Есть единственное исключение: если значение равно 0, то единицу измерения можно не указывать. Для некоторых свойств CSS разрешена отрицательная длина.

В CSS существуют абсолютные и относительные единицы измерения длины.

Абсолютные единицы измерения используются в том случае, когда известны физические параметры устройства вывода. К абсолютным единицам измерения относятся:

- in – дюйм (1 дюйм равен 2,54 сантиметра);
- cm – сантиметр;
- mm – миллиметр;
- pt – пункт (1 пункт равен 1/72 дюйма);
- pc – пика, типографская мера (1 пика равна 12 пунктам, или 4,23 миллиметра);
- px – пиксели (точки на экране компьютера).

Самой распространенной и широко используемой единицей измерения является пиксель.

Количество пикселей задается в настройках разрешения экрана, один px – это как раз один такой пиксель на экране. Все значения браузер в итоге пересчитает в пиксели.

Значения пикселей могут быть дробными: так, размер можно задать в 16.5px. Например, есть элемент шириной в 100px, который нужно разделить на три части, отсюда появляются 33.3px. При окончательном отображении дробные пиксели округляются и становятся целыми.

Для мобильных устройств, у которых много пикселей на экране, но сам экран маленький, чтобы обеспечить читаемость, браузер автоматически применяет масштабирование.

В реальной жизни пиксель может быть разным в зависимости от экрана. Однако в CSS под пикселем понимается абстрактный пиксель, или точка на «стандартизованном экране», характеристики которого описаны в спецификации. Согласно спецификации в одном сантиметре содержится ровно 38 пикселей. Поэтому ни о каком соответствии cm реальному сантиметру говорить нельзя. Это полностью синтетическая и производная единица измерения. Из вышесказанных соображений mm, cm, pt, pc уже практически не используются.

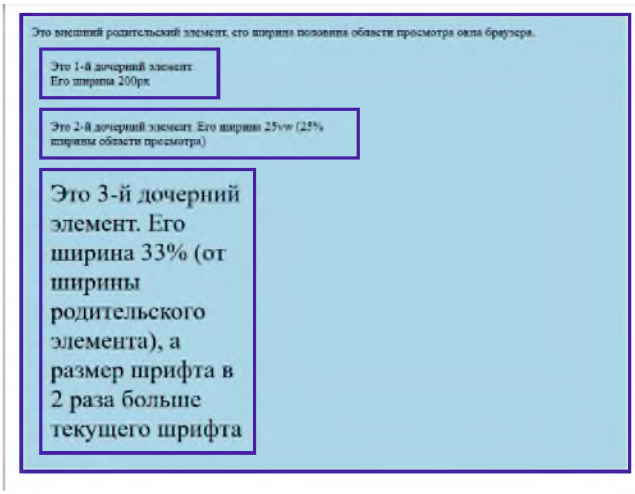
Относительные единицы длины выражают отношение к чему-то другому, например к размеру шрифта родительского элемента или области просмотра. Преимущество использования относительных единиц состоит в том, что при тщательном планировании можно сделать так, чтобы размер текста или других элементов масштабировался относительно всего остального на странице. Таблицы стилей, которые используют такой тип единиц, намного легче перенастраиваются с одного типа устройств на другой, например, с экрана компьютера на лазерный принтер и т. д. К относительным единицам измерения относятся:

- ch – относительно ширины символа "0" шрифта элемента;
- em – относительно размера шрифта родительского элемента, 2em означает в 2 раза больше размера текущего шрифта;
- ex – относительно высоты буквы x (икс) в латинском алфавите текущего шрифта (редко используется);
- rem – относительно размера шрифта корневого элемента <html>;
- vw – относительно 1 % от ширины области просмотра. Область просмотра (viewport) = размер окна браузера. Если окно просмотра имеет ширину 50 см, 1vw = 0,5 см;
- vh – относительно 1 % от высоты области просмотра (viewport);
- vmin – относительно 1 % от минимального размера области просмотра (наименьшее из vw и vh);
- vmax – относительно 1 % от максимального размера области просмотра (наибольшее из vw и vh);
- % – относительно размера родительского элемента.

Приведем пример.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Документ без названия</title>
```

```
    <style>
      .box { border: solid thick #461EA4;
             background: #ABD6E7;
             padding: 10px;  margin: 10px;}
            .wrapper {width: 50%;}
            .px {width: 200px;}
            .vw {width: 25vw;}
            .em {font-size: 2em; width: 33%;}
    </style>
  </head>
  <body>
    <div class="box wrapper">Это внешний родительский
      элемент, его ширина – половина области просмотра
окна
      браузера.
      <div class="box px">Это 1-й дочерний  элемент.
        Его ширина – 200px
      </div>
      <div class="box vw">Это 2-й дочерний  элемент. Его
        ширина – 25vw (25 % ширины области просмотра)
      </div>
      <div class="box em">Это 3-й дочерний  элемент.
       Его  ширина  –  33  %  (от  ширины  родительского
элемента),
       а размер шрифта в 2 раза больше текущего
      </div>
    </div>
  </body>
</html>
```



В данном примере во внешнем блоке, оборачивающем все остальные элементы, установлена ширина width: 50%, т. е. половина области просмотра окна браузера.

Во внешний родительский блок вложен первый дочерний элемент – тоже блок. Его ширина – 200px.

Для второго вложенного блока установлено стилевое правило width: 25vw, т. е. его ширина составляет 25 % ширины области просмотра окна браузера.

В третьем дочернем блоке установлена ширина width: 33 %, т. е. треть от ширины области просмотра браузера, а размер шрифта в 2 раза больше текущего шрифта.

1em обозначает текущий размер шрифта. Можно брать любые пропорции от текущего шрифта: 2em, 0.5em и т. п. Размеры в em относительные, они определяются по текущему контексту.

Приведем пример использования em.

```
<body style="font-size:20px">
  <p>Многим нравятся гарики.</p>
  <div style="font-size:1.5em">
    Бывает – проснешься, как птица, крылатой пружиной на
    взводе, и хочется жить и трудиться;
    <span style="font-size:1.5em">но к завтраку это
    проходит.</span>
  </div>
</body>
```

Многим нравятся гарики.

Бывает – проснешься, как птица, крылатой пружиной на взводе, и хочется жить и трудиться; но к завтраку это проходит.

Так как значение в em высчитывается относительно текущего шрифта, то текст, вложенный в , в 1,5 раза больше, чем текст в параграфе. Фраза в еще в 1,5 раза больше, чем .

Поскольку em – единица относительная, размеры, заданные в em, будут уменьшаться или увеличиваться вместе с размером базового шрифта. С учетом того что размер шрифта обычно определяется в родителе и может быть изменен ровно в одном месте, это бывает очень удобно.

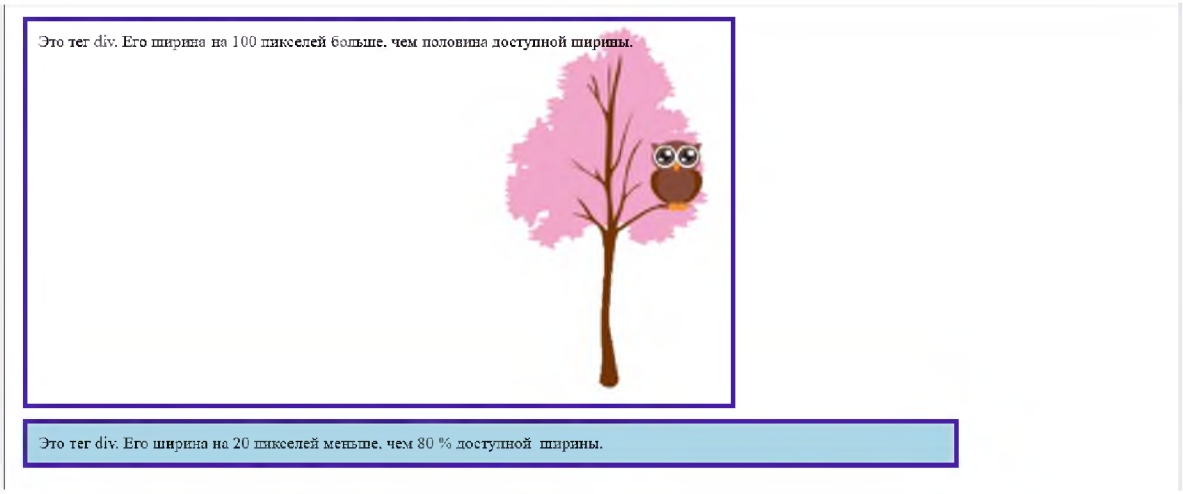
Единицы измерения vw, vh, vmin, vmax были созданы для поддержки мобильных устройств. Их основное преимущество в том, что любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

Вычисляемые значения длины

Функция calc() используется для указания вычисляемого значения свойств CSS, которые в качестве значения используют какое-либо число.

В первую очередь calc можно применять для вычисления размеров, углов, времени. Причем функция позволяет смешивать различные единицы измерений, например, ширину блока div можно задать следующим образом:
```
div {width: calc (100% - 40px);}
```
Ширина блока будет на 40 пикселей меньше размера родительского блока.

Еще пример. На странице есть два блока. Ширина первого блока на 100px больше, чем половина окна браузера. В этом блоке есть фоновый рисунок. Он смещен на 20px от правого края блока, при этом сам блок «резиновый»: calc(100% – 20px). Это пример того, как при адаптивной верстке встраиваются фоновые изображения. Ширина второго блока на 20px меньше, чем 80 % доступной ширины: calc(80% – 20px).
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Документ без названия</title>
    <style>
     .box {
           border: solid thick #461EA4;
           padding: 10px;
           margin: 10px; }
     .with_calc {
           width: calc(50% + 100px);
           height: 350px;
        background: url("images/img_tree.png") no-repeat;
           background-position: calc(100% - 20px) 0;}
     #foo {
           background: #ABD6E7;
           width: calc(80% - 20px);}
    </style>
  </head>
  <body>
    <div class="box with_calc">Это тег div. Его ширина На
      100 пикселей больше, чем половина доступной
      ширины.
    </div>
    <div class="box" id="foo">Это  тег div. Его ширина на
      20 пикселей меньше, чем 80 % доступной ширины.
    </div>
  </body>
</html>
```



Браузерные префиксы

Веб-разработчики перед некоторыми CSS-свойствами указывают браузерные префиксы: -webkit-, -moz-, -ms- и др. Например:

```
* {
    -moz-box-sizing: border-box; /* Для Firefox */
    -webkit-box-sizing: border-box; /* Для Opera, Safari и
    Chrome */
    -ms-box-sizing: border-box; /* Для IE */}
```

На данный момент используются следующие префиксы:

- -webkit- – для браузеров Chrome, Safari, Opera;
- -moz- – для браузера Mozilla Firefox;
- -ms- – для браузера Internet Explorer.

Если перед названием свойства стоит некоторый префикс, то это означает, что данное свойство реализовано и будет применяться исключительно в указанном браузере. Все остальные браузеры данное свойство будут игнорировать, так как для них данный префикс неизвестен. В примере выше в Mozilla Firefox будет использоваться свойство -moz-box-sizing, в Opera, Safari и Chrome – -webkit-box-sizing, а в Internet Explorer – -box-sizing.

Префиксы возникают:

- при включении в браузер экспериментальных свойств CSS, которые стандартом еще не утверждены. Таким образом производители браузеров производят тестирование и вносят предложения перед утверждением свойств CSS в стандарте;
- при создании собственных свойств, которые не входят в стандарт CSS, но, возможно, появятся в нем через некоторое время;
- для решения проблем с кросс-браузерностью.

Когда экспериментальное свойство утверждено в стандарте и прошло тестирование в браузере, у него обычно убирается префикс.

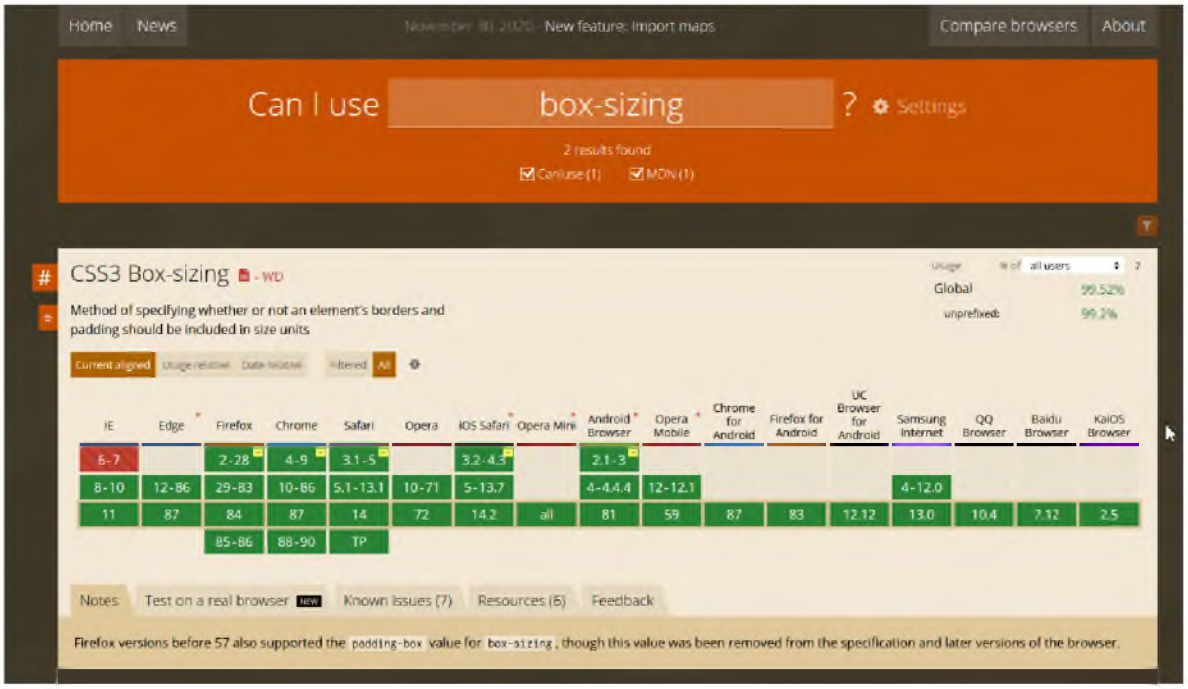
Рассмотрим в качестве примере следующий код:

```
* {
   -moz-box-sizing: border-box; /* Для Firefox */
   -webkit-box-sizing: border-box; /* Для Opera, Safari
    и Chrome */
    box-sizing: border-box;}
```

Данный код применяет свойства CSS, которые изменяют алгоритм расчета ширины и высоты для всех элементов веб-страницы. Первое CSS-свойство -moz-box-sizing со значением border-box предназначено для браузеров, использующих движок Gecko (Mozilla Firefox). Второе CSS-свойство -webkit-box-sizing со значением border-box предназначено для браузеров, использующих движок WebKit (Safari) или Blink (Chrome, Opera, «Яндекс.Браузер»). Последнее CSS-свойство предназначено для браузеров, в которых это свойство уже протестировано и внедрено в соответствии со стандартом.

При использовании префиксов для свойств CSS необходимо их размещать до свойства CSS без префикса. Это важно потому, что если когда-то в браузере будет реализовано оригинальное свойство (без префикса), то будет использоваться именно оно (так как оно располагается последним), а не его экспериментальная версия.

Проверить поддержку определенного свойства в браузере можно, например, на сайте https://caniuse.com.



Кроме этого, на сайте показывается количество пользователей, которые пользуются этой версией браузера, в процентах.

На сайте https://caniuse.com браузеры отмечаются различными цветами в зависимости от того, в каком состоянии находится поддержка определенных свойств или тегов:

- красный прямоугольник – браузер, в котором данное свойство не реализовано;
- зеленый прямоугольник с дефисом, расположенным в правом верхнем углу, – браузер, в котором данное свойство используется через префикс;
- светло-зеленый прямоугольник – браузер, в котором данное свойство реализовано частично;
- зеленый прямоугольник – браузер, в котором данное свойство реализовано в соответствии со стандартом.

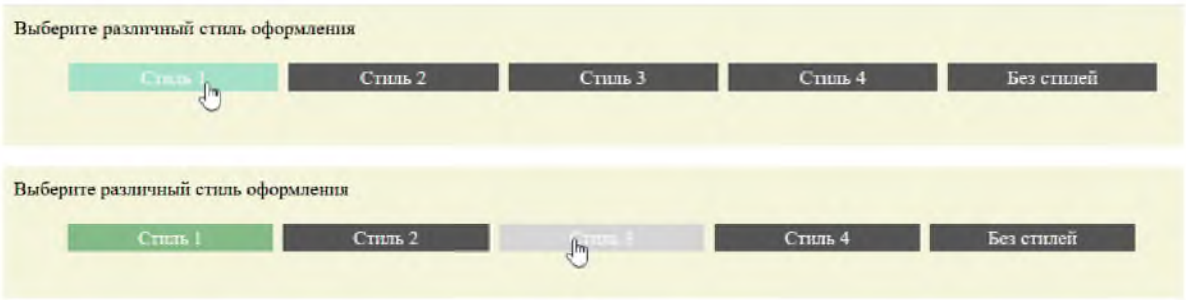
Подключение альтернативных стилей

Приведем пример подключения альтернативных стилей.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
/* Stylesheet 1: */
    <style>
      body {
            background-color: beige;}
      #s0{
            background-color: rgba(133,187,134,1.00);}
      #s0:hover {
            background-color:rgba(165,224,201,1.00) ;}
    </style>
>/* Stylesheet 2: */
    <style
      body {
            background-color: aqua;}
      #s1{
            background-color: rgba(133,187,134,1.00);}
      #s1:hover {
            background-color:rgba(165,224,201,1.00) ;}
    </style>
/* Stylesheet 3: */
    <style>
```

```
      body{
            background-color: blueviolet;}
      #s2{
            background-color: rgba(133,187,134,1.00);}
      #s2:hover {
            background-color:rgba(165,224,201,1.00) ;}
    </style>
/* Stylesheet 4: */
    <style>
      body {
            background-color: aquamarine;}
      #s3{
            background-color: rgba(133,187,134,1.00);}
      #s3:hover {
            background-color:rgba(165,224,201,1.00) ;}
    </style>
    <style>
      .menuitem {
            width: 165px;
            float: left;
            background-color: #555555;
            color: #ffffff;
            list-style-type: none;
            margin: 4px;
            padding: 2px;
            text-align: center;
            cursor: pointer;}
      .menuitem:hover {
            background-color:rgba(215,214,214,1.00);}
    </style>
  </head>
  <body>
    Выберите различный стиль оформления
    <ul >
     <li  class="menuitem" id="s0"
      onclick="reStyle(0)">Стиль 1</li>
     <li class="menuitem" id="s1"
      onclick="reStyle(1)">Стиль 2</li>
     <li class="menuitem" id="s2"
      onclick="reStyle(2)">Стиль 3</li>
     <li class="menuitem" id="s3"
      onclick="reStyle(3)">Стиль 4</li>
     <li class="menuitem" id="s4"
      onclick="noStyles();
```

```
      document.styleSheets[4].disabled = true;">
      Без стилей</li>
    </ul>
    <script>
      function noStyles() {
            document.styleSheets[0].disabled = true;
            document.styleSheets[1].disabled = true;
            document.styleSheets[2].disabled = true;
            document.styleSheets[3].disabled = true;
            document.styleSheets[4].disabled = false;  }
      function reStyle(n) {
            noStyles();
            document.styleSheets[n].disabled = false;
         /*только n-ю таблицу стилей делаем доступной*/}
            reStyle(n);
    </script>
  </body>
</html>
```



Выберите различный стиль оформления

- Стиль 1
- Стиль 2
- Стиль 3
- Стиль 4
- Без стилей

Стили отличаются только фоном.

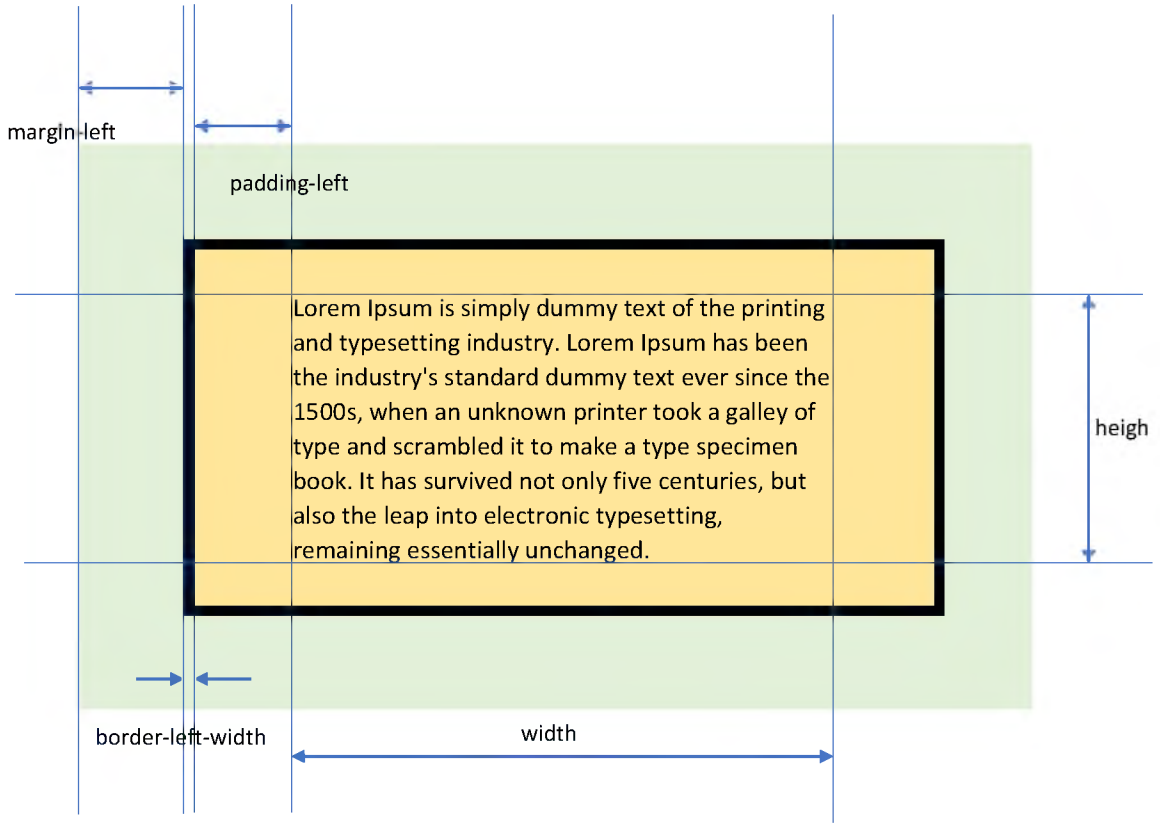
Стандартная блочная модель

В CSS используется модель представления частей HTML-документа в виде прямоугольных блоков. Каждый блок имеет информативную область, в которой заключено содержимое породившего его элемента (текст,

изображение и т. п.), а также может иметь области, отведенные для оформления полей (padding), рамок (border) и внешних отступов (margin).

Чтобы правильно установить ширину и высоту элемента, необходимо знать, как работает блочная модель.

Размеры окантовки существенно влияют не только на внешний вид блока, но и на размер пространства, которое блок будет занимать в браузере. Ширина информационной области, как правило, задается свойством width, а высота – height. Поля слева и справа от контента до рамки занимают padding-left и padding-right соответственно. Рамка также имеет ширину; margin-left и margin-right – отступы слева и справа от блока.



Ширина места, которое будет занимать блок в макете, рассчитывается следующим образом: общая ширина элемента = ширина (ширина контента) + левый отступ + правый отступ + левая граница + правая граница + + левое поле + правое поле, или width + padding-left + padding-right + + border-left + border-right + margin-left + margin-right.

Например, есть стиль

```
.box{
     width: 1110px;
     padding-left: 55px;
     padding-right: 54px;
```

```
    margin: 40x;
    border: solid 2px red;
}
```

Согласно этому правилу элемент с классом box по ширине будет занимать 1110 + 55 + 54 + 2 + 2 + 40 + 40 = 1303px.

Общая высота элемента рассчитывается аналогично: общая высота элемента = высота + верхний отступ + нижний отступ + верхняя граница + + нижняя граница + верхнее поле + нижнее поле.

Альтернативная блочная модель

При использовании альтернативной модели любая ширина – это ширина видимой части элемента на странице, поэтому ширина области содержимого высчитывается по формуле: ширина контента = ширина (общая ширина элемента) – (левая граница + правая граница) – (левое внутреннее поле + правое внутреннее поле).

Проблемы стандартной блочной модели

Стандартная блочная модель, когда свойство width задает ширину контента, не слишком удобна. Постоянно приходится заниматься вычислениями, когда требуется задать определенную ширину блока. Также начинаются проблемы при сочетании разных единиц измерения, в частности процентов и пикселей. Предположим, что ширина контента задана как 90 %, если сюда приплюсовать поля и рамки, заданные в пикселях, то нельзя вычислить суммарную ширину блока, поскольку проценты напрямую в пиксели не переводятся. В итоге может получиться так, что общая ширина блока превысит ширину веб-страницы, что приведет к появлению горизонтальной полосы прокрутки. В качестве выхода из подобной ситуации можно использовать вложенные блоки или изменить принцип построения блочной модели.

Рассмотрим оба варианта.

При использовании вложенных блоков для нужного блока создается блок-обертка. Для внешнего блочного элемента задается только необходимая ширина. Для вложенного блока задаются все остальные свойства: поля, рамки и отступы. Поскольку по умолчанию ширина блока равна доступной ширине родителя, получится, что блоки в каком-то смысле накладываются друг на друга, при этом фактическая ширина такого комбинированного элемента будет четко задана.

Можно изменить принцип построения блочной модели и использовать альтернативную блочную модель.

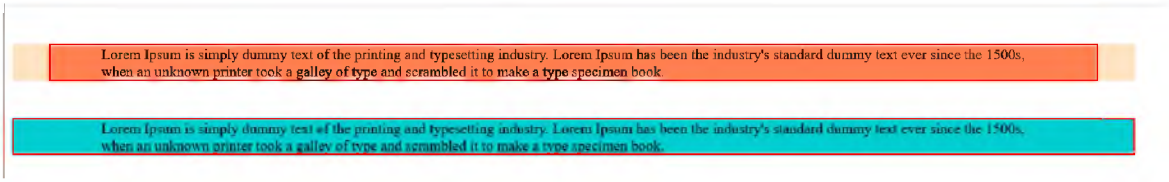
В CSS3 есть свойство box-sizing. По умолчанию значение свойства равно content-box, в этом случае width и height блока включают только контент, а border и padding не включают. При значении box-sizing, равном border-box, ширина начинает включать поля и рамки, но не отступы. Таким образом, подключая box-sizing со значением border-box к своему стилю, можно задавать ширину в процентах и спокойно указывать border и padding, не боясь, что изменится ширина блока.

Это прекрасное решение, но оно имеет существенный недостаток: не все браузеры его поддерживают.

Допустим, нам надо сверстать блок, который будет занимать на странице по ширине 1219px. Расстояние от левого края блока до текста должно быть 95px, от правого края блока до текста – 94px. В приведенном ниже примере дается два варианта верстки: с использованием стандартной блочной верстки и с использованием альтернативной блочной модели.

```
<!doctype html>
<html>
  <head>
    <style>
/*Вариант 1*/
      .box{
        padding-left: 55px;
        padding-right: 54px;
        margin: 40px;
        border: solid 2px red;
        background-color:coral;  }
      .wrapper{
        width: 1219px;
        background-color: bisque;}
/*Вариант 2*/
      .boxsizing{
        -moz-box-sizing: border-box; /* Для Firefox */
        -webkit-box-sizing: border-box; /* Для Opera,
        Safari и Chrome */
        box-sizing: border-box; /* Для IE */
        width: 1219px;
        padding-left: 95px;
        padding-right: 94px;
        margin: 0px;
        border: solid 2px red;
        background-color: darkturquoise;}
    </style>
  </head>
  <body>
```

```
<!-- Вариант 1-->
    <div class="wrapper">
      <div class="box">
        Lorem Ipsum is simply dummy text of the printing
        and typesetting industry. Lorem Ipsum has been the
        industry's standard dummy text ever since the
        1500s, when an unknown printer took a galley of
        type and scrambled it to make a type specimen
        book.
      </div>
    </div>
<!-- Вариант 2-->
    <div class="boxsizing">
      Lorem Ipsum is simply dummy text of the printing
      and typesetting industry. Lorem Ipsum has been the
      industry's standard dummy text ever since the 1500s,
      when an unknown printer took a galley of type and
      scrambled it to make a type specimen book.
    </div>
  </body>
</html>
```



Как видно, при верстке блоки занимают одинаковое место.

Свойства блока

width и height

Значения ширины и высоты блока могут указываться в абсолютных величинах, например в пикселях, или в относительных, например в процентах. При использовании процентной записи ширина элемента вычисляется в зависимости от ширины родительского элемента. Если родительский элемент явно не указан, то в его качестве берется окно браузера. Свойства width и height не применяются к строчным элементам display: inline. Ширина и высота содержимого встроенных блоков определяются шириной и высотой отображаемого содержимого внутри них. Отрицательные значения не допускаются.

min-width и max-width, min-height и max-height

Свойства min-width и max-width позволяют ограничивать ширину содержимого до определенного значения. Предельные значения не могут быть отрицательными. Для min-width значение по умолчанию 0, для max-width – none, что означает, что ограничения на ширину блока отсутствуют.

Аналогично свойства min-height и max-height ограничивают высоту элементов определенным диапазоном.

padding, border, margin

Свойство margin описывает ширину отступа между рамкой блока и границей элемента, в который помещен блок. Свойство border определяет свойства рамки блока. Свойство padding описывает поля или отступы от рамки блока до его содержимого.

Каждое из свойств padding, border, margin имеет соответствующие частные свойства: *-top, *-bottom, *-left, *-right, где вместо звездочки используется название одного из трех свойств. Каждое из частных свойств может быть задано отдельно. Одновременно в базовом свойстве может быть задано от одного до четырех значений частных свойств. Когда указано одно значение, оно применяется ко всем четырем сторонам. Например, следующие объявления эквивалентны:

```
p {margin:7px;}
p {
   margin-top:7px;
   margin-right:7px;
   margin-bottom:7px;
   margin-left:7px;
}
```

Когда указаны два значения, первый отступ применяется сверху и снизу, второй – справа и слева. Например, следующие объявления эквивалентны:

```
p {margin:7px 10px;}
p {
   margin-top:7px;
   margin-right:10px;
   margin-bottom:7px;
   margin-left:10px;
}
```

Когда заданы три значения, первый отступ применяется сверху, второй – слева и справа , третий – снизу.

```
p {margin:7px 10px 14 px}
p {
```

```
    margin-top:7px;
    margin-right:10px;
    margin-bottom:14px;
    margin-left:10px;
}
```

Если указаны четыре значения отступа, то они применяются сверху, справа, снизу и слева.

```
p {margin:7px 10px 14 px 5px;}
p {
    margin-top:7px;
    margin-right:10px;
    margin-bottom:14px;
    margin-left:5px;
}
```

Запомнить порядок применения значений просто: они присваиваются по часовой стрелке, начиная сверху. Например:

```
p {
    border-style: solid;
    border-top-width:2px;
    border-right-width:4px;
    border-bottom-width:6px;
    border-left-width:8px;
}
```

Так как поля и отступы элемента не являются обязательными, по умолчанию их значение равно нулю. Тем не менее браузеры добавляют этим свойствам положительные значения по умолчанию на основе своих таблиц стилей. Очистить стили браузеров для всех элементов можно при помощи универсального селектора:

```
* {
    margin: 0;
    padding: 0;
}
```

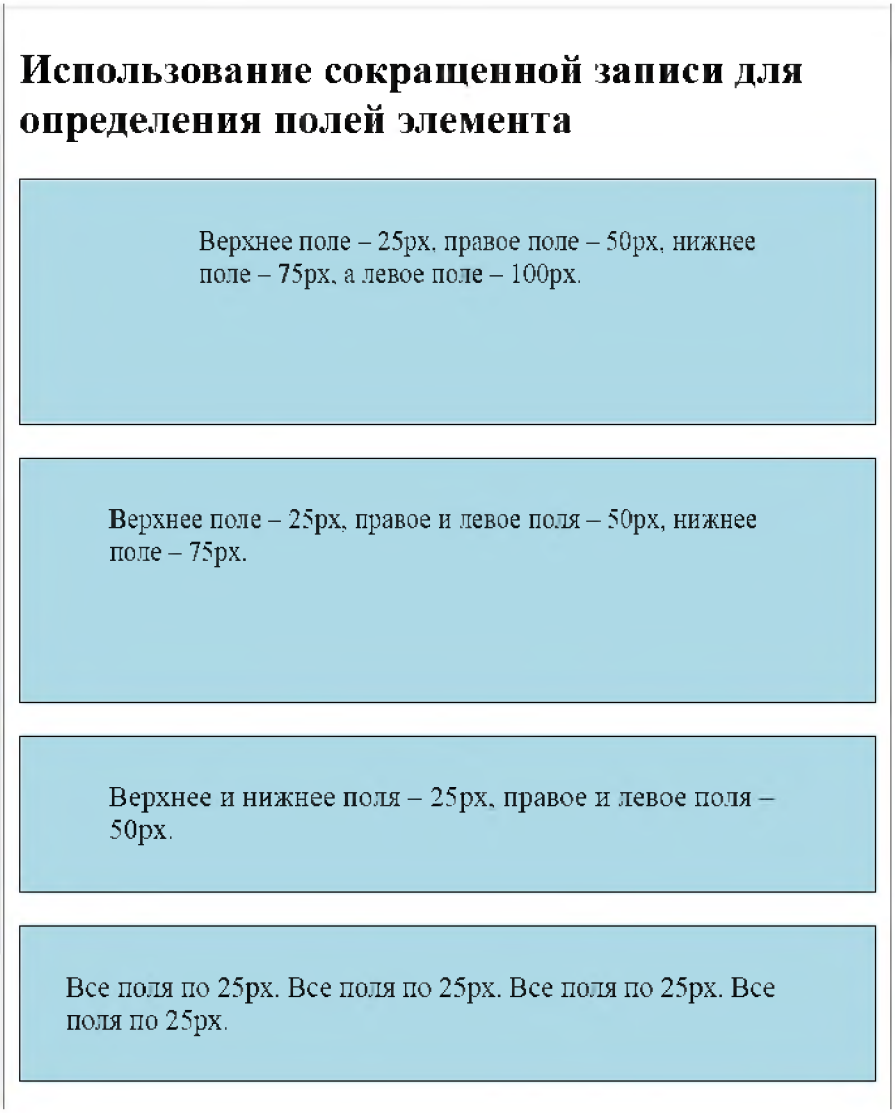
padding

Поля, или padding, представляют собой пространство между краем области содержимого и рамкой элемента. Они прозрачные, и фон элемента по умолчанию закрашивает его поля и пространство под его

рамкой. Значения полей определяют толщину их области. Сокращенное свойство padding задает поля для всех четырех сторон, а подсвойства устанавливают только их соответствующие стороны.

Демонстрация использования сокращенной записи свойства padding:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
           border: 1px solid black;
           background-color: lightblue;
           }
      .ex1 {padding: 25px 50px 75px 100px;}
      .ex2 {padding: 25px 50px 75px;}
      .ex3 {padding: 25px 50px;}
      .ex4 {padding: 25px; }
    </style>
  </head>
  <body>
    <h2>Использование сокращенной записи для определения
     полей элемента
    </h2>
    <div class="ex1">Верхнее поле - 25px, правое поле -
    50px,нижнее поле - 75px, а левое поле - 100px.
    </div>
    <br>
    <div class="ex2">Верхнее поле - 25px, правое и левое
      поля - 50px, нижнее поле - 75px.
    </div>
    <br>
    <div class="ex3">Верхнее поле и нижнее поля - 25px,
      правое и левое поля - 50px.
    </div>
    <br>
    <div class="ex4">Все поля по 25px. Все поля по 25px.
      Все поля по 25px. Все поля по 25px.
    </div>
    <br>
  </body>
</html>
```



margin

Свойство margin используется для создания пустого пространства вокруг элементов за пределами определенных границ. Внешние отступы прозрачные. Применяется ко всем элементам, кроме внутренних элементов таблицы.

Иногда необходимо центрировать элемент в его контейнере в горизонтальном направлении. В этом случае можно установить свойство margin в auto. Элемент будет занимать указанную ширину, а остальное пространство будет разделено поровну между левым и правым полями. В вертикальном направлении это не работает.

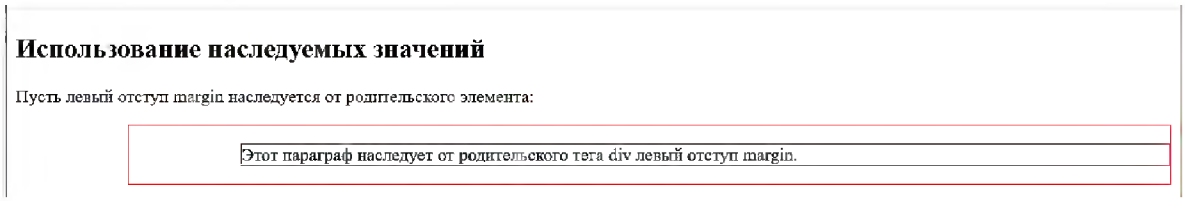
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
```

```
    <style>
      div {
           width:300px;
           margin: auto;
           border: 1px solid red;
           }
    </style>
  </head>
  <body>
    <h2>Использование значения auto для margin</h2>
    <p>Можно установить свойство margin в auto для
     горизонтального центрирования элемента внутри
     контейнера. Элемент будет занимать указанную ширину,
     а остальное пространство будет разделено поровну
     между левым и правым полями:
    </p>
    <div>
     Этот div будет выровнен по центру, потому что у него
     margin: auto;
    </div>
  </body>
</html>
```

Свойство margin автоматически не наследуется. Следующий пример показывает, как можно унаследовать левый отступ от родительского элемента:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div.container {
           margin-left: 100px;
           border: 1px solid red;}
      p.one {
           margin-left: inherit;
           border: 1px solid;}
    </style>
```

```
  </head>
  <body>
    <h2>Использование наследуемых значений</h2>
    <p>Пусть левый отступ margin наследуется от
      родительского элемента:
    </p>
    <div class="container">
      <p class="one">
        Этот параграф наследует от родительского тега div
        левый отступ margin.
      </p>
    </div>
  </body>
 </html>
```



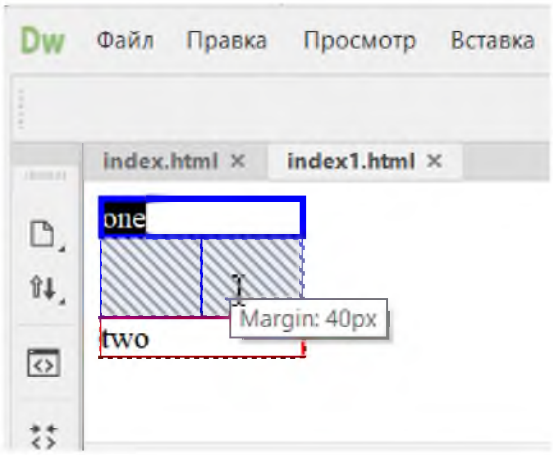
Значение свойства margin может быть как положительным, так и отрицательным. Отрицательное значение для внешнего отступа может привести к перекрытию элементов страницы. Независимо от того, используется стандартная или альтернативная блочная модель, margin всегда добавляется после расчета размера видимой части блока.

Особенности вертикального суммирования margin

Верхние и нижние отступы элементов сжимаются в один margin, который равен наибольшему из двух полей.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
           margin: 0;
           width: 100px;
           border: 1px solid red;
           }
      .one {margin-bottom: 40px;}
      .two {margin-top: 10px;}
    </style>
  </head>
```

```
  <body>
    <div class="one">one</div>
    <div class="two">two</div>
  </body>
</html>
```

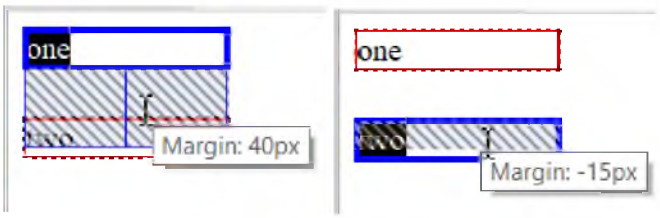


В данном примере есть два блока div фиксированной ширины. Для наглядности у блоков задана граница. Первый блок имеет отступ снизу 40px. У второго блока верхний отступ равен 10px. Однако, как видно из картинки, реально расстояние между первым и вторым блоками составляет 40px, т. е. вертикальные отступы двух элементов уровня блока перекрываются. При этом ширина общего отступа равна ширине большего из исходных.

Если один из отступов отрицательный и величина прилегающих отступов элементов – x и y, то происходит складывание отступов по правилам математики: x + (–y) = x – y.

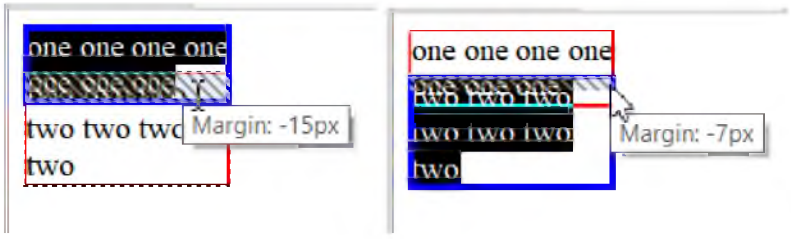
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
         margin: 0;
         width: 100px;
         border: 1px solid red;}
   .one {margin-bottom: 40px;}
   .two {margin-top: -15px;}
    </style>
  </head>
  <body>
    <div class="one">one</div>
    <div class="two">two</div>
```

```
  </body>
</html>
```



Если оба отступа отрицательны, то из двух значений выбирается наибольшее по модулю, оно же и выступает в качестве отрицательного отступа между элементами. Так, если отступы равны –15px и –7px, то итоговое значение будет –15px.

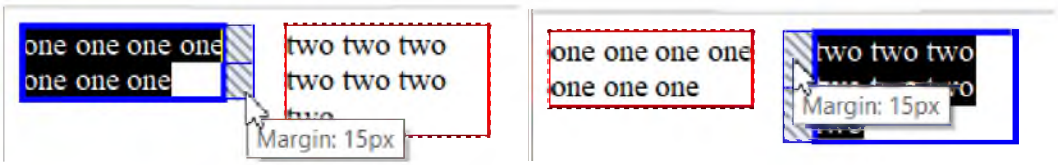
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
            margin: 0;
            width: 100px;
            border: 1px solid red;
           }
      .one {margin-bottom: -15px;}
      .two {margin-top: -7px;}
    </style>
  </head>
  <body>
    <div class="one">one one one one one</div>
    <div class="two">two two two two two two two</div>
  </body>
</html>
```



Горизонтальное суммирование margin

Горизонтальные отступы между элементами просто складываются. Например, горизонтальный отступ между двумя элементами с отступами 15px будет равен 30px.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
           margin: 0;
           width: 100px;
           float: left;
           border: 1px solid red;
          }
      .one {margin-right: 15px;}
      .two {margin-left: 15px;}
    </style>
  </head>
  <body>
    <div class="one">one one one one one one one</div>
    <div class="two">two two two two two two two</div>
  </body>
</html>
```



Вертикальное схлопывание внешних полей, когда элементы связаны родительской связью

Если элементы связаны родительской связью, то схлопывание работает немного сложнее. Если внутри одного блока расположить другой и задать ему margin-top, то внутренний блок прижмется к верхнему краю родительского, а у родительского элемента появится отступ сверху, т. е. внутренний блок как бы «выпадет» из родительского. Если у родительского элемента также был задан верхний отступ, то выберется наибольшее из значений.

Для рассмотрения данной ситуации представим, что имеется два блока: блок-родитель и дочерний блок. Для того чтобы сработало схлопывание, нужно, чтобы блок-родитель не имел ни полей, ни границ (их значения были нулевые). Таким образом, внешние поля этих элементов соприкасаются. Отметим, что нулевые значения внутренних полей и границ для родителей – обязательное условие срабатывания схлопывания.

```
<!doctype html>
<html>
```

```
<head>
  <meta charset="utf-8">
  <style>
    body {
          background: url("разлинованные клетки.png")
                      repeat;}
    div {
          width:200px;
          text-align:center;
        }
    .parent {
          /*Полей у родителя быть не должно*/
          padding:0px;
          /*Границ у родителя быть не должно */
          border:0px solid;
          /* Этот отступ схлопнулся с дочерним */
          margin-top: 40px;
          background-color:aqua;
          }
    .daughter {
          width:186px;
          /* Актуальное значения отступа */
          margin-top:60px;
          /* У дочернего может быть граница*/
          border:4px solid red;
          /* У дочернего блока внутренние отступы могут
           быть не равны 0 */
          padding:3px;
          background-color:yellow;
          }
  </style>
</head>
<body>
  <div class="parent">
    <div class="daughter">Дочерний</div>
      Родительский
  </div>
</body>
</html>
```

Для наглядности в качестве фона установим изображение с разлинованными клетками. Размер одной клетки равен 20 пикселям, поэтому без схлопывания отступ между элементами был бы 60px + 40px = 100px, но из-за применения данного эффекта он равен максимальному значению одного из элементов (дочернего), в данном случае – 60 пикселям (3 клетки). Видим, что отступ дочернего блока заменяет верхний отступ для своего родителя.

Если, например, добавить границу к родительскому блоку, то схлопывания не произойдет:

```
.parent {
        padding:0px;/* Полей у родителя быть не должно */
        border:2px solid;
        margin-top: 40px;
        background-color:aqua;
        }
```

Поэтому, чтобы избавиться от схлопывания, достаточно задать родительскому элементу padding-top или добавить border-top: 1px solid transparent.

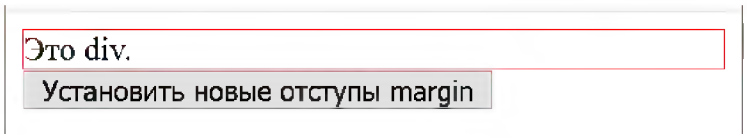
Динамическое изменение полей блока

Используя JavaScript, можно изменить поля элемента динамически. Доступ к свойству margin элемента с известным id можно получить следующим образом: document.getElementById("box").style.margin.

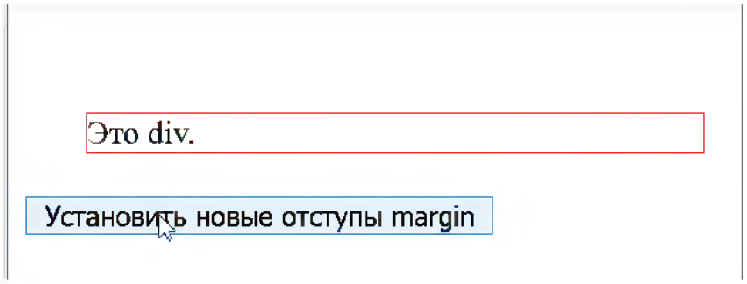
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      #box {border: 1px solid #FF0000;}
    </style>
```

```
    <script>
     function myFunction() {
           document.getElementById("box").style.margin =
           "50px 10px;
    </script>
  </head>
  <body>
    <div id="box">Это div.</div>
    <button type="button"
     onclick="myFunction()">
      Установить новые отступы margin</button>
  </body>
</html>
```

Вид страницы сразу после загрузки:



Вид страницы после нажатия на кнопку:



Рамки элемента border

Свойство border задает рамку элемента и представляет собой одну или несколько линий, окружающих содержимое элемента и его поля (padding). Стиль рамки задается с помощью трех свойств: стиль, цвет и ширина.

border-style

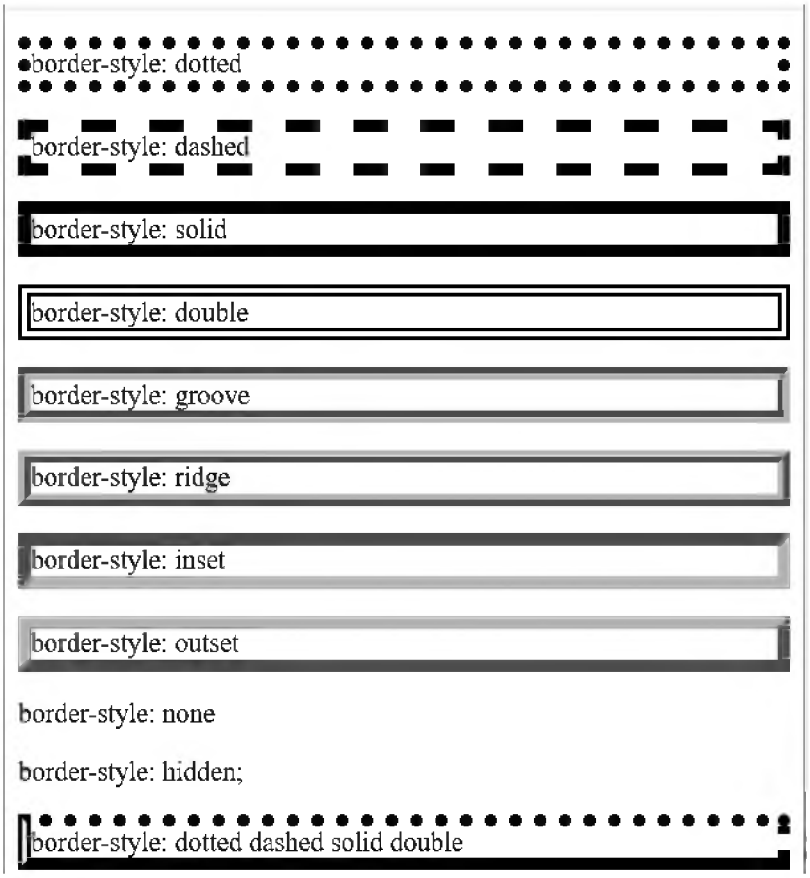
Свойство border-style устанавливает тип рамки одновременно на всех сторонах элемента разметки или индивидуально для каждой стороны. Способ изменения типа рамки зависит от числа аргументов. Может быть задано от одного до четырех значений. По умолчанию рамки всегда отрисовываются поверх фона элемента. Если не задан стиль рамки, то без этого свойства рамка не будет отображаться вообще.

Свойство border-style может принимать следующие значения:

- none – граница не отображается;
- hidden – определяет скрытую границу;
- dotted – определяет пунктирную границу (точками);
- dashed – определяет пунктирную границу (прерывистой линией);
- solid – определяет сплошную границу;
- double – определяет сплошную двойную границу;
- groove – определяет трехмерную рифленую границу (вогнутая рамка);
- ridge – определяет трехмерную рифленую границу (выпуклая рамка);
- inset – понижение содержимого блока по отношению к границе;
- outset – повышение содержимого блока по отношению к границе;
- initial – устанавливает в значение по умолчанию, т. е. none;
- inherit – использует значение свойства родительского элемента.

В разных браузерах вид отдельных стилей границы может отличаться. Например, при применении стиля dotted к рамке точка может изображаться в виде ромба, квадрата или круга.

Для управления стилем рамки только одной стороны элемента разметки можно воспользоваться свойствами border-top-style, border-right-style, border-bottom-style и border-left-style.



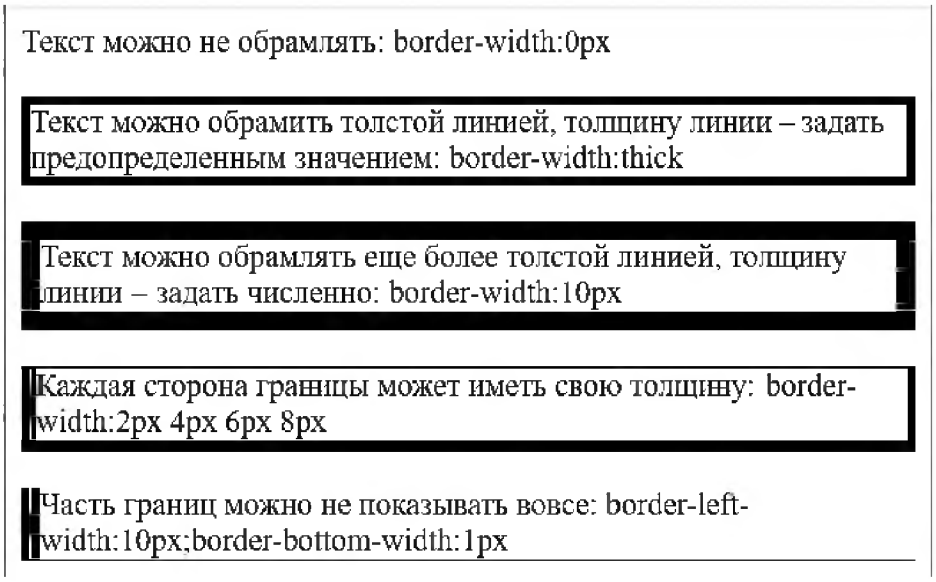
border-width

Свойство border-width задает ширину границы одновременно на всех сторонах элемента или индивидуально для каждой стороны. Способ изменения ширины зависит от числа аргументов. Может быть задано от одного до четырех значений. Ширина границы может быть задана числовым значением. Также доступны три предопределенных значения: thin (2px), medium (4px), thick (6px), рисующие тонкую, среднюю и толстую линии соответственно. Значение по умолчанию medium.

Для управления шириной рамки только одной стороны элемента разметки можно воспользоваться свойствами border-top-width, border-right-width, border-bottom-width и border-left-width.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
     div {
             border-style:solid;
             border-width:0px;
         }
     div#s1 {border-width:thick;}
     div#s2 {border-width:10px;}
     div#s3{ border-width:2px 4px 6px 8px;}
     div#s4 {
             border-left-width:10px;
             border-bottom-width:1px;
            }
    </style>
  </head>
  <body>
    <div>Текст можно не обрамлять:
      border-width:0px
    </div>
    <br>
    <div id="s1"> Текст можно обрамить толстой линией,
      толщину линии - задать предопределенным значением:
      border-width:thick
    </div>
    <br>
    <div id="s2"> Текст можно обрамлять еще более
      толстой линией, толщину линии - задать численно:
      border-width:10px
    </div>
```

```
    <br>
    <div id="s3">Каждая сторона границы может
      иметь свою толщину: border-width:2px 4px 6px 8px
    </div>
    <br>
    <div id="s4">Часть границ можно не показывать
      вовсе: border-left-width:10px;
      border-bottom-width:1px
    </div>
    <br>
  </body>
</html>
```



Если для свойства border-style задано значение none, а для ширины рамки элемента установлено допустимое значение, то в этом случае рамка не выводится. Свойство не наследуется.

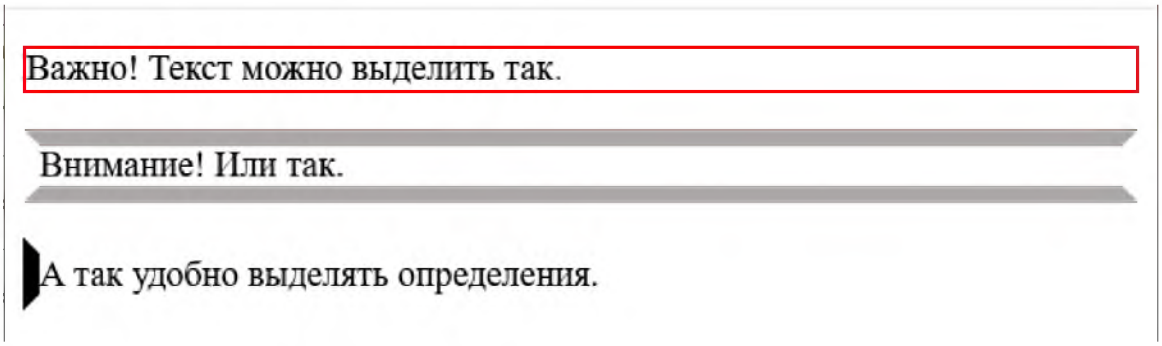
border-color

Свойство border-color устанавливает цвет границы. Если цвет границы опущен, то он будет таким же, как и цвет текста элемента. Если в элементе нет текста, то цвет рамки будет таким же, как и цвет текста родительского элемента. Данное свойство не наследуется. Параметр позволяет задать цвет границы сразу на всех сторонах элемента или только на указанных его сторонах. Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом.

Цвет может задаваться стандартной константой или шестнадцатеричным значением RGB-палитры. В этом случае описание цвета начинается с символа #, затем следуют две шестнадцатеричные цифры, выражающие

интенсивность красного цвета, после них – еще две шестнадцатеричные цифры, выражающие интенсивность зеленого цвета, и последними идут две шестнадцатеричные цифры, выражающие интенсивность синего цвета.

```
<!doctype html>
<html>
  <head>
    <meta charset=utf-8">
    <style>
     p {
        border-style:solid;
        border-width:8px;
       }
    .s1 {
        border-width:2px;
        border-color:red;
        }
     .s2 {border-color:rgba(171,167,167,1.00) #FFFFFF;}
     .s3 {border-color: #FFFFFF #FFFFFF #FFFFFF #000000;}
    </style>
  </head>
  <body>
    <p class="s1">Важно! Текст можно выделить так.</p>
    <p class="s2">Внимание! Или так.</p>
    <p class="s3">А так удобно выделять определения.</p>
  </body>
</html>
```



Управлять цветом одной стороны рамки можно через свойства border-top-color, border-bottom-color, border-left-color и border-right-color.

border

Свойство border можно рассматривать как сокращенный вариант одновременного использования свойств border-color, border-width и border-

style. Значения этих свойств могут располагаться в любом порядке и разделяться пробелом. Необязательно указывать значения всех трех свойств, можно указать одно, два или все три свойства. Значение по умолчанию – `border:medium none`.

Для установки границы только на определенных сторонах элемента можно воспользоваться параметрами border-top, border-bottom, border-left, border-right.

Ниже приведен пример использования границ для оформления объявления.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Документ без названия</title>
    <style>
      article {
         background-color: #33A1B6;
         border: 5px solid #0b385f;
         border-bottom-style: dashed;
         }
      h2 {
         color: #3E1F97;
         padding-left: 60px;
         margin-top: 20px;
         border-top: 2px dotted #2FDECA;
         border-bottom: 1em double rgba(43,194,15,1.00);
         }
    </style>
  </head>
  <body>
    <article>
     <h2>Внимание!</h2>
     <p>Сроки зачетов переносятся. Бла-бла-бла.</p>
    </article>
  </body>
</html>
```

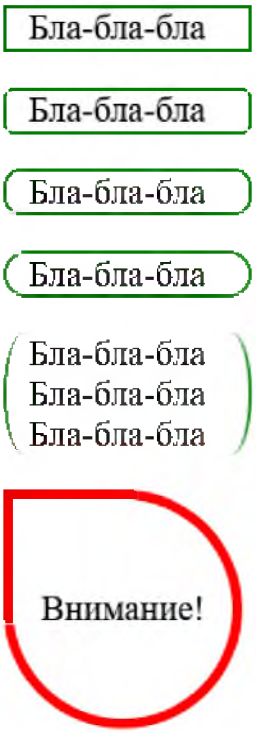
border-radius

Свойство border-radius используется для закругления углов элементов. Свойство применяется, даже если элемент не имеет границ, в этом случае оно распространяется на фон. Можно задать значения свойства через /, в этом случае первое число определяет горизонтальный, в второе – вертикальный радиусы эллипса.

Если задать два значения для свойства border-radius, то первое значение закруглит верхний левый и нижний правый углы, а второе – верхний правый и нижний левый. Общее свойство border-radius позволяет закруглить все углы одновременно, а с помощью свойств border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius можно закруглить каждый угол отдельно. Ниже приведены примеры применения обрамления границ блоков.

```
<!doctype html>
<html>
  <head>
    <style>
      p {
         border: 2px solid green;
         width: 100px;
         padding-left: 10px;
        }
      .round1 {border-radius: 5px;}
      .round2 {border-radius: 8px;}
      .round3 {border-radius: 12px;}
      .round4 {
         border-top: none;
         border-bottom: none;
         border-radius:
         30px/90px;
         }
      .atention{
         padding: 0;
         width: 100px;
         height: 100px;
         border: 5px solid #FF0000;
         text-align: center;
         line-height: 100px;
         border-radius: 0 50% 50% 50%;
         }
    </style>
  </head>
  <body>
```

```
    <p>Бла-бла-бла</p>
    <p class="round1">Бла-бла-бла</p>
    <p class="round2">Бла-бла-бла</p>
    <p class="round3">Бла-бла-бла</p>
    <p class="round4">
     Бла-бла-бла<br>
     Бла-бла-бла<br>
     Бла-бла-бла<br>
    </p>
    <p class="atention">Внимание!</p>
  </body>
</html>
```



Закругление можно применять не только к границам, но и к самим блокам. Это удобно при оформлении пунктов меню в виде кнопок.

```
<!doctype html>
<html>
  <head>
    <style>
      .menuitem {
              width: 165px;
              float: left;
              background-color: #555555;
              color: #ffffff;
              list-style-type: none;
```

```
                margin: 4px;padding: 2px;
                text-align: center;
                cursor: pointer;
                border-radius: 15px;
                }
        .menuitem:hover {
                background-color:rgba(215,214,214,1.00) ;
                }
    </style>
  </head>
  <body>
    <menu>
      <ul >
        <li  class="menuitem">Пункт 1</li>
        <li  class="menuitem">Пункт 2</li>
        <li  class="menuitem">Пункт 3</li>
        <li  class="menuitem">Пункт 4</li>
      </ul>
    </menu>
  </body>
</html>
```

Рамки-изображения border-image

В качестве рамки можно установить симметричное изображение. Для этого используется свойство border-image, которое, в свою очередь, является краткой записью свойств border-image-source, border-image-slice, border-image-width, border-image-outset и border-image-repeat.

Свойство border-image-source задает путь к изображению, которое будет использоваться для оформления границ блока.

Свойство border-image-width задает ширину изображения для границы элемента. Если ширина не задана, то по умолчанию она равна 1.

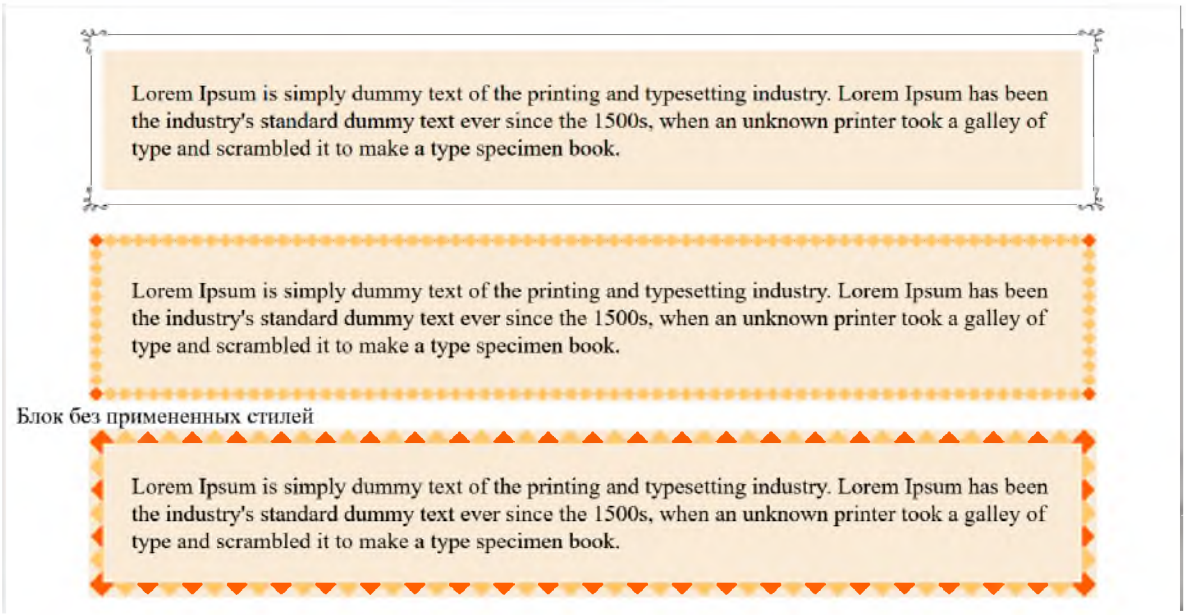
Свойство border-image-outset позволяет сместить рамку-изображение за пределы границ элемента на указанную длину.

Размер угловой части изображения задается с помощью значения свойства border-image-slice.

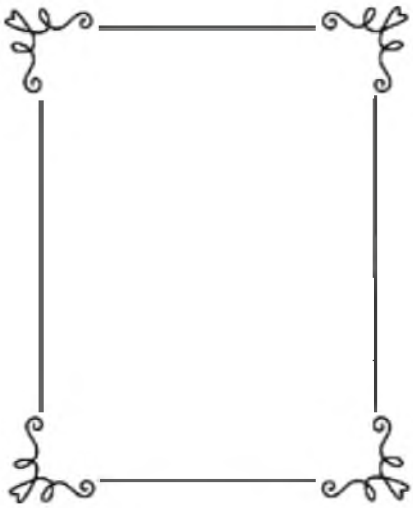
Свойство border-image-repeat управляет заполнением фоновым изображением пространства между углами рамки. Значение по умолчанию stretch растягивает изображение на все пространство блока.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      .r1 {
           width: 80%;
           margin: auto;
           margin-top: 30px;
           margin-bottom: 30px;
           border-image-outset: 15px;
           background-color: antiquewhite;
           border-image-width: 20px;
           border-image-source: url("рамка1.png");
           border-image-slice: 50;
           }
      .r2 {
           width: 80%;
           margin: auto;
           background-color: antiquewhite;
           border: 10px solid transparent;
           border-image: url("рамка2.png");
           border-image-width: 10px;
           border-image-slice:27 27;
           border-image-repeat: round;
           padding: 20px;
           }
      .r3 {
           width: 80%;
           margin: auto;
           background-color: antiquewhite;
           padding: 10px;
           border: 10px solid transparent;
           border-image: url("рамка2.png");
           border-image-slice:20%;
           border-image-repeat: round;
           padding: 20px;
           }
    </style>
  </head>
  <body>
    <div class="r1">Lorem Ipsum ...</div>
    <div class="r2">Lorem Ipsum ...</div>
    <div>Блок без примененных стилей</div>
    <div class="r3">Lorem Ipsum ...</div>
```

```
  </body>
</html>
```



Для первого блока для рамки взято изображение 200 × 245 пикселей с прозрачным фоном.



Блок «резиновый», его ширина установлена в 80 % (width: 80%). Блок отцентрирован по горизонтали: margin: auto. По вертикали сделаны внешние отступы: `margin-top: 30px` и `margin-bottom: 30px`. Это необходимо, поскольку рамка смещается за пределы границ блока `border-image-outset: 15px`. Для наглядности выведен цвет фона блока (`background-color: antiquewhite`): видно, что рамка-изображение смещена за пределы границ элемента. Взята ширина самой рамки в 20px

(`border-image-width: 20px`). Необходимо, чтобы рамка растягивалась по вертикали и горизонтали. Размер угловой части изображения – `border-image-slice: 50`. Чтобы заполнить пространство между углами рамки, можно поставить `border-image-repeat: round`. Тогда горизонтальный кусочек рамки, а в нашем случае это просто линия, будет повторяться. Можно установить `border-image-repeat: repeat`, в этом случае горизонтальный кусочек рамки растягивается на все пространство. Поскольку `border-image-repeat: repeat` – значение по умолчанию, то можно ничего не указывать.

Для второго блока для рамки взято изображение 81 × 81 пикселей.

Блок, как и первый, «резиновый», отцентрирован по горизонтали. Для наглядности выведен цвет фона блока (`background-color: antiquewhite`): видно, что рамка-изображение размещена поверх фона элемента. Взята ширина самой рамки в 20px (`border-image-width: 15px`). Необходимо, чтобы рамка растягивалась по вертикали и горизонтали. Размер угловой части изображения `border-image-slice: 27`. `border-image-repeat: round`, тогда горизонтальный кусочек рамки, а в нашем случае это более светлый ромб, будет повторяться. В случае если изображение будет отсутствовать, чтобы не изменился размер, занимаемый блоком, можно вывести прозрачную границу на ширину рамки-изображения: `border: 10px solid transparent`.

В четвертом блоке отличие в подобранном значении свойства `border-image-slice:20%`, которое позволяет так задать размер угловой части изображения, что получается ровный внутренний край рамки-изображения.

Внешний контур outline

Контур – это линия, которая проводится вокруг элементов за пределами границ и повторяет форму элемента. Контуры широко используются для улучшения пользовательского интерфейса. Например, они позволяют выделять активные элементы интерфейса, такие как кнопки, поля формы, карты изображений и т. п.

Контур всегда выводится поверх всех элементов и не влияет на положение или размер блока или любых других блоков. Поэтому отображение или скрытие контуров не вызывает перекомпоновку страницы.

Свойство outline представляет краткую запись свойств outline-color, outline-style и outline-width, которые имеют такой же смысл, как и аналогичные свойства для рамки border-color, border-style и border-width.

Значение по умолчанию – `outline: invert none medium`. Значение invert выполнит инверсию цвета пикселей на экране. Это обеспечивает видимость границы фокуса, независимо от цвета фона.

В приведенном ниже примере при помощи свойства outline выделяются ячейки таблицы, над которыми находится курсор.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      table, th, td {
        border: 1px solid black;
        border-collapse: collapse;}
      td:hover{outline: 2px solid red;}
    </style>
  </head>
  <body>
    <table>
      <tr>
        <th> фио </th>
        <th> алг. </th>
        <th> геом. </th>
        <th> мат.ан. </th>
        <th> прогр. </th>
      </tr>

      <tr>
        <td> Иванов И. И.</td>
        <td> 8 </td>
        <td> 7 </td>
        <td> 9 </td>
        <td> 8 </td>
      </tr>

      <tr>
        <td> Петров П. П.</td>
        <td> 5 </td>
        <td> 9 </td>
        <td> 6 </td>
        <td> 5 </td>
      </tr>
```

```
    <tr>
        <td> Сидоров С. С.</td>
        <td> не явился </td>
        <td> 4 </td>
        <td> 4 </td>
        <td> 7 </td>
      </tr>
    </table>
  </body>
</html>
```

ФИО	алг.	геом.	мат. ан.	прогр.
Иванов И. И.	8	7	9	8
Петров П. П.	5	9	6	5
Сидоров С. С.	не явился	4	4	7

Обтекание. Свойство float

Свойство float устанавливает горизонтальное выравнивание элемента относительно родительского элемента, при этом остальные элементы должны обтекать его с других сторон.

Свойство float может принимать следующие значения:

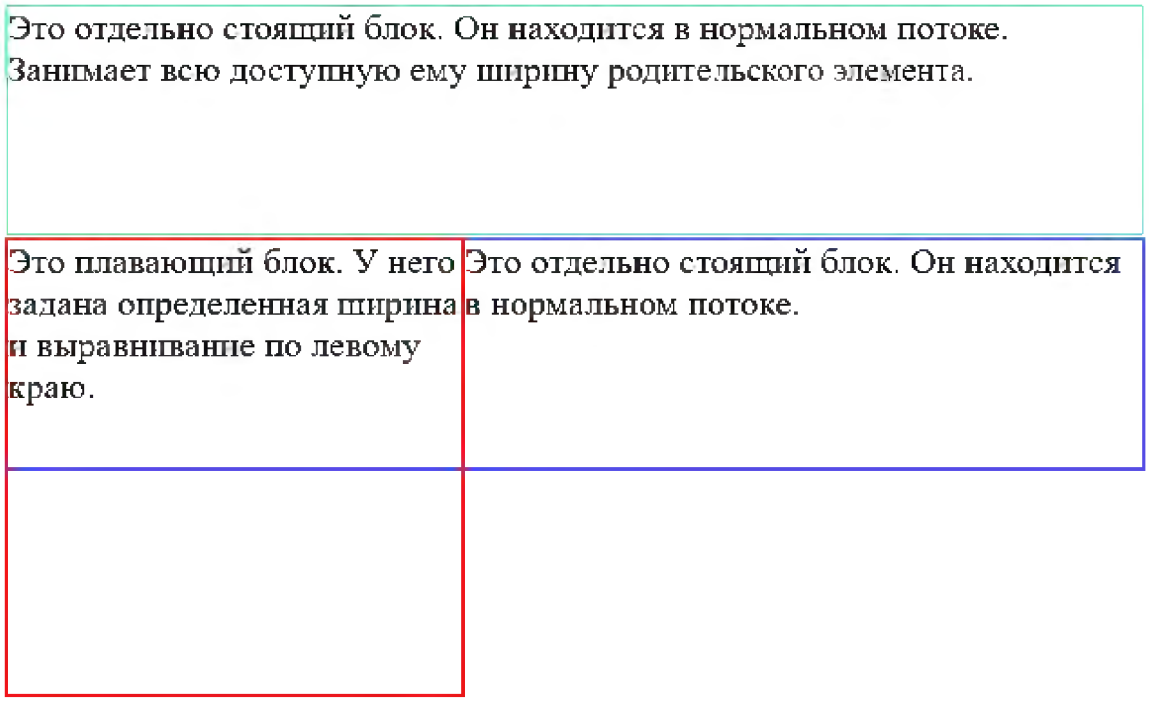
- left – выравнивание элемента по левому краю, а все остальные элементы, например текст, огибают его по правой стороне;
- right – выравнивание элемента по правому краю, а все остальные элементы, например текст, огибают его по левой стороне;
- none – отсутствие выравнивания текущего элемента, элемент выводится там, где он задан. Является значением по умолчанию;
- inherit – наследует значение свойства от родительского элемента.

Управляя обтеканием, можно смещать «плавающие блоки» влево или вправо на текущей строке. Плавающий блок смещается влево или вправо до тех пор, пока его внешний край не коснется края содержащего блока или внешнего края другого плавающего блока. Если для плавающего блока недостаточно места по горизонтали, он будет смещаться вниз до тех пор, пока не уместится.

В приведенном ниже примере сначала выводится блок с атрибутом class="normal1". Он находится в нормальном потоке, имеет заданную высоту и зеленую границу. Это отдельно стоящий блок. Он занимает всю

доступную ему ширину родительского элемента. Затем выводится плавающий блок с атрибутом class="floating_block". Задано выравнивание по левому краю. Заданы высота и ширина блока и красная граница. Следующим выводится третий блок с атрибутом class="normal2". Он также находится в нормальном потоке и имеет синюю границу. Блок занимает всю доступную ему ширину родительского элемента, он игнорирует плавающий блок, а текст в третьем блоке обтекает плавающий блок справа.

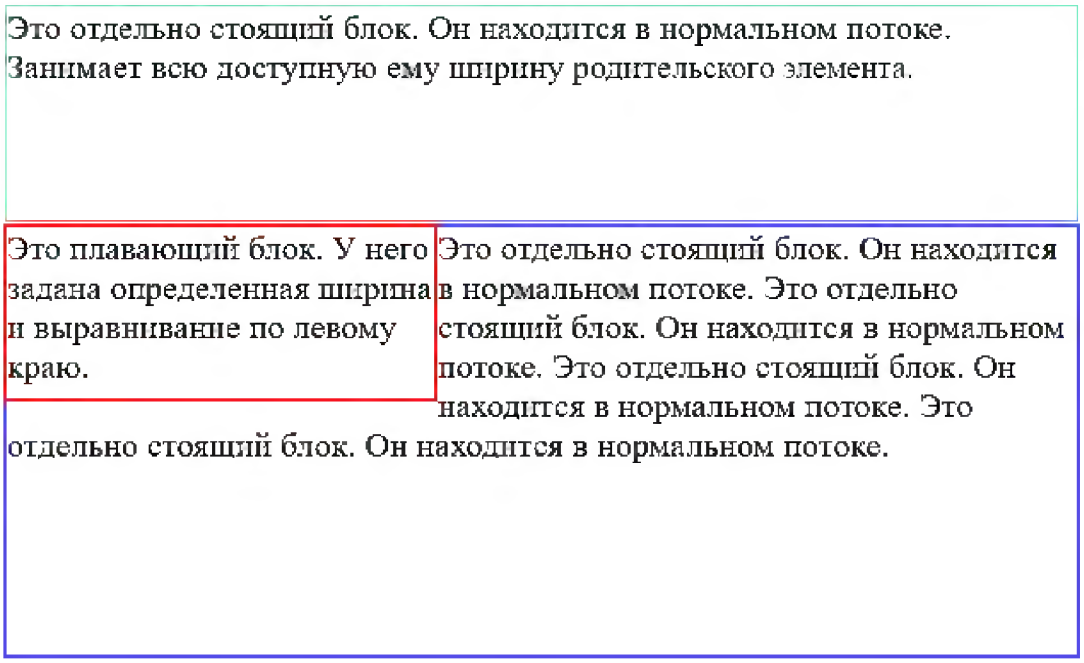
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Свойство float</title>
    <style>
      .normal1{
          height: 100px;
          border: 2px solid #75EFC3;
          }
      .floating_block{
          height: 200px;
          width: 200px;
          border: 2px solid #F5171A;
          float: left;
          }
      .normal2{
          height: 100px;
          border: 2px solid #564DE9;
          }
    </style>
  </head>
  <body>
    <div class="normal1">Это отдельно стоящий блок.
      Он находится в нормальном потоке. Занимает всю
      доступную ему ширину родительского элемента.
    </div>
    <div class="floating_block">Это плавающий блок. У
      него задана определенная ширина и выравнивание по
      левому краю.
    </div>
    <div class="normal2">Это отдельно стоящий блок. Он
      находится в нормальном потоке.
    </div>
  </body>
</html>
```



Если немного изменить высоту блоков и добавить в третий блок больше текста, то обтекание будет более очевидно.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Свойство float</title>
    <style>
      .normal1{
          height: 100px;
          border: 2px solid #75EFC3;}
      .floating_block{
          height: 80px;
          width: 200px;
          border: 2px solid #F5171A;
          float: left;}
      .normal2{
          height: 200px;
          border: 2px solid #564DE9;}
    </style>
  </head>
  <body>
    <div class="normal1">Это отдельно стоящий блок. Он
      находится в нормальном потоке. Занимает всю
      доступную ему ширину родительского элемента.
```

```
    </div>
    <div class="floating_block">Это плавающий блок.
      У него задана определенная ширина и выравнивание по
      левому краю.
    </div>
    <div class="normal2">Это отдельно стоящий блок. Он
      находится в нормальном потоке. Это отдельно стоящий
      блок. Он находится в нормальном потоке. Это отдельно
      стоящий блок. Он находится в нормальном потоке. Это
      отдельно стоящий блок. Он находится в нормальном
      потоке.
    </div>
 </body>
</html>
```



clear

Свойство clear запрещает обтекание элемента другими элементами разметки относительно родительского.

Свойство clear может принимать следующие значения:

- both – запрет на обтекание слева и справа;
- left – запрет на обтекание слева;
- right – запрет на обтекание справа;
- none – отменяет запрет на обтекание, установленное значениями both, left, right. Обтекание возобновляется согласно старым установкам.

Если отменить его, автоматически происходит переход на следующую строку.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      span {border:2px solid;}
      span.s1 {float:left;}
      span.s2 {float:right;}
    </style>
  </head>
  <body>
    <div>
     <span class="s1">Блок 1</span>
     <span class="s1" style="clear: left;">Блок 2</span>
     <span class="s1" style="clear: right;">Блок 3</span>
     <span class="s1" style="clear: both;">Блок 4</span>
     <span class="s1">Блок 5</span>
     <span class="s2">Блок 6</span>
     <span class="s2" style="clear: left;">Блок 7</span>
     <span class="s2" style="clear: right;">Блок 8</span>
     <span class="s2" style="clear: both;">Блок 9</span>
     <span class="s2">Блок 10</span>
    </div>
  </body>
</html>
```

В приведенном примере определено два стиля для тега : в первом значение свойства float установлено в left, а во втором – в right. Для вывода блоков с первого по пятый используется первый стиль, а для вывода последующих пяти блоков – второй стиль. При выводе первого блока свойство clear не используется, блок ведет себя как встроенный и располагается в начале абзаца. Для вывода второго блока свойство clear установлено в left, поэтому он выводится таким образом, чтобы его верхний край находился ниже нижнего внешнего края всех ранее созданных левосторонних перемещаемых блоков. Фактически зрительно это привело к переходу на следующую строку. Для третьего блока свойство clear установлено в right, что означает, что этот блок должен быть выведен ниже любого нижнего внешнего края любого правостороннего перемещаемого блока, который был сгенерирован в исходном документе предыдущим элементом. Правосторонних перемещаемых блоков еще не выводилось, поэтому третий блок становится справа за вторым во второй линии. При выводе четвертого установлен запрет на обтекание слева и справа. Поэтому он помещается ниже блока 2 (clear:left) и ниже блока 3 (clear:right),

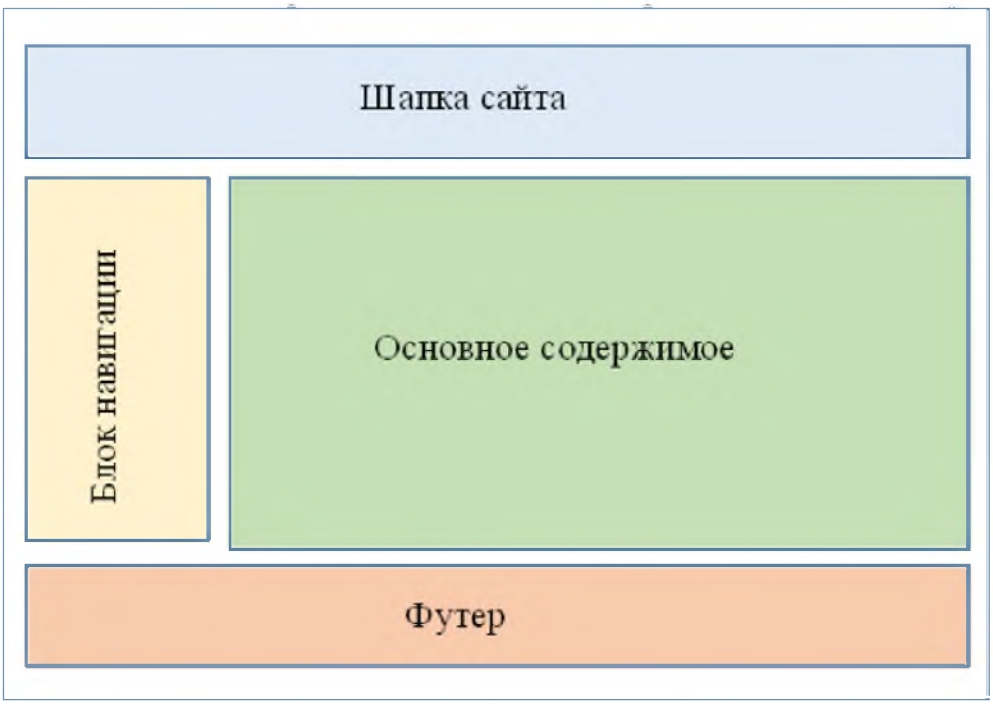
а именно в третий ряд. Для пятого блока свойство clear не используется, блок ведет себя как встроенный и становится на одну линию с предыдущим, четвертым блоком, сразу за ним.

Для вывода блоков с шестого по десятый используется второй стиль, где значение свойства float установлено в right. При выводе шестого блока свойство clear не используется, блок ведет себя как встроенный и располагается в самой правой позиции в той же линии, где уже есть четвертый и пятый блоки. Для вывода седьмого блока свойство clear установлено в left, поэтому он выводится таким образом, чтобы его верхний край находился ниже нижнего внешнего края всех ранее созданных левосторонних перемещаемых блоков, а именно блока 4, поскольку у него обтекание запрещено с обеих сторон. Поэтому седьмой блок становится самым правым на четвертой линии. Для восьмого блока свойство clear установлено в right, поэтому он будет выведен ниже любого нижнего внешнего края любого правостороннего перемещаемого блока, который был сгенерирован в исходном документе предыдущим элементом, т. е. ниже блока 7 на пятой линии. По этой же причине блок 9 выводится самым крайним справа на шестой линии: у него clear установлено в both. Для десятого блока свойство clear не используется, блок ведет себя как встроенный и становится на одну линию с предыдущим, девятым блоком, сразу слева перед ним. Результат выполнения представлен ниже.

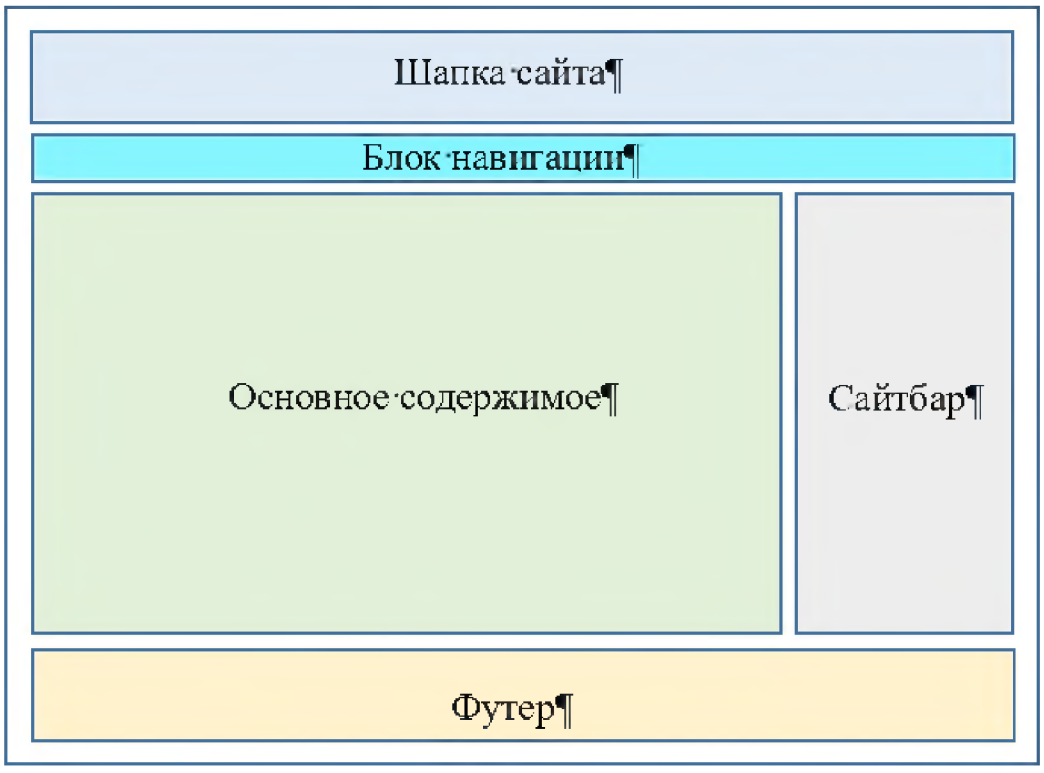
Упражнения

Используя блочную верстку, создайте следующие макеты HTML-страниц.

1. Фиксированная ширина страницы, блок навигации расположен слева.
2. «Резиновая» ширина страницы, блок навигации расположен слева.



3. Фиксированная ширина, блок навигации горизонтальный, расположен под шапкой сайта, справа от основного содержимого расположен сайтбар, внизу футер.



4. Расположите на странице три рядом стоящих на одной линии «резиновых» блока.

CSS цвета

Цвета в CSS могут определяться:

- как константы;
- значения RGB или RGBA;
- шестнадцатеричные значения;
- значения HSL и HSLA (CSS3);
- ключевым словом currentcolor.

Константы

Цвет может быть задан через имя константы, например "red". HTML и CSS поддерживают 140 стандартных названий цветов (http://w3schools.com/colors/colors_names.asp). Названия цветов нечувствительны к регистру.

Несмотря на то что названия цветов могут быть заданы 140 константами, HTML распознает только 16 основных цветовых ключевых слов, определенных еще в CSS1, остальные значения обрабатываются с помощью специального алгоритма для преобразования нераспознанных значений, иногда это совершенно разные цвета в различных браузерах. В CSS могут использоваться и все остальные ключевые слова для обозначения цвета. Если в HTML неизвестные ключевые слова для обозначения цвета обрабатываются специальным алгоритмом, то в CSS они будут полностью проигнорированы. Все ключевые слова цвета представляют собой сплошные цвета без прозрачности.

Несколько ключевых слов являются псевдонимами друг для друга, например aqua и cyan, fuchsia и magenta и т. д.

```
<!doctype html>
<html>
```

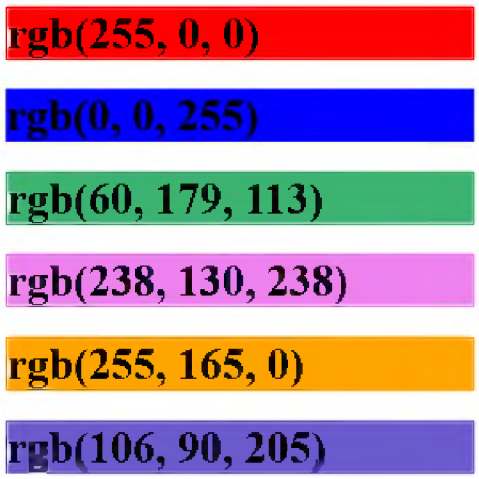
```
 <head>
  <meta charset="utf-8">
 </head>
 <body>
  <h1 style="background-color:Tomato;">Tomato</h1>
  <h1 style="background-color:Orange;">Orange</h1>
  <h1 style="background-
color:DodgerBlue;">DodgerBlue</h1>
  <h1 style="background-
color:MediumSeaGreen;">MediumSeaGreen</h1>
  <h1 style="background-color:Gray;">Gray</h1>
  <h1 style="background-color:SlateBlue;">SlateBlue</h1>
  <h1 style="background-color:Violet;">Violet</h1>
  <h1 style="background-color:LightGray;">LightGray</h1>
 </body>
</html>
```

RGB-цвета

В CSS цвет можно указать как значение RGB, используя соответствующую формулу. Каждый параметр (красный, зеленый и синий) определяет интенсивность цвета от 0 до 255. Например, rgb(255, 0, 0) отображается красным, потому что для красного установлено самое высокое значение (255), а для других – 0. Чтобы отобразить черный цвет, все параметры цвета нужно установить на 0, например rgb(0, 0, 0). Чтобы отобразить белый цвет, все параметры цвета нужно установить на 255, например rgb(255, 255, 255).

```
<!doctype html>
  <html>
    <head>
```

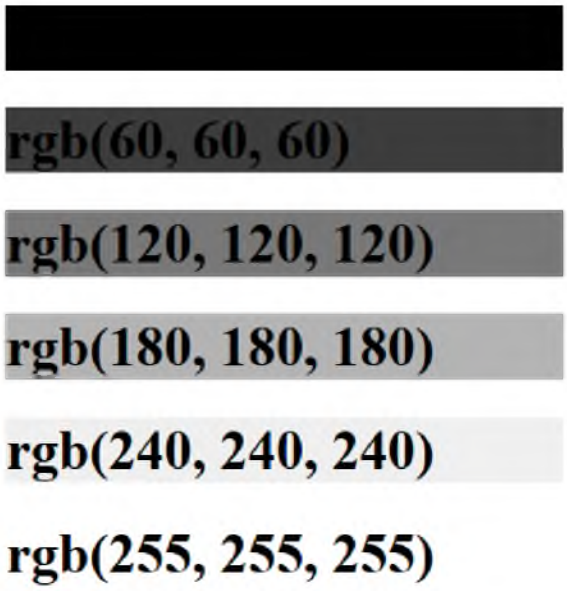
```
    <meta charset="utf-8">
  </head>
  <body>
    <h1 style="background-color:rgb(255, 0, 0);">
      rgb(255, 0, 0)
    </h1>
    <h1 style="background-color:rgb(0, 0, 255);">
      rgb(0, 0, 255)
    </h1>
    <h1 style="background-color:rgb(60, 179, 113);">
      rgb(60, 179, 113)
    </h1>
    <h1 style="background-color:rgb(238, 130, 238);">
      rgb(238, 130, 238)
    </h1>
    <h1 style="background-color:rgb(255, 165, 0);">
      rgb(255, 165, 0)
    </h1>
    <h1 style="background-color:rgb(106, 90, 205);">
       rgb(106, 90, 205)
    </h1>
  </body>
</html>
```



Оттенки серого часто определяются с использованием равных значений для всех трех источников света:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
```

```
    <h1 style="background-color:rgb(0, 0, 0);">
      rgb(0, 0, 0)
    </h1>
    <h1 style="background-color:rgb(60, 60, 60);">
      rgb(60, 60, 60)
    </h1>
    <h1 style="background-color:rgb(120, 120, 120);">
      rgb(120, 120, 120)
    </h1>
    <h1 style="background-color:rgb(180, 180, 180);">
      rgb(180, 180, 180)
    </h1>
    <h1 style="background-color:rgb(240, 240, 240);">
      rgb(240, 240, 240)
    </h1>
    <h1 style="background-color:rgb(255, 255, 255);">
      rgb(255, 255, 255)
    </h1>
  </body>
</html>
```



RGBA-цвета

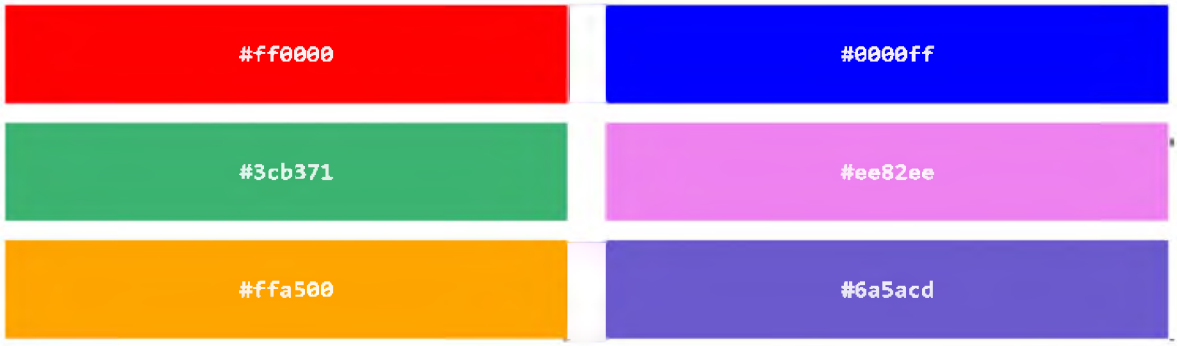
Можно задать цвет как значение RGBA (красный, зеленый, синий, прозрачность).

Параметры «красный», «зеленый» и «синий» определяют интенсивность цвета от 0 до 255. Могут выражаться процентами: число 255 соответствует 100 %. Прозрачность может быть выражена числом от 0 до 1 или процентами, где число 1 соответствует 100 % (полная непрозрачность).

```
rgba (51, 170, 51, .1) / * 10 % непрозрачный зеленый * /
rgba (51, 170, 51, .4) / * 40 % непрозрачный зеленый * /
RGBA (51, 170, 51, 0,7) / * 70 % непрозрачный зеленый * /
rgba (51, 170, 51, 1) / * полностью непрозрачный зеленый * /
```

Шестнадцатеричные значения

В CSS цвет можно указать с помощью шестнадцатеричного значения в форме #rrggbb, где rr (красный), gg (зеленый) и bb (синий) – шестнадцатеричные значения от 00 до ff (такие же, как десятичное 0–255). Например, #ff0000 задает красный цвет, потому что для красного установлено самое высокое значение (ff), а для других – самое низкое (00).



HSL- и HSLA-цвета

Цветовая модель HSL определяет цвет в соответствии с его компонентами оттенка, насыщенности и яркости. Необязательный альфа-компонент представляет прозрачность цвета.

`hsl(оттенок, насыщенность, яркость).`
`hsla(оттенок, насыщенность, яркость, прозрачность).`

Оттенок представляет собой угол (задается в градусах без указания единицы измерения) в диапазоне от 0° до 360° цветового круга относительно красного цвета. Красный цвет имеет угол 0°, или 360°, зеленый – 120°, синий – 240°. Отрицательные значения допускаются, например, 300° можно записать как –60°.

Насыщенность (saturation) и яркость (lightness) задаются в процентах.

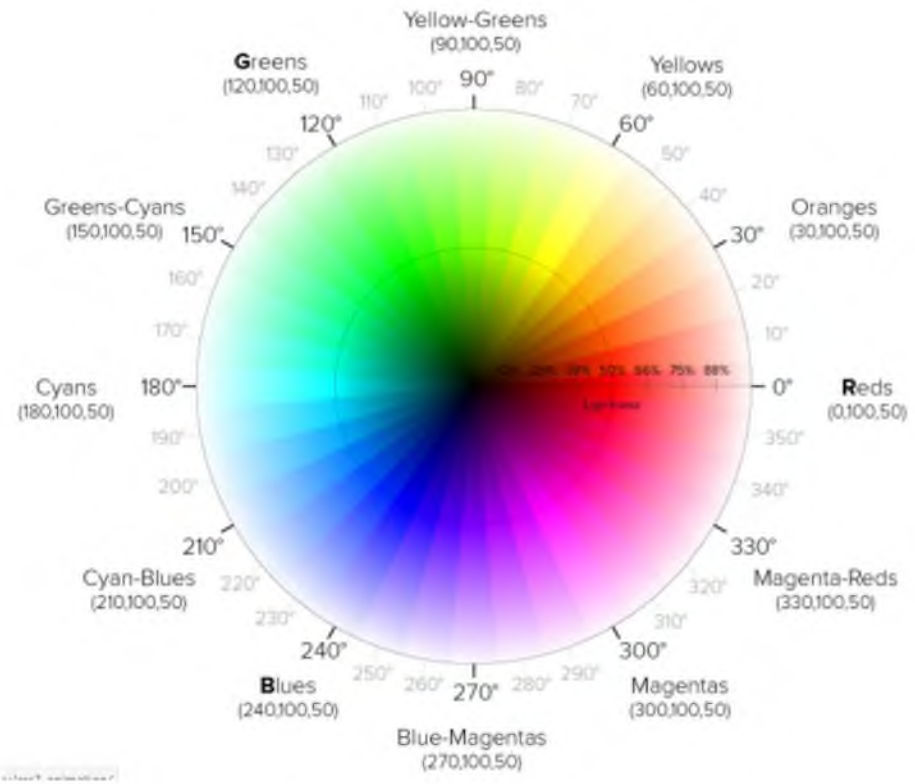
Цвета могут быть яркими (насыщенными) или тусклыми. Чем меньше цвета, тем более серый оттенок получается: 0 % насыщенности цвета – это серый, 100 % – чистый цвет.

Яркость (lightness) добавляет к цвету белый или черный, чтобы изменить его. Значение параметра яркости меньше 50 % означает, что добавляется черный цвет, выше – белый. Тон при этом остается неизменным.

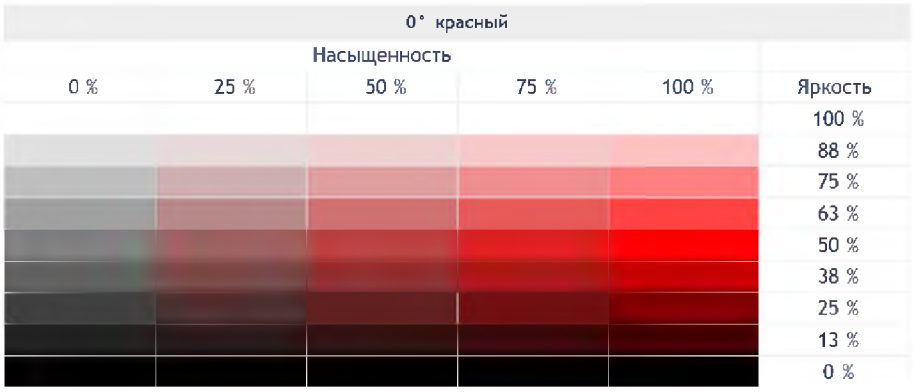
Прозрачность может быть задана числом от 0 до 1 или процентами, где число 1 соответствует 100 % (полная непрозрачность).

Многие дизайнеры считают модель HSL более интуитивной, чем RGB, поскольку она позволяет настраивать оттенок, насыщенность и яркость каждого цвета независимо. HSL также может упростить создание набора подходящих цветов (например, когда вам нужно несколько цветов одного оттенка).

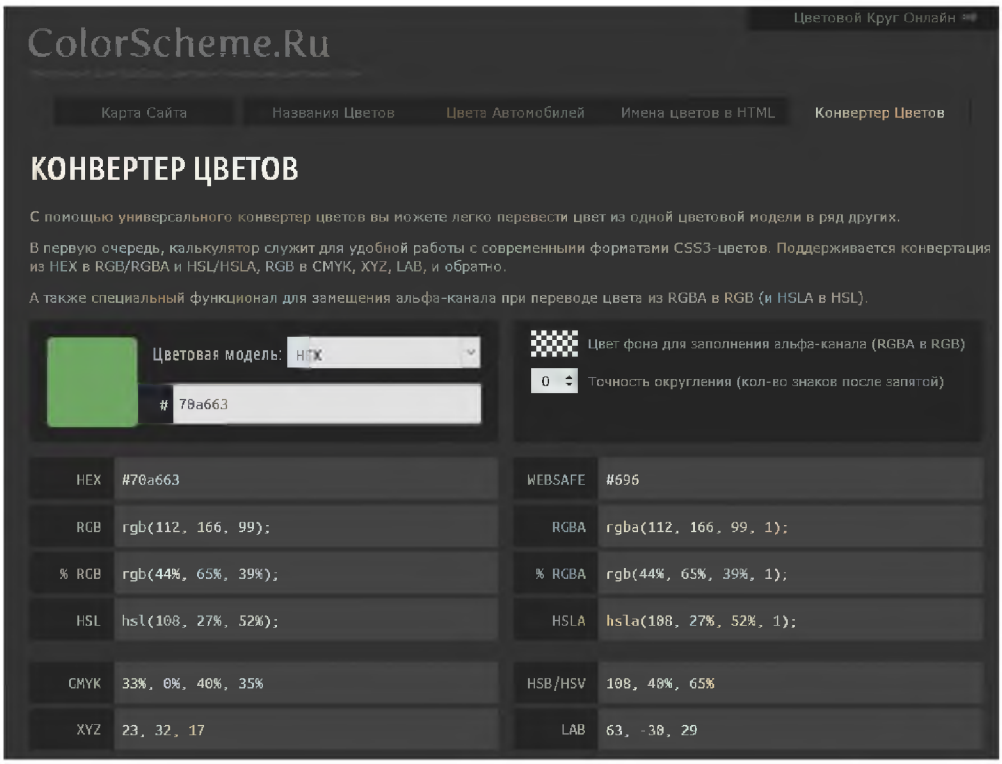
Цветовой круг:



Чтобы получить черный цвет, нужно присвоить показателям оттенка, насыщенности и яркости нулевое значение – hsla(0, 0%, 0%, 1). Белый цвет получается при 100 %-м значении яркости – hsla(0, 0%, 100%, 1), а серый цвет – при нулевом значении насыщенности – hsla(0, 0%, 50%, 1).

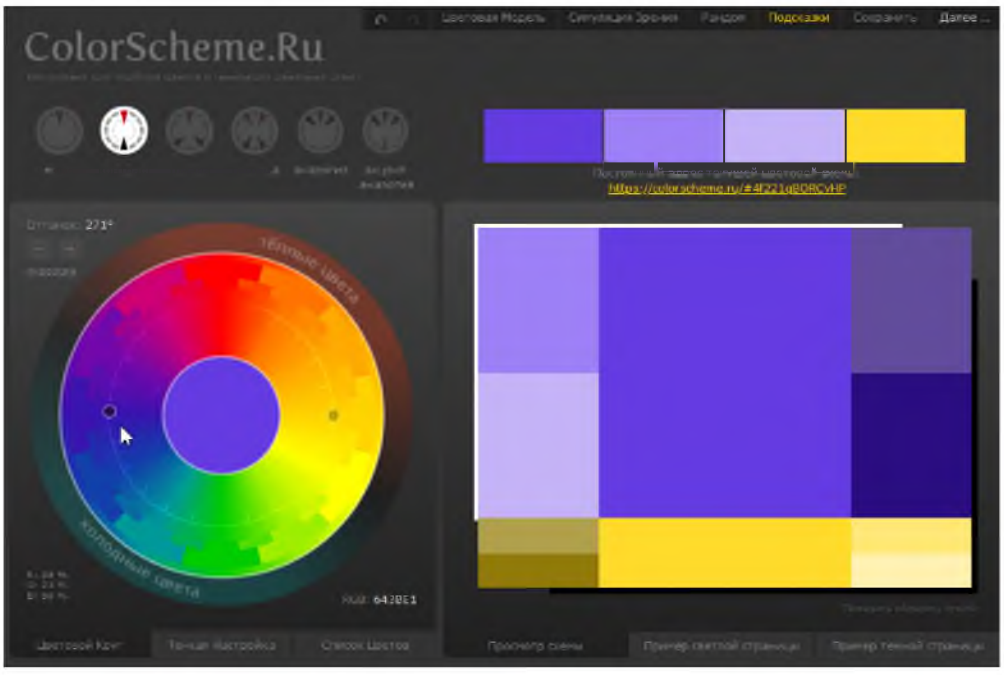


Есть конвертеры цветов из одного представления в другое, например https://colorscheme.ru/color-converter.html.



Кроме этого, данный ресурс позволяет выбрать цвет, а к нему автоматически подбирается один или несколько (по выбору) подходящих цветов и показывается, как выбранные цвета и их варианты сочетаются между собой.

Пример подбора одного контрастного цвета для синего:



currentcolor

Если у блока указать цвет границы как ***currentcolor***, то граница будет такого цвета, как цвет текста в блоке.

```
body {color: grey;}
div {border: 1px solid currentcolor;}
```

color

Свойство color устанавливает цвет символов текста элемента разметки. Цвет может задаваться стандартной константой или шестнадцатеричным значением RGB-палитры.

Фон элементов

У каждого HTML-элемента есть слой фона. Фон может быть полностью прозрачным (это значение по умолчанию), а может быть залит цветом и/или одним или несколькими изображениями.

Свойства фона не наследуются. Он не отображается у пустых элементов с нулевой высотой. Отрицательные значения свойства margin не влияют на фон элемента. Им управляет общее свойство background, которое является сокращенным свойством для background-color, background-image, background-position, background-size, background-repeat, background-origin, background-clip, background-attachment.

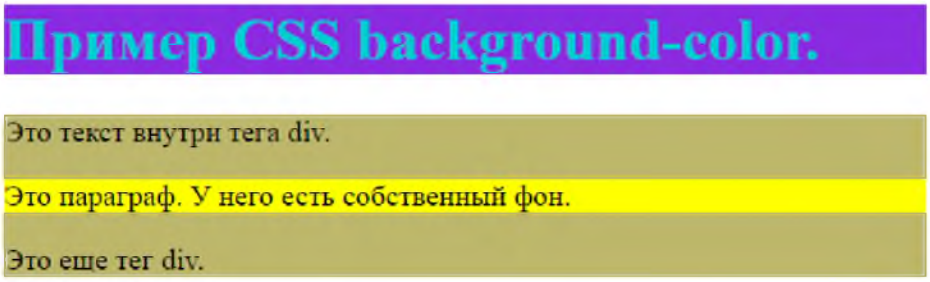
background-color

Свойство background-color устанавливает цвет фона элемента разметки. Если есть фоновый рисунок, то цвет заливается за ним. Для блочных элементов цвет фона распространяется на всю ширину и высоту блока элемента, включая padding и border, но не margin, для строчных – только на область их содержимого.

Свойства фона не наследуются, но если никаких свойств фона элемента не менять, то фон родительского блока будет просвечивать, поскольку по умолчанию значение background-color установлено в transparent.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      h1 {
```

```
            background-color: BlueViolet;
            color:DarkTurquoise;
           }
      div {background-color: DarkKhaki;}
      p {background-color: yellow;}
    </style>
  </head>
  <body>
    <h1>Пример CSS background-color.</h1>
    <div>
      Это текст внутри тега div.
      <p>Это параграф. У него есть собственный фон.</p>
      Это еще тег div.
    </div>
  </body>
</html>
```



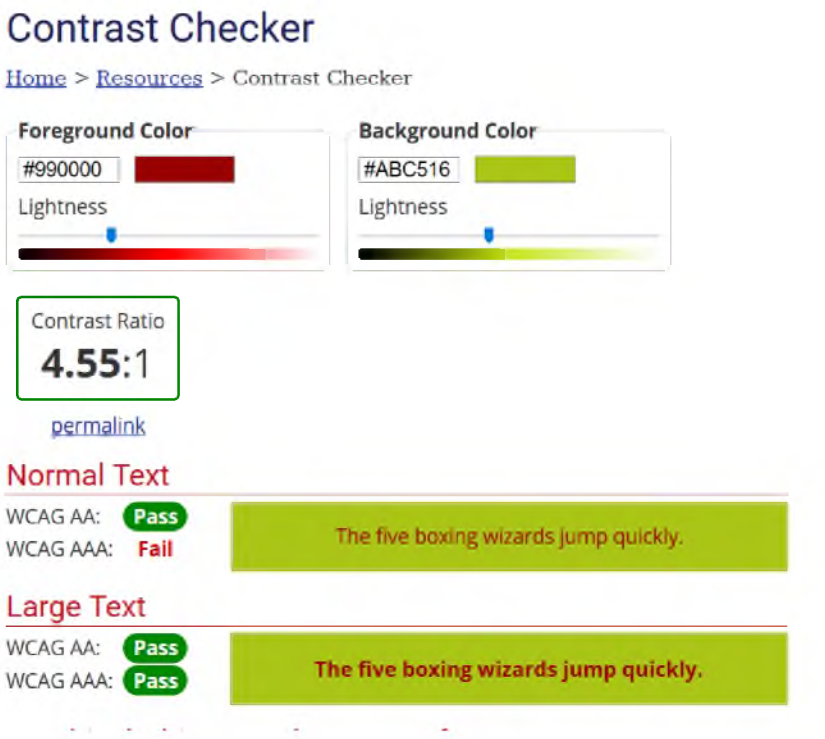
При выборе цветовой схемы для своего веб-сайта важно убедиться, что контрастность между цветом фона и цветом текста, размещенного над ним, достаточно высока, чтобы люди, испытывающие проблемы со зрением, могли прочитать содержимое страницы. Дизайн сайта может выглядеть отлично, но быть не слишком удобным при чтении текста. Или люди с нарушениями зрения, такими как дальтонизм, не смогут читать контент.

Существует простой способ проверить, достаточно ли хорош контраст, чтобы не вызывать проблем с восприятием. В интернете существует множество инструментов для проверки контрастности, например WebAIM: https://webaim.org/resources/contrastchecker/.

Кроме того, высокий коэффициент контрастности также позволяет лучше читать страницы с устройств, имеющих глянцевый экран, например смартфонов или планшетов, а также в условиях яркого освещения, такого как солнечный свет.

Коэффициент контрастности цвета определяется путем сравнения значений яркости текста и цвета фона. Для текстового содержимого требуется соотношение 4,5 : 1 и 3 : 1 для более крупного текста, такого как

заголовки. Крупный текст определяется как 18.66px жирным шрифтом или больше либо 24px или больше. Пример сравнения контрастности есть на https://webaim.org/resources/contrastchecker/.

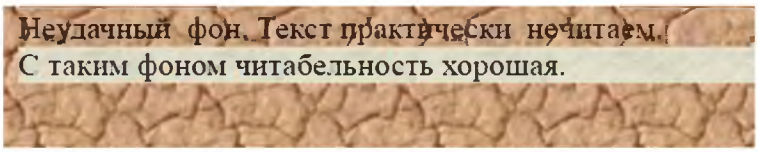


background-image

Свойство background-image определяет одно или несколько фоновых изображений для элемента. Если одновременно для элемента заданы и цвет фона, и фоновое изображение, то фон заданного цвета будет виден, пока фоновая картинка не загрузится полностью. Даже если изображения непрозрачны и цвет не будет отображаться в нормальных условиях, веб-разработчики всегда должны указывать background-color. Если изображения не могут быть загружены, например когда сеть не работает, цвет фона будет использоваться в качестве запасного варианта. Значением свойства является адрес файла изображения. Размер фонового рисунка не подстраивается автоматически под размер элемента разметки. Если размер изображения меньше размера элемента разметки, то по умолчанию фоновый рисунок повторяется. Если размер изображения больше размера элемента разметки, то будет видна только часть фонового рисунка. При использовании фонового изображения надо использовать изображение, которое не мешает тексту.

```
<!doctype html>
<html>
 <head>
```

```
   <style>
    body {   background-image: url(images/bgdesert.jpg);}
    .paper {    background-image: url(images/paper.gif);}
   </style>
 </head>
 <body>
  <div >Неудачный фон. Текст практически нечитаем.</div>
  <div   class="paper">С   таким   фоном   читабельность
хорошая.</div>
 </body>
</html>
```



background-repeat

Свойство background-repeat определяет, будет ли дублироваться фоновое изображение, установленное с помощью параметра background-image, если размер изображения меньше размера элемента разметки.

Свойство background-repeat может принимать следующие значения:

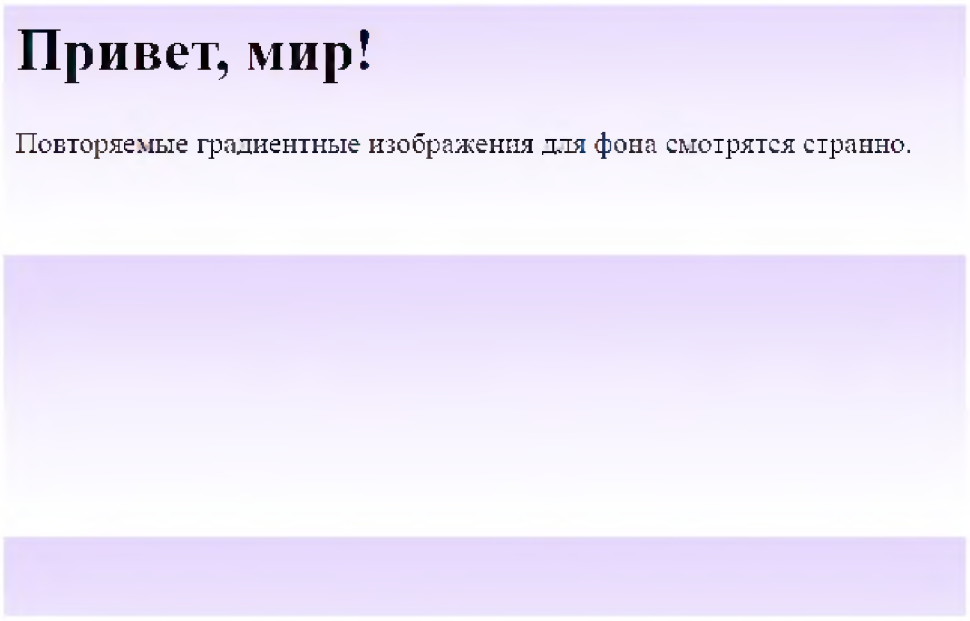
- repeat – изображение дублируется по вертикали и горизонтали так часто, чтобы покрыть область отрисовки фона. Если изображение не помещается, оно обрезается. Является значением по умолчанию;
- repeat-x – изображение дублируется только по горизонтали;
- repeat-y – изображение дублируется только по вертикали;
- no-repeat – изображение не дублируется;
- space – изображение дублируется столько раз, сколько оно помещается в области фона, не обрезаясь, изображения расположены на равном расстоянии друг от друга. Первое и последнее изображения касаются краев области. Если область рисования фона больше, чем область позиционирования фона, шаблон повторяется, чтобы заполнить область рисования фона. Если недостаточно места для двух копий изображения, то размещается только одно изображение, а свойство background-position определяет его положение;
- round – изображение повторяется так часто, чтобы заполнить область фона, масштабируясь и не обрезаясь;
- no-repeat – изображение размещается один раз и не повторяется;
- initial – устанавливает значение свойства в значение по умолчанию;
- inherit – наследует значение свойства от родительского элемента.

Не всегда значение по умолчанию уместно. Повторяемые градиентные изображения для фона смотрятся странно.

```
<body style="background-
image:url(images/gradient_bg.png)">
  <h1>Привет, мир!</h1>
Повторяемые градиентные изображения для фона смотрятся
странно.
</body>
```

Здесь в качестве фонового рисунка взято градиентное изображение.

В итоге получаем полосатый фон, что не всегда удобно и красиво.



Если использовать небольшое фоновое изображение, то, отрисовывая его определенным образом, можно получить декоративный элемент, например ленту-тесьму.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      p{
        background-image:url(BD10253_.GIF);
```

```
        width:100px;
        height:100px;
        border-style:double;
        border-color:olive;
        border-width:4px;
        font-size:10px;
        }
    </style>
  </head>
  <body>
    background-position:center 25%;
    background-repeat:repeat-x
    <p style="background-position:center 25%;
              background-repeat:repeat-x">
    </p>
  </body>
</html>
```

background-position:center 25%; background-repeat:repeat-x

background-position

Свойство background-position содержит координаты левого верхнего угла фонового изображения. Это комбинированное свойство, заменяющее атрибуты background-position-x и background-position-y. Значения могут задаваться в процентах и абсолютных координатах. Если задана только одна координата, то она считается горизонтальной, а для вертикальной координаты принимается значение в 50 %. Значение по умолчанию – 0 %. Ниже приведены результаты использования свойства.

В приведенном примере установлено фоновое изображение, размер которого значительно меньше размера блока. Дублирование фоновой картинки не разрешено: `background-repeat:no-repeat`. Затем выводится блок с различными значениями свойства background-position.

Можно указать, что фоновое изображение не должно повторяться в background-repeat:no-repeat. При этом при помощи свойства background-position можно указать для него место вывода, где оно не будет мешать воспринимать информацию. Тогда фоновое изображение будет показываться только один раз и располагаться вдали от текста, как в примере ниже.

```
body {
  background-image: url(images/img_tree.png);
  background-repeat:no-repeat;
  background-position:right top;
  margin-right: 200px;
  }
...
<body>
  <h1>Привет, мир!</h1>
  Пример … тексту.
</body>
```

background-attachment

Свойство background-attachment определяет, будет ли прокручиваться фоновое изображение вместе с содержимым элемента или оно будет зафиксировано. Применение этого свойства имеет смысл, если контент элемента достаточно большой и целиком не виден, а для полного просмотра содержимого есть полосы прокрутки.

Допустимы следующие значения свойства:

- scroll – фоновое изображение прокручивается вместе с контентом;

• fixed – фоновое изображение остается неподвижным во время прокрутки содержимого элемента;

• local – фоновое изображение прокручивается вместе с содержимым элемента;

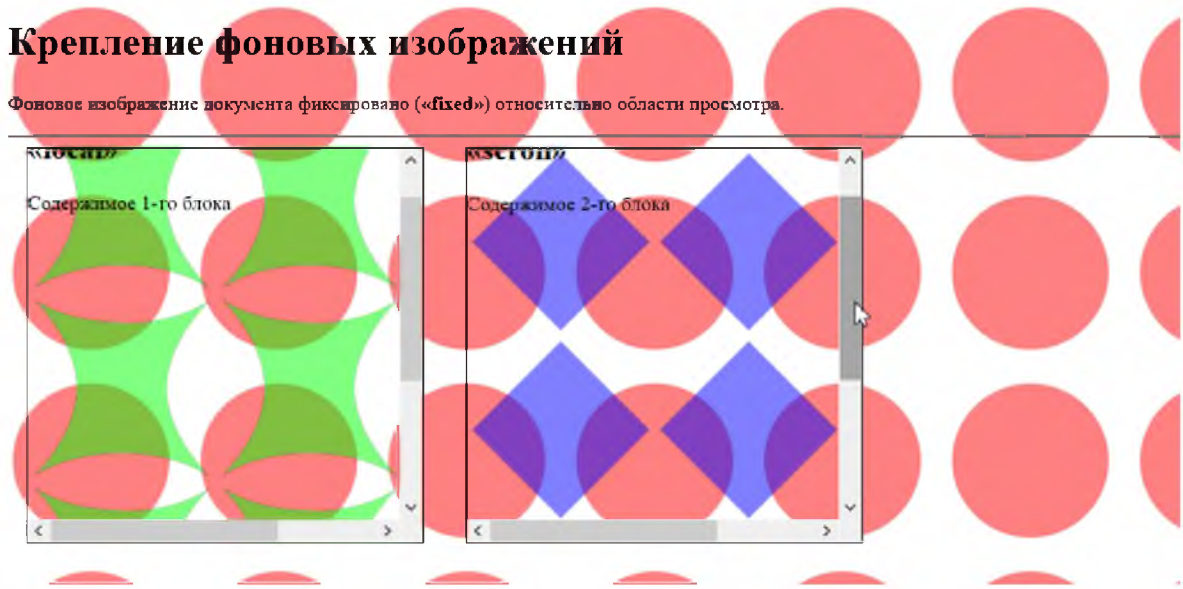
• initial – устанавливает значение свойства в значение по умолчанию;

• inherit – наследует значение свойства от родительского элемента.

В приведенном ниже примере основной фон с текстом не прокручивается. В первом блоке зеленый фон прокручивается с текстом. Во втором блоке фон фиксирован.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style type="text/css">
     body {
       min-height: 550px;
       background-image: url("images/fig-red.png");
       background-attachment: fixed;
       }
     div {
       display: inline-block;
       margin: 0 15px;
       width: 315px;
       height: 315px;
       border: 1px solid black; overflow: scroll;
       }
     div p {
       width: 400px;
       height: 400px;
       }
    </style>
  </head>
  <body>
    <h1>Крепление фоновых изображений</h1>
    <p>Фоновое изображение документа фиксировано
      (<b>«fixed»</b>) относительно области просмотра.
    </p>
    <hr>
    <div style="background-image:
         url('images/fig-green.png');
         background-attachment: local;">
      <h2>«local»</h2>
      <p>Содержимое 1-го блока</p>
```

```
    </div>
    <div style="background-image:
                url('images/fig-blue.png');
                background-attachment: scroll;">
      <h2>«scroll»</h2><p>Содержимое 2-го блока</p>
    </div>
  </body>
</html>
```



В примере намеренно были взяты яркие и крупные фоновые изображения, чтобы продемонстрировать действие свойства. Как видно из рисунка, такой фон сильно затрудняет восприятие.

background-clip

Свойство background-clip определяет область рисования фона. Фон всегда рисуется под рамкой элемента, если таковая имеется.

Корневой элемент <html> имеет другую область рисования фона, поэтому свойство background-clip на него не влияет.

Возможны следующие значения свойства:

- border-box – фон закрашивает область в пределах рамки элемента, значение по умолчанию;
- padding-box – фон закрашивает область в пределах внутренних полей элемента;
- content-box – фон закрашивает только область содержимого;
- initial – устанавливает значение свойства в значение по умолчанию;.
- inherit – наследует значение свойства от родительского элемента.

background-origin

Определяет исходную позицию (область позиционирования фона) фонового изображения. Допустимые значения:

- padding-box – фон позиционируется относительно верхних границ области внутренних полей элемента. Значение по умолчанию;
- border-box – фон позиционируется относительно верхних границ рамки элемента;
- content-box – фон позиционируется относительно верхних границ области содержимого элемента;
- initial – устанавливает значение свойства в значение по умолчанию;
- inherit – наследует значение свойства от родительского элемента.

свойство background-size

Свойство background-size устанавливает размер фоновых изображений. Значения свойства:

• auto – значение по умолчанию, высота и ширина изображения равны его оригинальным размерам;

• размер – задается парой значений, первое значение устанавливает ширину изображения, второе – высоту (для того чтобы фон масштабировался вместе с текстом, размеры изображения нужно задавать в em);

• % – задает размер фонового изображения в процентах от ширины или высоты элемента, который заполняется фоном;

• cover – масштабирует изображение с сохранением пропорций так, чтобы его ширина или высота равнялась ширине или высоте блока;

• contain – масштабирует изображение с сохранением пропорций таким образом, чтобы оно целиком поместилось внутри блока;

• initial – устанавливает значение свойства в значение по умолчанию;

• inherit – наследует значение свойства от родительского элемента.

background

Свойство background является объединением свойств background-color, background-image, background-position, background-size, background-repeat, background-origin, background-clip и background-attachment. Значения должны быть перечислены через пробел. Значения свойства background-size нужно записывать через слеш (/), чтобы отделить его от свойства background-position. Необязательно указывать все перечисленные свойства. Сначала свойство background устанавливает всем перечисленным свойствам фона их значения, а неупомянутым атрибутам устанавливает значения по умолчанию.

Динамическое задание фонового рисунка

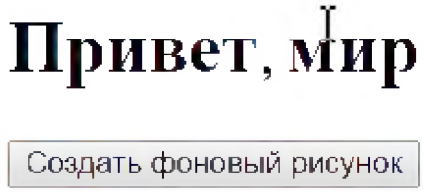
Фон можно добавлять или изменять при помощи JavaScript.

Вариант первый:

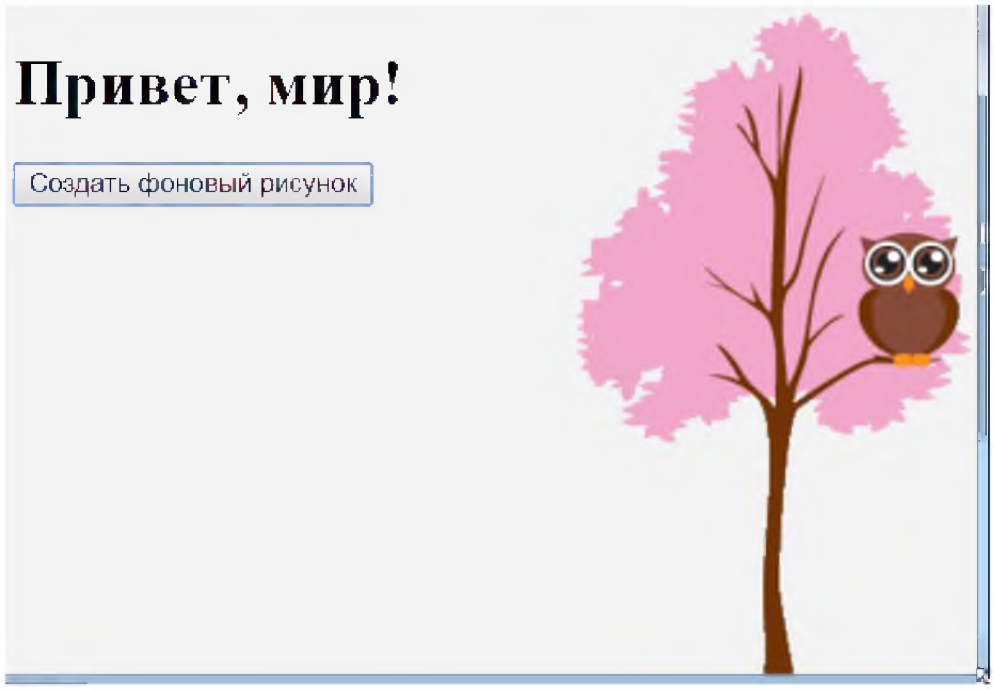
```
<!doctype html>
<html>
  <head>
…
  </head>
  <body>
    <h1>Привет, мир!</h1>
    <button onclick="myFunction()">
      Создать фоновый рисунок
    </button>
    <script>
```

```
      function myFunction() {
       document.body.style.background =
       "#f3f3f3
       url('https://www.w3schools.com/css/img_tree.png')
       no-repeat right top";}
    </script>
  </body>
</html>
```

Первоначально на странице есть некоторый контент и кнопка «Создать фоновый рисунок».



После нажатия кнопки для элемента ***body*** устанавливается фон.



Вариант второй (с использованием DOM):

```
<!doctype HTML>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      #myDIV {height: 380px;}
    </style>
```

```
  </head>
  <body>
    <p>
      Нажмите кнопку, чтобы изменить цвет фона в теге div.
    </p>
    <button onclick="myFunction()">
      Изменить фон
    </button>
    <div id="myDIV">
      <p>Тег div</p>
    </div>
    <script>
      function myFunction() {
document.getElementById("myDIV").style.backgroundColor =
"#f3f3f3";
document.getElementById("myDIV").style.backgroundImage
="url('images/img_tree.png')";
document.getElementById("myDIV").style.backgroundRepeat=
"no-repeat";
document.getElementById("myDIV").style.backgroundPositio
n="right top";}
    </script>
  </body>
</html>
```

Вид веб-страницы такой же, как и в первом варианте.

overflow

Свойство overflow управляет отображением содержания блочного элемента, если контент элемента целиком не помещается в отведенные ему размеры. Такая ситуация может возникнуть, если для блока были установлены фиксированные значения width и height и они оказались недостаточными.

Свойство overflow может принимать следующие значения:

- visible – полосы прокрутки не отображаются. Текст выводится полностью, а значения width и height остаются прежними, поэтому контент блока может наползать на содержимое следующего блока;
- scrool – значения width и height остаются постоянными, независимо от величины содержимого блока. При этом всегда отображаются полосы прокрутки, даже если содержимое элемента разметки полностью помещается в отведенных ей пределах;
- hidden – элемент разметки не меняет размеров, а отображается в соответствии с фиксированными значениями width и height. При этом если

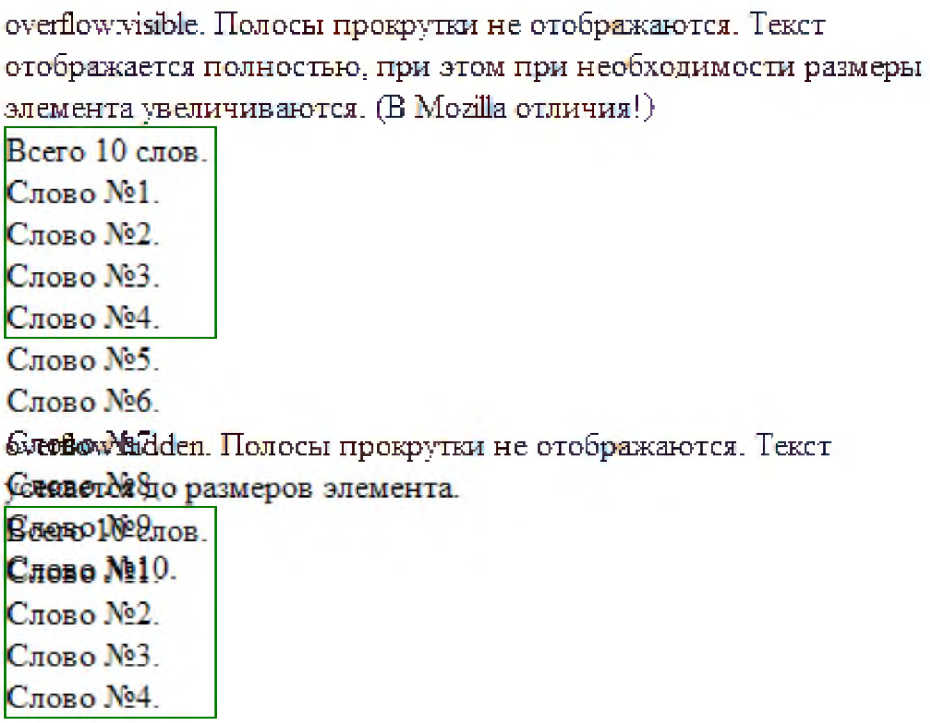
содержимое элемента не помещается в отведенную ему область, то оно усекается, полосы прокрутки не отображаются.

- auto – элемент разметки размеров не меняет и отображается в соответствии с фиксированными значениями width и height. При необходимости появляются полосы прокрутки.

```
<!doctype html>
<html>
  <head>
    <style>
      div{
        width:100px;
        height:100px;
        border:solid thin green;
        }
    </style>
  </head>
  <body>
    overflow:visible. Полосы прокрутки не отображаются.
    Текст отображается полностью, при этом при
    необходимости размеры элемента увеличиваются.
    (В Mozilla отличия!)
    <div style="overflow:visible">Всего 10 слов.
      Слово №1. Слово №2. Слово №3. Слово №4. Слово №5.
      Слово №6. Слово №7. Слово №8. Слово №9.
      Слово №10.
    </div>
    <br>
    <br>
    overflow:hidden. Полосы прокрутки не отображаются.
    Текст усекается до размеров элемента.
    <div style="overflow:hidden">
      Всего 10 слов. Слово №1. Слово №2. Слово №3.
      Слово №4. Слово №5. Слово №6. Слово №7. Слово №8.
      Слово №9. Слово №10.
    </div>
    <br>
    <br>
    overflow:scroll. Полосы прокрутки отображаются
    всегда.
    <div style="overflow:scroll">
      Всего 3 слова. Слово №1. Слово №2.
      Слово №3.
    </div>
    <br>
```

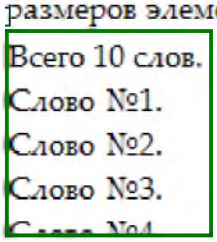
```
    <br>
    overflow:auto. Полосы прокрутки появляются при
    необходимости.
    <div style="overflow:auto">
      Всего 10 слов. Слово №1. Слово №2. Слово №3.
      Слово №4. Слово №5. Слово №6. Слово №7.
      Слово №8. Слово №9. Слово №10.
    </div>
    <br>
    <br>
    overflow:auto. Полосы прокрутки появляются
    при необходимости.
    <div style="overflow:auto">
      Всего 3 слова. Слово №1. Слово №2. Слово №3.
    </div>
  </body>
</html>
```

В приведенном примере для всех блоков установлены фиксированные значения таким образом, что содержимое блоков не помещается в отведенные им пределы. Размер блока не увеличился, но его контент выводится полностью, что приводит к наложению текста на последующие блоки и потере читабельности.

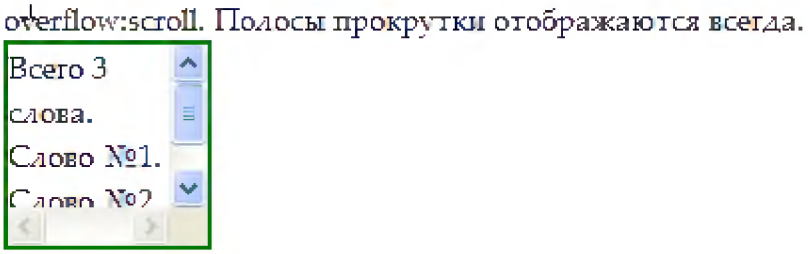


Для второго блока `overflow:hidden`, поэтому блок не изменяет своих размеров, а его содержимое видно только частично.

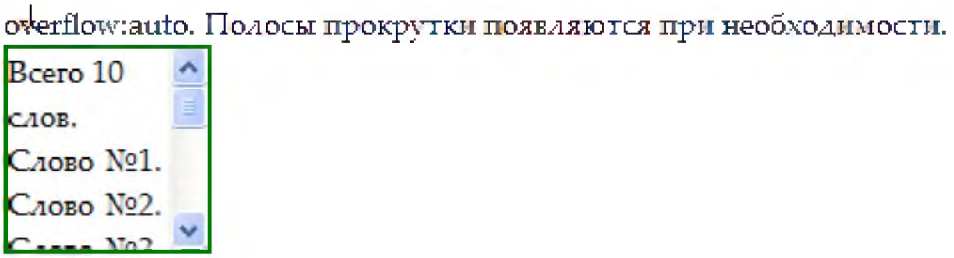
overflow:hidden. Полосы прокрутки не отображаются. Текст усекается до размеров элемента.



`overflow:scroll`. В этом случае полосы прокрутки отображаются всегда, даже если в этом нет необходимости. Здесь вертикальная полоса прокрутки необходима, а горизонтальная – нет.



`overflow:auto`. Появилась только вертикальная полоса прокрутки, горизонтальной полосы прокрутки нет, так как в ней нет необходимости.



position

Свойство position устанавливает способ позиционирования элемента относительно окна браузера или других объектов на HTML-странице.

Свойство может принимать следующие значения:

- absolute – устанавливает абсолютные координаты относительно левого верхнего угла контейнера блока. Координаты указываются с помощью параметров left, right, top и bottom. Абсолютно позиционируемые блоки изымаются из нормального потока. Это значит, что они не влияют на размещение последующих сестринских элементов;
- fixed – устанавливает абсолютные координаты относительно левого верхнего угла окна браузера. Координаты указываются с помощью параметров left, right, top и bottom. При прокрутке содержимого окна браузера элемент разметки не смещается;

• relative – положение элемента устанавливается относительно его исходного места. Добавление атрибутов left, top, right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения в зависимости от применяемого параметра;

• static – игнорирует установку координат с помощью свойств left, top, right и bottom.

visibility

Свойство visibility указывает браузеру, отображать ему элемент разметки или нет. Свойство может принимать следующие значения:

• visible – отображать;

• hidden – не отображать;

• collapse – если это значение применяется не к строкам или колонкам таблицы, то результат его использования будет таким же, как hidden. В случае использования collapse для содержимого ячеек таблиц они реагируют, словно к ним было добавлено стилевое свойство display: none. Иными словами, заданные строки и колонки убираются, а таблица перестраивается заново.

Свойства шрифта и текста

direction

Свойство direction устанавливает порядок вывода текста: слева направо (значение по умолчанию), как в европейских языках, или справа налево, как в арабских языках.

Свойство может принимать одно из двух значений:

• ltr – выводить текст слева направо;

• rtl – выводить текст справа налево.

Как правило, используется в паре со свойством unicode-bidi.

unicode-bidi

Свойство unicode-bidi определяет поведение встроенных элементов при изменении направления вывода текста с помощью атрибута direction.

Возможные значения свойства unicode-bidi:

• normal – направление вывода текста должно измениться и у родительского элемента (значение по умолчанию);

• embed – направление вывода текста изменяется только у самого элемента;

• bidi-override – направление вывода текста изменяется только у данного элемента согласно значению атрибута direction, независимо от локальных установок веб-обозревателя.

Приведем пример использования свойств direction и unicode-bidi.

font-family

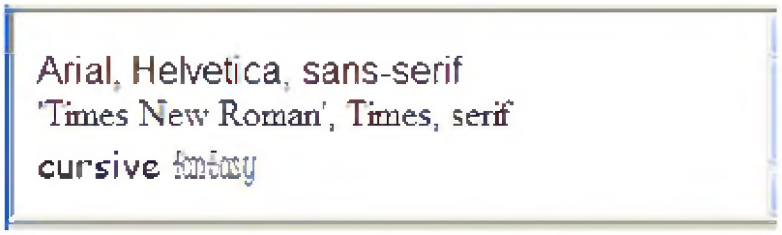
Свойство font-family определяет название шрифта или семейства шрифтов, необходимых для корректного вывода HTML-документа. Например, если в контенте будет встречаться русский текст и математические символы, то необходимо указать по крайней мере два шрифта: для вывода текста и для вывода формул. Если название шрифта состоит из нескольких слов, то оно должно заключаться в кавычки. Предпочтительнее всегда указывать одновременно несколько шрифтов, разделив их имена запятыми, поскольку на компьютере пользователя может не оказаться того шрифта, в котором был подготовлен документ. В этом случае браузер сможет выбрать тот шрифт из списка, который установлен на компьютере клиента. Кроме того, указывается гарнитура шрифта:

• serif – шрифты с засечками, например Times;
• sans-serif – рубленые шрифты, например Arial;
• cursive – курсивные шрифты;
• fantasy – декоративные шрифты;
• monospace – моноширинные шрифты, в которых ширина каждого символа одинаковая.

По умолчанию шрифт Times соответствует значению serif, шрифт Helvetica – sans-serif, шрифт Zapf-Chancery – cursive, шрифт Western – fantasy, шрифт Courier – monospase.

Приведем пример использования свойства font-family.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  <body>
    <div style="font-family:Arial, Helvetica,
           sans-serif">Arial, Helvetica, sans-serif
    </div>
    <div style="font-family:'Times New Roman',
           Times, serif">'Times New Roman', Times, serif
    </div>
    <span style="font-family:cursive;">cursive </span>
    <span style="font-family:fantasy;">fantasy</span>
</body>
</html>
```



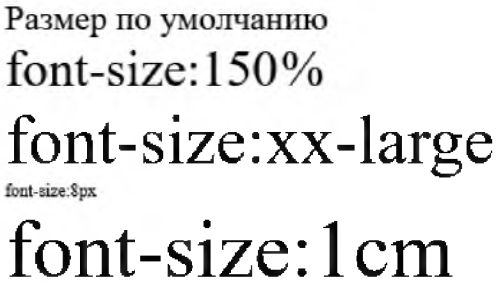
font-size

Свойство font-size управляет размером шрифта. Размер шрифта может быть задан явно. В этом случае он может задаваться в любых единицах, определенных CSS, как абсолютных, так и относительных. При указании размера в процентах за 100 % берется размер шрифта родительского элемента. Отрицательные значения не допускаются.

Для задания абсолютных размеров шрифта определены семь констант: xx-small, x-small, small, medium, large, x-large, xx-large. Значения этих констант зависят от настроек браузера и операционной системы. Относительные размеры шрифта можно определить с помощью двух констант: larger, smaller. В этом случае размер шрифта текущего элемента определяется относительно размера шрифта родительского элемента.

Приведем пример использования свойства font-size.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Свойство FONT-SIZE</title>
  </head>
  <body>
    <div>Размер по умолчанию</div>
    <div style="font-size:150%">font-size:150%</div>
    <div style="font-size:xx-large">
      font-size:xx-large
    </div>
    <div style="font-size:8px">font-size:8px</div>
    <div style="font-size:1cm">font-size:1cm</div>
  </body>
</html>
```



font-style

Свойство font-style задает тип начертания шрифта. Свойство может принимать следующие значения:

- normal – обычное начертание шрифта;
- italic – курсивное начертание;
- oblique – наклонный шрифт. Наклонный шрифт внешне похож на курсивный, но отличается от него. Курсив – это специальный шрифт, имитирующий рукописный, наклонный шрифт образуется путем наклона символов обычного шрифта вправо.

Приведем пример использования свойства font-style.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Свойство FONT-STYLE</title>
   </head>
   <body>
    <div style="font-style:italic">
      font-style:italic курсив
    </div>
    <div style="font-style:normal">
      font-style:normal
    </div>
    <div style="font-style:oblique">
      font-style:oblique курсив + полужирный (но в браузере
      курсив)
    </div>
    <div style="font-style:italic; font-family:fantasy">
      font-style:italic курсив
    </div>
    <div style="font-style:normal; font-family:fantasy">
      font-style:normal
    </div>
    <div style="font-style:oblique; font-family:fantasy">
      font-style:oblique курсив + полужирный (но
в браузере
      курсив)
    </div>
  </body>
</html>
```

font-style:italic курсив
font-style:normal
font-style:oblique курсив + полужирный (но в браузере курсив)
font-style:italic курсив
font-style:normal
font-style:oblique курсив + полужирный (но в браузере курсив)

font-variant

Свойство font-variant определяет, как будут выводиться строчные символы: в виде прописных букв, но меньшего размера, или обычным образом. Способ вывода строчных символов в виде прописных называется капителью. Свойство font-variant определяет выбор варианта шрифта, обладающего двумя наборами знаков (т. е. капителью), и не имеет видимого эффекта для шрифтов, обладающих одним набором знаков. Свойство может принимать следующие допустимые значения:

- normal – регистр вывода строчных символов не изменяется (значение по умолчанию);
- small-caps – строчные символы выводятся капителью, как заглавные уменьшенного размера.

Приведем пример использования свойства font-variant.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <div style="font-variant:normal">font-variant:normal
      ПРОПИСНЫЕ БУКВЫ. строчные буквы.</div>
    <div style="font-variant:small-caps">font-
      variant:small-caps ПРОПИС-НЫЕ БУКВЫ. строчные
      буквы.</div>
  </body>
</html>
```

font-variant:normal ПРОПИСНЫЕ БУКВЫ. строчные буквы.
FONT-VARIANT:SMALL-CAPS ПРОПИС-НЫЕ БУКВЫ. СТРОЧНЫЕ БУКВЫ.

font-weight

Свойство font-weight устанавливает степень выделения, или жирность, символов текста при выводе. Свойство может принимать значения:

• 100, 200, 300, 400, 500, 600, 700, 800, 900 – это условные единицы, каждое значение представляет степень выделения текста по возрастанию. Нормальное начертание шрифта, установленное по умолчанию, – 400. Самое светлое начертание шрифта, отображаемое браузером, – 100, самое жирное – 900;

• bold – полужирное начертание, соответствует значению 700;

• bolder – более жирное начертание символов, чем в родительском элементе. Если в родительском элементе значение font-weight равно 900, то результирующим значением будет также 900;

• lighter – более светлое начертание символов, чем в родительском элементе. Если в родительском элементе значение font-weight равно 100, то результирующим значением будет также 100;

• normal – нормальное начертание шрифта, принятое по умолчанию.

• Приведем пример использования свойства font-weight.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
<div style="font-weight:200">font-weight:200</div>
<div style="font-weight:800">font-weight:800</div>
<div style="font-weight:900">font-weight:900</div>
<div style="font-weight:bold">font-weight:bold</div>
<div style="font-weight:bolder">font-weight:bolder</div>
<div style="font-weight:normal">font-weight:normal</div>
<div     style="font-weight:lighter">font-weight:lighter
</div>
  </body>
</html>
```

font-weight:200
font-weight:800
font-weight:900
font-weight:bold
font-weight:bolder
font-weight:normal
font-weight:lighter

line-height

Свойство line-height, устанавливает межстрочный интервал для текста, или расстояние между базовыми линиями соседних строк текста. По

умолчанию расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически, что соответствует значению свойства line-height, равному normal. Может задаваться абсолютным значением в любых единицах длины, поддерживаемых CSS. Может также задаваться числом без указания единиц измерения. В этом случае значение воспринимается как множитель от размера шрифта текущего текста. Отрицательное значение межстрочного расстояния не допускается. Может быть задано относительной величиной в процентах. За 100 % принимается размер шрифта текущего текста.

font

Свойство font дает возможность задать сразу шесть свойств шрифта: font-family, font-size, font-style, font-variant, font-weight и line-height. Значения должны быть перечислены через пробел и могут идти в любом порядке: браузер сам определит, какое из них соответствует нужному свойству. Поскольку по крайней мере три параметра допускают числовые значения, следует каждый раз проверять интерпретацию браузера.

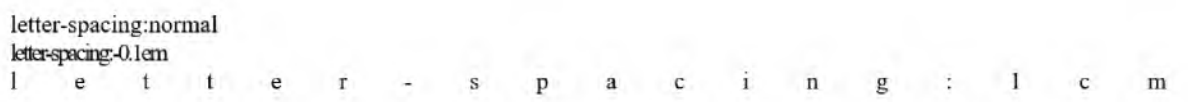
letter-spacing

Свойство letter-spacing устанавливает интервал между символами текста. Свойство может принимать значение normal, что означает, что задается стандартный интервал между символами текста. Величина этого интервала определяется браузером, исходя из типа шрифта, его размеров и настроек операционной системы. Можно задать расстояние между символами и вручную, указав в свойстве letter-spacing значение в любых абсолютных единицах длины, принятых в CSS, или в относительных единицах, основанных на размере шрифта, em и ex. Это значение задает дополнительное межсимвольное расстояние, которое добавляется к стандартному. Значения могут быть отрицательными. При использовании выравнивания содержимого по ширине для блока браузеры могут не увеличивать или не уменьшать расстояние между символами.

Приведем пример использования свойства letter-spacing.

```
<!doctype html>
<html>
 < head>
    <meta charset="utf-8">
  </head>
  <body>
    <div style="letter-spacing:normal">
      letter-spacing:normal
    </div>
    <div style="letter-spacing:-0.1em">
      letter-spacing:-0.1em
```

```
    </div>
    <div style="letter-spacing:1cm">
      letter-spacing:1cm
    </div>
  </body>
</html>
```



text-align

Свойство text-align задает тип горизонтального выравнивания текста в пределах блока. Свойство может принимать следующие значения:

- left – выравнивание текста по левому краю браузера или контейнера, в котором он расположен, при этом правый край текста может располагаться лесенкой;
- center – выравнивание текста по центру;
- right – выравнивание текста по правому краю браузера или контейнера, в котором он расположен, при этом левый край текста может располагаться лесенкой;
- justify – выравнивание текста по ширине браузера или контейнера, в котором он расположен, при этом в строку текста могут быть равномерно добавлены пробелы, для того чтобы и левый, и правый края текста были выровнены.

text-decoration

Свойство text-decoration может добавлять такие эффекты оформления текста, как подчеркивание, надстрочное подчеркивание, перечеркивание и мигание. Им отвечают значения свойства underline, overline, line-through и blink соответственно. Значение none отменяет все эффекты оформления, в том числе и подчеркивание ссылок, которое задается по умолчанию. Свойство выборочно поддерживается различными браузерами.

text-indent

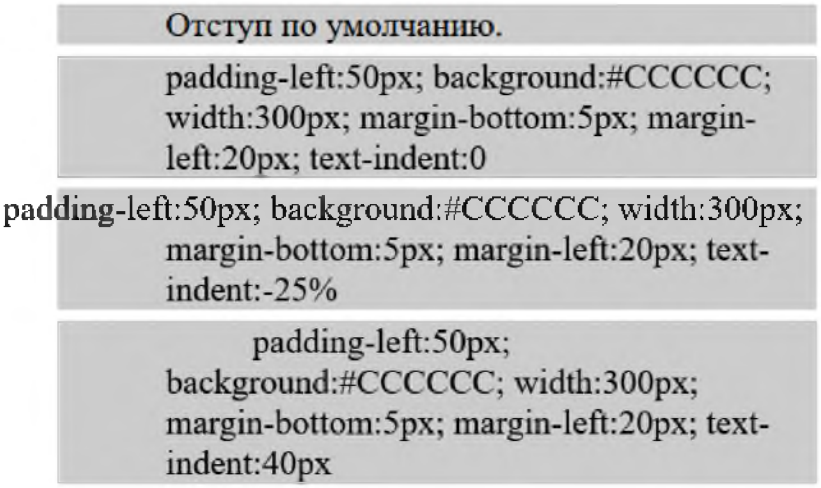
Свойство text-indent устанавливает величину отступа первой строки блока текста (красная строка). Значение отступа может задаваться в любых единицах длины, принятых в CSS. При задании значения в процентах отступ вычисляется в зависимости от ширины блока. Допускается указывать отрицательные значения. Если значение положительное, то формируется отступ. Если значение отрицательное, то формируется выступ, при этом

часть текста может выходить за рамки блока. Различные браузеры по-разному могут отображать этот кусок текста, выходящий за рамки блока.

Приведем пример использования свойства text-indent.

```
<!doctype html>
  <html>
    <head>
     <meta charset="utf-8">
     <style  >
       div {
         padding-left:50px; background:#CCCCCC;
         width:300px;
         margin-bottom:5px;
         margin-left:20px;}
       div#style1 { text-indent:0 }
       div#style2 { text-indent:-25% }
       div#style3 { text-indent:40px }
    </style>
  </head>
  <body>
    <div>Отступ по умолчанию.</div>
    <div                    id="style1">padding-left:50px;
background:#CCCCCC;    width:300px;    margin-bottom:5px;
margin-left:20px; text-indent:0</div>
    <div                    id="style2">padding-left:50px;
background:#CCCCCC;    width:300px;    margin-bottom:5px;
margin-left:20px; text-indent:-25%</div>
    <div                    id="style3">padding-left:50px;
background:#CCCCCC;    width:300px;    margin-bottom:5px;
margin-left:20px; text-indent:40px</div>
  </body>
</html>
```

Результат отображения в Mozilla Firefox:



text-transform

Свойство text-transform управляет регистром вывода символов текста. Допустимые значения свойства:

- none – не происходит никаких преобразований регистра символов текста в блоке, значение по умолчанию;
- capitalize – каждое слово в блоке текста выводится с заглавной буквы;
- uppercase – все символы текста преобразуются в прописные;
- lowercase – все символы текста преобразуются в строчные.

Приведем пример использования свойства text-transform.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {margin-bottom:4px}
      div.StyleCapitalize{text-transform:capitalize}
      div.StyleLowercase{text-transform:lowercase}
      div.StyleUppercase{text-transform:uppercase}
      span{text-transform:none}
    </style>
  </head>
  <body>
    <div>Отступ по умолчанию.</div>
ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ рЕгИсТрЕ. Вывод по
умолчанию.<br>
    <div                    class="StyleCapitalize">text-
transform:capitalize:ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ
рЕгИсТрЕ. <span>text-transform:none:ТЕКСТ набран БУКВАМИ
в РаЗлИчНоМ рЕгИсТрЕ</span></div>
    <div                     class="StyleLowercase">text-
transform:lowercase:ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ
рЕгИсТрЕ.</div>
    <div                     class="StyleUppercase">text-
transform:uppercase:ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ
рЕгИсТрЕ.</div>
  </body>
</html>
```

Отступ по умолчанию.
ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ рЕгИсТрЕ. Вывод по умолчанию.
Text-transform:capitalize:ТЕКСТ Набран БУКВАМИ В РаЗлИчНоМ РЕгИсТрЕ. text-transform:none:ТЕКСТ набран БУКВАМИ в РаЗлИчНоМ рЕгИсТрЕ
text-transform:lowercase:текст набран буквами в различном регистре.
TEXT-TRANSFORM:UPPERCASE:ТЕКСТ НАБРАН БУКВАМИ В РАЗЛИЧНОМ РЕГИСТРЕ.

В приведенном примере в первом блоке текста осуществлен вывод текста по умолчанию. Как текст был написан в HTML-документе, так он и выводится браузером. Во втором блоке было применено свойство `text-transform:capitalize`. Каждое слово, независимо от того, как оно было записано в HTML-документе, выводится с заглавной буквы. Внутри второго блока, в рамках тега , были отменены установки для вывода каждого слова текста с заглавной буквы, поэтому текст снова был выведен так же, как и набран. В третьем блоке было применено свойство `text-transform:lowercase`, что привело к тому, что весь текст был отображен браузером строчными буквами. Для четвертого блока текста было установлено `text-transform:uppercase`, в результате чего все содержимое блока было выведено браузером заглавными буквами.

vertical-align

Свойство vertical-align управляет вертикальным выравниванием содержимого блока относительно родительского блока или окружающего теста. Свойство может принимать следующие значения:

- baseline – осуществляется выравнивание базисной линии блока относительно базисной линии содержащего его родительского блока. Если у блока нет базисной линии, то выполняется выравнивание его нижней границы относительно базисной линии родительского блока;
- sub – базисная линия блока смещается вниз до уровня нижнего индекса родительского блока, в результате чего блок отображается в виде нижнего индекса, при этом размер шрифта текста не изменяется;
- super – базисная линия блока смещается вверх до уровня верхнего индекса родительского блока, в результате чего блок отображается в виде верхнего индекса, при этом размер шрифта текста не изменяется;
- top – происходит выравнивание верхнего края блока по верхней границе родительского элемента (строки);
- text-top – происходит выравнивание верхнего края блока по верху самого высокого элемента строки родительского блока;
- middle – осуществляется выравнивание средней по вертикали точки блока относительно суммы уровня базисной линии родительского блока и половины высоты родительского блока;
- bottom – происходит выравнивание нижнего края блока по нижней границе родительского элемента (строки);
- text-bottom – происходит выравнивание нижнего края блока по нижней кромке шрифта родительского блока;
- числовое значение в процентах – осуществляется смещение блока на заданную величину, вычисляемую как процент от значения свойства

line-height, вверх при положительном значении или вниз при отрицательном значении. Задание 0 % аналогично значению baseline;

- числовое значение – осуществляется смещение блока на заданную величину вверх при положительном значении или вниз при отрицательном значении. Значение 0 равносильно значению baseline.

white-space

Свойство white-space определяет тип обработки идущих подряд пробельных символов внутри текста. Допускаются следующие значения свойства:

- normal – пробельные символы обрабатываются стандартным образом, т. е. любая последовательность подряд стоящих пробелов, символов табуляции и пустых строк отображается браузером как один пробел, если речь идет не о содержимом тега <pre>;
- pre – последовательность подряд стоящих пробелов, символов табуляции и пустых строк отображается браузером без преобразований, так как она была задана в коде исходного HTML-документа;
- nowrap – пробельные символы обрабатываются стандартным образом, как при значении normal, но запрещен перенос слов.

Приведем пример использования свойства white-space.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      div {
           width:400px;
           background:#99CCFF;
           margin-bottom:4px;}
      div.classNormal{white-space:normal}
      div.classNowrap{white-space:nowrap}
      div.classPre{white-space:pre}
    </style>
  </head>
  <body>
    <div class="classNormal">Обработка пробелов
      обычным образом.       Несколько

     подряд
идущих пробелов заменяются
одним,  при  необходимости  происходит  автоматический
перенос текста на новую строку.
```

```
    </div>
    <div class="classNowrap">Вывод текста       без

переносов. Вывод текста       без переносов. Вывод текста
без переносов.
  </div>
  <div class="classPre">
Вывод
 текста
  в
   авторской
    редакции.</div>
  </body>
</html>
```

Обработка пробелов обычным образом. Несколько подряд идущих пробелов заменяются одним, при необходимости происходит автоматический перенос текста на новую строку.

Вывод текста без переносов. Вывод текста без переносов. Вывод текста без переносов.

```
Вывод
 текста
  в
   авторской
    редакции.
```

Из приведенного примера видно, что в первом блоке, к которому применено свойство white-space со значением normal, несколько подряд идущих пробельных символов трактуются браузером как один разделитель, а также, поскольку установлена фиксированная ширина блока, происходит автоматический перенос слов на новую строку по достижении текстом правой границы блока.

Во втором блоке свойство white-space установлено в nowrap, что приводит к выводу текста в одну строку.

В третьем блоке применено свойство white-space со значением pre, что эквивалентно применению тега <pre>, и текст выводится в авторском представлении.

word-spacing

Свойство word-spacing устанавливает величину пробельного символа, или расстояние между словами в тексте. Свойство может принимать следующие значения:

• normal – величина пробела стандартная и зависит от выбранного шрифта и настроек браузера;

• числовое значение – величина пробела задается напрямую как сумма стандартной величины и заданного значения, числовое значение может указываться в любых единицах, принятых в CSS, кроме процентов. Числовые значения могут быть и отрицательными, но их отображение зависит от браузера.

Если параметр, управляющий выравниванием текста text-align, установлен в значение justify, то величина расстояния между отдельными словами может быть больше значения, указанного в свойстве word-spacing, поскольку интервал между словами подбирается и устанавливается браузером принудительно таким образом, чтобы строка текста была выровнена по ширине родительского блока.

Приведем пример использования свойства word-spacing.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
     .WordSpacingNormal{word-spacing:normal}
     .WordSpacing10px{word-spacing:10px}
     .WordSpacing7em{word-spacing:7em}
     .WordSpacingNegativ3{word-spacing:-3px}
     .WordSpacingNegativ30{word-spacing:-30px}
     #AlignJustgify{text-align:justify}
    </style>
  </head>
  <body>
    <div>Расстояние между словами по умолчанию.</div>
    <div      class="WordSpacingNormal">WordSpacingNormal
WordSpac-ingNormal</div>
    <div class="WordSpacing7em">WordSpacing7em WordSpac-
ing7em</div>
    <div          class="WordSpacing10px">WordSpacing10px
WordSpac-ing10px</div>
    <div  class="WordSpacingNegativ3">WordSpacingNegativ3
WordSpac-ingNegativ3</div>
    <div
class="WordSpacingNegativ30">WordSpacingNegativ30   Word-
SpacingNegativ30</div>
    <div class="WordSpacing10px" id="AlignJustgify">text-
align:justify text-align:justify</div>
  </body>
```

```
</html>
```

Расстояние между словами по умолчанию.
WordSpacingNormal WordSpac-ingNormal
WordSpacing7em WordSpac-ing7em
WordSpacing10px WordSpac-ing10px
WordSpacingNegativ3 WordSpac-ingNegativ3
WordSpacingNegativ30 Word-SpacingNegativ30
text-align:justify text-align:justify text-align:justify text-align:justify text-align:justify text-align:justify text-align:justify

В примере для второй строки текста установлено значение свойства `word-spacing:normal`. Это эквивалентно приданию пробелу стандартной величины, такой же, как и при выводе текста без заданных значений свойства word-spacing в первой строке.

В третьей и четвертой строках устанавливается пробел в `7em` и `10px` соответственно. В пятой и шестой строках применяются отрицательные значения свойства word-spacing. При небольшом значении в `-3px` это приводит к более сжатому выводу текста в пятой строке. Отрицательное значение свойства `-30px` приводит к потере читабельности текста и наползанию слов друг на друга в шестой строке.

Содержимое последнего блока, занимающего две последние строки текста, выравнивается по ширине страницы, поэтому величина пробела между словами различна в седьмой и восьмой строках.

Упражнение

Это своего рода практикум по использованию селекторов. Попробуйте, не заглядывая в справочник по CSS, проделать все описанные ниже шаги по преобразованию дизайна страницы. В качестве исходного текста для страницы используем заготовку.

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Селекторы</title>
  </head>
  <body>
    <article>
      <div id="logo">CSS в действии</div>
      <h1>Потрясающий мир CSS</h1>
      <p>Sed ut perspiciatis unde omnis iste natus error
sit voluptatem accusantium doloremque laudantium, totam
rem aperiam, eaque ipsa quae ab illo inventore veritatis
```

et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

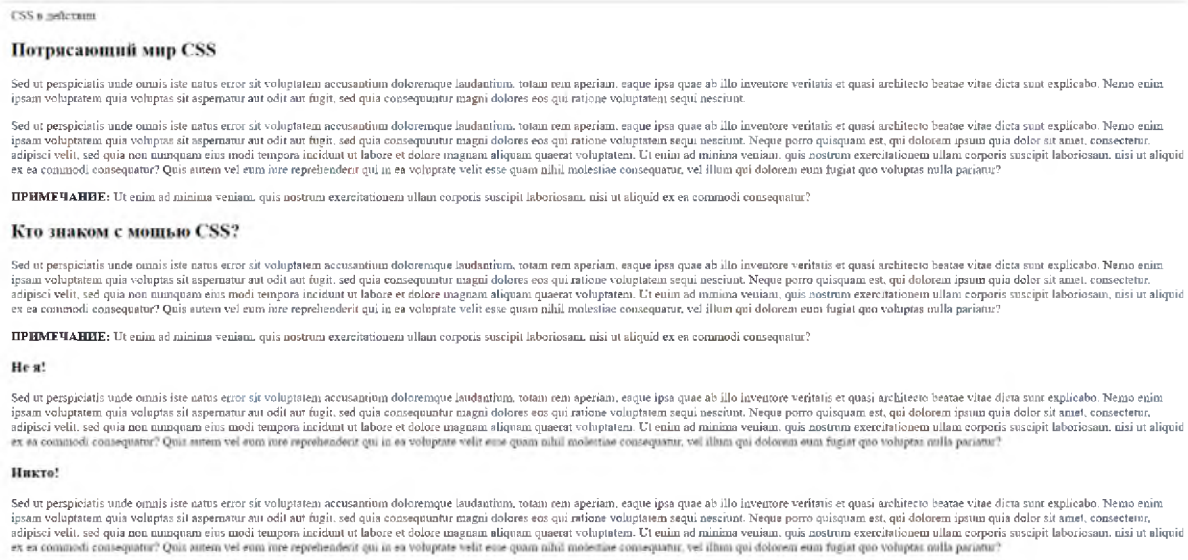
ПРИМЕЧАНИЕ: Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?

<h2>Кто знаком с мощью CSS?</h2>

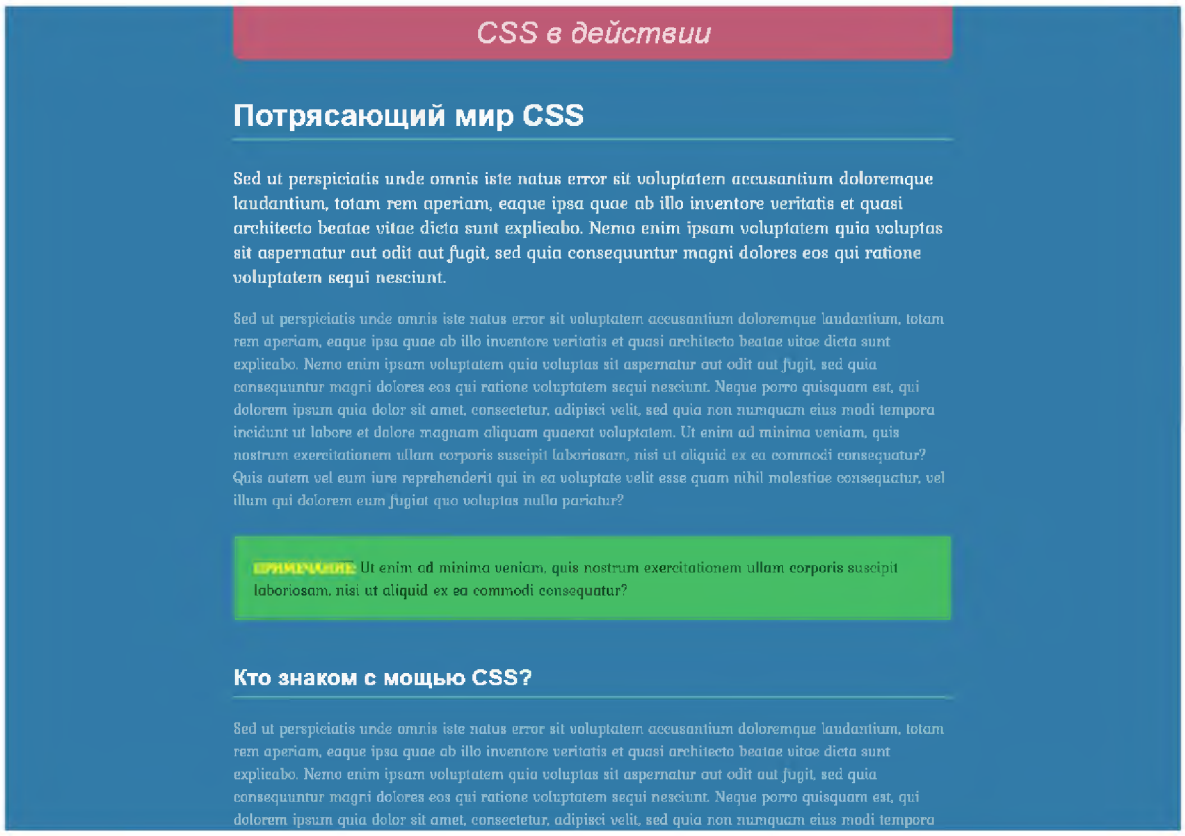
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

```
      <p   class="note"><strong>ПРИМЕЧАНИЕ:</strong>  Ut
enim ad minima veniam, quis nostrum exercitationem ullam
corporis suscipit laboriosam, nisi ut aliquid ex ea
commodi consequatur?
      </p>
      <h3>Не я!</h3>
      <p>Sed ut perspiciatis unde omnis iste natus error
sit voluptatem accusantium doloremque laudantium, totam
rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.
Nemo enim ipsam voluptatem quia voluptas sit aspernatur
aut odit aut fugit, sed quia consequuntur magni dolores
eos qui ratione voluptatem sequi nesciunt. Neque porro
quisquam est, qui dolorem ipsum quia dolor sit amet,
consectetur, adipisci velit, sed quia non numquam eius
modi tempora incidunt ut labore et dolore magnam aliquam
quaerat voluptatem. Ut enim ad minima veniam, quis nostrum
exercitationem ullam corporis suscipit laboriosam, nisi
ut aliquid ex ea commodi consequatur? Quis autem vel eum
iure reprehenderit qui in ea voluptate velit esse quam
nihil molestiae consequatur, vel illum qui dolorem eum
fugiat quo voluptas nulla pariatur?
      </p>
      <h3>Никто!</h3>
      <p>Sed ut perspiciatis unde omnis iste natus error
sit voluptatem accusantium doloremque laudantium, totam
rem aperiam, eaque ipsa quae ab illo inventore veritatis
et quasi architecto beatae vitae dicta sunt explicabo.
Nemo enim ipsam voluptatem quia voluptas sit aspernatur
aut odit aut fugit, sed quia consequuntur magni dolores
eos qui ratione voluptatem sequi nesciunt. Neque porro
quisquam est, qui dolorem ipsum quia dolor sit amet,
consectetur, adipisci velit, sed quia non numquam eius
modi tempora incidunt ut labore et dolore magnam aliquam
quaerat voluptatem. Ut enim ad minima veniam, quis nostrum
exercitationem ullam corporis suscipit laboriosam, nisi
ut aliquid ex ea commodi consequatur? Quis autem vel eum
iure reprehenderit qui in ea voluptate velit esse quam
nihil molestiae consequatur, vel illum qui dolorem eum
fugiat quo voluptas nulla pariatur?
      </p>
    </article>
  </body>
</html>
```

Сама по себе страница выглядит невзрачно и непритязательно.



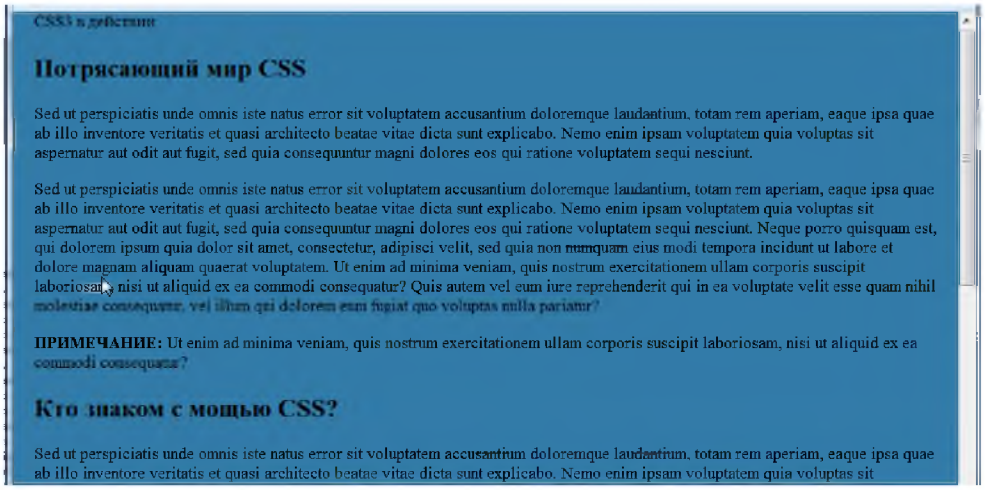
Если вы все проделаете верно, то страница преобразуется в более приятный и удобный вид.



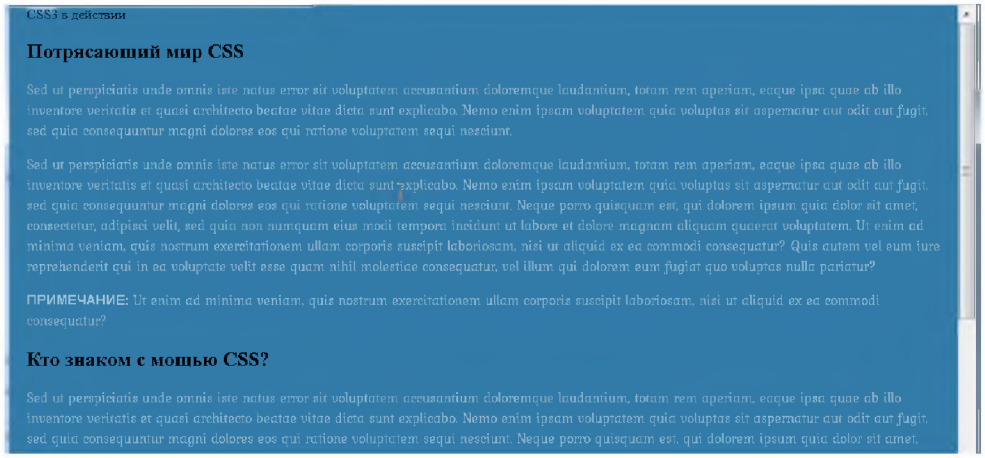
Сохраните заготовку из пункта 1 как файл selectors.html. Откройте файл selectors.html в редакторе HTML-кода, например в Adobe Dreamweaver.

Во всем документе используем шрифт Kurale из онлайн-шрифтов Google (https://fonts.google.com/). Для этого зайдите на сайт, найдите шрифт Kurale для кириллицы, сгенерируйте необходимый для вставки код и вставьте его в страничку.

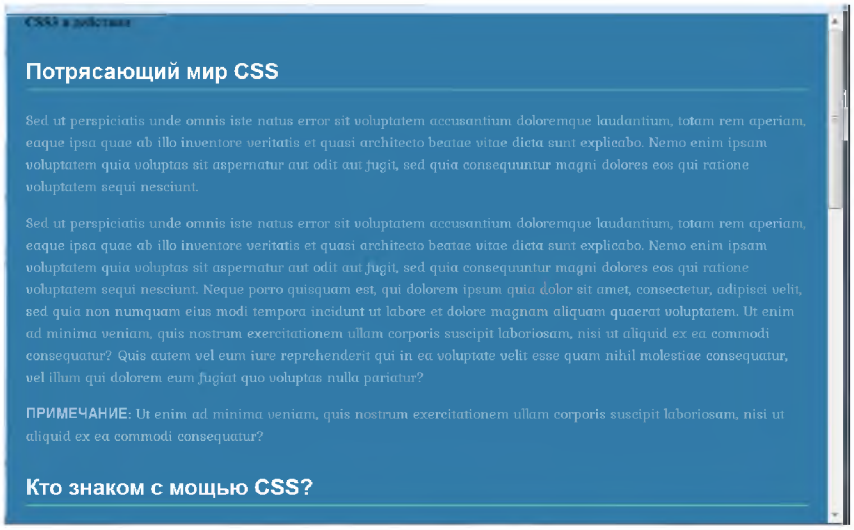
Для тега body добавьте три правила форматирования: цвет, отступ и поле. Цвет фона сделайте темно-синим rgb(50,122,167). Установите внутренние поля для элемента body: верхнее – 0px, остальные – по 20px. Внешние отступы убрать.



В данном случае цвет фона темно-синий. Он затрудняет чтение, поэтому необходимо изменить цвет текста элементов абзаца. Добавьте стиль для всех элементов p: определите цвет, размер и шрифт текста. Для указания цвета используйте значение rgba(), для абзаца выберите белый цвет с непрозрачностью 60 % (значение .6), тогда сквозь текст будет проступать синий цвет, поэтому шрифт будет казаться светло-синим. Размер шрифта укажите в относительных единицах em как равный базовому. В качестве шрифта укажите Kurale, при его отсутствии – Arial, если нет и его – Helvetica, если отсутствуют все три шрифта – гарнитуру sans-serif.



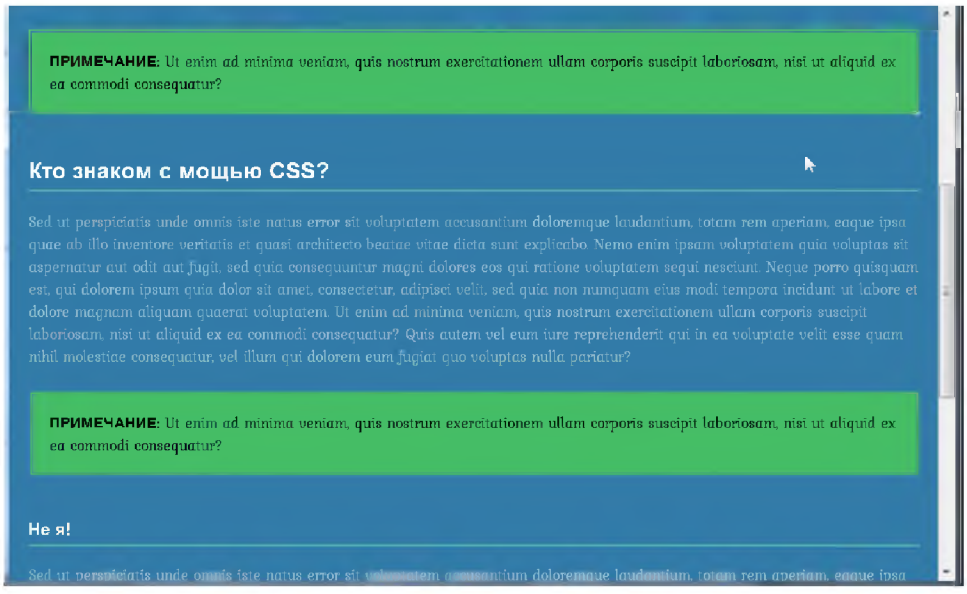
Сделаем так, чтобы все заголовки – а у нас используются заголовки трех уровней – отображались шрифтом одного вида и цвета. Для этого создайте групповой селектор для h1, h2 и h3. Установите белый цвет шрифта. По нижнему краю заголовков установите непрерывную границу толщиной 2px цвета rgb(87,185,178). Верхнее поле блоков заголовков – 10px, нижнее поле – 5px. Заголовок h1 в начале веб-страницы и заголовки h2 и h3 ниже на странице имеют одинаковое начертание и цвет шрифта, а также зеленую рамку внизу.



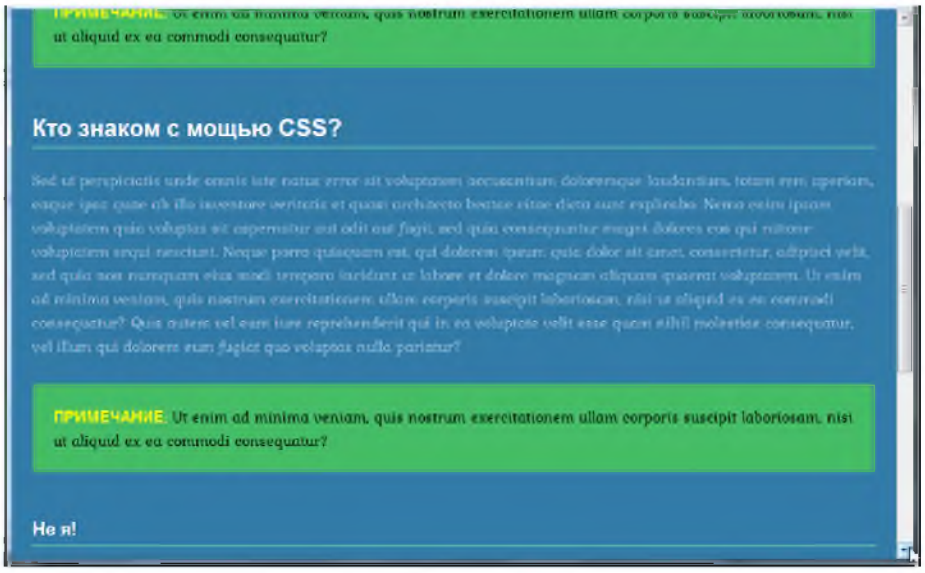
Элемент h1 выглядит немного мелковатым, но мы легко можем увеличить его. Добавим еще один стиль под только что созданным групповым селектором для h1. В нем установим двойной стандартный размер шрифта. Обратите внимание, что сначала применяются сразу несколько стилей к одному элементу: групповой селектор для h1, h2, h3 и h1 и новый селектор тега применяются к элементу h1. Такой процесс в языке CSS называется каскадом.

Создайте стиль, который будет определять вид текста «CSS3 в действии» в самом верху страницы. Этот текст играет роль логотипа. Воспользуйтесь идентификатором для его форматирования. Добавьте после последнего созданного класса h1 новый стиль для идентификатора logo. В нем в качестве шрифта укажите Baskerville, при его отсутствии – Palatino, а если и его нет, то гарнитуру sans-serif. Установите двойной стандартный размер шрифта. Цвет белый с прозрачностью .8. Начертание шрифта курсивное. Выравнивание по центру. Нижний внешний отступ – 30px. Цвет фона в RGB-палитре – 191, 91, 116. Закругление углов блока для верхних углов не делайте, для нижних углов радиус закругления – 10px. Внутренние поля – по 10px со всех четырех сторон. Далее в теге с текстом «CSS3 в действии» (он расположен после тега <article>) добавьте код id="logo".

Используя атрибут class со значением note, измените стиль третьего абзаца. Установите черный цвет текста, непрерывную белую границу в 2px, цвет фона – rgb(69,189,102), внутренние поля – по 20px со всех сторон, внешний верхний отступ – 25px, а нижний внешний отступ – 35px. Этот же класс примените к абзацу, расположенному перед элементом h3, с текстом «Не я!»



На странице вы применили класс note к двум абзацам. Каждый из них начинается словом «Примечание:», выделенным полужирным начертанием. Отформатируйте эти слова еще и желтым цветом. Лучший способ решить эту проблему – создать селектор потомков, который относится только к нужным нам элементам strong. Создайте стиль для .note strong {}. В этом случае стиль отформатирует элемент strong, только если он расположен внутри другого элемента, к которому применен класс .note. Цвет шрифта задайте желтым.



Текст на полученной странице расширяется, чтобы заполнить окно браузера при изменении его размера. Обратите внимание, что по мере растягивания окна строки текста становятся шире. Если монитор достаточно велик, то при достижении определенной ширины строки текста становятся слишком длинными, чтобы читать их с комфортом. Ограничим ширину контента страницы, чтобы она не превышала определенную величину. Для этого создайте стиль для тега <article> и укажите в нем, что максимальная ширина элемента статьи не будет превышать 760 пикселей. Сохраните файл и просмотрите его в браузере. Если растянуть окно браузера шире 760 пикселей, то можно увидеть, что по бокам от текста появился синий фон, а сам текст больше не расширяется. С другой стороны, если вы уменьшите окно браузера, сделав его ширину меньше 760 пикселей, то строки текста сожмутся. Это важный элемент адаптивного дизайна. Кроме этого, выровняйте текст по центру страницы вместо привязки к левому краю, через поля margin: 0 auto. В данном примере значения левого и правого полей определяются автоматически, то есть браузер сам вычисляет величину левого и правого полей элемента article. Когда ширина окна браузера превысит 760 пикселей, элемент article прекратит расширяться, поэтому браузер автоматически добавит пустое пространство слева и справа от элемента, по сути выравнивая его по центру относительно своего окна.

Чтобы акцентировать внимание пользователей на первом абзаце после заголовка первого уровня (как правило, там находится самое важное), создайте смежный родственный селектор. Этого же эффекта можно достигнуть, создав класс и применив его к этому абзацу, но смежный родственный селектор не требует изменения HTML-кода. Добавьте последний стиль для h1+p. Он будет применен к любому абзацу, следующему сразу за элементом h1, т. е. к первому абзацу после верхнего заголовка страницы. Он не будет применен ко второму или последующим абзацам. С помощью этого селектора можно легко изменить внешний вид вводного абзаца, чтобы выделить его визуально и обозначить начало статьи. Измените цвет шрифта на белый без прозрачности. Размер шрифта в 1,2 раза больше базового. Пространство между строками в абзацах (параметр, также известный как интерлиньяж или высота строки) установите в 140 %.

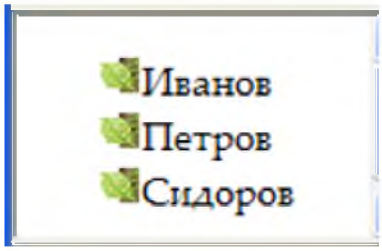
Свойства списков

list-style-image

Свойство list-style-image устанавливает произвольное изображение в качестве маркера списка. Атрибут наследуется, поэтому для отдельных элементов списка для восстановления маркера используется значение none. Значением свойства является адрес файла с изображением маркера. Аргумент none отменяет изображение в качестве маркера для родительского элемента.

Приведем пример использования свойства list-style-image.

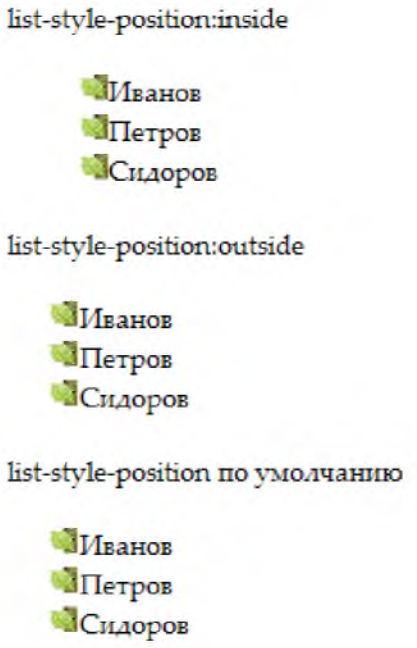
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <ul style="list-style-image:url(image/bullet.GIF)">
    <li>Иванов</li>
      <li>Петров</li>
      <li>Сидоров</li>
    </ul>
</body>
</html>
```



list-style-position

Свойство list-style-position устанавливает, где должен находиться маркер по отношению к элементу списка. Возможные значения свойства:

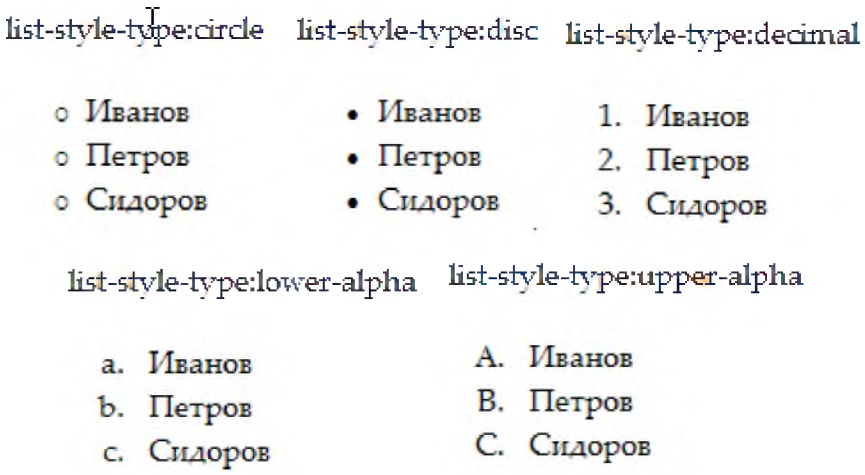
- inside – внутри блока элемента списка;
- outside – снаружи, перед блоком элемента списка.



list-style-type

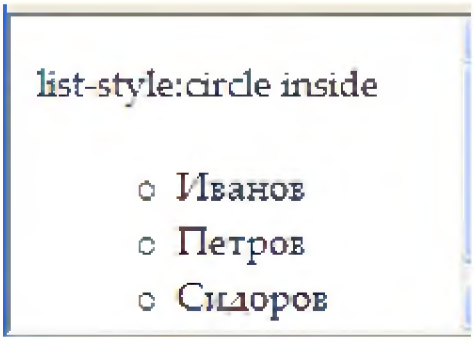
Свойство list-style-type отвечает за установку типа маркера элементов списка. Свойство может принимать следующие значения:

- none – маркер списка отсутствует;
- disc – закрашенная окружность в качестве маркера списка;
- circle – незакрашенная окружность в качестве маркера списка;
- square – квадрат в качестве маркера списка;
- decimal – список промаркирован арабскими цифрами;
- lower-roman – список промаркирован маленькими римскими цифрами;
- upper-roman – список промаркирован большими римскими цифрами;
- lower-alpha – список промаркирован малыми латинскими буквами;
- upper-alpha – список промаркирован большими латинскими буквами.



list-style

Свойство list-style является сокращенным вариантом одновременного использования свойств list-style-image, list-style-position, list-style-type.



Свойства таблиц

caption-side

Свойство caption-side используется для установки позиции вывода заголовка таблицы, который определен в теге <caption>. Допустимы следующие значения свойства:

- top – поле заголовка располагается над таблицей;
- bottom – поле заголовка располагается под таблицей;
- initial – значение по умолчанию;
- inherit – поле заголовка располагается как у родительского элемента.

border-collapse

Свойство border-collapse определяет модель вывода границ таблицы. Это свойство применяется, если установлен вывод рамки для ячеек таблицы и для самой таблицы. Свойство может принимать следующие значения:

- separate – для каждой ячейки отображается своя граница, поэтому в местах стыка двух ячеек или ячейки и границы таблицы выводится линия двойной толщины. Значение по умолчанию;
- collapse – для каждой ячейки отображается своя граница, но в местах стыка двух ячеек или ячейки и границы таблицы выводится одна линия.
- initial;
- inherit.

Приведем пример использования свойства border-collapse.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      table,th { border:2px solid blue }
      td{ border:2px solid red }
      table { margin-bottom:10px }
      table.ClassCollapse { border-collapse:collapse }
    </style>
  </head>
  <body>
    <table>
      <tr>
        <th>ФИО</th>
        <th>Средний балл</th>
```

```
      </tr>
      <tr>
        <td>Иванов И.И.</td>
        <td>7.5</td>
      </tr>
      <tr>
        <td>Петров П.П.</td>
        <td>8.4</td>
      </tr>
    </table>
    <table class="ClassCollapse">
      <tr>
        <th>ФИО</th>
        <th>Средний балл</th>
      </tr>
      <tr>
        <td>Иванов И.И.</td>
        <td>7.5</td>
      </tr>
      <tr>
        <td>Петров П.П.</td>
        <td>8.4</td>
      </tr>
    </table>
  </body>
</html>
```

ФИО	Средний балл
Иванов И.И.	7.5
Петров П.П.	8.4

ФИО	Средний балл
Иванов И.И.	7.5
Петров П.П.	8.4

Для верхней таблицы свойство border-collapse явно не устанавливается. По умолчанию `border-collapse:separate`, что приводит к двойному выводу границ.

В нижней таблице определено `border-collapse:collapse`, что заставляет браузер на стыке двух границ выводить одну линию. При этом

при различном значении параметра границы у разных блоков значение этого же параметра для общей границы выбирается равным значению параметра родительского элемента. Например, для границы ячеек таблицы установлен красный цвет, а для заголовочных ячеек и границы самой таблицы – синий. На стыке границ ячеек и таблицы за цвет границы принимается синий – цвет границы всей таблицы.

empty-cells

Свойство empty-cells управляет выводом пустых ячеек таблицы. По умолчанию пустая ячейка таблицы не отображается, для ее вывода в нее необходимо поместить неотображаемый символ неразрывного пробела . Используя свойство empty-cells, можно выводить (show) или не выводить (hide) пустые ячейки, не содержащие символа неразрывного пробела. Если свойство empty-cells установлено в hide, то пустые ячейки не отображаются.

ФИО	Средний балл
Иванов И.И.	7.5
Петров П.П.	8.4
Итого	

table-layout

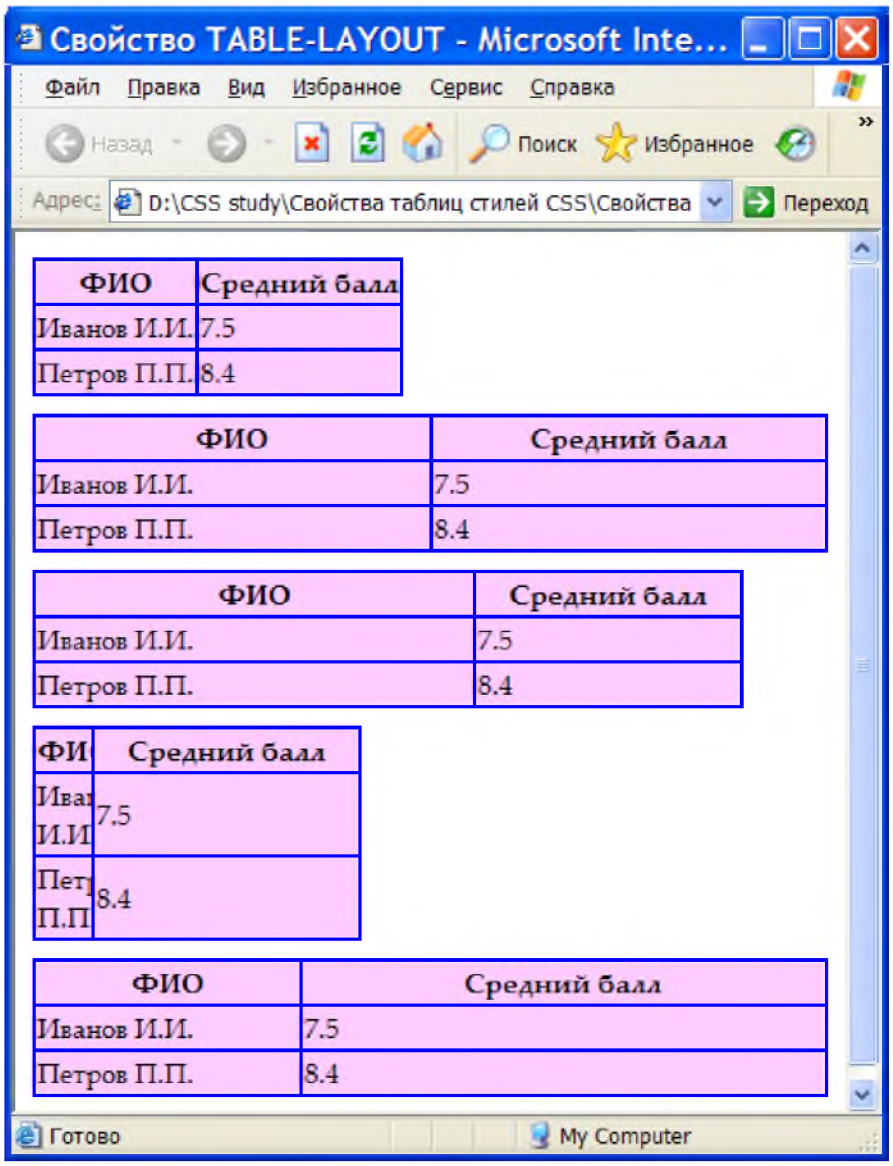
Свойство table-layout определяет алгоритм, по которому браузер будет определять размеры ячеек таблицы при ее выводе. Возможные значения свойства:

- auto – браузер загружает всю таблицу, анализирует ее, определяет размер ячеек (высота и ширина ячеек определяются в зависимости от их содержимого) и только после этого отображает всю таблицу;
- fixed – браузер начинает вывод таблицы сразу, при этом ширина колонок определяется либо с помощью тега <col>, либо вычисляется на основе первой строки. Если данные о форматировании первой строки таблицы не заданы явно, то в этом случае таблица по ширине занимает все свободное место родительского элемента и делится на колонки равной ширины. Если при фиксированной ширине колонок их содержимое не помещается в ячейку заданной ширины, то оно либо частично не будет видно, либо наложится поверх соседней ячейки в зависимости от используемого браузера.

Пример использования свойства table-layout:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      table,td,th { border:2px solid blue }
      table{
            margin-bottom:10px;
            border-collapse:collapse;
            background-color:#FFCCFF; }
      .StLayoutFixed { table-layout:fixed }
      .FixedWidth30 {width:30px }
      .FixedWidth15 {width:150px }
      .FixedWidth25 {width:250px }
   </style>
  </head>
  <body>
  <!--Размеры ячеек устанавливаются исходя из их
содержимого по умолчанию. -->
  <table>
    <tr><th>ФИО</th><th>Средний балл</th></tr>
    <tr><td>Иванов И.И.</td><td>7.5</td></tr>
    <tr><td>Петров П.П.</td><td>8.4</td></tr>
  </table>
  <!--Ширина ячеек устанавливается одинаковой. -->
  <table class="StLayoutFixed">
    <tr><th>ФИО</th><th>Средний балл</th></tr>
    <tr><td>Иванов И.И.</td><td>7.5</td></tr>
    <tr><td>Петров П.П.</td><td>8.4</td></tr>
  </table>
  <!--Размеры ячеек устанавливаются по размеру ячеек
заголовка, заданному явно. -->
  <table class="StLayoutFixed">
    <tr>
      <th  class="FixedWidth25">ФИО</th>
      <th  class="FixedWidth15">Средний балл</th>
    </tr>
    <tr><td>Иванов И.И.</td><td>7.5</td></tr>
    <tr><td>Петров П.П.</td><td>8.4</td></tr>
  </table>
  <!--Размеры ячеек устанавливаются по размеру ячеек
заголовка, заданному явно, даже если содержимое ячеек
отображается не полностью. -->
  <table class="StLayoutFixed">
    <tr>
```

```
      <th  class="FixedWidth30">ФИО</th>
      <th  class="FixedWidth15">Средний балл</th> </tr>
      <tr><td>Иванов И.И.</td><td>7.5</td></tr>
      <tr><td>Петров П.П.</td><td>8.4</td></tr>
  </table>
  <!--Размеры ячеек первого столбца устанавливаются по
размеру заданному явно в теге COL. -->
  <table class="StLayoutFixed">
    <col width="150">
    <tr><th>ФИО</th><th>Средний балл</th> </tr>
    <tr><td>Иванов И.И.</td><td>7.5</td></tr>
    <tr><td>Петров П.П.</td><td>8.4</td></tr>
  </table>
  </body>
</html>
```



В первой из выводимых таблиц размеры ячеек устанавливаются исходя из их содержимого по умолчанию.

Во второй таблице свойство table-layout установлено в fixed, но ширина не задана явно, поэтому таблица заняла по ширине все пространство окна, а ширина столбцов взята одинаковой.

В третьей таблице свойство table-layout установлено в fixed, размеры ячеек устанавливаются по размеру ячеек заголовка, заданному явно.

В четвертой таблице свойство table-layout установлено в fixed и размеры ячеек также устанавливаются по размеру ячеек заголовка, заданному явно, даже если содержимое ячеек первого столбца видно только частично. Браузер обрезал их содержимое при выводе.

В пятой таблице свойство table-layout установлено в fixed, при этом ширина первого столбца устанавливается по размеру, заданному явно в теге <col>, а под второй столбец отводится все пространство до правого края родительского элемента (окна браузера).

cursor

Устанавливает вид курсора мыши, когда он находится в пределах элемента разметки. Вид курсора по умолчанию зависит от операционной системы и установленных параметров. Свойство cursor может принимать следующие значения:

- default – курсор, используемый для данной операционной системы по умолчанию, часто представляется в виде большой стрелки;
- crosshair – курсор-прицел в виде крестика;
- help – курсор, означающий, что для объекта, на который он указывает, имеется справочная информация, часто представляется в виде вопросительного знака или воздушного шара;
- move – курсор, определяющий объект, который можно переместить, пересечение горизонтальной и вертикальной двунаправленных стрелок;
- pointer – курсор представляется указателем, обозначающим ссылку, обычно в виде указывающей руки;
- progress – курсор, указывающий на выполнение программы, обычно в виде стрелки с песочными часами;
- text – курсор, используемый при выделении текста, часто представляется в виде вертикальной линии, ограниченной короткими горизонтальными линиями сверху и снизу;
- wait – курсор, указывающий на занятость программы, часто представляется в виде песочных часов или циферблата;

• e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize – курсоры, определяющие перемещение некоторого края, обычно в виде двунаправленной стрелки соответствующей ориентации;

• <uri> – браузер загружает курсор из ресурса, задаваемого этим URI. Значение состоит из трех частей: <uri> базового курсора, <uri> резервного курсора, стандартного курсора браузера. Если браузеру не удается загрузить курсор, расположенный первым в списке курсоров, он должен попытаться загрузить резервный. Если браузеру не удается обработать ни одного курсора, заданного пользователем, он должен использовать общий курсор, расположенный в конце этого списка.

Пример демонстрирует использование различных видов курсора, в том числе и расширенного набора курсоров для Internet Explorer.

quotes

Свойство quotes устанавливает тип открывающей и закрывающей кавычек, который применяется в тексте документа для выделения содержимого тега ***<q>***. Тип кавычек может быть задан символами, символами Unicode или специальными символами HTML.

Тип	Спецкод HTML	Символ Unicode	Описание
"	&#34;	\0022	Двойная кавычка, применяется обычно для обозначения символа дюйма
'	&#39;	\0027	Апостроф
«	« или «	\00AB	Открывающая двойная угловая кавычка
»	» или »	\00BB	Закрывающая двойная угловая кавычка
‘	&#8216;	\2018	Открывающая одинарная кавычка
’	&#8217;	\2019	Закрывающая одинарная кавычка
“	&#8220;	\201C	Открывающая двойная кавычка в англоязычных текстах или закрывающая для русского языка
”	&#8221;	\201D	Закрывающая двойная кавычка в англоязычных текстах
„	&#8222;	\201E	Открывающая двойная кавычка в русскоязычных текстах

```
<!doctype html>
<html>
  <head>
```

```
    <meta charset="utf-8">
    <style>
      .s1{ quotes:"\00ab" "\00bb" }
      .s2{ quotes:"\2018" "\2019" }
      .s3{ quotes:"\201c" "\201d" }
    </style>
  </head>
  <body>
Просто текст. <q class="s1">Это цитата. <q class="s2">А
это цитата в цитате.</q> Продолжение цитаты.</q> Снова
просто текст.<br>
<p class="s3">А в этом абзаце <q style="quotes: '<'
'>'">свое</q> выделение цитат. <q>Это цитата.</q></p>
  </body>
</html>
```

В приведенном примере определено три стиля для выделения цитат во внедренной таблице стилей и один стиль во встроенной таблице стилей.

z-index

Свойство z-index определяет порядок отображения элементов разметки. Свойство может принимать значение auto, в этом случае порядок перекрытия элементов задается по умолчанию: элементы, определенные в HTML-коде раньше, перекрываются заданными позже. Кроме константы auto, значением свойства может быть любое целое число, которое и является индексом в последовательности отображения элементов. Элементы с бóльшим значением индекса будут перекрывать элементы с меньшим значением. Спецификация разрешает использовать отрицательные значения z-index.

zoom

Свойство zoom управляет масштабом вывода элементов разметки. Значением свойства может быть число с плавающей точкой, задающее множитель увеличения или уменьшения, также свойство может быть задано в процентах или константой normal. Значение normal является значением по умолчанию и задает масштаб 1,0, или 100 %.

Псевдоклассы гипертекстовых ссылок

Псевдоклассы гипертекстовых ссылок управляют отображением гипертекстовых ссылок и применяются только к тегу <a>. Обычно браузеры

различным образом отображают просмотренные и непросмотренные ссылки, ссылки, находящиеся в фокусе, и ссылки в момент их активации.

Псевдокласс :link применяется для описания стиля отображения непосещенных ссылок.

Псевдокласс :visited применяется для описания стиля отображения ссылок, которые уже посещались.

Псевдокласс :hover применяется для описания стиля отображения ссылок в момент, когда они находятся в фокусе, но не активизированы. Например, браузер может применять этот псевдокласс, когда указатель мыши находится над ссылкой, но щелчка мыши не произошло.

Псевдокласс :active применяется в момент активизации ссылки, а именно между моментами, когда пользователь нажимает кнопку мыши и отпускает ее.

Браузеры не обязаны проводить новое форматирование HTML-документа во время выбора ссылок и переходов по ссылкам. Например, в таблице стилей может быть указано, что размер шрифта у посещенной ссылки больше, чем у непосещенной. Поскольку это может привести к изменению положения текста при возвращении в данную точку документа после посещения ссылки, браузер может проигнорировать соответствующее стилевое правило. Псевдоклассы гипертекстовых ссылок по-разному поддерживаются различными браузерами. При описании псевдоклассов гипертекстовых ссылок важен порядок их описания. Правило для :hover должно располагаться после правил :link и :visited, так как в противном случае правила каскадирования скроют свойства правила :hover. Аналогично :active должно находиться после :hover, так как когда пользователь устанавливает указатель поверх ссылки, то одновременно активизирует ее. Поскольку описанные псевдоклассы управляют только отображением гипертекстовых ссылок и применяются только к тегу <a>, то в стилевой таблице можно явно не указывать название тега. Следующие два правила эквивалентны.

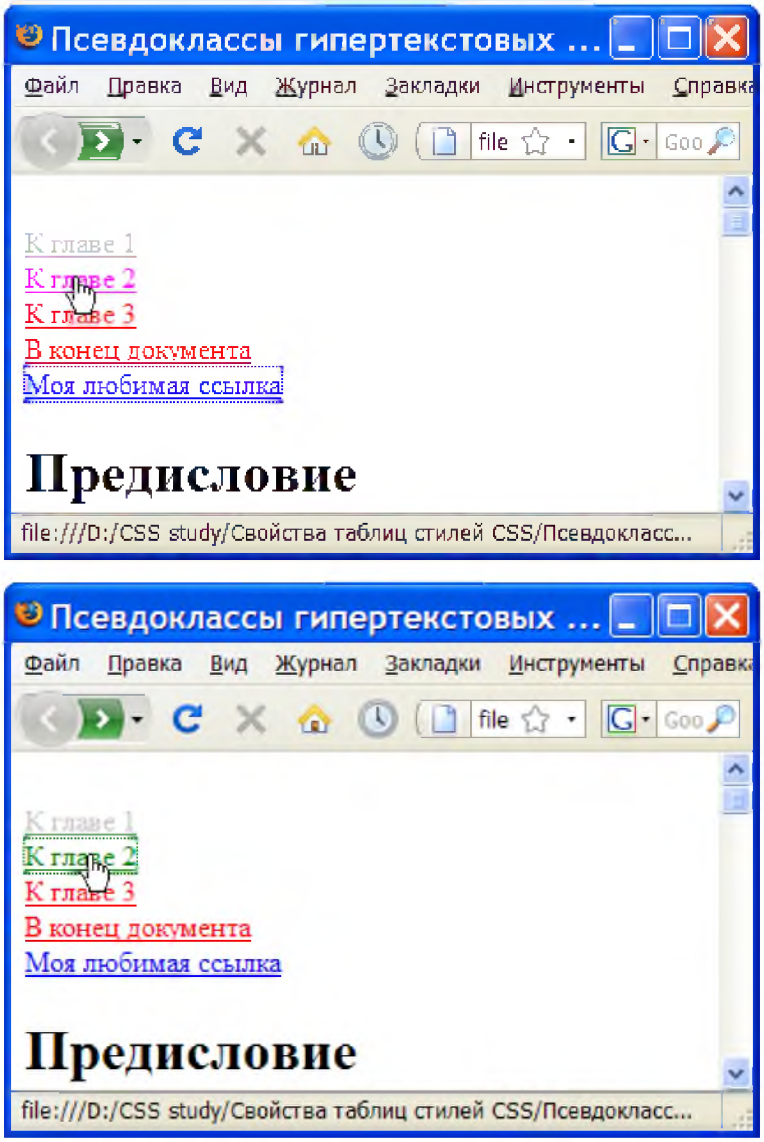
```
:visited {color:silver}
a:visited {color:silver}
```

Приведем пример использования :link, :visited, :hover, :active.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Псевдоклассы гипертекстовых ссылок</title>
    <style>
      a:link {color:red}
      a:visited {color:silver}
      a:hover{color:magenta}
```

```
      a:active{color:green}
      a.MyFavorite:visited {color:blue}
    </style>
  </head>
  <body>
    <a name="Top"></a><br>
    <a href="#Chapter1">К главе 1</a><br>
    <a href="#Chapter2">К главе 2</a><br>
    <a href="#Chapter3">К главе 3</a><br>
    <a href="#Bottom"> В конец документа </a> <br>
    <a class="MyFavorite" href="Untitled-6.html">Моя
любимая ссылка</a><br>
    <h1>Предисловие</h1>
    На предисловие ссылаться не будем.
    <a name="Chapter1"><h1>Глава 1</h1></a><br>
Содержание главы 1.<br>Содержание главы 1.<br>Содержание
главы 1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>
    <a href="glava.html">К дополнительным материалам
главы 1</a><br>
Содержание главы 1.<br>Содержание главы 1.<br>Содержание
главы 1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>Содержание главы
1.<br>Содержание главы 1.<br>
    <a name="Chapter2"><h1>Глава 2</h1></a><br>
Содержание главы 2.<br>Содержание главы 2.<br>Содержание
главы 2.<br>Содержание главы 2.<br> Содержание главы
2.<br>Содержание главы 2.<br>Содержание главы
2.<br>Содержание главы 2.<br> Содержание главы
2.<br>Содержание главы 2.<br>Содержание главы
2.<br>Содержание главы 2.<br> Содержание главы
2.<br>Содержание главы 2.<br>Содержание главы
2.<br>Содержание главы 2.<br> Содержание главы
2.<br>Содержание главы 2.<br>Содержание главы
2.<br>Содержание главы 2.<br>
    <a name="Chapter3"><h1>Глава 3</h1></a><br>
Содержание главы 3.<br>Содержание главы 3.<br>Содержание
главы 3.<br>Содержание главы 3.<br> Содержание главы
3.<br>Содержание главы 3.<br>Содержание главы
3.<br>Содержание главы 3.<br>Содержание главы
3.<br>Содержание главы 3.<br>Содержание главы
```

```
3.<br>Содержание главы 3.<br>Содержание главы
3.<br>Содержание главы 3.<br>Содержание главы
3.<br>Содержание главы 3.<br> Содержание главы
3.<br>Содержание главы 3.<br>Содержание главы
3.<br>Содержание главы 3.<br><br>
    <a  href="#Top"> К началу документа </a>
    <a name="Bottom"></a>
  </body>
</html>
```



В приведенном примере задается красный цвет для непосещенных ссылок, серый – для посещенных, бледно-фиолетовый – для ссылок, находящихся в фокусе, и зеленый – для ссылок в момент их активации. Кроме этого, для определенной, избранной ссылки ее цвет после посещения станет синим.

Псевдоэлементы :first-letter и :first-line

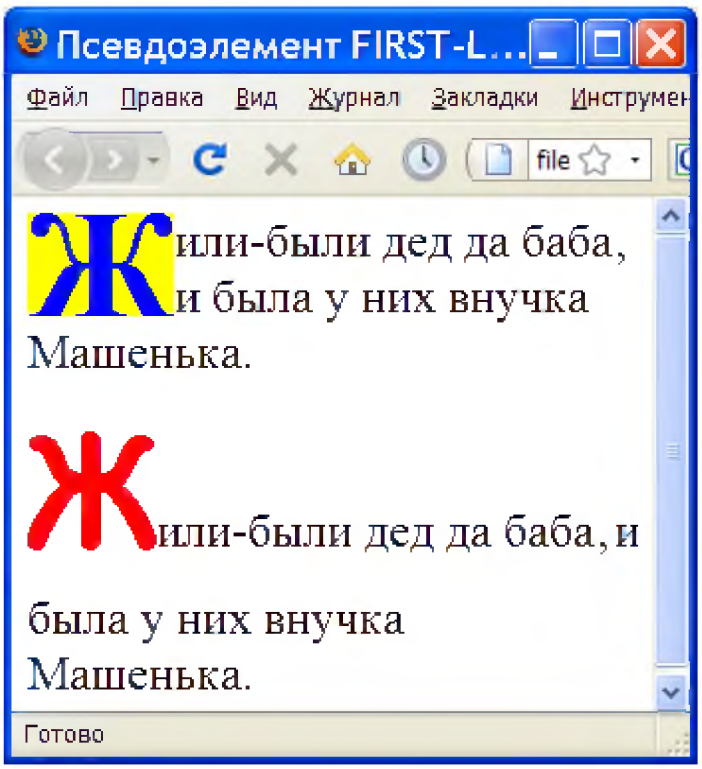
:first-letter

Псевдоэлемент :first-letter используется для форматирования первого символа элемента разметки для создания типографских эффектов заглавной буквы и буквицы.

Эффект заглавной буквы достигается, если свойство float установлено в none (значение по умолчанию). В этом случае первая буква ведет себя как элемент строки, т. е. находится на одной линии с другими символами строки. Если значение свойства float изменено, то первая буква будет вести себя как перемещаемый объект и можно добиться эффекта буквицы.

В разных браузерах эффект применения псевдоэлемента :first-letter может быть различным.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Псевдоэлемент FIRST-LETTER</title>
    <style>
      div{font-size:24px;}
      div.fl1:first-letter {
           font-family:"Times New Roman", Times, serif;
           font-size:76px;
           font-weight:bold;
           background:yellow;
           color:blue;
           float:left
           }
      div.fl2:first-letter {
           font-family:cursive; font-size:76px;
           font-weight:bold; color:red;
           }
      </style>
  </head>
  <body>
<!--Смотри в Mozilla Firefox! -->
    <div class="fl1">Жили-были дед да баба, и была у них
внучка Машенька.</div>
    <div class="fl2">Жили-были дед да баба, и была у них
внучка Машенька.</div>
  </body>
</html>
```



В приведенном примере для первого блока применен псевдоэлемент :first-letter для создания буквицы. Для этого свойству float было придано значение left, в результате чего первая буква стала располагаться не на одной линии с другими символами строки, а затем был изменен тип шрифта, его размер, степень выделения (жирность), цвет самого символа и цвет фона для его вывода.

Для второго блока был применен эффект заглавной буквы, поэтому свойство float в этом случае не изменялось, а принималось равным значению по умолчанию, т. е. none.

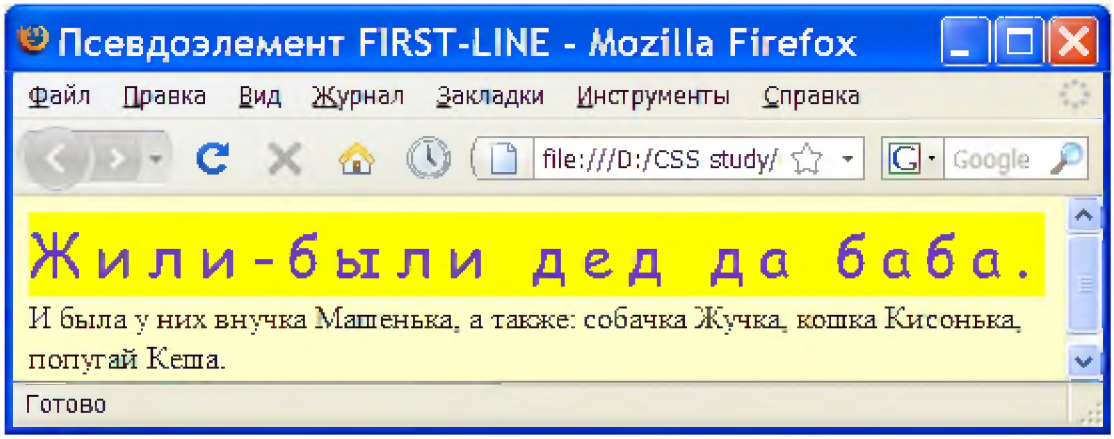
:first-line

Псевдоэлемент :first-line используется для изменения стиля первой строки абзаца. Он может применяться только к элементам уровня блока.

Браузеры по-разному интерпретируют инструкции, описанные в :first-line.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Псевдоэлемент FIRST-LINE</title>
    <style>
      body {background:#FFFFCC;}
      .Style1:first-line{
            background:#FFFF00;
```

```
            color:#6633CC;
            font-family:cursive;
            font-size:30px;
            letter-spacing:7px;
            }
    </style>
  </head>
  <body>
    <div class="Style1">
      Жили-были дед да баба. И была у них внучка Машенька,
      а также: собачка Жучка, кошка Кисонька, попугай
      Кеша.
    </div>
  </body>
</html>
```



В приведенном примере для первой строки был изменен тип шрифта, его цвет и размер, цвет фона и расстояние между символами текста.

JavaScript

Работа в интернете основана на схеме «клиент – сервер». Серверы – это программы или компьютеры, предоставляющие клиентам различные сервисы, например сервер электронной почты, файловый сервер или веб-сервер. Клиенты – это потребители, программы или компьютеры, использующие то, что предлагают им серверы, например различные браузеры.

Первоначально веб-страницы были статическими и всегда имели один и тот же вид. Затем с развитием популярности интернета выросла и изменилась область применения веб-страниц. Активно стали применяться сценарии. Сценарии – это небольшие программы, записанные на каком-

либо сценарном языке программирования, автоматизирующие определенную задачу, которую без сценария пользователь делал бы вручную.

Первыми появились серверные сценарии. Они позволили получать и обрабатывать информацию от пользователей, что, в свою очередь, привело к тому, что появилась возможность создавать веб-страницы «на лету», в зависимости от запроса пользователя. Серверные сценарии во многом расширили возможности веб-разработчиков. Однако при этом значительно выросла нагрузка на веб-серверы и увеличился сетевой трафик. Кроме того, пользователи часто допускали ошибки или вводили неверную информацию, которая все равно поступала на веб-сервер и обрабатывалась им. Все это привело к необходимости обрабатывать и проверять информацию на стороне клиента до ее отправки на сервер.

JavaScript был создан в 1995 г. программистами компании Netscape. Сам язык изобрел Брендан Эйх (Brendan Eich). Помимо Брендана Эйха, в разработке участвовали сооснователь Netscape Communications Марк Андриссен и сооснователь Sun Microsystems Билл Джой. Перед ними стояла цель: за 10 дней создать язык для «склеивания» составляющих частей веб-ресурса (изображений, плагинов, Java-апплетов), который был бы удобен для веб-дизайнеров и программистов, не обладающих высокой квалификацией. Первоначально язык назывался LiveScript и предназначался для программирования как на стороне клиента, так и на стороне сервера. На синтаксис языка оказали влияние языки C и Java, и, поскольку в то время было модным слово Java, 4 декабря 1995 г. LiveScript переименовали в JavaScript, получив соответствующую лицензию у Sun.

Впервые новый язык был использован в браузере Netscape Navigator 2.0. Это дало браузеру Netscape Navigator дополнительные преимущества на рынке сбыта. Из-за этого разработчики Internet Explorer включили в него поддержку другого, но по большей части совместимого с JavaScript языка. Поскольку JavaScript имел лицензию Sun, компания Microsoft назвала свою разработку JScript. После этого JScript стал использоваться во всех последующих браузерах от Microsoft, начиная с Internet Explorer 3.0.

Далее в погоне за прибылью в каждой из последующих версий Netscape и Internet Explorer стали появляться новые возможности, несовместимые с браузером-конкурентом. Делалось это для того, чтобы заставить разработчиков включать в свои страницы новые технологии, доступные только в одном из браузеров, поддерживающем язык JavaScript или JScript. Фактически компании выкидывали и продолжают выкидывать деньги на то, чтобы создать несовместимый код. Для обеспечения совме-

стимости версий языка независимых разработчиков Генеральной ассамблеей ECMA (Ecma International – ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий) был создан стандарт. Стандартизированная версия имеет название ECMAScript и описывается стандартом ECMA-262. Она основана на нескольких базовых технологиях, наиболее известными из которых являются JavaScript (Netscape) и JScript (Microsoft). Стандартизированная версия JavaScript, называемая ECMAScript, работает одинаково во всех приложениях, поддерживающих стандарт.

JavaScript – это язык управления сценариями просмотра веб-страниц. Сценарий – это программа, имеющая дело с готовыми программными компонентами, которые, однажды загруженные, в своей работе не зависят от дальнейшего подключения к интернету. Наиболее часто JavaScript обеспечивает программирование на стороне клиента. Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра веб-страницы пользователем. При этом перезагрузки страницы не происходит. На практике это выражается в том, что можно, например, изменить цвет фона страницы или интегрированную в документ картинку, открыть новое окно или выдать предупреждение.

JavaScript – это интерпретируемый язык программирования с объектно ориентированными возможностями. Изначально он создавался только для браузера, но сейчас используется на многих других платформах, например на сервере или любом другом устройстве, которое имеет специальную программу, называющуюся движком JavaScript. У браузеров есть собственные движки.

Скрипты (сценарии), написанные на JavaScript, не компилируются в исполняемую программу, а хранятся в текстовых файлах с расширением .js или в виде вставленных в HTML-код фрагментов и передаются движку каждый раз при запуске сценария на исполнение. Программа на JavaScript не имеет классической стартовой функции main, присущей любому компилируемому языку, с которой начинается работа программы.

JavaScript включает стандартную библиотеку объектов, например Array, Date и Math, а также базовый набор операторов и управляющих конструкций. Ядро JavaScript может быть расширено для различных целей путем добавления в него новых объектов. JavaScript на стороне клиента расширяет ядро языка, предоставляя объекты для контроля браузера и его Document Object Model (DOM). Клиентские расширения позволяют приложению, например, размещать элементы в HTML-форме и обрабатывать пользовательские события, такие как щелчок мыши, ввод данных в форму

и навигация по страницам и др. Расширение JavaScript на стороне сервера позволяет приложению, например, соединяться с базой данных, обеспечивать непрерывность информации между вызовами приложения или выполнять манипуляции над файлами на сервере и др.

Возможности JavaScript в браузере

JavaScript называют безопасным языком программирования, поскольку он не предоставляет низкоуровневый доступ к памяти или процессору, так как изначально был создан для браузеров, не требующих этого.

С помощью JavaScript в браузере доступно все, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером. Например, можно:

- добавлять новый HTML-код на страницу;
- изменять контент страницы;
- изменять стили на странице;
- реагировать на действия пользователя (щелчки мыши, перемещения указателя, нажатия клавиш и т. д.);
- отправлять запросы на сервер;
- скачивать и загружать файлы;
- получать и устанавливать куки;
- задавать вопросы пользователю;
- показывать сообщения;
- запоминать данные на стороне клиента.

Ограничения JavaScript в браузере

Возможности JavaScript в браузере были сознательно ограничены в целях безопасности, а именно в предотвращении доступа недобросовест-ной веб-страницы к личной информации или нанесения ущерба данным пользователя:

- JavaScript на веб-странице не может читать и записывать произвольные файлы на жестком диске, копировать их или запускать программы;
- JavaScript не имеет прямого доступа к системным функциям операционной системы;

• работа с файлами предполагает ограниченный доступ: только при совершении пользователем определенных действий, таких как перетаскивание файла в окно браузера или его выбор;

• при организации взаимодействия с дополнительными устройствами, например камерой или микрофоном, требуется явное разрешение пользователя, что исключает возможность незаметно включать эти устройства;

• JavaScript с одной страницы или вкладки в браузере не имеет доступа к другой;

• в случае если при помощи JavaScript, открывается другая страница, то JavaScript с открывающей страницы не имеет доступа к открытой им странице, если страницы принадлежат разным сайтам;

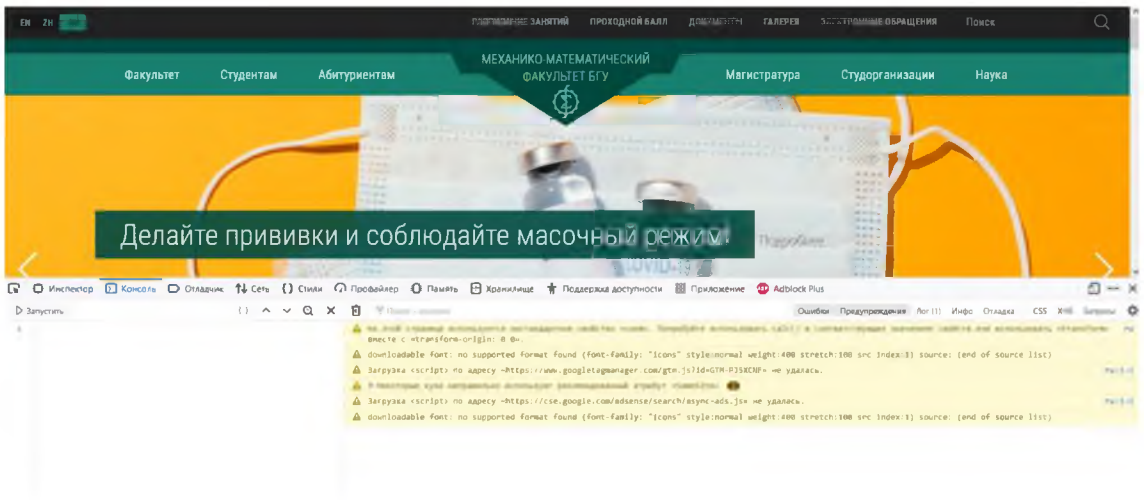
• JavaScript может отправлять и получать данные с сервера, с которого пришла текущая страница, но его способность получать данные с других сайтов ограничена, поскольку для этого требуется явное согласие, выраженное в заголовках HTTP.

У современных браузеров есть расширения, с помощью которых можно запрашивать дополнительные разрешения для действий, на которые наложены ограничения.

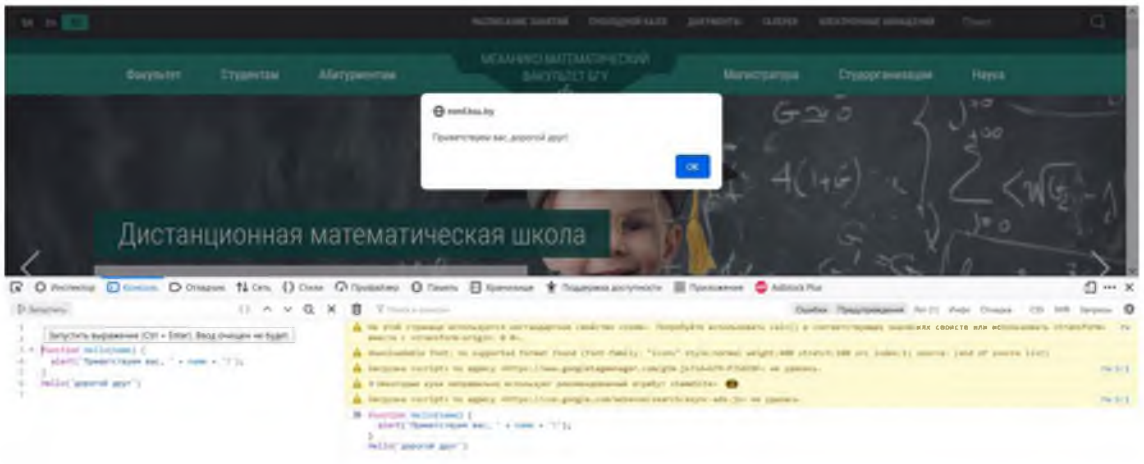
Эти ограничения на JavaScript действуют только в браузерах. Вне браузеров, например на сервере, таких ограничений нет.

Консоль JavaScript

По умолчанию все ошибки, допущенные в коде, браузер пытается решить сам. Он интерпретирует код тем или иным образом и исправляет по собственным алгоритмам либо игнорирует неверные инструкции, как в случае с CSS. Сами ошибки в браузере не видны. О наличии ошибок разработчик может догадаться только по тому, что страница отображается неверно, не работает какой-то функционал. А это значит, что, если что-то не будет работать или будет работать не так, как было задумано, мы явно не увидим причину, из-за которой возникла проблема. В браузер встроены ***Инструменты разработчика***, которые позволяют работать с кодом страницы, смотреть на ошибки, выполнять различные команды, изменять код и многое другое. В операционной системе Windows в браузерах Chrome и Firefox открыть «Инструменты разработчика» можно, нажав клавишу F12. Работать с JavaScript можно через консоль разработчика. Для этого надо перейти на вкладку ***Консоль***. Консоль разработчика отображает информацию о загруженной веб-странице.



Как видно на рисунке выше, отображаются в том числе и ошибки, допущенные в коде JavaScript. Кроме этого, «Консоль разработчика» включает командную строку, которую можно использовать, чтобы выполнить код JavaScript.



Размещение JavaScript-кода на веб-странице

Для того чтобы сценарии JavaScript могли выполняться на стороне клиента, в настройках браузера должен быть выбран соответствующий пункт. Время и условия выполнения скрипта зависят от того, когда и как этот интерпретатор JavaScript получает управление, что, в свою очередь, зависит от размещения кода. В общем случае можно выделить четыре способа размещения JavaScript-кода:

- в гипертекстовых ссылках (схема URL);
- в обработчиках событий (в атрибутах событий);
- во вставках (контейнеры <script>);
- внешний JavaScript.

URL-схема javascript:

При обнаружении браузером на веб-странице ссылки вызывается стандартная программа реализации гипертекстового перехода. JavaScript позволяет поменять стандартную программу на программу пользователя. Для этого в атрибуте href надо указать JavaScript-код следующим образом: `<a href="javascript: код_программы">...</a>`, где `код_программы` – это программа JavaScript, которая исполняется при выборе гипертекстовой ссылки.

В приведенном ниже примере при нажатии на гипертекстовую ссылку «Это важно» появляется окно предупреждения «Внимание!».

```
<a href="javascript:alert('Внимание! Не забудьте сделать
домашнее задание.')">Это важно</a>
```

Такую схему можно использовать и в некоторых других атрибутах, например в атрибуте ***action***:

```
<form name="mainForm" action="javascript:
    if document.getElementById('textFieldId').value==''')
    alert('Вы забыли заполнить поле!'); void(0);">
  <input type="text" id="textFieldId" size="30">
  <p>
    <input type="submit" value="Заполнить">
    <input type="reset" value="Очистить">
  </p>
</form>
```

JavaScript в обработчиках событий

JavaScript-программы могут размещаться в обработчиках событий. Например, в момент завершения полной загрузки документа происходит событие Load и вызывается обработчик события onLoad, при нажатии на кнопку происходит событие Click и, соответственно, вызывается обработчик этого события onClick. Или

```
<body onLoad="alert('Приветствуем тебя, гость!');">
  <input type="button" value="Жми сюда!"
    onClick="alert('Вы нажали кнопку');">
</body>
```

Контейнеры <script>

Программы на JavaScript могут внедряться в HTML-документ при помощи тега <script>. Теги <script> могут быть размещены в любом месте внутри тега <body> или внутри тега <head> и использоваться любое количество раз.

Если при разборе документа HTML-парсер встречает тег <script>, он передает управление JavaScript-интерпретатору. После этого JavaScript-

интерпретатор выполняет код внутри контейнера <script> и возвращает управление HTML-парсеру.

Если JavaScript-код находится внутри тега <script>, то внутри его ни в каком контексте не может находиться сочетание </script>, поскольку HTML-парсер сначала определяет границы скрипта, а затем передает его интерпретатору JavaScript. Если HTML-парсер встретит где-либо сочетание </script>, он на этом месте заканчивает идентификацию участка текста как скрипта и выбранный участок кода передает JavaScript-интерпретатору. Чтобы избежать этой ошибки, в данном случае достаточно экранировать служебный символ слэша в строке <\/script> или разбить сочетание на несколько строк </script>, например так: '</'+'script>'.

Поскольку существует несколько языков для написания скриптов, то в теге <script> можно указать язык скрипта. Для этого можно использовать атрибут type. В HTML5 атрибут type не указывается, поскольку JavaScript является языком сценариев по умолчанию, но при необходимости можно явно указать другие значения типа IANA.

Внешний JavaScript

Сценарии можно размещать во внешних файлах. Это практично, поскольку один и тот же сценарий можно использовать на разных веб-страницах. Присоединяются внешние файлы со скриптами при помощи тега <script>. Внутри тега <script> может быть размещен как непосредственно код скрипта, так и адрес файла, в котором размещен скрипт. Адрес заносится в атрибут src. Если присутствует атрибут src, то содержимое контейнера <script> должно быть пустым, поскольку, согласно спецификации HTML, если скрипт подключается из внешнего файла, то все, что написано между тэгами <script> и </script>, будет проигнорировано браузером. Внешний скрипт можно присоединять внутри элементов <head> или <body>. Можно присоединять несколько файлов с JavaScript-кодом на страницу.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script>
      document.writeln("Выполнен скрипт из head");
    </script>
    <script src="Script1.js"></script>
  </head>
  <body>
    <p id="demo"></p>
    <script>
```

```
      document.getElementById("demo").innerHTML =
        "Выполнен скрипт из body";
      </script>
    <noscript>Извините, ваш браузер не поддерживает
      JavaScript! Отображение страницы будет
      неполным.
    </noscript>
    <script src="Script2.js"></script>
    <script src="Script3.js"></script>
  </body>
</html>
```

Файлы сценариев:

```
// JavaScript Document Scripts1.js
document.writeln("Выполнен скрипт из Scripts1.js,
подключенный в head");
// JavaScript Document Script2.js
document.writeln("Выполнен скрипт из Script2.js,
подключенный в body");
// JavaScript Document Script3.js
document.writeln("Выполнен скрипт из Script3.js,
подключенный в body");
```

Преимущества внешнего JavaScript

Размещение сценариев JavaScript во внешних файлах имеет ряд преимуществ:

- это разделяет HTML и JavaScript;
- воспринимать такой код проще;
- это удобно при поддержке кода;
- загрузка последующих веб-страниц, куда подключен JavaScript-код, происходит быстрее, поскольку файлы сценариев уже есть в кэше.

Упражнение

Продемонстрируйте подключение JavaScript к странице всеми известными вам способами на примере вывода сообщений.

Способы ввода информации и вывода результатов

Поскольку мы изучаем использование JavaScript на веб-страницах, то результат выполнения кода можно вывести:

- на страницу при помощи innerHTML;
- на страницу при помощи document.write();

• в окно сообщений, например окно предупреждения при помощи window.alert();

• в консоль разработчика при помощи console.log().

Следующий код выводит результат работы скрипта на страницу при помощи innerHTML.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script>
      function dateDemo(){
        let today = new Date();
        let dd = String(today.getDate());
        let mm = String(today.getMonth() + 1);
        let yyyy = today.getFullYear();
        today = mm + '/' + dd + '/' + yyyy;
        return today;}
    </script>
  </head>
  <body>
    <p id="demo">Этот текст пропадет, а выведется дата</p>
    <script>
  document.getElementById("demo").innerHTML = dateDemo()
    </script>
    <script>
      document.write('2. ' + dateDemo());
    </script>
    <script>
      alert('3. ' + dateDemo());
    </script>
    <script>
      console.log('4. ' + dateDemo());
    </script>

  </body>
</html>
```

В свойстве innerHTML хранится в виде строки содержимое элемента, к которому применяется свойство. Свойство доступно для чтения и записи, поэтому можно получать и изменять содержимое элемента. Однако следует помнить, что если вы выводите что-либо в элемент при помощи innerHTML, то весь HTML-код, который был внутри этого элемента до вывода, будет потерян. Метод выводит на страницу переданные ему аргументы. На этапе загрузки страницы метод добавляет контент. Если document.write() вызвать после того, как страница загрузилась, то результатом

будет перезаписанная страница с текстом, который был добавлен с помощью document.write().

Объект window является глобальным в браузерах, но к нему нельзя обратиться напрямую. Обратиться к нему можно при помощи его свойства window, которое ссылается на сам объект. Метод alert() объекта window выводит модальное диалоговое окно с сообщением и кнопкой OK.

Объект console служит для доступа к средствам отладки браузера. Доступ к console можно получить через свойство глобального объекта window: window.console или просто console. Console.log выводит сообщение в веб-консоль.

Системные диалоговые окна

Окна сообщений позволяют выводить различные сообщения, в том числе и результаты работы, а также при необходимости вводить данные. Диалоговые окна бывают трех типов:

- окно с сообщением и кнопкой OK;
- окно с сообщением и кнопками OK и Cancel;
- окно с сообщением, полем ввода и кнопками OK и Cancel.

Эти диалоговые окна выводятся при помощи методов alert, confirm и prompt соответственно. Каждое из них является синхронным и модальным, т. е. загрузка дальнейшей части страницы приостанавливается на время показа такого окна и возобновляется после его закрытия. Кроме этого, они приостанавливают загрузку дальнейшей части страницы.

Дизайн этих диалоговых окон очень простой и непритязательный. Внешний вид окна зависит от браузера, и разработчик не может изменить его и переделать дизайн в стиле сайта. Это не всегда допустимо, но это самый простой способ вывести сообщение или получить информацию от пользователя, поэтому их используют в тех случаях, когда дизайн не особо важен. Место, куда будет выведено окно сообщений, также выбирает браузер.

Окно с сообщением и кнопкой OK

Стандартное окно с сообщением и кнопкой OK можно вывести при помощи метода alert() класса window. В метод в качестве параметра передается строка, которая и будет выведена в диалоговом окне.

```
alert("message");
```

Если сообщение в окне должно быть выведено в несколько строк, то в местах разделения строк должен быть вставлен символ перехода на следующую строку \n.

После нажатия кнопки OK окно метода alert() закрывается.

Метод используется для отображения сообщения, не требующего решения пользователя.

Следующий скрипт выводит сообщение.

```
<script> alert("Раз, \ndва, \nтри, \n четыре, \nпять\n
вышел \nзайчик \nпогулять. \nПора завершать работу и также
идти гулять");
</script>
```

Окно с сообщением и кнопками OK и Cancel

Стандартное диалоговое окно с сообщением и кнопками OK и Cancel выводится методом confirm() класса window. Как и для метода alert(), имя класса почти всегда можно опускать. Синтаксис метода имеет вид

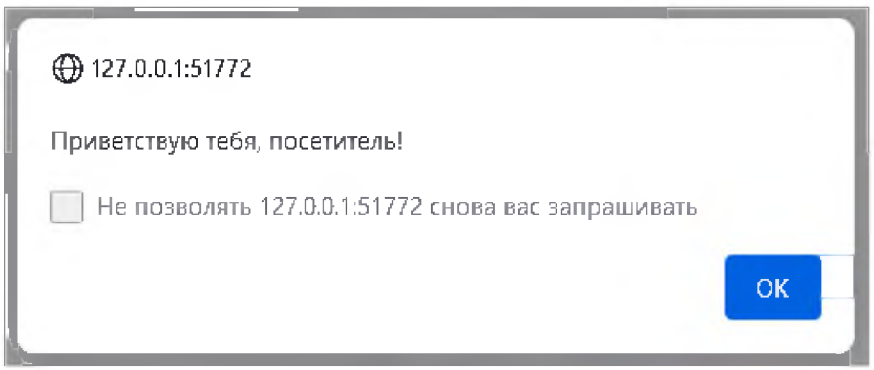
```
confirm("message");
```

Аргумент message определяет сообщение, которое выводится в диалоговом окне и требует выбора пользователя, принять или отказаться. Метод возвращает true, если пользователь выбрал OK, и false – во всех остальных случаях.

```
<script>
  if (confirm("Здороваться будем?")==true)
    document.writeln('Приветствую тебя, посетитель!');
  else
    document.writeln('Не больно-то и хотелось!');
</script>
```

Если JavaScript-интерпретатор посчитает, что в скрипте выводится слишком много диалоговых окон сообщений, то на окне будет дополнительно размещен флажок, позволяющий блокировать дополнительные диалоговые окна. Это касается также и методов alert() и promt().

```
<script>
  if (confirm("Здороваться будем?")==true)
    alert('Приветствую тебя, посетитель!');
  else
    alert('Не больно-то и хотелось!');
</script>
```



Окно с сообщением, полем ввода и кнопками OK и Cancel

Метод prompt() выводит стандартное диалоговое окно с сообщением и кнопками OK и Cancel, требующее от пользователя ввода текста. Как и два предыдущих метода, prompt() является методом класса window, причем имя класса почти всегда можно опускать. Синтаксис метода имеет вид

```
prompt(message, default);
```

Здесь message – строка, которая будет выведена в качестве сообщения, а необязательный параметр default – это строка, число или значение свойства, определяющее значение поля ввода по умолчанию. Если параметр default не определен, то диалоговое окно отображает значение типа undefined (поле пустое). Метод возвращает введенное значение или null, если пользователь отказался от ввода и нажал на кнопку Cancel.

```
<script>
  let years=prompt('Сколько вам лет?','18');
  if (years == ''){
    alert('Так много лет, что затрудняетесь посчитать?
    Или так мало, что плохо считаете?');}
  else if(years == null){
    alert('Смешно скрывать свой возраст!');}
  else {alert('Ого! Вам '+years+'! Поздравляем!');}
</script>
```

Упражнение

> Создайте простейшую веб-страницу. Спросите у пользователя, сколько ему лет. Если пользователь введет число меньше 18, то вместо контента на странице выведите фразу «Вам еще рано просматривать этот контент».

Основные синтаксические правила

Инструкции – это синтаксические конструкции и команды, которые выполняют действия. Код JavaScript может содержать множество инструкций, каждая из которых завершается точкой с запятой. Чтобы код было легче читать, каждую инструкцию пишут на отдельной строке. В этом случае точку с запятой можно не ставить, поскольку браузеры могут распознать переход на следующую строку как неявную точку с запятой. Однако в JavaScript есть возможность переносить часть слишком длинных инструкций на следующую строку (обычно это делается после знака операции). В случае такого разрыва инструкции JavaScript может ошибочно поставить точку с запятой, и код будет интерпретироваться не так, как вам бы того хотелось, или вовсе станет ошибочным. В следующем

примере JavaScript добавляет точку с запятой после return, а при отсутствии выражения после return инструкция возвращает значение undefined. Поэтому синтаксической ошибки нет, но функция будет работать не так, как ожидалось.

```
<script>
  function f(x){
    if(x>0)
      return 2*x;
    else
      return
    x+0.1;
  }
  x=1;
  document.write('x='+x+'<br>');
  x=f(x);
  document.write('новое значение x='+x+'<br>');
  x=-2;
  document.write('x='+x+'<br>');
  x=f(x);
  document.write('новое значение x='+x+'<br>');
</script>
```

x=1
новое значение x=2
x=-2
новое значение x=undefined

Поэтому, несмотря на то что заключение операторов точкой с запятой не требуется, а только настоятельно рекомендуется, все же стоит ставить точки с запятой после каждого оператора.

JavaScript, как и многие языки, в некоторых случаях игнорирует дополнительные пробелы в инструкциях. Поэтому желательно добавлять пробелы в код, чтобы сделать его более читабельным. Хорошая практика – ставить пробелы вокруг операторов (= + – * /):

```
let y = 1; // let y=1; воспринимается хуже
let x = y + z; // let x=y+z; воспринимается хуже
```

JavaScript – регистрозависимый язык.

В JavaScript используется набор символов Unicode, поэтому при написании кода их можно использовать.

Идентификаторы

Имена переменных могут содержать буквы, цифры, знаки подчеркивания и доллара. Цифра не может быть первым символом. В качестве имен

нельзя использовать зарезервированные слова. В идентификаторах допустимо использовать символы Unicode, например å или ü. Например, слово Gemüse означает по-немецки «овощ». Его можно использовать в качестве имени переменной.

```
var Gemüse = "Kartoffel";
```

По этой же причине можно использовать русские или белорусские буквы.

```
var овощ = "картофель";
var воўк  = "Серы";
```

Использовать символы национальных алфавитов в идентификаторах можно, но не рекомендуется.

Комментарии

Комментарии в JavaScript С-подобные. Для обозначения однострочного комментария используется двойной слеш. Для выделения многострочного комментария используется /* и */. Вложенные комментарии не поддерживаются.

```
<script>
/*
  /* Ошибка! Вложенный комментарий не допускается. */
*/
  x1 = 1; //Первое число Фибоначчи
  x2 = 1; /*Второе число Фибоначчи*/
  document.write(x1 + "  ");
  document.write(x2 + "  ");
  /*Начиная с третьего,
    находим числа Фибоначчи в цикле, */
  for(i = 3; i < 11 ; i++){
    x3 = x1 + x2;
    document.write(x3 + "  ");
    x1 = x2;
    x2 = x3;
  }
</script>
```

Зарезервированные слова

Зарезервированные слова не могут быть именами переменных, функций и меток циклов. Зарезервированные слова JavaScript: abstract, arguments await, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, eval, export, extends, false, final,

finally, float, for, function, goto, if, implements, import, in, instanceof, int, interface, let, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, void, volatile, while, with, yield.

Кроме того, следует избегать использования идентификаторов встроенных объектов, свойств и методов, предопределенных в языке JavaScript: Array, Date, eval, function, hasOwnProperty, Infinity, isFinite, isNaN, isPrototypeOf, length, Math, NaN, name, Number, Object, prototype, String, toString, undefined, valueOf. Если попытаться создать переменную или функцию с таким именем, то это будет приводить либо к ошибке (если свойство определено как доступное только для чтения), либо к переопределению.

Не стоит использовать как идентификаторы также имена объектов и свойств HTML, объекта Window и имена обработчиков событий HTML. Нежелательно использовать ключевые слова из языков программирования, поскольку JavaScript можно использовать как язык программирования во многих приложениях.

Переменные

Переменные в JavaScript могут быть объявлены тремя способами.

Первый способ: с использованием ключевого слова let. Например, `let userLogin;`. Данный синтаксис может быть использован для объявления локальной переменной в области видимости блока.

Второй способ: объявление без указания ключевых слов. Например, `x = 42`. Переменные, объявленные данным способом, являются глобальными. Такое объявление генерирует строгое предупреждение, поэтому его не рекомендуется использовать.

Третий способ: с использованием ключевого слова var. Например, `var userName`. Такой синтаксис используется для объявления как локальных, так и глобальных переменных. Ключевое слово var является устаревшим, и его не рекомендуется использовать.

Один раз использовав ключевое слово var или let, можно объявить несколько переменных, перечислив их через запятую:

```
var userName, userLogin;
```

При объявлении можно сразу присвоить переменной значение, совместить объявление с инициализацией:

```
let userLogin = "guest";
```

Если при объявлении переменной с использованием ключевого слова var или let ей не было присвоено никакого значения, она будет иметь специальное значение undefined до тех пор, пока ей не будет присвоено другое значение:

```
var a;
let b;
console.log(a);    // undefined
console.log(b);    // undefined
```

В следующем примере переменная определена как глобальная до использования, но не инициализирована. Значение переменной – undefined.

```
var x;
console.log("x = " + x);
```

При попытке доступа к необъявленной переменной или переменной до ее объявления будет выброшено исключение ReferenceError:

```
console.log("y = " + y);
```

В JavaScript можно использовать переменные, которые будут объявлены в коде ниже при помощи ключевого слова var. Это называется поднятием или всплыванием переменных. Переменные, которые еще не были инициализированы, возвратят значение undefined:

```
console.log("z = " + z);
var z = 20;
```

Переменные, определенные с помощью let, должны быть объявлены перед использованием. Следующий код вызовет ошибку ReferenceError при попытке использовать переменную, объявленную в коде ниже при помощи ключевого слова let.

```
console.log("The value of x is " + x); //Uncaught
ReferenceError: x не определена
let x;
```

Переменные в JavaScript не имеют типа, поэтому переменной может быть присвоено значение любого типа, а затем ей же может быть присвоено значение уже другого типа:

```
var x = 1;
let y = 1.1;
x = 5.6;
y = "content";
```

Языки, в которых типы данных существуют, но переменные не привязаны ни к одному из типов данных, называются динамически типизированными. JavaScript – динамически типизированный язык программирования.

В реальном проекте бóльшая часть времени тратится на сопровождение кода, т. е. на его изменение и расширение, а не на написание нового кода с нуля. Причем ищут первоначальный код одни люди, а сопровождают его совершенно другие. Когда просматривается чужой код или свой, но после какого-то промежутка времени, гораздо легче понять его, если он хорошо размечен. Этому способствуют хорошие имена переменных.

Перечислим правила хорошего наименования. Имена должны легко читаться, например `userName`. Избегайте использования аббревиатур или

коротких имен, таких как a, b, c. Если идентификатор содержит несколько слов, обычно используется верблюжья нотация, т. е. слова следуют одно за другим, где каждое следующее слово начинается с заглавной буквы: `warningSigns`. Имена должны быть максимально описательными и лаконичными. Например, не стоит использовать `data` для обозначения даты рождения или `value` для обозначения затрат. Такие имена ничего не говорят. Лучше использовать `birthday` или `amountCosts`. Именование переменных – это один из самых важных и сложных навыков в программировании. Быстрый взгляд на имена переменных может показать, какой код был написан новичком, а какой – опытным разработчиком.

Константы

При объявлении констант используется слово const:

```
const MY_PET = 'puppy';
```

Константы используются в качестве псевдонимов значений, которые известны до начала исполнения скрипта. Это позволяет избежать использования «магических чисел». Названия констант пишутся с использованием заглавных букв и подчеркивания. Например, для обозначения призывного возраста подойдет константа MILITARY_AGE:

```
const MILITARY_AGE = 18;
```

Названия таких констант пишутся с использованием заглавных букв и подчеркивания.

Типы данных

Тип данных переменной зависит от значений, которые она принимает. Тип переменной может изменяться в ходе выполнения программы. Преобразование типов выполняется автоматически и зависит от контекста в котором используется переменная. Например, при суммировании числовых и строковых значений числовые значения преобразуются в строковые:

```
let message = 10 + " дней до отпуска";
typeof (10 + " дней до отпуска"); // вернет "10 дней до
отпуска"
```

В ходе выполнения операций происходит динамическое приведение типов. Однако следует помнить, что JavaScript оценивает выражения слева направо, производя приведение типов последовательно.

```
1 + 38 + ' попугаев'; // 39 попугаев
'1' + 38 + ' попугаев'; // 138 попугаев
```

В случае `1 + 38 + ' попугаев'` JavaScript обрабатывает 1 и 38 как числа, пока не достигнет « попугаев». В случае `'1' + 38 + ' попугаев'` первый операнд является строкой, поэтому все операнды обрабатываются как строки.

Получить тип данных переменной или выражения можно с помощью оператора typeof.

Ядро языка JavaScript работает с такими простыми типами данных, как числа, строки и булевские значения, поддерживает работу с массивами, датами и регулярными выражениями. Язык JavaScript содержит восемь типов данных:

- number – числа, как целые, так и с плавающей точкой;
- BigInt – целые числа произвольной длины;
- string – строки;
- boolean – логический тип данных;
- object – объектный тип данных;
- null;
- undefigned;
- symbol (символ) – используется для создания уникальных идентификаторов в объектах.

Типы данных в JavaScript делятся на простые, или примитивные, и составные. К простым типам данных относят строковый, числовой, логический. Также к простым типам относятся значения null и undefined.

Разница между простыми и составными типами видна при копировании значений. Когда переменной присваивается значение простого типа, например число, то в переменную записывается само значение. При выполнении присваивания одной переменной со значением простого типа другой переменной происходит копирование значения. В результате каждая переменная имеет свою копию значения, и изменения в одной из переменных никак не сказывается на значении другой:

```
let amountFirst = 100;
let amountSecond = amountFirst; // Копируем значение
document.write("У вас на счету: " + amountFirst + // 100
     "<br> Вы можете потратить: " + amountSecond); // 100
amountFirst = 20; // Изменяем значение
document.write("<br><br>У вас на счету:"+amountFirst+//20
      "<br> Вы можете потратить: " + amountSecond); // 100
```

Когда переменной присваивается значение составного типа, например объект, то в переменную записывается ссылка на значение. При выполнении присваивания одной переменной копируется ссылка, а не значение. В результате обе переменные ссылаются на одно и то же место в памяти, и любые изменения одной из переменных автоматически изменят вторую переменную.

```
let ob1 = {x:0,y:0};
let ob2 = ob1; // Копируем ссылку на объект
document.write("(" + ob1.x + ", " + ob1.y + ")<br>");
//(0,0)
ob1.x = 10; //Изменяем значение первой координаты ob2
ob1.y = 10;// Изменяем значение второй координаты ob2
document.write("(" + ob2.x + ", " + ob2.y + ")<br>"); //
Изменился ob1
```

Числовой тип

В JavaScript к числовому типу number относятся целые числа и числа с плавающей точкой.

```
typeof 12345 // "number"
typeof -12.345 // "number"
typeof 12345e-3 // "number"
typeof 07 // "number"
typeof 0xa // "number"
```

Если в записи числа содержатся только десятичные цифры и оно не начинается с префикса 0 или 0x, то это десятичное целое число. Целые числа могут быть положительными, отрицательными или нулем. Десятичные числа с плавающей точкой содержат в качестве разделителя точку или записываются в научной нотации. Буква e в экспоненциальном формате может быть прописной или строчной.

Шестнадцатеричные целые числа начинаются с префикса 0x (нуль + + икс), причем буква X может быть как большой, так и малой. В запись шестнадцатеричного числа могут входить цифры от 0 до 9 и буквы от A до F, либо строчные, либо прописные. Поскольку буква e обозначает цифру в шестнадцатеричной записи, то шестнадцатеричные числа не могут быть представлены в экспоненциальном формате. Шестнадцатеричные числа могут быть как положительными, так и отрицательными.

Восьмеричные целые числа начинаются с нуля и могут содержать цифры от 0 до 7. Если первая цифра числа – нуль, но в записи числа присутствуют цифры 8 или 9, то считается, что число записано в десятичной системе счисления с незначащим нулем впереди. Восьмеричные числа могут быть отрицательными, но не могут быть числами с плавающей точкой или быть записанными в экспоненциальном формате.

В JavaScript определены также специальные числовые значения:

- NaN (нет числа);
- Infinity (положительная бесконечность);
- –Infinity (отрицательная бесконечность);
- 0 (положительный 0);
- –0 (отрицательный 0).

В отличие от других языков программирования математические операции в JavaScript не могут привести к ошибке или аварийно завершить работу программы. Если математическая операция не может быть совершена, то возвращается специальное значение NaN (Not A Number), которое обозначает математическую ошибку. Например, деление 0/0 в математическом смысле не определено, поэтому результатом операции будет NaN, то же касается и Infinity/Infinity. Значение NaN не равно ничему, включая себя. Его можно проверить только специальной функцией isNaN(n), которая возвращает true, если аргумент NaN, и false – для любого другого значения. Результат любой операции с NaN также будет равен NaN. Значение NaN можно получить также как Number.NaN.

```
typeof (1/0) // NaN
0/0 // NaN
isNaN(0/0) // true
Number.NaN // NaN
NaN+1 // NaN
NaN == NaN // false
NaN === NaN // false
```

В теоретических математических выкладках при делении получается бесконечность. Аналогично в JavaScript значения выражений, в которых происходит деление на нуль, также равны бесконечности со знаком плюс или минус. Infinity – особенное численное значение, которое ведет себя в точности как математическая бесконечность. Когда вещественное число превышает самое большое представимое конечное значение, результату присваивается специальное значение бесконечности, которое в JavaScript обозначается как Infinity. А когда отрицательное число становится меньше наименьшего представимого отрицательного числа, результатом является отрицательная бесконечность –Infinity. Infinity больше любого числа. Добавление к бесконечности любого числа не меняет ее. Бесконечность можно присвоить в явном виде:

```
let x = Infinity;
```

Также специальные значения, обозначающие плюс и минус бесконечность, можно получить как константы Number.POSITIVE_INFINITY и Number.NEGATIVE_INFINITY.

Значение Infinity/Infinity не определено. Сумма бесконечностей одного знака есть бесконечность того же знака. Разность бесконечностей противоположного знака не определена.

```
1/0 // Infinity
-1/0 // -Infinity
Infinity > 1e100 // true
Infinity + 5 // Infinity
Infinity - 1e100 // Infinity
```

```
Infinity / Infinity // NaN
Infinity + Infinity // NaN
Infinity - Infinity // Infinity
```

Для чисел с плавающей запятой в JavaScript определены два нуля: положительный и отрицательный. Однако, как правило, в программе это не нужно учитывать. Оба нуля при сравнении равны и на выводе тоже выглядят одинаково. Знак нуля важен только для операций деления и умножения. Можно определить константу со значением −0, и ее знак будет учитываться при делении.

```
1/ Infinity // 0
-1/ Infinity // -0
1/ Infinity == -1/ true
1 / 0 // Infinity
1 / (-0) // Infinity
```

BigInt

В JavaScript тип number не может содержать числа больше, чем $2^{53} – 1$, или меньше, чем $–2^{53} – 1$. Это ограничение вызвано их внутренним представлением. С изменение разрядности операционной системы диапазон изменится, но он останется. BigInt позволяет работать с целыми числами произвольной длины.

Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала:

```
typeof 1n // "bigint"
```

Данные типа BigInt также можно создать, вызвав функцию BigInt:

```
BigInt("12345678912345678 9");// 12345678912345678 9n
```

В математических операциях нельзя смешивать BigInt и обычные числа. Операция деления возвращает округленный результат без дробной части. Операции сравнения < и > работают как нужно. Однако равенство с обычными типами достигается только при нестрогом сравнении === без приведения типов.

```
1 == 1n  // true
1 === 1n // false
```

Строковый тип

Имя типа – string. Строки – это произвольные текстовые данные, заключенные в кавычки. Кавычки могут быть одинарными или двойными.

Отдельно символьного типа в JavaScript нет. Один символ также является строкой. Все строки, вне зависимости от кодировки, хранятся в формате Unicode. В строки могут включаться специальные символы:

- \b – Backspace, или возврат курсора на одну позицию;

- \f – Form feed, или переход на новую страницу;
- \n – New line, или переход на новую строку;
- \r– Carriage return, или возврат каретки;
- \t – Tab, или табуляция.

Если в строку необходимо включить символ двойных кавычек, то исходную строку надо заключить в одинарные кавычки, и наоборот, при включении внутрь строки одинарных кавычек строка должна быть заключена в двойные кавычки.

```
alert('"Нет" любимый ответ лодыря');
alert("'Нет' любимый ответ лодыря");
```

Если есть необходимость включить в строку кавычки того же типа, в который обрамлена строка, то в этом случае символы кавычек внутри строки должны экранироваться слэшем. Сам символ обратного слэша \ является служебным, поэтому всегда экранируется \\:

```
alert('\'Нет\' любимый ответ лодыря');
alert("\"Нет\" любимый ответ лодыря");
alert("\"Нет\"\\\"Да\" любимый ответ лодыря?");
```

Также в строку могут быть включены любые символы через их код:

- \XXX – символ, заданный тремя восьмеричными цифрами XXX в диапазоне от 0 до 377;
- \xXX – символ, заданный двумя шестнадцатеричными цифрами XX от 00 до FF;
- \uXXXX – символ Unicode, заданный четырьмя шестнадцатеричными цифрами.

```
alert("\251 - знак копирайта");
alert("\xA9 - знак копирайта");
alert("\u00A9 - знак копирайта");
```

Если необходимо включить в строку вычисляемое выражение, то можно использовать обратные кавычки. Включаемые выражения должны быть обернуты в ${...}:

```
function salary(hourSalary, hour) {
  return hourSalary * hour;}
`Почасовая оплата считается как произведение стоимости
часа на число отработанных часов. Ваша сумма к выплате =
${salary(1, 2)}.` // "Почасовая оплата считается как
произведение стоимости часа на число отработанных часов.
Ваша сумма к выплате = 2."
```

Строки с обратными кавычками могут занимать более одной строки.

```
let pasport = `Стихи о советском паспорте
Я волком бы
          выгрыз
               бюрократизм.
```

```
К мандатам
          почтения нету.`;
alert(pasport);
```

Строки являются постоянными. После того как строка создана, она не может быть изменена, но может быть создана новая строка.

Логический тип

Ключевое слово, обозначающее логический тип, – boolean. Логические переменные могут принимать всего два значения: true (истина) и false (ложь).

```
let check = true; // флажок на форме отмечен галочкой
check = false; // флажок на форме не содержит галочки
```

Объекты

Объект в JavaScript можно понимать как коллекцию свойств, ассоциативный массив или список, состоящий из пар «ключ – значение». Причем ключом может быть только строка, даже у элементов массива.

В JavaScript объекты бывают нескольких типов:

- объекты базового языка, или объекты, определяемые спецификацией ECMAScript, например Array, String, Date, Number, Function, Boolean, Math и т. д.;
- клиентские объекты, входящие в модель DOM (представление документа в виде дерева тегов), т. е. отвечающие тому, что содержится или происходит на веб-странице в окне браузера, например window, document, location, navigator;
- серверные объекты, отвечающие за взаимодействие «клиент – сервер», например Server, Project, Client;
- пользовательские объекты.

Объект может быть создан с использованием литерального синтаксиса, когда в фигурных скобках через запятую перечисляются пары из имени свойства и значения. После имени свойства ставится двоеточие и затем указывается значение свойства.

```
address = {city:"Минск", street:"проспект Жукова",
house:50, apartment:11};
```

Основные операции над объектами: добавление новых свойств, изменение уже существующих свойств, удаление свойств и обращение к свойствам.

Доступ к свойствам осуществляется через точку:

```
alert(address.city);
```

Или с использованием квадратных скобок:

```
alert(address["city"]);
```

Пустой объект или пустой ассоциативный массив может быть создан с явным и неявным использованием конструктора.

```
ob1 = new Object();
ob2 = {};
```

Объект может содержать в себе любые значения, называемые свойствами объекта. Доступ к свойствам осуществляется по имени. Синтаксически добавить свойство в объект можно двумя способами. Первый – записав имя нового свойства через точку после имени объекта, второй – указав имя свойства в квадратных скобках.

Например, создадим объект student для хранения информации о студенте: фамилии, имени, отчества, адреса, номера студенческого билета и среднего балла.

```
var student = {};
student.name='Иванов И.И.';
student.adress='ул. Ленина, 23-7';
student["num"]=123456;
student['average']=7.5;
```

Квадратные скобки используются в основном, когда название свойства находится в переменной или состоит из двух слов:

```
var average = 'ball'
student['average']=7.5;
```

Здесь имя свойства 'ball' является ключом в ассоциативном массиве, по которому расположено значение 7.5.

Доступ к свойству осуществляется точно так же, как и при его добавлении, – через точку или квадратные скобки:

```
alert(student['name'] + '\n' + student["address"] + '\n'
+ student.num + '\n' + student.average);
```

Если у объекта нет такого свойства, то никакой ошибки при обращении по несуществующему свойству не будет, просто вернется специальное значение undefined:

```
alert(student.fio);
```

Свойства можно указывать непосредственно при создании объекта, через список в фигурных скобках вида {..., ключ : значение, ...}:

```
let book = {autor: 'Пушкин', title: 'Евгений Онегин',
pages: 132};
```

Можно перебрать все свойства объекта. Для этого используется специальная конструкция for…in:

```
str='';
for(var key in book){
  // key - название свойства
  // object[key] - значение свойства
str=str+book[key]+'  ';
```

```
}
alert(str);
```

Можно удалить свойство объекта с помощью оператора delete.

```
delete (book.pages);
str='';
for(var key in book){
  str=str+key + ': ' + book[key]+'  ';
}
alert(str);
```

У объектов JavaScript могут быть методы. Метод объекта – это функция, которая является значением свойства.

```
let book = {autor: 'Пушкин', title: 'Евгений Онегин',
pages: 132, read : function(n){alert("Прочитано "+n+"
страниц.");}};
book.read(40);
```

Метод может не просто вызываться из объекта, но иметь доступ к самому объекту, влиять на сам объект и менять находящиеся в нем данные. Для этого используется ключевое слово this.

```
let kat = { name:'Мурзик' }
let dog = { name:'Шарик' }
myPet = function() {
 alert("Мой любимый домашний питомец - "+ (this.name ?
this.name : 'безымянный') )
}
// один и тот же метод в двух объектах
kat.myPet = myPet
dog.myPet =kat.myPet;
// в этом вызове - this будет kat
//Мой любимый домашний питомец - Мурзик
kat.myPet();
// в этом вызове - this будет dog
//Мой любимый домашний питомец - Шарик
dog.myPet();
// а тут - вызывается метод глобального объекта window,
у которого нет имени
myPet();
// Мой любимый домашний питомец - безымянный
```

null и undefined

Нулевой тип включает только значение null. Это не отдельный тип, а специальное значение. Считается, что у значения *null* объектный тип и оно говорит об отсутствии объекта.

undefined – еще одно специальное значение, используемое в JavaScript. Оно возвращается при обращении либо к переменной, которая была

объявлена, но которой никогда не присваивалось значение, либо к свойству объекта, которое не существует.

```
typeof(undefigned) // "undefined"
let x; typeof(x); // "undefined"
y=5; typeof(y.prop); // "undefined"
```

Специальное значение undefined – это не то же самое, что null. Хотя значения null и undefined не эквивалентны друг другу, оператор эквивалентности == считает их равными:

```
x.prop==null // true
```

Когда требуется отличить null от undefined, используется оператор идентичности === или оператор typeof():

```
x.part===null // false
```

Symbol

Symbol (символ) — это примитивный тип данных, экземпляры которого уникальны и неизменяемы. В некоторых языках программирования символы также называются атомами. Функция Symbol() создает экземпляры типа symbol. Единственное разумное использование – сохранить символ, а затем использовать сохраненное значение для создания свойства объекта.

```
let handlerMethod  = Symbol();
this[handlerMethod] = function() {...};
```

Преобразования типов данных

В JavaScript значения одного типа могут быть преобразованы в значения другого типа. Это могут быть явные и неявные преобразования.

Неявное преобразование происходит, когда интерпретатор автоматически выполняет преобразование типов без участия программиста. Например, если какой-нибудь оператор ожидает получить значение определенного типа, а ему передается значение другого типа, то интерпретатор автоматически попытается выполнить преобразования к нужному типу:

```
33 + " коровы" // "33 коровы". Число неявно преобразуется
в строку
'6' * '6';// 36. Обе строки неявно преобразуются в числа
```

Явное преобразование – это когда преобразование выполняет сам программист. Также его называют приведением типов.

```
Number('2')  *  parseInt("12.5"));      //  25.  Явное
преобразование
```

В JavaScript можно представлять числа в виде строк и преобразовывать строки в числа.

Преобразование чисел в строки

Преобразование чисел в строки производится автоматически. Например, если число используется в операции конкатенации строк, оно будет преобразовано в строку:

```
x=100;
str=' руб.';
document.write('x+str='+(x+str)+'
'+typeof(x+str)+'<br>');
```

На практике, чтобы преобразовать число в строку, достаточно сложить его с пустой строкой:

```
n_as_string = n + '';
```

Функция String(number) явно преобразовывает число number в строку.

```
y_as_string = String(y);
```

Для явного преобразования числа в строку можно для объекта number вызвать метод toString().

```
z=11;
z_as_string = z.toString();
```

При этом примитивные типы чисел автоматически преобразуются в объекты типа number, благодаря чему можно воспользоваться методом toString(). Метод toString() может принимать один необязательный аргумент, который определяет основание системы счисления, в которой записано переводимое число (от 2 до 36). Если основание системы счисления не указывается, по умолчанию она предполагается равной 10.

```
z=7;
z_as_binary_string = z.toString(2);
document.write('z_as_binary_string = '+z_as_binary_string
+'  ' + typeof(z_as_binary_string) + '<br>';
```

Метод toFixed() класса number преобразует число в строку и отображает заданное число знаков после запятой.

```
var n = 1234.56789;
n.toFixed(2);
```

Метод toExponential() класса number выполняет преобразование числа в экспоненциальную форму с одним знаком перед точкой и с заданным числом десятичных знаков после точки.

```
n.toExponential(1);
```

Метод toPrecision() класса number используется для отображения определенного числа значащих разрядов. Он возвращает строку с экспоненциальным представлением числа, если заданного количества значащих разрядов недостаточно для точного отображения целой части числа.

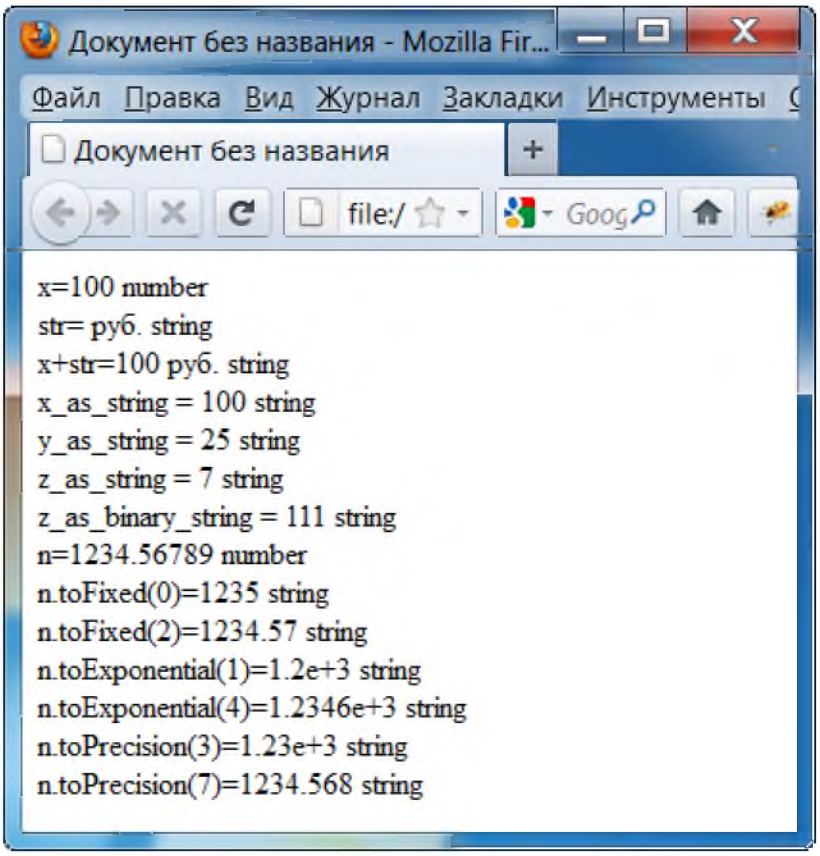
```
n.toPrecision(4);
```

Все три метода: toFixed(), toExponential() и toPrecision() – корректно выполняют округление результата.

Пример преобразования числа в строку:

```
<script>
x=100;
document.write('x='+x+'  '+typeof(x)+'<br>');
str=' руб.';
document.write('str='+str+'  '+typeof(str)+'<br>');
document.write('x+str='+(x+str)+'   ' + typeof(x+str) +
'<br>');

x_as_string = x + '';
document.write('x_as_string  =  '+x_as_string  +'    ' +
typeof(x_as_string) +'<br>');

y=25;
y_as_string =String(y);
document.write('y_as_string  =  '+y_as_string  +'    ' +
typeof(y_as_string) +'<br>');

z=7;
z_as_string =z.toString();
document.write('z_as_string  =  '+z_as_string  +'    ' +
typeof(z_as_string) +'<br>');
z_as_binary_string =z.toString(2);
document.write('z_as_binary_string      =      '      +
z_as_binary_string +'   ' + typeof(z_as_binary_string) +
'<br>');

var n = 1234.56789;
document.write('n='+n+'  '+typeof(n)+'<br>');
document.write('n.toFixed(0)='+n.toFixed(0)+'      '   +
typeof(n.toFixed(0)) + '<br>');
document.write('n.toFixed(2)='+n.toFixed(2)+'      '   +
typeof(n.toFixed(2)) +'<br>');

document.write('n.toExponential(1)=' + n.toExponential(1)
+ '  ' + typeof(n.toExponential(1)) + '<br>');
document.write('n.toExponential(4)=' + n.toExponential(4)
+ '  ' + typeof(n.toExponential(4)) + '<br>');

document.write('n.toPrecision(3)='+n.toPrecision(3)+ '  '
+ typeof(n.toPrecision(3))+'<br>');
```

```
document.write('n.toPrecision(7)='+n.toPrecision(7)  +  '
' + typeof(n.toPrecision(7))+'<br>');
</script>
```



Преобразование строк в числа

Когда строка используется как операнд в числовом выражении, она автоматически преобразуется в число. Исключение составляет сумма операндов. В этом случае плюс воспринимается как знак конкатенации строк.

```
product = "17" * "2"; //34
sum = 17 + '2'; //"172"
```

При необходимости преобразовать строку в число достаточно вычесть из строки значение 0.

```
n = '123'-0;//123
```

Для преобразования строки в число можно воспользоваться конструктором Number(str). При этом в качестве строки str может быть представлено число в строковом формате, допускается наличие ведущих и конечных символов пробела, появление других нецифровых символов не допускается.

```
n = Number('345'); //345
n = Number(' 2.34   '); //2.34
n = Number('567x'); //NaN
```

Более гибкий способ преобразования обеспечивается функциями parseInt() и parseFloat(). Эти функции преобразуют и возвращают произвольные числа, стоящие в начале строки, игнорируя любые нецифровые символы, расположенные после числа. Функция parseInt() выполняет только целочисленное преобразование, а parseFloat() может преобразовывать как целые, так и вещественные числа. Если строка начинается с символов 0x или 0X, функция parseInt() интерпретирует строку как шестнадцатеричное число. В качестве второго аргумента функция parseInt() может принимать основание системы счисления (числа в диапазоне от 2 до 36). Если методы parseInt() и parseFloat() не могут выполнить преобразование, они возвращают значение *NaN*.

```
parseFloat("3.14 метров"); // 3.14
parseInt("3 орешка для Золушки"); // 3
parseInt("12.34"); // 12
parseInt("0xFF"); // 255
parseInt("077",8); // 63
parseInt("077",10); // 77
parseInt("two"); // NaN
parseInt("$1"); // NaN
```

Преобразование логических значений

Булевские, или логические, значения автоматически преобразуются в значения других типов. В какой тип происходит трансформация, зависит от того, в каком контексте используется логическое значение. Если логическое значение используется в числовом контексте, тогда значение true преобразуется в 1, а false – в 0.

```
y=5+true; // y=6
y=1-false; // y=1
```

Если логическое значение используется в строковом контексте, тогда значение true преобразуется в строку 'true', а false – в строку 'false'.

```
'логическое значение истина ' + true ; //'логическое
значение истина true'
```

1 равно true, а 0 равно false. Любое другое значение не равно ни true, ни false.

```
1==true // true
1==false // false
0==true // false
0==false // true
17==true // false
17==false // false
NaN==true // false
NaN==false // false
```

Когда в качестве логического значения используется число, оно преобразуется в значение true, если не равно значениям 0 или NaN, которые преобразуются в логическое значение false.

```
1&&true//true
17&&true//true
0&&true//0
NaN||true//true
NaN||false//false
```

Если в качестве логического значения используется не пустая строка, то она преобразуется в значение true, пустая строка преобразуется в false.

```
"abc"&&true//true
```

Специальные значения null и undefined преобразуются в false, а любые функция, объект или массив, значения которых отличны от null, преобразуются в true.

Для явного преобразования используется функция Boolean(). Результат преобразований функции Boolean():

- следующие значения в результате преобразования дают значение false: undefined, null, 0, -0, NaN, "";
- значение false возвращается без изменения;
- все остальные значения в результате преобразования дают значение true.

```
Boolean(0&&true) // false
Boolean(""&&true) // false
```

Другой способ явного преобразования заключается в использовании двойного оператора логического отрицания:

```
!!12; // true
!!0 // false
!!NaN // false
```

Преобразование специального значения null

Если значение null используется в логическом контексте, оно преобразуется в false, в числовом контексте – в значение 0, а в строковом контексте – в строку 'null'.

```
null&&true // null
Boolean(null&&true) // false
null||false // false
'null-12=' + (null-12) // 'null-12=-12'
'специальное значение ' + null // 'специальное значение
null'
```

Преобразование специального значения undefined

Если значение undefined используется в логическом контексте, оно преобразуется в значение false, в числовом контексте – в значение NaN, а в строковом – в строку 'undefined'.

```
undefined&&true // undefined
Boolean(undefined && true) // false
undefined-12 // NaN
'специальное значение' + undefined // 'специальное значение undefined'
```

Операторы JavaScript

В таблице перечислены операторы JavaScript в порядке убывания приоритета.

Приоритет оператора	Оператор	Типы операндов	Выполняемая операция
20	()	Выражение	Скобки для группировки в выражениях
19	.	Объект	Обращение к свойству
	[]	Массив, целое число	Индексация массива
	()	Функция, аргументы	Вызов функции
	new	Вызов конструктора	Создание нового объекта
18	++	Выражение	Постфиксный инкремент
	--	Выражение	Постфиксный инкремент
17	++	Выражение	Префиксный инкремент
	--	Выражение	Префиксный инкремент
	-	Число	Унарный минус
	+	Число	Унарный плюс
	~	Целое число	Поразрядная инверсия
	!	Логическое значение	Логическое отрицание
	delete	Левостороннее значение	Аннулирование определения свойства (унарный)
	typeof	Любой	Возвращает тип данных (унарный)
	void	Любой	Возвращает неопределенное значение (унарный)
16	**	Числа	Возведение в степень
15	*, /, %	Числа	Умножение, деление, целочисленный остаток от деления
14	+, -	Числа	Сложение, вычитание
	+	Строки	Конкатенация строк
13	<<	Целые числа	Сдвиг влево

Приоритет оператора	Оператор	Типы операндов	Выполняемая операция
	>>	Целые числа	Сдвиг вправо с расширением знакового разряда
	>>>	Целые числа	Сдвиг вправо с дополнением нулями
12	<, <=	Числа или строки	Меньше, меньше либо равно
	>, >=	Числа или строки	Больше, больше либо равно
	instanceof	Объект, конструктор	Проверка на принадлежность к определенному типу
	in	Строка, объект	Проверка вхождения в объект
11	==	Любой	Проверка на равенство (допускается преобразование типов)
	!=	Любой	Проверка на неравенство
	===	Любой	Проверка на идентичность (не допускается преобразование типов)
	!==	Любой	Проверка на неидентичность
10	&	Целые числа	Поразрядная операция И
9	^	Целые числа	Поразрядная операция исключающего ИЛИ
8	|	Целые числа	Поразрядная операция исключающего ИЛИ
7	&&	Логические значения	Логическое И
6	||	Логические значения	Логическое ИЛИ
5	??	Оператор нулевого слияния	Логический оператор, возвращает значение правого операнда, когда значение левого операнда равно null или undefined, в противном случае возвращает значение левого операнда
4	?:	Логическое значение, любое, любое	Условный тернарный оператор
3	=	Левостороннее значение, любое	Присваивание
	*=, /=, %=, +=,-=, <<=, >>=, >>>=, &=, ^=, |=	Левостороннее значение, любое	Присваивание с операцией
2	yield	Любое	Функция паузы, используется для остановки и возобновления функций-генераторов
1	,	Любой	Множественное вычисление выражений

```
/* Оператор () */
(10+2)*3 // 36
/* Оператор =, .,[], (), new, delete, typeof, instanceof,
in */
function Person(firstName, lastName, age, eyeColor) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age,
  this.eyeColor = eyeColor;
}
let person = new Person("Василий", "Теркин", 21,
"голубой");
delete person["age"];   // or delete person.age;
typeof person // "object"
person instanceof Person // true
"firstName" in person // true
/* Оператор ++, -- */
y = 6, x = 10, x = ++y; // x=7, y=7
y = 6, x = 10, x = y++; // x=6, y=7
y = 6, x = 10, x = y--; // x=6, y=5
y = 6, x = 10, x = --y; // x=5, y=5
~ 5 // -6
!Boolean(17)//false
!17//false
!true//false
5/2 // 2.5
5%2 // 1
8<<1 // 16
8>>1 // 4
5>>>1 // Сдвиг с дополнением нулями
```

В JavaScript оператор void вычисляет переданное ему выражение. При этом, независимо от того, какое именно выражение вычисляется, void всегда возвращает undefined.

```
let x = void 12; // x === undefined
```

Использование void позволяет вернуться из функции без возврата какого-либо значения (undefined), но с вызовом, например, новой функции (коллбэка):

```
// возврат чего-нибудь, кроме undefined, приведет
к аварийной остановке приложения
function foo(Callback) {
  if(condition()) {return void Callback();}}
5**2 // 25
```

Оператор «запятая» предоставляет возможность последовательно вычислять несколько выражений, разделенных запятой. Каждое выражение выполняется, но возвращается результат только последнего.

```
let a = (10 + 2, 33 + 14);
alert( a ); // 47 (результат вычисления 33 + 14)
```

Такой код, как в предыдущем примере, не имеет смысла, но оператор «запятая» широко используется в составе более сложных конструкций, чтобы сделать код компактнее, например в операторе for. Ниже приведен код для нахождения и вывода первых 10 чисел Фибоначчи.

```
document.write(0 + "  " + "1" + " ");
  /*Начиная с третьего,
    находим числа Фибоначчи в цикле, */
for(x1 = 0, x2 = 1, i = 3; i < 11 ; i++){
  x3 = x1 + x2;
  document.write(x3 + "  ");
  x1 = x2;
  x2 = x3;}
```

В первом разделе заголовка цикла записаны и выполняются последовательно три оператора. Используя оператор «запятая», этот же код можно записать таким образом, чтобы тело цикла было пустым. Однако «можно» не значит «лучше», особенно в плане легкости восприятия.

```
for(x1 = 0, x2 = 1, i = 3; i < 11 ;x3 = x1 + x2,
document.write(x3 + "  "), x1 = x2, x2 = x3, i++){}
```

if … else, else if

Условный оператор if … else имеет вид такой же, как и в java и C++.

```
if (условие)
оператор 1;
[else оператор 2;]
if (условие){
  блок кода 1
}
[else {
   блок кода 2
 }]
```

Если в случае выполнения (или невыполнения) условия необходимо выполнить группу операторов, то они берутся в фигурные скобки; else может отсутствовать.

Если необходимо проверить несколько уточняющих условий, с каждым из которых связан блок кода, можно воспользоваться лесенкой else if. Формально это не отдельный оператор JavaScript, а распространенный стиль программирования, состоящий в применении повторяющихся инструкций if … else.

```
if (n == 1) {
```

```
// блок кода 1
}
else if (n == 2) {
// блок кода 2
}
else if (n == 3) {
// блок кода 3
}
else {
// Если все остальные условия else не выполняются,
исполняем блок кода 4
}
```

Лесенка else if – это просто последовательность инструкций if, где каждая инструкция if является частью конструкции else предыдущей инструкции. Стиль else if предпочтительнее и понятнее записи в синтаксически эквивалентной форме.

```
if (n == 1) {
// блок кода 1
}
else {
if (n == 2) {
// блок кода 2
}
else {
if (n == 3) {
// блок кода 3
}
else {
// Если все остальные условия else не выполняются,
исполняем блок кода 4
}
}
}
```

Условный оператор `if (...)` вычисляет выражение в скобках и преобразует результат к логическому типу.

Напомним, что, согласно правилу преобразования типов:

- число 0, пустая строка "", null, undefined и NaN преобразуются в false.
- остальные значения преобразуются в true.

Например, кусок кода в фигурных скобках никогда не выполнится при условии `if (0) { ... }`, а при условии `if (1) { ... }` будет выполняться всегда.

? :

Тернарный оператор условия

```
переменная = (условие) ? значение1 : значение2;
```

Если выполняется условие, то в переменную заносится «значение1», иначе – «значение2». Круглые скобки необязательны, но делают код читабельнее.

switch

Оператор выбора switch:

```
switch (переменная или выражение) {
case значение1: // Выполняется, если переменная или
выражение совпадает со значением1
операторы блока кода 1
break; // остановка и выход за пределы switch
case значение2: // Выполняется, если переменная или
выражение совпадает со значением2
операторы блока кода 2
break; // остановка и выход за пределы switch
…
case значениеN: // Выполняется, если переменная или
выражение совпадает со значениемN
операторы блока кода N
break; // остановка и выход за пределы switch
[default: // Выполняется, если переменная или выражение
не совпадает ни с одним из перечисленных в case значений
операторы блока кода N+1
break;] // остановка и выход за пределы switch
}
```

Оператор switch сначала вычисляет значение выражения, а затем переходит на case, соответствующий этому значению. Если case найден, исполняется блок кода, начиная с первого оператора, следующего за case. Если соответствующий case не найден, то выполняется блок кода, расположенный за default. Если метки default нет, то блок кода пропускается целиком. Ключевое слово break в конце каждого блока case приводит к передаче управления в конец инструкции switch. Конструкции case в switch задают только начальную точку исполняемого кода, но не задают конечной точки. В случае отсутствия break оператор switch начинает исполнение блока кода с case, соответствующего значению выражения, и продолжает исполнение до тех пор, пока не дойдет до конца блока (не встретит break или не выполнит весь код в switch). При использовании switch внутри функции можно вместо break использовать return.

Приведем пример использования оператора switch. Функция convert преобразует значение в строку способом, зависящим от типа значения: число преобразуется в строку, содержащую шестнадцатеричную запись числа; к строке дописываются кавычки, а логическое значение записывается в верхнем регистре.

```
<script>
function convert(x) {
switch(typeof(x)) {
case 'number': // Преобразуем число в шестнадцатеричное
целое
return x.toString(16);
case 'string': // Возвращаем строку, заключенную в кавычки
return '"' + x + '"';
case  'boolean':  return  x.toString().toUpperCase();//
Преобразуем в TRUE или FALSE в верхнем регистре
default: // Любой другой тип преобразуем обычным способом
return x.toString()
}
}
document.write(convert(200)+'<br>');
document.write(convert('qwerty')+'<br>');
document.write(convert(true));
</script>
```

В качестве результата получим

```
c8
"qwerty"
TRUE
```

while

Цикл с предусловием с неизвестным количеством повторений представлен оператором while. Синтаксис оператора цикла с предусловием while совпадает с синтаксисом C++ и Java.

```
while (условие){
оператор;}
```

Чтобы цикл не продолжался вечно, условие должно изменяться, например в теле цикла.

```
let i = 3;
while (i) { // когда i будет равно 0, условие станет ложным
и цикл остановится
  let years=prompt("Вам дается три попытки ввода ответа.
Попытка " + (3-i+1) + ":\nСколько дней в високосном
году?",'360');
```

```
  if(years==366){alert("Правильно! Угадали с " + (3-i+1)
+ " попытки."); break;}
  i--;
}
if(i==0)alert("Стыдно не знать таких вещей!");
```

do while

Для организации цикла с предусловием существует оператор

```
do
  оператор;
while (условие);
```

Этот цикл проверяет условие продолжения после выполнения цикла, поэтому тело цикла выполняется как минимум один раз, независимо от того, выполняется условие изначально или нет. Ниже приведен код примера игры, где пользователь может проверить свою удачу. Проверка удачливости в любом случае проверяется по крайней мере один раз.

```
alert(" Проверим кто удачливее! Кто задумает большее число
от 0 до 1, тот и выиграл.");
let myWin=0, yourWin=0, str, yourAnswer;
do {
  let myNumber = Math.random()*10;
  let yourNumber=prompt("Какое число задумал ты?");
  if(yourNumber>myNumber)
    {str = "Ты победил!";yourWin++;}
  else if (yourNumber===myNumber)
    {str = "Ничья!";yourWin++;myWin++;}
  else {str = "Я выиграл!";myWin++;}
  yourAnswer=prompt(str + "Продолжим (да/нет)?");
} while(yourAnswer == "да");
let win=yourWin-myWin;
if(win) {str = "Тебе везет чаще!"}
else if (yourNumber===myNumber)
  {str = "Удача любит нас одинаково!";}
else {str = "Мне везет чаще!";}
alert(str);
```

for

Цикл с известным количеством повторений for.

```
for (блок инициализации; проверка условия; изменение
счетчика цикла)
оператор;
```

Сначала один раз выполняется блок инициализации, затем проверяется условие, выполняется оператор тела цикла, изменяется счетчик и снова проверяется условие, выполняется оператор тела цикла, изменяется счетчик и т. д.

```
let pet = ["кот", "собака", "попугай", "хомяк"];
for (var i = 0; i < pet.length; i++){
  alert(pet[i] + " - это домашнее животное.");}
```

for in

Оператор цикла for in используется для перебора в случайном порядке перечисляемых свойств объекта.

```
for (переменная in объект)
оператор;
```

Например, просмотреть все свойства объекта document можно следующим образом.

```
for(prop in document)
document.write("имя:  "  +  prop  +  "; значение:  "
+document[prop] + "<br>");
```

При помощи for in можно легко скопировать имена всех свойств объекта в массив.

```
let ob = {x:1, y:2, z:3};
let a = new Array();
let i = 0;
for(a[i++] in ob); // тело цикла  пусто
for(prop in a)
document.write("a[" + prop + "] = " + a[prop] + "<br>");
```

Метки

Любой оператор может быть помечен указанным перед ним именем идентификатора и двоеточием.

```
идентификатор: оператор;
```

Имена меток могут быть любыми допустимыми в JavaScript идентификаторами. Метки применяются при необходимости выхода из вложенных операторов с глубиной вложенности больше двух.

continue

Позволяет перейти к следующей итерации цикла еще до завершения выполнения всех операций предыдущей итерации. Может использоваться в любом цикле.

Например, приведенный ниже код использует continue в цикле for, чтобы выводить только нечетные значения.

```
for (let i = 0; i < 10; i++) {
  // если true, пропустить оставшуюся часть тела цикла
  if (i % 2 == 0) continue;
  alert(i);}
```

Для четных значений i директива continue прекращает выполнение текущей итерации и управление передается проверке условия цикла. Если условие цикла true, то выполняется следующая итерация и т. д. Таким образом, alert вызывается только для нечетных значений.

break

Специальная директива break приводит к выходу из цикла или из операторов switch. JavaScript допускает указание имени метки за ключевым словом break.

```
break: имя_метки;
```

Когда break используется с меткой, происходит переход на именованный оператор, внешний по отношению к break. Между ключевым словом break и именем метки перевод строки не допускается. Если разбить строку кода между ключевым словом break и следующей за ним меткой, интерпретатор JavaScript предположит, что имелась в виду простая форма break без метки, и добавит точку с запятой.

Директива break часто используется в бесконечных циклах, где условие, по которому нужно прерваться, находится в теле цикла. В следующем примере ведется подсчет суммы чисел, которые вводит пользователь. Числа суммируются до тех пор, пока посетитель их вводит, а затем выводится результат.

```
let sum = 0;
while (true) {
  let value = +prompt("Введите число", '');
  if (!value) break; // (*)
  sum += value;
}
alert( 'Сумма: ' + sum );
```

Функции

В программировании есть правило: «Программный код, который используется два и более раз, должен быть оформлен в функцию».

Функции в JavaScript можно рассматривать как специальный тип объекта, т. е. функция – это объект, с которым связан исполняемый код, определенный в JavaScript-программе или в интерпретаторе JavaScript. Функции могут быть встроенными, например, в коде ниже вызывается функция sin стандартного встроенного объекта Math.

```
document.write('sin(0)=' + Math.sin(0)); // sin(0)=0
```

Есть пользовательские функции. Например, для определения максимального из двух чисел можно создать следующую функцию.

```
function max(x,y){
  return x>y ? x : y;
}
document.write('max(8,-3)=' + max(8,-3) + '<br>'); //
max(8,-3)=8
```

В JavaScript функции представляют собой тип данных, их можно определять и вызывать. Вследствие этого они представляют собой значения и могут храниться в переменных, массивах и объектах, а также передаваться в качестве аргументов другим функциям. Поскольку функции представляют собой значения, такие же, как числа или строки, они могут присваиваться свойствам объектов.

Чтобы создать функцию, ее надо объявить. Это можно сделать несколькими способами. Первый способ еще называют классическим. Вначале записывают ключевое слово function, после него – имя функции, затем в круглых скобках через запятую перечисляют список параметров, после списка параметров в фигурных скобках помещают тело функции. Возвращаемое значение помещают в оператор return.

```
function имя(параметры) {
тело
return результат}
```

Если функция ничего не возвращает, то return можно опускать. Если использовать return без значения, то это приведет к немедленному выходу из функции. Отсюда следует, что нельзя размещать return и возвращаемое значение на разных строках.

```
function Hello(person) {
  alert("Hello, "+ person);
}
```

Программный код, расположенный в теле функции, выполняется не в момент объявления функции, а в момент ее вызова. При вызове указывается имя функции и параметры.

```
Hello("Маша")
```

Функции, объявленные первым, классическим способом, создаются интерпретатором до начала выполнения кода, поэтому их можно вызывать еще до объявления, но только в области их видимости.

```
activWork();// Ошибка!
Hello("Маша"); // Ошибки не будет
function Hello(person) {
  alert("Hello, "+ person);
    activWork();
  function activWork() {
    alert("Работай активнее!");  }
}
```

Так, например, вызов функции activWork() первый раз приведет к ошибке, поскольку вызов производится вне области видимости функции. А вот вызов функции Hello() до объявления происходит в области ее видимости, поэтому ошибки не вызовет. Аналогично вызов activWork до объявления внутри функции Hello не вызовет ошибки по той же причине.

Второй способ объявления функций – это когда объявление функции является частью какого-либо выражения, например присваивания.

```
Hello = function (person) {alert("Hello, "+ person);}
Hello("Маша");
```

В этом случае функции создаются в момент выполнения кода, поэтому их нельзя вызывать до их объявления.

Переменные, объявленные внутри функции с помощью ключевого слова let, являются локальными и доступны только внутри этой функции.

Внутри функций есть доступ к внешним переменным, и можно изменить значение внешней переменной прямо внутри функции.

```
let person = "Маша";
function Hello() {
  alert("Hello, "+ person);// Маша
  person = "Катя"; // изменяем значение внешней переменной
  let message = 'Привет, ' + person;
  alert(message);
}
alert(person); // person - Маша перед вызовом функции

Hello();
alert(person);  // person - Катя, значение внешней
переменной было изменено функцией
```

Параметры передаются в функцию копией, поэтому изменение значений параметров внутри функции не приведет к их изменению вне функции.

```
function Hello(person) {
  person = "Катя"; // изменяем значение параметра
  alert("Hello, "+ person);// Hello, Катя
}
let person = "Маша";
```

```
alert(person); // Маша перед вызовом функции
Hello(person);
alert(person);  //  Маша,  значение  внешней  переменной
осталось прежним
```

Если параметр не указан, то его значением становится undefined.

Если функция вызывается как метод объекта, то параметр this – это ссылка на объект, который производит вызов функции. Если функция не является методом объекта, тогда this равняется undefined.